МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ, ПРОГРАММИРОВАНИЕ

УДК 004.4, 378.147

ОСОБЕННОСТИ ПРОГРАММНОЙ РЕАЛИЗАЦИИ СИСТЕМЫ МОНИТОРИНГА И ФОРМИРОВАНИЯ ПАКЕТА ДОКУМЕНТОВ ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММ ПРАКТИЧЕСКОЙ ПОДГОТОВКИ ВУЗА

Л. В. Апалькова

Воронежский государственный университет

Аннотация. В статье представлен опыт создания десктопного приложения для автоматизации мониторинга практической подготовки в вузе. Описаны особенности консолидации данных из разнородных источников и формирования на их основе отчетных документов и аналитических дашбордов. В данной работе рассматриваются особенности программной реализации системы. Особое внимание уделено обоснованию выбора технологического стека: Python, Tkinter, Pandas, python-docx и Power BI Embedded. Подчеркивается практическая ценность решения для сотрудников учебно-методических подразделений.
Ключевые слова: автоматизация, практическая подготовка, мониторинг, аналитика, отчетность, десктопное приложение, технологический стек.

Введение

В работе в качестве предметной области рассматривается процесс организации и проведения практической подготовки в вузах. Процесс мониторинга и анализа результативности программ практической подготовки основан на обработке данных из разнородных источников, таких как реестры ОПОП, учебные планы, контингент студентов и отчетные документы. Задача автоматизации решается за счет создания единого реестра практик, на основе которого формируются регламентированные отчетные документы и строится аналитическая визуализация. Программная реализация такого инструментария предполагает разработку десктопного приложения, предназначенного для использования сотрудниками, обеспечивающими организацию и реализацию ОП, и сотрудниками учебно-методического управление вуза (УМУ).

1. Постановка задачи

Ставится задача: разработать систему, которая позволит автоматизировать формирования отчетности и осуществлять мониторинг и аналитику ключевых показателей программ практической подготовки вуза на основе консолидированных данных о практиках.

При формировании отчетности и аналитики входными источниками данных являются: реестр ОПОП, учебные планы, выгрузки из информационной системы вуза о контингенте и распределения нагрузки факультета по ППС, база МТО для прохождения практики и результаты аттестации. С учетом разнородности исходных данных, решение должно быть представимо в виде комплекса взаимосвязанных сервисов, формирующих интерактивный аналитический отчет (дашборд) и итоговые документы – документы о направлении на практику, отчеты руководителя практики и факультета.

Концептуальная модель разрабатываемого решения представлена на рис. 1.

2. Функциональные требования и особенности технического задания на разработку

Согласно постановке задачи, реализуемый сервис должен обеспечивать следующую функциональность:

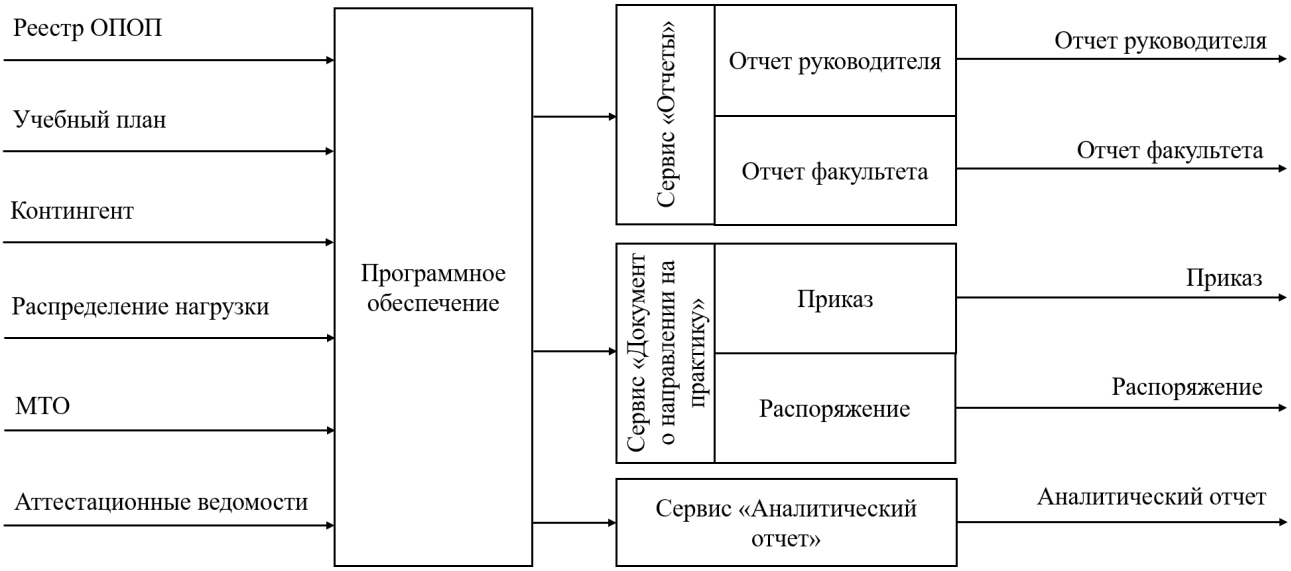

Рис. 1. Концептуальная модель системы

1) загрузка и обновление внутреннего реестра практической подготовки (реестра ПП) на основе исходных данных;

2) сохранение сформированного реестра ПП для последующего анализа;

3) формирование документов о направлении на практику с возможностью выборки данных по периоду, направлению подготовки, группе, виду и названию практики;

4) формирование документа «Отчет руководителя практики» с выбором ответственного руководителя и периода, с возможностью заполнения текстовых полей (проблемы, предложения, формы поощрения);

5) формирование документа «Отчет факультета» с возможностью выборки данных по периоду, направлению подготовки, виду и названию практики;

6) формирование дашборда с визуализацией данных за выбранный период – «Аналитического отчета», включая:

a. подсчет соотношения практик по видам;

b. детализация по кафедрам, направлениям подготовки и руководителям;

c. анализ результатов аттестации студентов;

d. статистику по загрузке аудиторного фонда;

e. мониторинг участия компаний-партнеров в практической подготовке;

7) сохранение сформированных отчетных документов.

Сценарий использования отражены в виде диаграммы, представленной на рис. 2.

Учитывая, что система предназначена для внутреннего использования сотрудниками вуза, она должна быть реализована в виде десктопного или веб-приложения с интерфейсом, адаптированным для работы с документами. Для реализации модуля визуализации должна быть предусмотрена интеграция с платформой Power BI.

3. Архитектура приложения и технологический стек

Для реализации сервиса мониторинга и аналитики практической подготовки в вузах выбрана архитектура десктопного приложения. Приложение будет иметь модульную структуру, включающую модуль работы с данными, модуль формирования отчетов, модуль аналитики и визуализации, а также графический пользовательский интерфейс (GUI). Архитектура предлагаемого решения представлена на рис. 3.

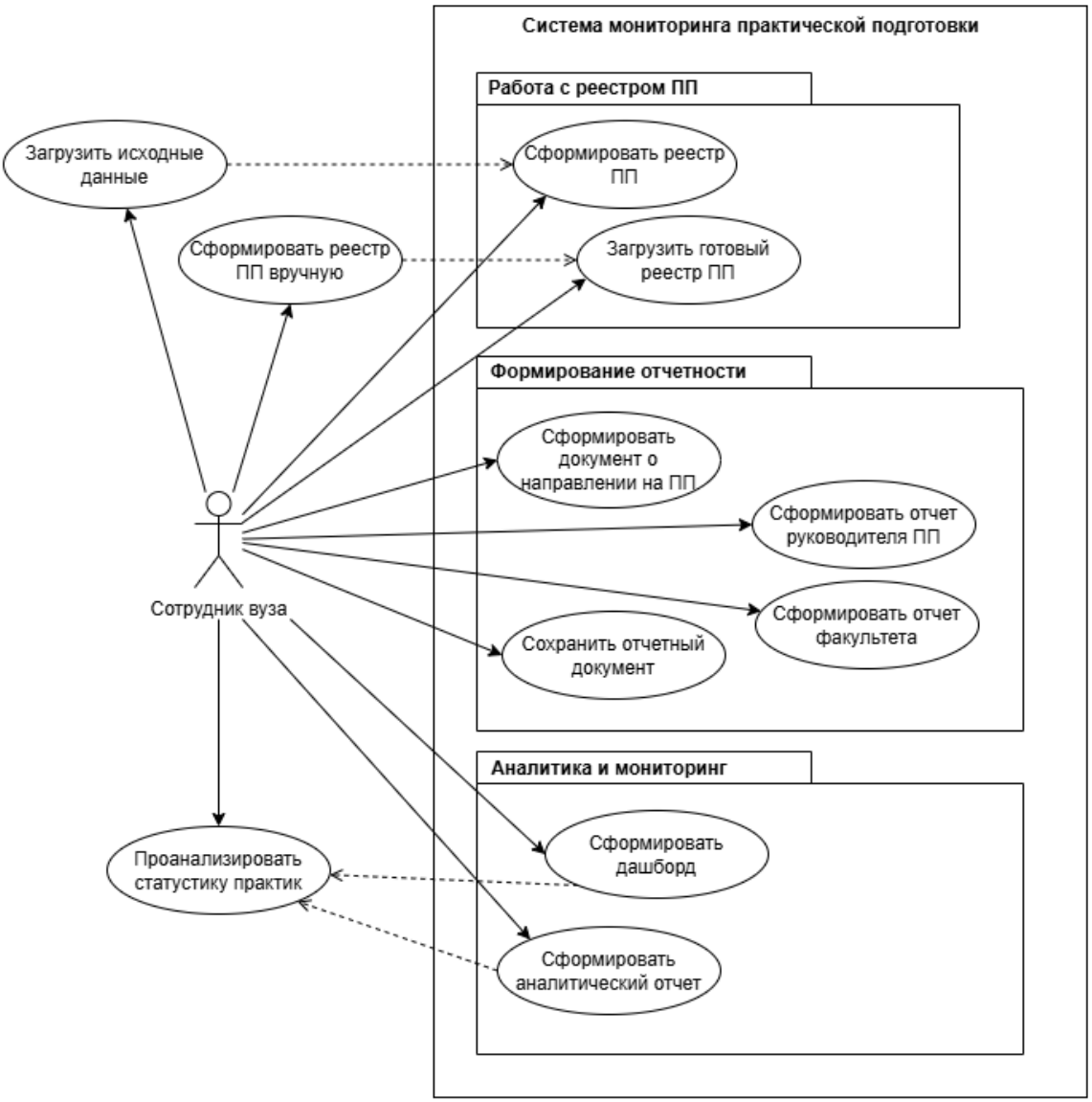

Рис. 2. Сценарии использования сервиса

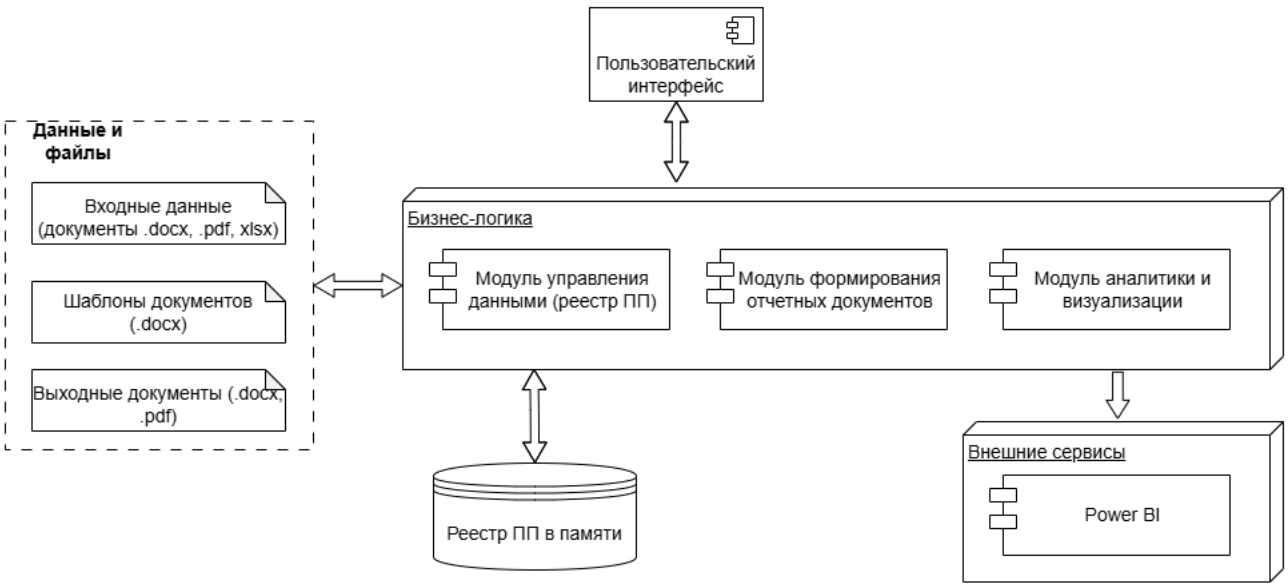

Рис. 3. Архитектура сервиса

Согласно требованиям, описанным выше, технологический стек представим следующими программными средствами:

1) язык программирования Python;
2) библиотека для создания графических интерфейсов Tkinter;
3) библиотека для работы с данными Pandas;
4) библиотека для формирования отчетов в формате .docx — python-docx;
5) библиотека для генерации PDF-документов ReportLab;
6) инструмент для встраивания аналитических дашбордов Power BI Embedded;
7) интерактивная среда разработки Visual Studio Code.

Технологический стек в привязке к компонентам системы представлен на рис. 4.

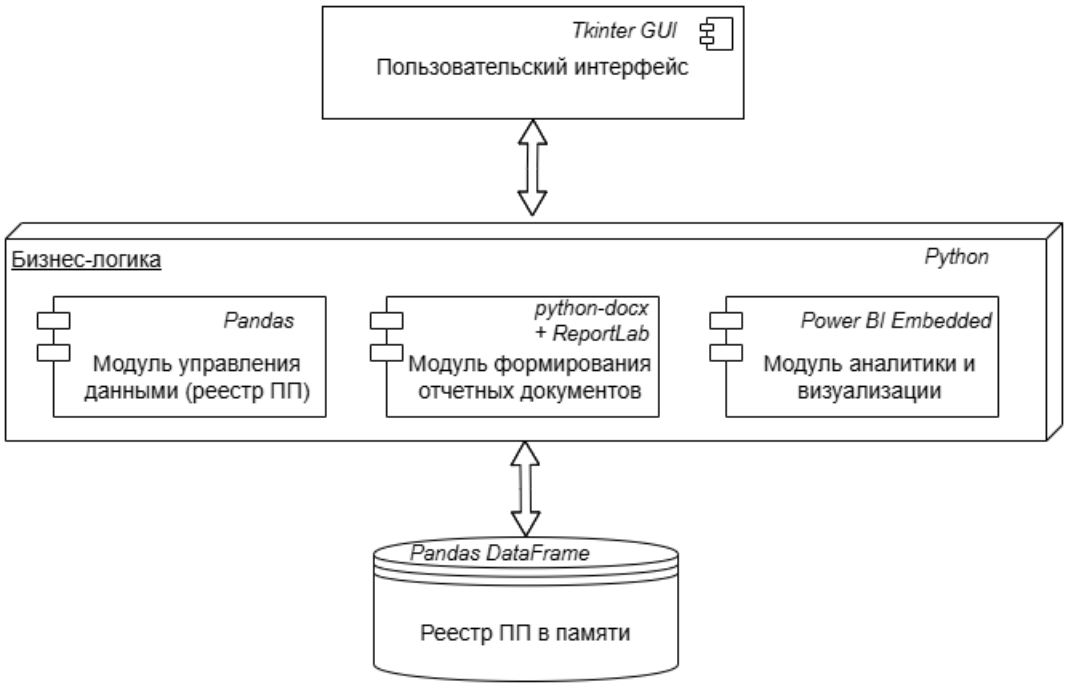

Рис. 4. Технологический стек сервиса

Для разработки ядра и графического интерфейса используется язык программирования Python. Среди доступных фреймворков для GUI был выбран Tkinter. Ключевым фактором стала его доступность — как стандартная библиотека Python, он не требует установки дополнительных компонентов на рабочие компьютеры сотрудников вуза, что значительно упрощает развертывание и поддержку в корпоративной среде вуза.

Модуль управления данными строится в основном на библиотеке Pandas. Ее выбор был обусловлен тем, что исходные данные поставляются в виде таблиц Excel. Pandas эффективно справляется с загрузкой, очисткой и агрегацией этих разнородных данных перед формированием единого реестра практик.

Для генерации отчетных документов в рамках модуля формирования документов используются две библиотеки. С помощью библиотеки python-docx происходит заполнение шаблонов документов (.docx) данными из реестра. Итоговые документы конвертируются в PDF с использованием ReportLab, что гарантирует их единообразное отображение.

Аналитический модуль реализован через интеграцию с платформой Power BI с помощью технологии Power BI Embedded. Это решение позволяет встроить готовые интерактивные дашборды непосредственно в интерфейс приложения. Таким образом, пользователь получает все преимущества данного инструмента визуализации без необходимости осваивать новое ПО или переключаться между окнами.

Заключение

Описанное десктопное приложение призвано решать задачу консолидации разнородных данных о практиках и их последующего преобразования в регламентированные отчетные документы. Интеграция с Power BI обеспечивает наглядность и глубину анализа показателей практической подготовки без необходимости использовать стороннее ПО.

Литература

1. Бесшовная интеграция Microsoft Excel и Word с помощью Python – URL: https://habr.com/ru/companies/skillfactory/articles/553224/ (дата обращения: 06.10.2025).

2. Визуализация данных и создание интерактивных дашбордов – URL: https://apptask.ru/blog/vizualizaciia-dannyx-i-sozdanie-interaktivnyx-dasbordov (дата обращения: 24.11.2025).

3. О порядке организации практической подготовки обучающихся по основным образовательным программам : Инструкция Воронежского государственного университета от 27.11.2020 №10 // ФГБОУ ВО «Воронежский государственный университет». – 2020. – Ст. 2.1.12.

4. *Паклин Н. Б.* Бизнес-аналитика от данных к знаниям / Н. Б. Паклин, В. И. Орешков. – 2-е изд., исправленное. – Санкт-Петербург : Питер, 2013. – с. 706

5. *Садовникова Н. П.* Технологии анализа данных: учебное пособие / Н. П. Садовникова, М. В. Щербаков. – ВолгГТУ. – Волгоград, 2021. – С. 75

6. *Соколов Е. А.* Электронный документооборот вуза / С. Н. Середа, Е. А. Соколов // Перспективы развития информационных технологий. – 2012. – № 9. – С. 39–44

УДК 004.051

СЖАТИЕ МЕДИА-ДАННЫХ С ИСПОЛЬЗОНИЕМ ВИДЕОКОДЕКОВ H.264, H.265, VP9 И AV1

А. А. Арутюнян

Воронежский государственный университет

Аннотация. В работе проведено экспериментальное сравнение видеокодеков H.264, H.265, VP9 и AV1 при сжатии видеоданных различного типа. Исследование выполнено на 40 видеороликах с различными характеристиками. Для каждого кодека оценивались размеры файлов, битрейт, качество по метрикам SSIM и PSNR, а также время кодирования. Для автоматизации процесса использовались набор свободных библиотек с открытым исходным кодом FFmpeg и специализированный Python. Полученные результаты позволили определить преимущества и недостатки кодеков при динамичных и статичных сценах. На основе анализа сформулированы рекомендации по выбору оптимального алгоритма сжатия.

Ключевые слова: видео-кодирование; сжатие данных; H.264; H.265; HEVC; VP9; AV1; битрейт; PSNR; SSIM; качество видео; мультимедиа; алгоритмы сжатия.

Введение

В современном мире объём мультимедийных данных растёт с большой скоростью. Видеоконтент стал неотъемлемой частью сетевой инфраструктуры, от потоковых сервисов и видеоконференций до социальных сетей и онлайн-обучения. При этом передача и хранение видео требуют значительных вычислительных и сетевых ресурсов. Эффективное сжатие видеоинформации является ключевым фактором, определяющим экономичность и качество современных медиа-систем. Без эффективных алгоритмов кодирования невозможно обеспечить массовое распространение высококачественного видео в сети Интернет, особенно в разрешениях Full HD и 4K.

Цель настоящего исследования заключается в проведении экспериментального сравнения эффективности современных стандартов видео-кодирования: H.264/AVC, H.265/HEVC, VP9 и AV1. Анализ охватывает такие параметры, как итоговый размер видеофайлов, средний битрейт, вычислительная сложность кодирования и показатели объективного качества (PSNR и SSIM). Практическая часть работы выполнена с использованием программного комплекса FFmpeg и разработанного на языке Python скрипта, обеспечивающего автоматическую конвертацию и последующую оценку качества сжатого видео.

1. Постановка задачи

Основная задача работы заключается в исследовании соотношения между степенью сжатия и качеством изображения для разных видеокодеков.

В ходе исследования провести оценку конвертации исходных видеофайлов с использованием четырех видеокодеков H.264, H.265, VP9 и AV1. Необходимо оценить: итоговый размер (степень сжатия), скорость обработки (средний битрейт), длительность кодирования, а также метрики качества SSIM (Structural Similarity Index) и PSNR (Peak Signal-to-Noise Ratio), отражающие близость полученного изображения к оригиналу.

2. Общие принципы сжатия данных

Сжатие видео основано на устранении избыточности, присутствующей в мультимедийных данных. Избыточность бывает статистической, пространственной, временной и перцептив-

ной. Статистическая избыточность проявляется в том, что соседние пиксели или блоки изображения часто имеют сходные значения.

Пространственная избыточность устраняется с помощью внутрикадрового предсказания, при котором значения пикселей в текущем блоке вычисляются на основе уже известных данных из соседних областей.

Временная избыточность характерна для последовательных кадров, где большая часть изображения остаётся неизменной. Для её устранения используются межкадровое предсказание и компенсация движения, при которых кодируется не сам кадр, а разница между ним и предыдущими. Таким образом, при наличии схожих кадров можно существенно сократить объём данных, передавая только информацию о смещениях объектов и небольшие различия.

Перцептивное сжатие учитывает особенности человеческого зрения: человек более чувствителен к изменениям яркости, чем цвета, и хуже различает высокочастотные детали. Поэтому в большинстве кодеков используется субдискретизация цветности и квантование, при котором малозаметные компоненты изображения кодируются с меньшей точностью. В совокупности эти принципы обеспечивают значительное снижение объёма видеопотока без заметного ухудшения визуального качества.

Современные стандарты кодирования, такие как H.264, H.265, VP9 и AV1, применяют сходную архитектуру, основанную на блочном преобразовании, предсказании движения и энтропийном кодировании. Отличия между ними заключаются в степени гибкости, глубине и адаптивности применяемых алгоритмов.

3. Сравнительный анализ

Результаты экспериментального исследования представлены в четырёх сериях измерений, отражающих поведение кодеков с разными видеоматериалами. Для каждого кодека оценивались размер файла, битрейт, метрики качества и время кодирования. Результаты проводились на 40 видеороликах различного характера, что позволило оценить стабильность алгоритмов при изменении структуры контента.

Видео ролики разделены на 4 различные группы по 10 штук, для анализа влияния исходных свойств видео на результат сжатия.

Первая группа — Динамичные видео, разрешение 480p, 30 кадров в секунду, длительностью 10 секунд.

Вторая группа — Динамичные видео, разрешение 1080p, 30 кадров в секунду, длительностью 10 секунд.

Третья группа — Динамичные видео, разрешение 1080p, 30 кадров в секунду, длительностью 1 минута.

Четвертая группа — Статичные сцены, разрешение 4к, 30 кадров в секунду, длительностью 10 секунд.

Таблица 1

Результаты измерений сжатия группы «dynamic 480p 30fps 10 сек»

Кодек	Метрика	Размер (MB)	Битрейт (Мбит/с)	SSIM	PSNR	Время кодирования (с)
h264	min	0.715	0.59	0.9892	41.97	0.72
	max	1.33	1.10	0.9951	45.79	1.07
	avg	1.05	0.88	0.9914	44.25	0.93
hevc	min	0.496	0.41	0.9829	40.11	3.08
	max	0.845	0.69	0.9929	44.42	4.60

	avg	0.689	0.56	0.9863	42.42	3.94
vp9	min	0.792	0.65	0.8760	21.29	3.81
	max	1.58	1.32	0.9704	27.09	6.55
	avg	1.24	1.03	0.9255	24.66	5.58
av1	min	0.702	0.58	0.8785	21.30	15.35
	max	1.42	1.19	0.9742	27.12	29.36
	avg	1.10	0.91	0.9281	24.67	23.07

Таблица 2

Результаты измерений сжатия группы «dynamic 1080p 30fps 10 сек»

Кодек	Метрика	Размер (MB)	Битрейт (Мбит/с)	SSIM	PSNR	Время кодирования (с)
h264	min	2.26	1.96	0.9798	41.04	3.93
	max	17.65	14.85	0.9935	49.62	7.95
	avg	5.72	4.73	0.9890	45.70	5.27
hevc	min	0.623	0.53	0.9618	38.44	11.26
	max	11.77	9.91	0.9923	48.49	37.27
	avg	2.90	2.39	0.9831	44.01	16.84
vp9	min	1.32	1.14	0.6832	22.57	14.30
	max	29.44	24.78	0.9885	37.92	48.09
	avg	7.39	6.09	0.9308	32.49	23.60
av1	min	0.951	0.80	0.6860	22.59	98.75
	max	21.58	18.16	0.9890	38.07	304.02
	avg	5.51	4.55	0.9328	32.54	175.60

Таблица 3

Результаты измерений сжатия группы «dynamic 1080p 30fps 1 м»

Кодек	Метрика	Размер (MB)	Битрейт (Мбит/с)	SSIM	PSNR	Время кодирования (с)
h264	min	21.86	2.81	0.9841	40.39	24.68
	max	62.78	8.65	0.9896	45.33	35.74
	avg	37.03	5.14	0.9874	43.34	29.26
hevc	min	9.51	1.22	0.9759	38.67	73.62
	max	44.48	6.13	0.9841	43.86	127.36
	avg	21.81	3.03	0.9807	41.49	96.46
vp9	min	23.40	3.01	0.8678	19.69	133.86
	max	113.16	15.59	0.9810	35.84	189.13
	avg	52.88	7.34	0.9249	28.64	152.79
av1	min	21.66	2.79	0.8700	19.69	577.07
	max	91.35	12.58	0.9834	35.95	1345.21
	avg	46.21	6.42	0.9277	28.71	912.47

Таблица 4

Результаты измерений сжатия группы «static 4k 30fps 10 сек»

Кодек	Метрика	Размер (MB)	Битрейт (Мбит/с)	SSIM	PSNR	Время кодирования (с)
h264	min	4.63	4.00	0.9911	49.09	13.19
	max	9.54	7.98	0.9942	50.76	17.10
	avg	7.12	5.92	0.9927	49.83	14.69
hevc	min	0.835	0.71	0.9902	48.05	37.52
	max	1.82	1.52	0.9916	48.75	43.55
	avg	1.24	1.03	0.9911	48.41	39.72
vp9	min	1.76	1.52	0.9891	39.80	45.56
	max	4.76	3.98	0.9916	48.83	49.32
	avg	3.09	2.57	0.9907	44.36	47.03
av1	min	1.64	1.42	0.9896	39.84	223.94
	max	3.03	2.53	0.9931	49.76	288.86
	avg	2.54	2.11	0.9918	44.80	252.01

Конфигурация оборудования, на котором проводилось тестирование:
CPU — Ryzen 7 5800
RAM — 16 GB
OS — Windows 11

Анализ показал, что кодек H.265 (HEVC) обеспечивает наилучшее соотношение между качеством и степенью сжатия. При сопоставимых значениях SSIM и PSNR во всех выборках достигался наилучший результат сжатия, в каждой из представленных групп файлов сжатые кодеком H.265 имели наименьший размер, при умеренном увеличении времени кодирования.

H.264 демонстрирует стабильно высокое качество и минимальное время кодирования. Он остаётся самым быстрым и совместимым стандартом, хотя требует большего битрейта для достижения того же уровня визуального качества.

VP9 и AV1, показали лучшие результаты в роликах со статичными сценами. Результаты в последней группе видео роликов сопоставимы с H.265 по степени сжатия. Однако, кодирование AV1 требует значительно больших вычислительных ресурсов в сравнении с остальными кодеками.

В динамических сценах кодеки VP9 и AV1 показали себя значительно хуже, чем H.265 и H.264. В настоящих условиях эксперимента преимущество по совокупным показателям остаётся за H.265.

4. Рекомендации по использованию алгоритмов сжатия

Сравнение показало, что выбор оптимального кодека должен определяться конкретными условиями применения. H.264 остаётся универсальным решением с высокой скоростью кодирования, широкой аппаратной поддержкой и отличной совместимостью. Он подходит для онлайн-видеоконференций, стриминга и повседневных задач, где важна низкая задержка и стабильность.

Кодек H.265/HEVC целесообразно использовать в случаях, когда требуется высокая эффективность сжатия при сохранении качества. Он особенно оправдан для записи и распространения видео в разрешении 4K и выше, а также для медиаконтента, где объём хранилища или пропускная способность сети ограничены. Следует учитывать, что HEVC требует больших вычислительных затрат, что может ограничивать его применение в открытых проектах.

VP9 представляет собой хороший компромисс между открытостью, эффективностью и поддержкой в браузерах. Он широко используется в веб-платформах, таких как YouTube, и обеспечивает экономию трафика по сравнению с H.264. При этом его качество сильно зависит от параметров кодирования, лучше всего подходит для видео с большим количеством статичных сцен, таких как подкасты или интервью, где смена кадров происходит не так часто, что позволяет обеспечить неплохое сжатие за относительно небольшое время.

AV1, также как и VP9, хорошо себя показывает с видео с большим количеством статичных сцен, однако, в отличии от VP9 требует колоссального времени кодирования. Он наиболее оправдан в ситуациях, когда скорость не является критическим фактором: при подготовке больших архивов видео, контента для потоковых платформ с серверным перекодированием или длительным офлайн-рендерингом, также оптимален для фильмов, где сцена меняется не так часто, что позволяет достичь лучшего коэффициента сжатия, чем у VP9, при этом AV1 имеет более широкую поддержку, чем H.265.

Таким образом, для реального времени и массового распространения оптимален H.264; для 4K-видео и хранения — HEVC; VP9, AV1 — Плохо себя показывают в динамических сценах, однако, хорошо подходят для видео с длительными сценами.

Заключение

В работе проведено исследование эффективности сжатия видеоданных с использованием четырёх современных стандартов: H.264/AVC, H.265/HEVC, VP9 и AV1. Полученные результаты подтвердили, что прогресс в области видео-кодирования направлен на уменьшение битрейта без потери визуального качества, однако повышение эффективности сопровождается ростом вычислительных затрат.

Кодек H.264 по-прежнему обеспечивает оптимальное соотношение между качеством, скоростью и совместимостью. HEVC демонстрирует существенное преимущество по размеру и битрейту при умеренном увеличении времени кодирования. VP9 выступает открытой альтернативой с хорошим балансом между качеством и доступностью. AV1 обеспечивает хорошее сжатие видео с большим количеством статических последовательностей кадров, однако требует на порядок больше времени для обработки, что ограничивает его применение в реальном времени.

Проведённые эксперименты подтверждают, что выбор кодека всегда является компромиссом между качеством, скоростью и доступными вычислительными ресурсами. С развитием аппаратного ускорения и увеличением производительности устройств роль новых форматов, прежде всего AV1, будет возрастать. Однако, в настоящее время оправдано использование всех четырёх стандартов, каждый из которых оптимален в своей нише.

Литература

1. *Шелухин О. И.* Сжатие аудио и видео информации / О. И. Шелухин, А. В. Гузеев. – Москва, 2012. – 80 с.

2. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин ; МИФИ. – Москва : МИФИ, 2003. – 350 с.

3. *Архипцев С. В.* Сравнительный анализ методов видеокодирования стандартов ITU-T H. 264-avc/ MPEG-4 Part-10 и H. 265 HEVC/ С. В. Архипцев, Д. П. Лукьянов. – URL: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-metodov-videokodirovaniya-standartov-itu-t-h-264-avc-mpeg-4-part-10-i-h-265-hevc. (Дата обращения 12.11.25)

4. *Шелухин О. И.* Мультифрактальный анализ видеосигналов стандарта H. 264/AVC / О. И. Шелухин, А. В. Арсеньев, А. Е. Перегняк. – URL: https://cyberleninka.ru/article/n/multifraktalnyy-analiz-videosignalov-standarta-h-264-avc (дата обращения 12.11.25)

5. *Дауд Абдо* Мультифрактальный анализ видеосигналов стандарта H. 264/AVC / Абдо Дауд, В. М. Квашнин. – URL: https://cyberleninka.ru/article/n/obzor-rasprostranennyh-standartov-i-metodov-szhatiya-tsifrovogo-video (дата обращения 12.11.25)

УДК 004.6

ИНВЕРТИРОВАННЫЕ ИНДЕКСЫ В ПОЛНОТЕКСТОВОМ ПОИСКЕ: АРХИТЕКТУРА, ПРИНЦИПЫ РАБОТЫ И ПРИМЕНЕНИЕ В МАСШТАБИРУЕМЫХ ВЕБ-СИСТЕМАХ

И. Ф. Астахова, Д. В. Дедов, В. М. Баркалова

Воронежский государственный университет

Аннотация. Инвертированный индекс — фундаментальная структура данных для полнотекстового поиска, широко используемая в поисковых системах и информационном поиске. В данной статье рассматриваются основные принципы работы инвертированных индексов, их внутреннее устройство и архитектура, сравнение с альтернативными методами индексации, а также области применения и существующие ограничения. Особое внимание уделяется описанию компонентов индекса (словари терминов и списки вхождений), оптимизациям (сжатие, пропуски) и взаимодействию с системами на базе *Lucene*/*Elasticsearch*. В заключении подводятся итоги роли инвертированных индексов в современных системах поиска.

Ключевые слова: инвертированный индекс, полнотекстовый поиск, поисковые системы, *Lucene*, *Elasticsearch*, структура данных, информационный поиск, сигнатурные файлы, прямой индекс.

Введение

Инвертированные индексы являются ключевым механизмом организации полнотекстового поиска в современных информационных системах. Они позволяют эффективно обрабатывать запросы в условиях больших объёмов данных, заменяя прямой перебор документов структурой вида «термин → список документов», что существенно сокращает время поиска. Благодаря высокой производительности и масштабируемости инвертированные индексы широко применяются в поисковых системах, аналитических платформах, корпоративных хранилищах данных и сервисах, содержащих большие коллекции текстовой информации.

Инвертированный индекс обеспечивает точную и быструю навигацию по тексту, что делает его фундаментальным инструментом в задачах полнотекстового поиска и фильтрации данных. Цель данной статьи — рассмотреть принципы построения инвертированных индексов, их архитектурные особенности, преимущества и ограничения, а также показать их роль в современных системах, ориентированных на высокоскоростную обработку текстовой информации.

1. Основные принципы работы инвертированных индексов

Инвертированный индекс — структура данных, в которой для каждого уникального термина текстовой коллекции перечисляются документы, содержащие этот термин.

Существует две разновидности инвертированных индексов:

• вариант без позиций: для каждого термина хранится просто список документов, в которых он встречается. Подходит для простых логических запросов (*AND*, *OR*);

• вариант с позициями: помимо идентификаторов документов хранятся позиции (номера слов) в тексте документа. Это необходимо для поддержки точного поиска фраз, подсчёта близости терминов.

При обработке поискового запроса система обращается к инвертированному индексу: для каждого термина запроса берутся соответствующие списки документов, после чего они объединяются или пересекаются в зависимости от логики запроса.

Важный этап — индексация документов: прежде чем добавить документы в индекс, обычно выполняют текстовую предобработку. Слова нормализуются (приводятся к нижнему регистру), удаляются пунктуация и стоп-слова. Например, *Lucene* и *Elasticsearch* умеют выполнять морфологический анализ и лемматизацию. Это уменьшает размер индекса и повышает качество поиска: стоп-слова («и», «в», «но» и т. п.) часто исключают из индекса, так как они встречаются в большинстве документов.

Постинг-листы (списки документов) обычно хранятся в упорядоченном виде по идентификаторам документов, что позволяет эффективно выполнять пересечение списков через алгоритм слияния. Для ускорения поиска по длинным спискам внедряются специальные указатели-пропуски (*skip-pointers*), позволяющие быстро пропустить большие блоки документов при пересечении. При хранении также часто используют дельта-кодирование или другие методы сжатия: вместо хранения абсолютных номеров документов сохраняют разности между ними. Это сильно экономит место и ускоряет операции ввода-вывода диска, поскольку числовые данные в постинг-листах обычно возрастают, и дельты оказываются малыми числами [1].

Обобщенная диаграмма последовательности обработки поискового запроса в системе на основе инвертированного индекса представлена на рис. 1.

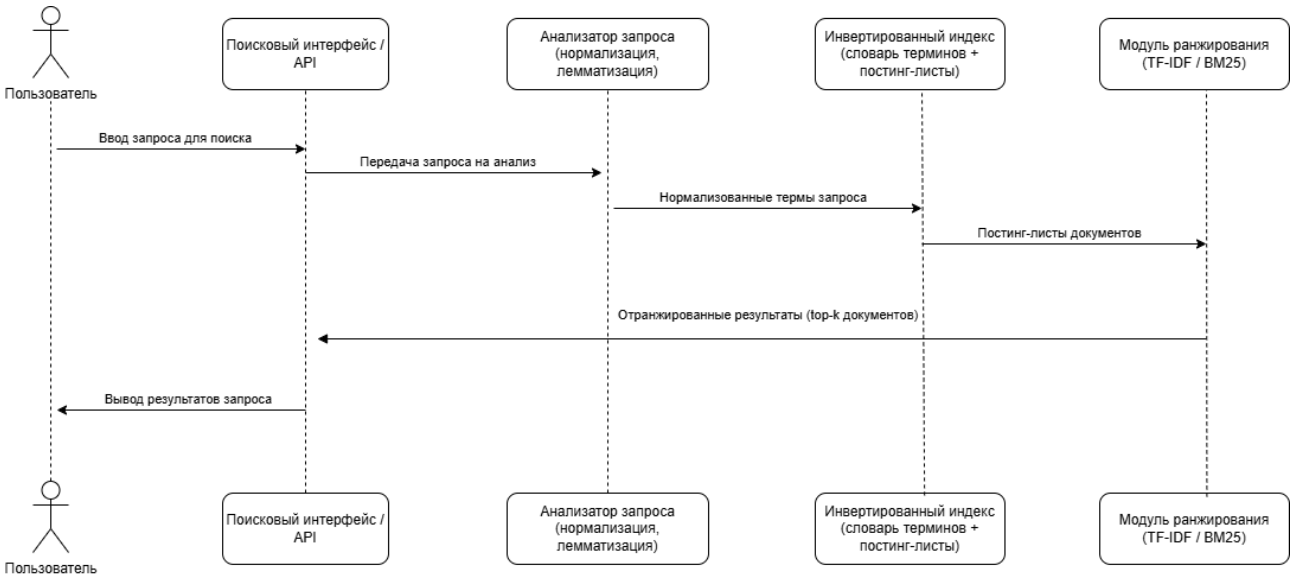

Рис. 1. Диаграмма последовательности обработки поискового запроса в системе на основе инвертированного индекса

Простой пример инвертированного индекса для трёх коротких документов представлен в табл. 1. Строки таблицы — уникальные слова (термины) из корпуса, а вторая колонка — номера документов, где слово встречается. Именно эти списки используются при поиске.

Таблица 1

Пример инвертированного индекса для трёх документов

Слово	Номера документов
Сегодня	1
Очень	1
Вчера	2
Было	2, 3
Тепло	2, 3
Прошлый	3
Месяц	3

Таким образом, при запросе «тепло AND вчера» достаточно взять список документов для «тепло» (2,3) и пересечь с «вчера» (2) — результат: документ 2. Эта схема работы обеспечивает гораздо более быстрый поиск, чем полный перебор контента.

2. Архитектура инвертированного индекса

На уровне концепции инвертированный индекс можно представить как словарь терминов с указанием их вхождений. На практике современные системы построения индекса (например, *Apache Lucene*) реализуют инвертированный индекс через набор неизменяемых сегментов. Каждый сегмент индекса — это самостоятельно инвертированный индекс: он хранит термины и постинг-листы. Такая сегментированная структура позволяет эффективно индексировать новые документы: добавления и изменения оформляются созданием новых сегментов, а старые по необходимости объединяются в фоновом режиме.

В реализации *Lucene* термины хранятся в компактной форме с помощью конечного автомата (*FST*), что обеспечивает быстрый поиск и экономию памяти [4]. Постинг-лист для каждого терма содержит номера документов и дополнительные данные, например частоту вхождений терма в документ. Списки документов упорядочены по возрастанию *ID*, что позволяет выполнять быстрые пересечения. Кроме того, для часто используемых операций ранжирования (*TF-IDF, BM25*) сами частоты хранятся в этом же листе или рядом с ним.

Отдельно стоит упомянуть о *doc values* — альтернативной структуре хранения, применяемой в *Lucene*/*Elasticsearch* для агрегаций и сортировки. В отличие от инвертированного индекса, который оптимизирован под полнотекстовый поиск, *doc values* — это колонно-ориентированный формат, в котором хранятся все значения конкретного поля по документам. Такой формат ускоряет операции сортировки и агрегации, поскольку позволяет читать поля документов последовательно. Именно поэтому, если нужно, например, сортировать результаты по текстовому полю, в *Elasticsearch* предлагают использовать поле типа *keyword c doc values* [2].

Взаимодействие с инвертированным индексом происходит через интерфейсы чтения/записи. При добавлении документов индекс обновляется путем записи новых сегментов, а при удалении документов эти изменения фиксируются пометкой «удалён» в соответствующих сегментах (фактические документы физически удаляются при последующем слиянии сегментов). В итоге структура индекса остаётся неизменяемой на уровне сегментов, что упрощает многопоточный доступ и отказоустойчивость [3].

3. Альтернативные структуры и сравнение

Помимо инвертированных индексов, существуют и другие подходы к индексации текстовых данных. Одним из них являются сигнатурные файлы (*signature files*), в которых для каждого документа формируется битовая подпись по набору ключевых слов, а поиск выполняется путём сопоставления масок. Однако на практике такие структуры существенно уступают инвертированным индексам как по скорости выполнения запросов, так и по объёму занимаемой памяти, поэтому применяются ограниченно [1].

Другим вариантом является прямой (*forward*) индекс, где для каждого документа хранится список терминов. Этот способ удобен при построении индекса, но при поиске требует перебора значительной части коллекции, что приводит к низкой производительности на масштабах больших данных.

Существуют и другие структуры, такие как суффиксные деревья, суффиксные массивы и n-граммные индексы. Суффиксные структуры позволяют эффективно искать подстроки, но требуют значительных объёмов памяти и больше подходят для анализа одного большого текста, чем множества отдельных документов. В ряде задач также применяется последовательный

поиск без индекса, однако он рассматривается как базовый случай и используется только при малых объёмах данных.

В сравнении с перечисленными методами инвертированный индекс остаётся наиболее универсальным и эффективным решением для полнотекстового поиска. Именно поэтому он используется в большинстве современных поисковых систем и библиотек, рассчитанных на работу с большими коллекциями текстовой информации.

4. Области применения

Инвертированные индексы используются везде, где требуется быстрый поиск по текстовым данным. Это, прежде всего, поисковые системы общего назначения — интернет-поисковики, внутренняя навигация по сайтам, электронным библиотекам и т. д. Поисковые движки используют инвертированные индексы для быстрого получения релевантных документов по запросам пользователей. Аналогично, корпоративные системы поиска (например, в интернете-магазинах) полагаются на инвертированные индексы для анализа названий и описаний продуктов.

Кроме «человеко-ориентированных» запросов, инвертированные индексы применяются в области больших данных и аналитики: например, для поиска по текстам логов или документов в *NoSQL*-хранилищах. Одним из интересных примеров является биоинформатика: при сборке ДНК-фрагментов часто строят инвертированные индексы всех подстрок эталонной ДНК, что позволяет эффективно находить соответствие фрагментов.

Важно отметить, что «обычные» реляционные СУБД не были оптимизированы для полнотекстового поиска: они ориентированы на точный поиск по полям (через *B*-деревья, хеши). При наличии нестрогих запросов («по смыслу», «с учетом опечаток») традиционные индексы оказываются малоэффективны. Инвертированный индекс же создан именно для полнотекста: он разбивает документы на термы и сохраняет их в виде компактного списка. Это позволяет быстро находить даже частичные совпадения и работать с большими массивами данных.

5. Проблемы и ограничения

Хотя инвертированный индекс обеспечивает высокую скорость полнотекстового поиска, у него есть ряд ограничений. Существенным фактором является объём хранимых данных: для каждого термина формируется отдельный постинг-лист, и при больших корпусах индекс может занимать больше места, чем исходный текст, несмотря на применение методов сжатия. Дополнительные трудности возникают при обновлении данных — создание новых сегментов и их последующее слияние приводит к накладным расходам на *CPU* и дисковую подсистему, что особенно заметно в динамичных системах.

Инвертированный индекс также не оптимизирован для сортировок и агрегирования по текстовым полям, поэтому в поисковых движках используются дополнительные структуры данных, увеличивающие общий объём хранилища. Ещё одним ограничением является чувствительность к морфологии и вариативности языка: разные формы и синонимы рассматриваются как разные термины, и без лингвистической предобработки снижается полнота поиска. Эти особенности требуют использования лемматизации и расширения запросов, если необходима корректная работа с естественным текстом.

Заключение

Инвертированные индексы являются фундаментальным механизмом для организации быстрого полнотекстового поиска и остаются ключевой структурой в современных информационных системах.

Практическая значимость инвертированных индексов подтверждается их широким применением: от поисковых движков и электронных библиотек до аналитических платформ и сервисов обмена сообщениями. Возможность обеспечивать быстрый доступ к текстовой информации делает этот подход незаменимым при проектировании масштабируемых систем.

Таким образом, понимание архитектуры и ограничений инвертированных индексов является важным элементом разработки высоконагруженных приложений, ориентированных на обработку больших текстовых массивов и обеспечение высокой производительности поиска.

Литература

1. *Маннинг К.* Введение в информационный поиск / К. Маннинг, П. Рагаван, Х. Шютце; пер. с англ. – М. : Вильямс, 2011. – 528 с.

2. *Конда М.* Elasticsearch в действии, 2-е изд. / М. Конда; пер. с англ. – Астана : АЛИСТ, 2024. – 656 с.

3. *Шривастава А.* Elasticsearch для разработчиков: индексирование, анализ, поиск и агрегирование данных, 2 е изд. / А. Шривастава; пер. с англ. – М. : Питер, 2025. – 336 с.

4. Apache Lucene – официальная документация [Электронный ресурс] // Apache Lucene Documentation. – URL: https://lucene.apache.org/core/documentation.html (дата обращения: 26.11.2025).

УДК 004.031.43

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ИНТЕРНЕТ-МАГАЗИНА С ВОЗМОЖНОСТЬЮ ФОРМИРОВАНИЯ ОТЧЕТОВ И МОДЕЛЬЮ ФИЛЬТРАЦИИ ОТЗЫВОВ

И. Н. Барбашина, М. В. Матвеева

Воронежский государственный университет

Аннотация. В статье рассматривается разработка веб-приложения для интернет-магазина на основе Spring Boot и PostgreSQL, обеспечивающего управление товарами, заказами, клиентами и формирование аналитических отчётов. Приложение включает модуль CRUD-операций, систему из семи видов отчётов для анализа продаж и интерфейс на Thymeleaf и Bootstrap. Также представлена концепция интеграции модели машинного обучения для автоматической фильтрации релевантных и нерелевантных товару отзывов.
Ключевые слова: веб-приложение, интернет-магазин, Spring Boot, аналитические отчёты, PostgreSQL, машинное обучение, фильтрация отзывов, Thymeleaf.

Введение

Современная электронная коммерция требует не только эффективных инсрументов управления, но и глубокой аналитики для принятия стратегических решений [1]. Существующие коммерческие решения, такие как 1С-Розница или Shopify, часто обладают избыточной функциональностью и высокой стоимостью, что ограничивает их доступность для малого бизнеса. В связи с этим актуальной задачей является разработка специализированных веб-приложений, сочетающих простоту использования, аналитические возможности и низкую стоимость владения.

В статье представлено веб-приложение для интернет-магазина, реализующее базовые операции управления данными и систему отчётности. Дополнительно рассматривается возможность интеграции модели машинного обучения для автоматической фильтрации отзывов, что повышает качество обратной связи и улучшает пользовательский опыт.

1. Постановка задачи

Разработать веб-приложение для интернет-магазина, позволяющее:

- вести учёт товаров;
- регистрировать клиентов и обрабатывать их заказы;
- формировать отчёты по продажам.

Границы проекта: разработка включает backend-систему и веб-интерфейс. Мобильное приложение и интеграция с платёжными системами выходят за рамки данной работы.

2. Сравнение технологических стеков

Для выбора технологического стека был проведён сравнительный анализ платформ (табл.1).

На основании проведённого сравнительного анализа для реализации backend-части веб-приложения был выбран фреймворк Spring Boot в связке с языком Java. Данный выбор обусловлен следующими ключевыми преимуществами:

- высокая производительность: Spring Boot демонстрирует превосходство по количеству обрабатываемых запросов в секунду (RPS) по сравнению с рассмотренными альтернативами;

Таблица 1

Сравнение технологических стеков

Критерий	PHP (Laravel)	Python(Django)	Java (Spring Boot)
Производительность	100-500 RPS	150-600 RPS	1000-10000 RPS
Безопасность	CSRF, шифрование	Встроенная защита от XSS, SQL-инъекций	Spring Security
Скорость разработки	Готовые шаблоны, миграции	Админ-панель, ORM	Spring Initializr ускоряет старт
Поддержка БД	MySQL, PostgreSQL, SQLite (Eloquent ORM)	MySQL, PostgreSQL, SQLite (Django ORM)	Любые SQL/NoSQL, включая Oracle, MongoDB

• мощные встроенные механизмы безопасности: наличие комплексного фреймворка Spring Security предоставляет готовые и легко настраиваемые средства для аутентификации, авторизации и защиты от распространённых уязвимостей (XSS, CSRF, SQL-инъекции и др.);

• гибкость и масштабируемость: поддержка различных СУБД (SQL и NoSQL) и модульная архитектура Spring Boot позволяют легко адаптировать и расширять приложение в будущем;

• скорость разработки: несмотря на то, что Java часто ассоциируется с более низкой скоростью разработки по сравнению с Python и PHP, использование Spring Initializr, Spring Boot Starter-ов и Lombok компенсируют этот недостаток, ускоряя создание каркаса приложения и сокращая объём шаблонного кода.

3. Средства реализации

Разработка приложения велась с использованием следующего стека технологий:

• backend: Spring Boot(Spring MVC, Spring Data, Spring Security);
• СУБД: PostgreSQL;
• фронтенд: Thymeleaf, Bootstrap 5;
• инструменты сборки: Maven;
• среда разработки: IntelliJ IDEA;
• система контроля версий: Git.

Ключевые зависимости, указанные в файле pom.xml, включают:

• spring-boot-starter-web — для создания веб-приложения;
• spring-boot-starter-data-jpa — для работы с базой данных через JPA и Hibernate;
• spring-boot-starter-thymeleaf — для генерации HTML-шаблонов;
• posgresql — JDBC-драйвер для подключения к PostgreSQL;
• lombok — для сокращения шаблонного кода;
• spring-boot-starter-test — для модульного и интеграционного тестирования.

Конфигурация подключения к базе данных выполнена в файле application.properties, где указаны параметры URL, имя пользователя, пароль и режим генерации DDL (create-drop).

4. Архитектура и реализация приложения

Приложение разработано с использованием многослойной архитектуры на основе Spring Boot [2]. На рис. 1 показано, что система организована по стандартной Maven-структуре с чётким разделением на пакеты:

• entities содержит классы-сущности (например, Product, Order, Category);
• repositories включает интерфейсы, наследуемые от JpaRepository;

• services содержит классы-сервисы, в которых инкапсулирована бизнес-логика приложения;
• controllers содержит классы, обрабатывающие HTTP-запросы;
• templates содержит HTML-шаблоны, используемые движком Thymeleaf для генерации конечных веб-страниц;
• resources содержит статические ресурсы, а также файл конфигурации application-properties.

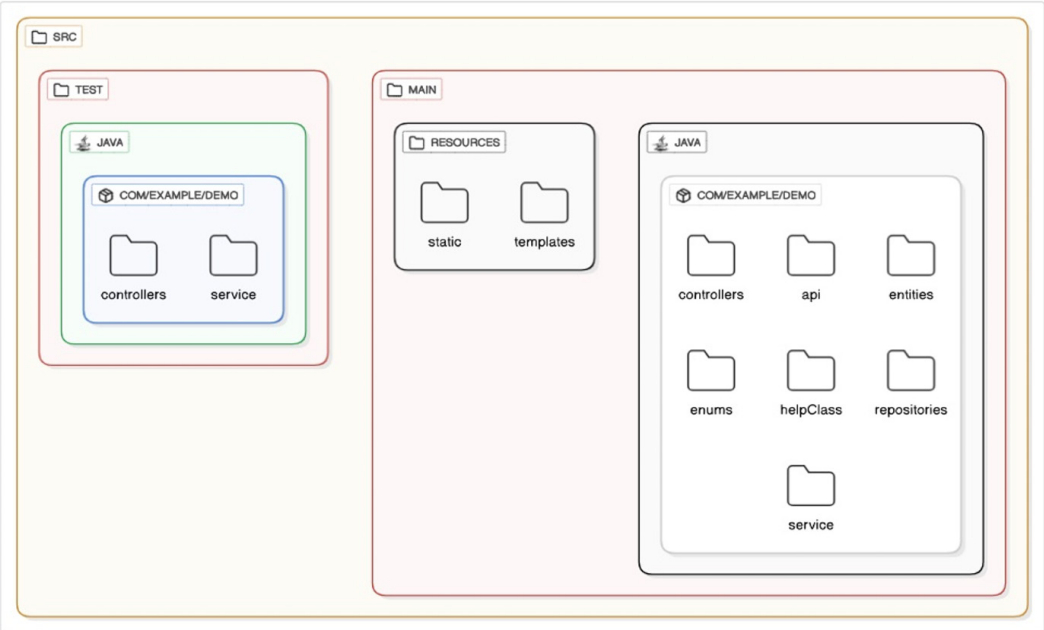

Рис. 1. Структура проекта

В качестве СУБД выбрана PostgreSQL [3], обеспечивающая надежность и производительность. Интерфейс реализован на Thymeleaf и Bootstrap [4]. На рис. 2 представлен интерфейс главной страницы веб-приложения.

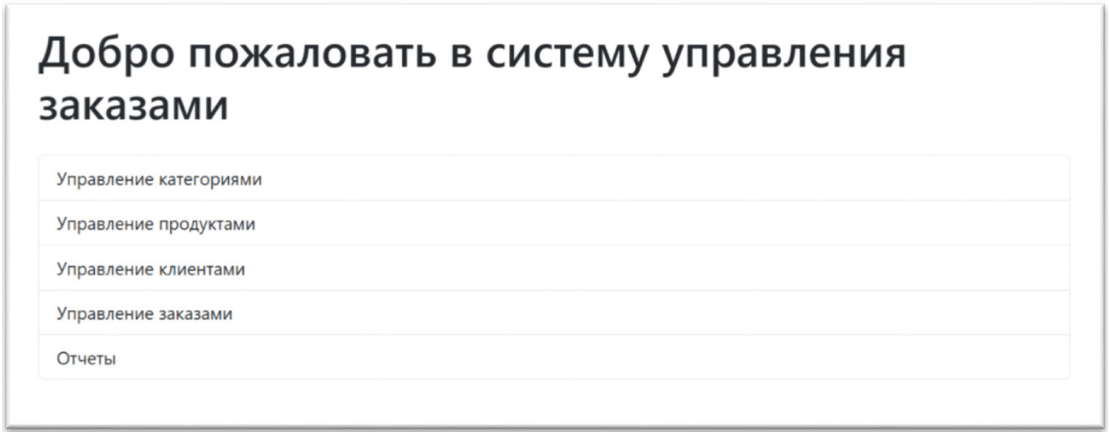

Рис. 2. Интерфейс главной страницы

1.1. Модель данных

На рис. 3 представлена структура базы данных.

Связи между сущностями соответствуют принципам нормализации (3НФ). Например, связь «один-ко-многим» реализована между Category и Product, а связь «многие-ко-многим» — между Order и Product через OrderIrem. Программная реализация сущностей на примере класса Product выглядит следующим образом:

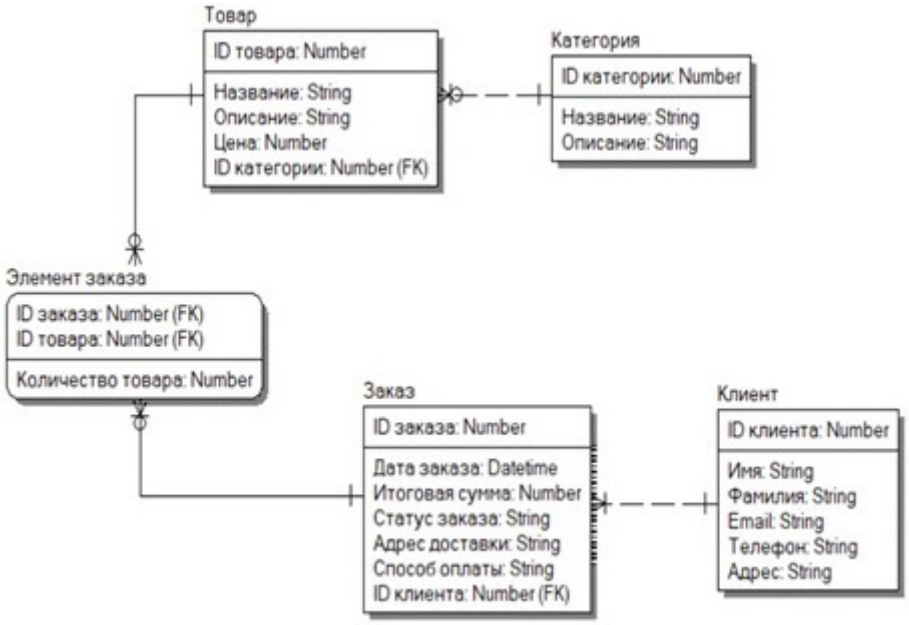

Рис. 3. Структура базы данных

```
package com.example.demo.entities;

import jakarta.persistence.*;
import lombok.*;

import java.math.BigDecimal;

@Entity
@Table(name = "Product")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String name;

    @Column(nullable = false)
    private BigDecimal price;

    private String description;

    @ManyToOne
    @JoinColumn(name = "category_id")
    private Category category;

    public void setPrice(BigDecimal price) {
        if (price == null || price.compareTo(BigDecimal.ZERO) <= 0) {
           throw new IllegalArgumentException("Price must be greater than zero.");
        }
        this.price = price;
    }
}
```

4.2. Многослойная архитектура и взаимодействие компонентов

Рис. 4. Схема взаимодействия слоёв приложения

Архитектура приложения состоит из следующих слоёв (рис. 4).

• слой представления (View): реализован на Thymeleaf в связке с Bootstrap 5. Отвечает за отображение данных пользователю и взаимодействие с интерфейсом. Включает как страницы для CRUD-операций, так и специализированные шаблоны для отображения отчётов;

• контроллеры (Controller): реализованы два типа контроллеров для гибкости взаимодействия: API-контроллеры — возвращают данные в формате JSON и предназначенные для потенциальной интеграции с внешними системами или фронтендом, а также View-контроллеры, обрабатывающие запросы от веб-интерфейса и возвращающие готовые HTML-страницы через Thymeleaf;

• сервисный слой (Service): в этом слое инкапсулирована вся бизнес-логика приложения. Сервисы обращаются к репозиториям, выполняют валидацию данных и транзакционные операции;

• слой доступа к данным (Repository): для взаимодействия с базой данных используются репозитории, наследуемые от JpaRepository. Это позволяет использовать готовые CRUD-методы, а для сложных аналитических запросов применяются методы с аннотацией @Query;

• слой сущностей (Model): сущности отображают таблицы базы данных и используют аннотации JPA (@Entity, @Table).

4.3. Функциональные модули

Приложение включает модули управления товарами, категориями, клиентами и заказами (CRUD-операции), а также систему отчётов, в которой реализованы следующие отчёты:

• продажи по товарам;
• продажи по категориям;
• продажи по клиентам;
• топ-N товаров;
• динамика продаж по месяцам и годам;
• заказы в статусе ожидания оплаты.

4.4. Реализация отчётов

Для формирования отчётов используются специализированные SQL-запросы с агрегатными функциями и группировками. Например, отчёт «Продажи по товарам» формируется с помощью запроса:

```
SELECT oi.product.name, SUM(oi.quantity * oi.product.price)
FROM OrderItem oi
JOIN oi.order o
WHERE o.orderDate BETWEEN :startDate AND :endDate
GROUP BY oi.product.name
```

На рис. 5. представлена визуализация одного из семи отчётов — «Продажи по товарам», где данные представлены в структурированном табличном виде с возможностью фильтрации по временным периодам.

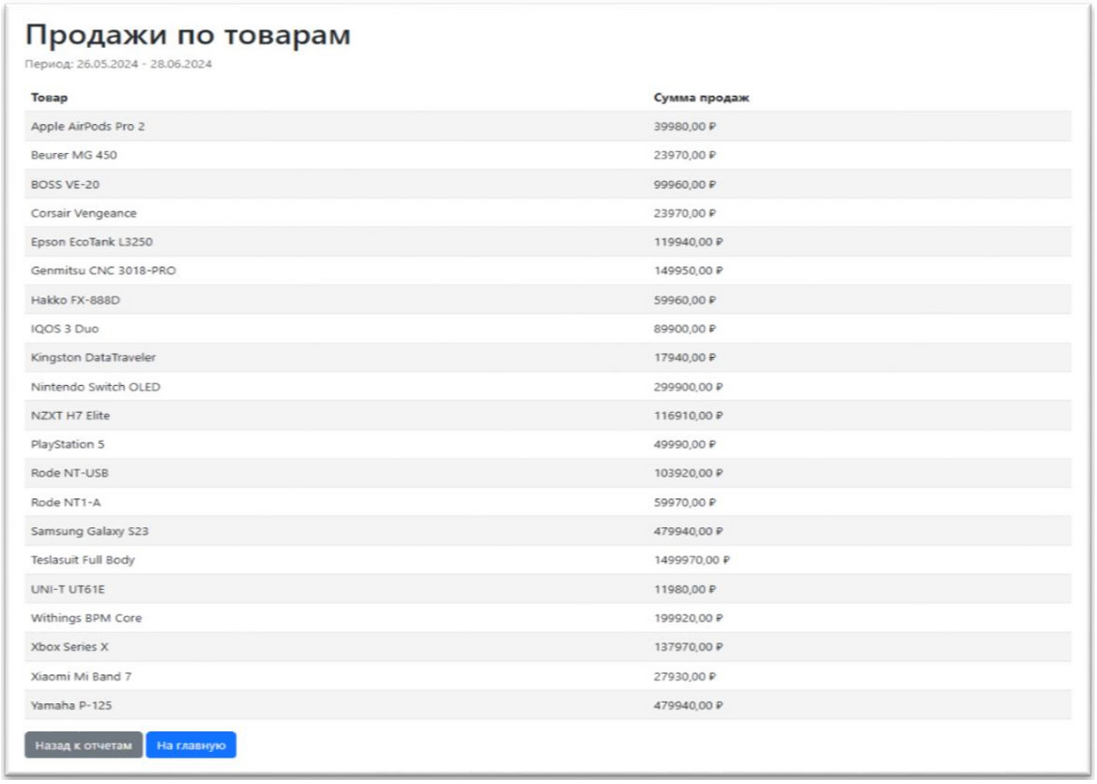

Рис. 5. Выходные данные отчёта «Продажи по товарам»

5. Интеграция модели для фильтрации отзывов

Для улучшения качества пользовательского контента планируется внедрение модели машинного обучения, которая будет классифицировать тексты отзывов на два класса:

• релевантные — содержат оценку качества, характеристик или опыта использования товара;

• нерелевантные — касаются службы доставки, упаковки или других аспектов, не связанных непосредственно с товаром.

5.1. Выбор подхода

Для решения задачи классификации текстов будут использованы методы NLP (Natural Language Processing) на основе предобученных моделей, таких как BERT или ruBERT для русского языка. В качестве альтернативы можно рассмотреть более лёгкие модели, например, на основе Random Forest или SVM с векторизацией текста (TF-IDF) [5].

5.2. Пример реализации

Предполагается создание отдельного микросервиса на Python с использованием библиотек transformers или scikit-learn. Модель будет принимать текст отзыва и возвращать метку класса и вероятность принадлежности.

Пример кода для инференса:

```
from transformers import pipeline

classifier = pipeline("text-classification", model="ruBERT-model")
review = "Товар отличный, соответствует описанию."
```

```
result = classifier(review)
print(result)  # [{'label': 'RELEVANT', 'score': 0.98}]
```

5.3. Интеграция с приложением

Микросервис будет взаимодействовать с основным приложением через REST API[6]. При добавлении нового отзыва он будет автоматически отправляться на классификацию, и в зависимости от результата отзыв будет публиковаться или помечаться для модерации.

6. Результаты и перспективы

Разработанное веб-приложение предоставляет платформу для управления интернет-магазином и анализа продаж. Внедрение модели фильтрации отзывов позволит:

- повысить релевантность пользовательского контента;
- снизить нагрузку на модераторов.

В перспективе планируется расширение функциональности за счёт:

- модуля поставщиков;
- системы скидок и промокодов;
- интеграции с платёжными системами;
- мобильного приложения.

Заключение

В статье представлено веб-приложение для интернет-магазина, сочетающее простоту использования, аналитические возможности и модульность архитектуры. Реализованная система отчётов позволяет владельцам бизнеса оперативно анализировать ключевые показатели, а планируемая модель фильтрации отзывов направлена на повышение качества пользовательского контента. Приложение может служить основой для дальнейшего расширения и адаптации под конкретные бизнес-требования.

Литература

1. *Мартин Р.* Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин ; [пер. с англ.]. – Санкт-Петербург : Питер, 2018. – 352 с.

2. Документация Spring Framework [Электронный ресурс]. – URL: https://spring.io/projects/spring-framework (дата обращения 01.07.2025).

3. Документация PostgreSQL 15 [Электронный ресурс]. – URL: https://www.postgresql.org/docs/15/ (дата обращения 01.07.2025).

4. Thymeleaf Documentation [Электронный ресурс]. – URL: https://www.thymeleaf.org/documentation.html (дата обращения 01.07.2025).

5. *Иванов Д. С.* Современные подходы к тестированию Spring Boot приложений / Д. С. Иванов // Программная инженерия. – 2022. – № 4. – С. 45–52.

6. *Роджерс Р.* RESTful Web Services / Р. Роджерс ; [пер. с англ.]. – Москва : Символ-плюс, 2019. – 456 с.

УДК 004.056

СОВРЕМЕННЫЕ ТЕХНОЛОГИЧЕСКИЕ РЕШЕНИЯ БЕЗОПАСНОГО ФОРМАТИРОВАНИЯ СТРОК В PYTHON 3.14

И. А. Батищев, А. А. Зуенко

МИРЭА – Российский технологический университет

Аннотация. В Python 3.14 были представлены template strings (t-строки) для использования в области интерполяции строк. Определённые в PEP 750, t-строки отделяют вычисление значений от их обработки, решая критические проблемы безопасности. Структурированный объект Template содержит метаинформацию о каждой интерполяции, позволяя обработчикам контролировать отделение персональной информации и трансформацию данных. Это решает проблемы XSS в HTML, SQL-инъекции в запросах и обеспечивает структурированное логирование. Статья рассматривает эволюцию форматирования строк в Python, архитектуру *t*-строк, практические применения в веб-разработке и безопасности баз данных.

Ключевые слова: python 3.14, PEP 750, template strings, *t*-строки, форматирование строк, безопасность, XSS, SQL-инъекции, HTML-санитизация, структурированное логирование, интерполяция, доменно-специфичные языки, структурированные данные, безопасность приложений, функциональное программирование.

Введение

Python 3.14, выпущенный в октябре 2025 года, представил template strings (t-строки) — значительный прогресс в области интерполяции строк. Посредством отделения вычислений значений от их обработки, t-строки представляют безопасную замену f-строкам. В отличии от f-строк, которые мгновенно преобразуют выражения в текст, t-строки возвращают структурированный объект Template, содержащий метаинформацию о каждой интерполяции. Это позволяет пользовательским обработчикам контролировать трансформацию данных в зависимости от контекста — будь то генерация HTML, защита SQL-запросов или структурированное логирование.

1. Эволюция форматирования строк в Python

История форматирования строк в Python демонстрирует постоянное стремление к улучшению безопасности и читаемости. Первый метод — оператор % из языка C — был функциональным, но ограниченным. С появлением Python 3.0 пришёл str.format(), предложивший более гибкий синтаксис с именованными аргументами, хотя и требовавший многословных конструкций.

Глобальные изменения произошли с f-строками после выхода Python 3.6, предоставившего новый способ встраивания выражений прямо в литералы. F-строки поддерживали спецификаторы конверсии (!r, !s, !a) и форматные спецификаторы. PEP 701 (Python 3.12) формализовал грамматику f-строк, убрав синтаксические ограничения.

Однако f-строки имели критическое ограничение: невозможность перехватить и трансформировать значения до их объединения в итоговую строку. PEP 501 предлагал решение через отложенные i-строки, но был отозван. PEP 750 реализовал этот подход через структурированную интерполяцию с сохранением метаинформации о выражениях [1, 2].

2. Архитектура и ключевые инновации

Фундаментальное отличие t-строк — их двухэтапная природа. F-строки работают по следующим этапам: Выражение Вычисление Форматирование

1. Итоговая строка В свою очередь t-строки реализуют:

1. Выражение
2. Вычисление
3. Объект Template
4. Пользовательская трансформация.

Первая инновация — новый тип данных Template из модуля string.templatelib. Объект содержит три основных атрибута:

- strings: tuple[str, ...] — статические части (всегда N+1 элементов для N интерполяций)
- strings: tuple[str, ...] — статические части (всегда N+1 элементов для N интерполяций)
- strings: tuple[str, ...] — статические части (всегда N+1 элементов для N интерполяций)

Второй тип — Interpolation — представляет одну интерполяцию с метаинформацией:

- value — вычисленное значение
- expression — исходный текст (например, «user_id»)
- conversion — спецификатор конверсии (!r, !s, !a)
- format_spec — форматная спецификация (например, «.2f»)

Третья инновация — разделение ответственности: компилятор Python парсит синтаксис и создаёт объекты Template, а пользовательские обработчики определяют обработку [1, 2]. Это обеспечивает контекстно-зависимую безопасность — одни данные обрабатываются по-разному для HTML, SQL или логирования.

Синтаксис использует префикс t или T:

- `template = t"Hello {name}"`
- `template = t"Value: {value:.2f}"`
- `template = t"Repr: {value!r}"`
- `template = rt"Path: C:\Users\{username}"`

3. Применение в веб-разработке

XSS-атаки (Cross-Site Scripting) остаются критической уязвимостью [6, 7]. F-строки делают код уязвимым:

```
user_comment = "<img src=x onerror='alert(1)'>"
html_output = f"<div>{user_comment}</div>"
```

T-строки решают проблему через автоматическую санитизацию [2, 6]:

```
from html import escape
from string.templatelib import Template, Interpolation
def html(template: Template) -> str:
    parts = []
    for item in template:
        if isinstance(item, str):
            parts.append(item)
        elif isinstance(item, Interpolation):
            parts.append(escape(str(item.value)))
    return "".join(parts)
```

Обработчик может применять контекстно-зависимую обработку — различные правила для содержимого, атрибутов, CSS-значений. Это предотвращает инъекции на уровне языка через контролируемую обработку.

4. Применение в базах данных и SQL-инъекциях

SQL-инъекции остаются одной из главных уязвимостей [4, 5]. F-строки опасны:

```
user_id = "1 OR 1=1"
query = f"SELECT * FROM users WHERE id = {user_id}"
```

T-строки обеспечивают параметризованные запросы [1, 5]:

```
from string.templatelib import Template, Interpolation

def sql_query(template: Template) -> tuple[str, list]:
    parts = []
    params = []
    param_num = 0
    for item in template:
        if isinstance(item, str):
            parts.append(item)
        elif isinstance(item, Interpolation):
            param_num += 1
            parts.append(f"${param_num}")
            params.append(item.value)

    return "".join(parts), params
```

Значение передаётся как параметр, не как часть SQL, что предотвращает инъекции через небезопасное разделение SQL-кода от данных [4, 5].

5. Применение в структурированном логировании

Традиционное логирование объединяет данные в неструктурированный текст, что затрудняет машинную обработку [8, 9]. T-строки позволяют создать двухканальное логирование — одновременно читаемый текст и JSON [8, 9]:

```
import json
from logging import Formatter, LogRecord
from string.templatelib import Template, Interpolation

class StructuredFormatter(Formatter):
    def format(self, record: LogRecord) -> str:
        if isinstance(record.msg, Template):
            readable = self._to_readable(record.msg)
            data = self._extract_data(record.msg)
            return f"{readable} | {json.dumps(data)}"
        return super().format(record)

    def _to_readable(self, template: Template) -> str:
        parts = []
        for item in template:
            if isinstance(item, str):
                parts.append(item)
            elif isinstance(item, Interpolation):
                parts.append(str(item.value))
        return "".join(parts)
```

Это обеспечивает автоматизированную обработку логов системами мониторинга. Структурированные поля индексируются быстрее неструктурированного текста, что улучшает поисковую эффективность и аналитику.

6. Доменно-специфичные языки и дополнительные применения

T-строки открывают возможность создания Domain-Specific Languages (DSL) прямо в Python [11, 12]. Пример DSL для CSS с валидацией показывает, как обработчик может проверять корректность значений для конкретного контекста.

Кэширование обеспечивает производительность. Статические части (template.strings) кэшируются, повышая скорость многократного использования шаблона с разными значениями.

Интеграция с системой типов через typing.Annotated позволяет IDE определять язык внутри шаблона [17, 18]. Это включает подсветку синтаксиса и валидацию конкретного языка. Pattern matching, введённый в Python 3.10, предоставляет чистый способ обработки интерполяций с учётом их типов и метаинформации.

7. Сравнение с альтернативами

T-строки идеальны для встраиваемых шаблонов в Python-коде, в то время как Jinja2 остаётся предпочтительным для файловых шаблонов и пользовательских конфигураций. JavaScript предлагает близкую функцию — tagged templates, но Python выбрал явный вызов функции. Python также сохраняет метаинформацию в Interpolation.expression, чего нет в JavaScript.

Каждый подход имеет свои плюсы. F-strings остаются быстрыми для простого форматирования в доверенном контексте. str.format() применяется для динамических строк. Jinja2 используется для пользовательских шаблонов. T-strings становятся стандартом для встраиваемых шаблонов, DSL и критичных к безопасности областей.

8. Ограничения и вызовы

Синтаксическая несовместимость: t-строки требуют Python 3.14+. Код с t-строками не будет работать на предыдущих версиях, создавая барьер для проектов, поддерживающих несколько версий.

Новый объект Template: Для Template всегда требуется обработчик, что отличает t-строки от привычных f-строк.

Производительность: обработка значений требует больше времени, чем f-строки, но компенсируется кэшированием статических частей. F-строки остаются быстрее для некритичных к безопасности кода.

Отладка может быть сложнее из-за разделения на этапы создания и обработки.

Заключение

PEP 750 Template Strings — новый этап форматирования Python. Ключевыми изменениями с добавлением t-строк станут: безопасность по умолчанию через контекстно-зависимую санитизацию [2, 6, 7]; гибкость с полной поддержкой Python-выражений и доменно-специфичных языков [1, 11, 12]; структурированность сохранением метаинформации о выражениях [1, 8]; производительность через кэширование статических частей; типизация для статического анализа и IDE-поддержки [17,18].

Более частым для критичных областей (HTML-генерация, SQL-запросы, shell-команды), простого форматирования текста в доверенном контексте, специализированных обработчиков для решения задач конкретных предметных областей станет использование t-strings. Экосистема быстро развивается с появлением библиотек для HTML (tdom), баз данных

(sql-tstring) и других инструментов, что подтверждает актуальность инновации. По мере распространения Python 3.14 t-строки станут стандартной практикой безопасной разработки, сравнима с повсеместным использованием f-строк сегодня. Главное понимание заключается в том, что t-строки не заменяют f-строки, а дополняют их, расширяя возможности разработчиков в создании безопасных и надёжных приложений.

Литература

1. PEP 750 – Template Strings / Python Enhancement Proposals. – 2025. – URL: https://peps.python.org/pep-0750/ (дата обращения: 15.11.2025)
2. Real Python. Python 3.14 Preview: Template Strings (T-Strings). – 2025. – URL: https://realpython.com/python-t-strings/ (дата обращения: 18.11.2025)
3. PEP 701 – Syntactic formalization of f-strings / Python Enhancement Proposals. – 2021. – URL: https://peps.python.org/pep-0701/ (дата обращения: 18.11.2025)
4. BetterStack. What's New in Python 3.14. – 2025. – URL: https://betterstack.com/community/guides/scaling-python/python-3-14-new-features/ (дата обращения: 21.11.2025)
5. Mitigation of SQL Injection Attack using Prepared Statements / GeeksforGeeks. – 2017. – URL: https://www.geeksforgeeks.org/sql/mitigation-sql-injection-attack-using-prepared-statements-parameterized-queries/ (дата обращения: 21.11.2025)
6. How to Safely Sanitize HTML Before Rendering on Your Website or App / ButterCMS. – 2015. – URL: https://buttercms.com/knowledge-base/html-sanitization-best-practices/ (дата обращения: 21.11.2025)
7. Why Server-Side HTML Sanitization is Doomed to Fail / SonarSource Blog. – 2024. – URL: https://www.sonarsource.com/blog/sanitize-client-side-why-server-side-html-sanitization-is-doomed-to-fail/ (дата обращения: 21.11.2025)
8. Guide to structured logging in Python / NewRelic Blog. – 2025. – URL: https://newrelic.com/blog/how-to-relic/python-structured-logging (дата обращения: 21.11.2025)
9. Python Logging Format Tutorial with Examples / Middleware.io. – 2025. – URL: https://middleware.io/blog/python-logging-format/ (дата обращения: 18.11.2025)
10. Python string formatting history / PowerfulPython. – URL: https://powerfulpython.com/blog/python-string-formatting/ (дата обращения: 20.11.2025)
11. Writing a Domain Specific Language (DSL) in Python / GeeksforGeeks. – 2024. – URL: https://www.geeksforgeeks.org/python/writing-a-domain-specific-language-dsl-in-python/ (дата обращения: 21.11.2025)
12. Creating a DSL In Python / Dev.to. – 2023. – URL: https://dev.to/fractalis/creating-a-dsl-in-python-dj6 (дата обращения: 18.11.2025)
13. Structural Pattern Matching in Python: A Comprehensive Guide / BetterStack. – 2025. – URL: https://betterstack.com/community/guides/scaling-python/python-pattern-matching/ (дата обращения: 20.11.2025)
14. Tagged Template Literals / Wes Bos. – 2016. – URL: https://wesbos.com/tagged-template-literals (дата обращения: 18.11.2025)
15. Template literals (Template strings) - JavaScript | MDN. – 2025. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals (дата обращения: 20.11.2025)
16. Jinja2 / PyPI. – 2025. – URL: https://pypi.org/project/Jinja2/ (дата обращения: 20.11.2025)
17. Type annotations — typing documentation / Python Typing. – 2020. – URL: https://typing.python.org/en/latest/spec/annotations.html (дата обращения: 21.11.2025)
18. Explaining Python Type Annotations / Dev.to. – 2025. – URL: https://dev.to/leapcell/explaining-python-type-annotations-a-comprehensive-guide-to-the-typing-module-495i (дата обращения: 18.11.2025)

УДК 004.9

ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ ТЕСТИРОВАНИЯ НАВЫКОВ РАБОТЫ С SQL

А. С. Белых

Воронежский государственный университет

Аннотация. Данная работа посвящена проектированию и разработке веб-приложения для тестирования учащихся на примере проверки знаний языка структурированных запросов (SQL). В статье описан анализ задачи, методов и средств ее реализации и непосредственно процесс реализации. Результатом работы является веб-приложение, позволяющее проводить оценку уровня владения теоретической базой и практическими навыками по работе с SQL путем прохождения интерактивных тестов с автоматической проверкой ответов. Используемый стек: СУБД PostgreSQL, Java для серверной части, JavaScript для клиентской. Использование данного приложения способствует повышению эффективности преподавания информатики в вузах и школах.
Ключевые слова: веб-приложение, программирование, программное обеспечение, СУБД, SQL, PostgreSQL, Node.js, JavaScript, React, тестирование знаний, преподавание информатики, образовательные технологии.

Введение

В современном мире цифровых технологий навыки работы с базами данных, в частности, с языком SQL, являются ключевыми для специалистов в области информатики и разработки программного обеспечения. По данным РБК, к 2024 году рынок IT-образования в России вырос на 46,3 % по сравнению с показателями 2022 года[7], что подчеркивает необходимость подготовки квалифицированных кадров и использования инновационных инструментов для обучения. Традиционные методы проверки знаний учащихся (такие как тестирование на бумаге, простые онлайн-формы и т.п.) не позволяют в полной мере оценить практические навыки.

В связи с этим возникает потребность в разработке специализированного веб-приложения для тестирования, которое интегрирует язык структурированных запросов в образовательный процесс, делая обучение интерактивным и доступным. Описанный продукт не заменит полноценные курсы, однако сможет дополнить их, фокусируясь на практике. Данная работа основана на дипломном проекте автора по разработке веб-приложения для тестирования учащихся.

1. Постановка задачи

Необходимо разработать веб-приложение для тестирования навыков работы с SQL, обладающее следующей функциональностью:

1) Регистрация пользователя (студента или преподавателя) с сохранением личных данных в базу;

2) Возможность авторизации и выхода из аккаунта с возвратом на главную страницу;

3) Просмотр списка доступных тестов;

4) Прохождение теста: ответы на закрытые и открытые вопросы, вопросы с множественным выбором; ввод SQL-запросов в интерактивном редакторе с автоматической проверкой на правильность;

5) Для преподавателей: добавление новых тестов и вопросов.

Приложение должно обеспечивать безопасность хранения пользовательских данных и интеграцию с реляционной СУБД для хранения и выполнения запросов.

2. Анализ существующих решений

В ходе работы были изучены популярные платформы для тестирования в образовании: *Moodle*, *Quizlet*, *Google Forms* и специализированные SQL-тренажеры, такие как *SQLZoo*, *sql-ex.ru*, *LeetCode SQL*.

Выявлено, что Moodle предлагает широкую функциональность, однако интерфейс зачастую перегружен и требует администрирования. Quizlet удобен для составления флеш-карт, но не поддерживает проверку ответов. *SQLZoo*, *sql-ex* и *LeetCode* фокусируются на практике, однако оба варианта не предполагают составление и назначение авторских тестов; кроме того, первый из перечисленных вариантов не интегрируется с пользовательскими аккаунтами. Среди преимуществ указанных сервисов можно выделить интерактивные редакторы кода, визуализацию результатов, наличие статистики по решенным заданиям, из недостатков — отсутствие локализации для России в большинстве вариантов, недостаток необходимых для эффективного использования в образовательном процессе функций. Предлагаемый продукт сочетает простоту интерфейса с возможностью создавать и автоматически проверять тесты, что делает его более доступным для студентов.

3. Анализ средств реализации

Для реализации приложения были выбраны следующие инструменты:

1) *Node.js* — платформа для разработки серверной части приложения;
2) *React* — фреймворк для разработки клиентской части веб-приложения;
3) *JavaScript* — основной язык разработки;
4) *HTML* — язык разметки страниц;
5) *CSS* — средство стилизации;
6) *PostgreSQL* — реляционная СУБД для хранения данных о пользователях и тестах;
7) *Sequelize* — библиотека для взаимодействия с PostgreSQL;
8) *CodeMirror* — компонент для редактора кода в браузере;
9) *Jest* — фреймворк для модульного тестирования;
10) *Visual Studio Code* — среда разработки.

4. Схема взаимодействия модулей

Приложение построено по архитектуре *MVC* (*Model-View-Controller*):

- *Model*: модели данных для взаимодействия с *PostgreSQL*;
- *Controller*: обработка запросов, маршрутизация, выполнение SQL-запросов и проверка ответов;
- *View*: страницы, включая формы тестов и результаты;
- *PostgreSQL* — СУБД, хранящая данные приложения.

Схема взаимодействия программных компонентов приложения продемонстрирована на рис. 1.

Выбранный стек технологий демонстрирует оптимальное сочетание производительности и эффективности для образовательных целей.

5. Структура данных

Для хранения данных в веб-приложении используется реляционная СУБД PostgreSQL. База данных нормализована в соответствии с нуждами задачи. ER-диаграмма логического уровня изображена на рис. 2.

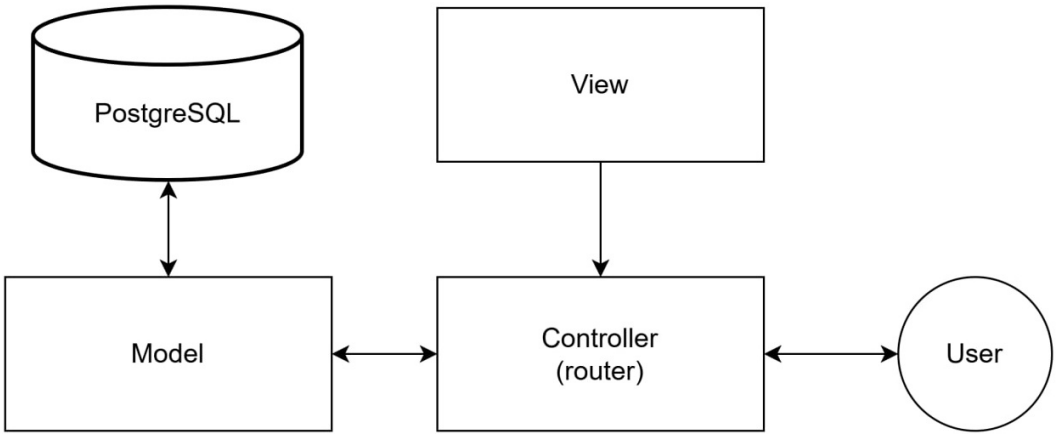

Рис. 1. Схема взаимодействия модулей приложения

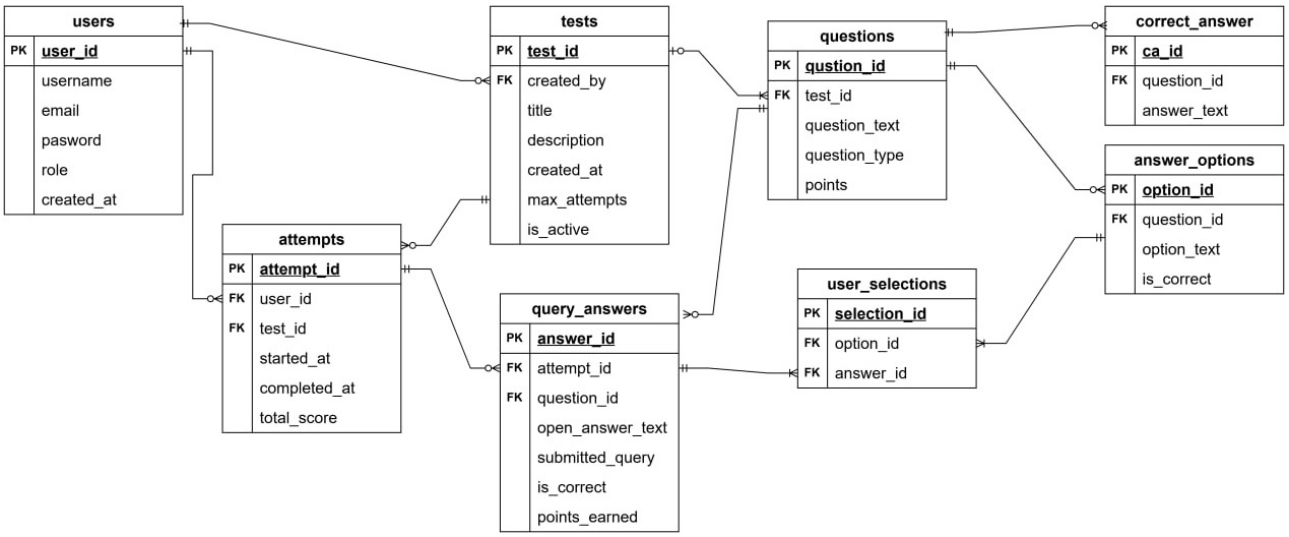

Рис. 2. ER-диаграмма базы данных приложения

6. Пользовательский интерфейс

Интерфейс приложения разработан с акцентом на минимализм. Со страницы регистрации/авторизации осуществляется переход на главную страницу, содержащую список доступных тестов, ссылку на профиль пользователя. Страница теста предоставляет возможность прохождения проверки теоретических знаний и практических навыков в браузерном редакторе кода.

На рис. 3 показан интерфейс основной страницы приложения.

7. Тестирование

Для того, чтобы убедиться в корректной работе как отдельных элементов веб-приложения, так и их взаимосвязей, необходимо провести модульное и интеграционное тестирование.

Тестирование включает проверку правильности выполнения ключевых функций продукта: регистрации в системе, авторизации, выхода из аккаунта, прохождения тестовых заданий (как с ожидаемыми, так и с заведомо неверными ответами), отображение результатов выполнения задач.

Помимо прочего, приложение прошло бета-тестирование группой из 15 студентов, показав высокую вовлеченность и удовлетворенность более чем 85 % опрошенных.

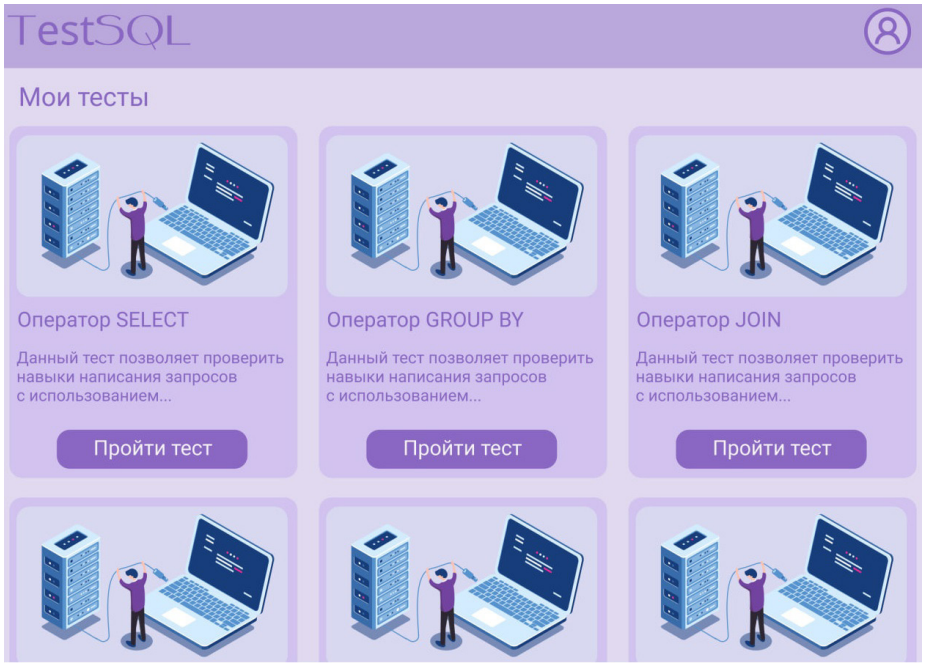

Рис. 3. Интерфейс основной страницы приложения

Заключение

Результатом работы является веб-приложение, позволяющее оценить навыки владения языком структурированных запросов (SQL) путем теоретического и практического тестирования. В процессе создания продукта были установлены ключевые требования к функциональности на основе анализа существующих решений, выбраны наиболее подходящие для реализации поставленной задачи средства и инструменты разработки, составлена архитектура системы. Разработанное приложение отвечает поставленным задачам, что было проверено путем тестирования.

Литература

1. *Гарсиа-Молина Г., Ульман Дж. Д., Уидом Дж.* Системы баз данных: полный курс. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2017. – 1088 с.

2. *Дронов В. А., Прохоренок Н. А.* JavaScript и Node.js для веб-разработчиков. – СПБ. : БХВ, 2022. – 768 с.

3. *Жао Э.* SQL. Pocket guide – 4-е изд. : пер. с англ. – Астана : «Спринт Бук», 2024. – 320 с.

4. *Матвеева М. В.* Проектирование баз данных. Нормализация. : учеб.-метод. пособие. – Воронеж : ВГУ, 2018. – 51 с.

5. *Османи Э.* Изучаем паттерны проектирования JavaScript. – 2-е изд. : пер. с англ. – Астана : АЛИСТ, 2024. – 272 с.

6. *Сухов К.* Node.js. Путеводитель по технологии. – М. : ДМК Пресс, 2015. – 416 с.

7. РБК Статьи. Рынок ИТ-образования в России: рост потребления и новые предпочтения [Электронный ресурс]. – URL: https://marketing.rbc.ru/articles/16082/ (дата обращения: 14.10.2025).

8. Официальная документация Express.js [Электронный ресурс] // Express.js Basic routing. – URL: https://expressjs.com/en/starter/basic-routing.html (дата обращения: 21.06.2025).

9. Официальная документация Node.js [Электронный ресурс] // Node.js. – Introduction to Node.js. – URL: https://nodejs.org/en/learn (дата обращения: 28.05.2025).

10. Официальная документация PostgreSQL [Электронный ресурс] // Документация к PostgreSQL 18.1. – URL: https://postgrespro.ru/docs/postgresql/current (дата обращения: 11.04.2025).

11. Официальная документация React [Электронный ресурс] // Passing props to a component. – URL: https://react.dev/learn/passing-props-to-a-component (дата обращения: 13.07.2025).

12. Официальная документация Sequelize [Электронный ресурс] // URL: https://codemirror.net/docs/ (дата обращения: 28.09.2025).

13. Современный учебник JavaScript [Электронный ресурс] // Регулярные выражения. – URL: https://learn.javascript.ru/regular-expressions (дата обращения: 11.05.2025).

14. *Brown E.* Web Development with Node and Express. [Веб-разработка с применением Node и Express.]. – USA: O'Reilly Media, Inc., 2014. – 329 p.

УДК 004.9

ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ПО ПОИСКУ ПОПУТЧИКОВ ДЛЯ МЕЖДУГОРОДНИХ ПОЕЗДОК

С. А. Бологов, Т. В. Курченкова

Воронежский государственный университет

Аннотация. В данной работе рассматривается процесс проектирования веб-приложения, предназначенного для поиска попутчиков в междугородних поездках и получения информации, а также покупки билетов городских автовокзалов. Веб-приложение спроектировано под браузерные движки Blink версия (1.0.1+), Gecko (версия 139.0+), WebKit (версия 139.0.7220.0. +) с использованием языка программирования C#, СУБД PostgreSQL и фреймворка ASP .NET Core. Поиск автобусных билетов осуществляется с помощью обращения к API Яндекс Расписаний. В работе приведены основные этапы проектирования, структура базы данных и ключевые элементы интерфейса пользователя.
Ключевые слова: веб-приложение, UML, API, PostgreSQL, C#, ASP .NET Core, междугородние поездки.

Введение

У многих людей есть необходимость по работе, учёбе или личным делам совершать поездки в разные города. Однако организация таких поездок не всегда удобна: расписание автобусов и поездов не всегда соответствуют индивидуальным планам, а цены на билеты отдельных рейсов могут быть высокими.

Большинство онлайн-сервисов поиска попутчиков или сайтов по продаже автобусных билетов, решают эти задачи частично, не предоставляя единого решения для возможности выбора способа добраться до места назначения.

Проектируемое веб-приложение значительно упростит организацию поездок и сделает передвижение между городами более гибким и доступным для пользователей.

1. Постановка задачи

Необходимо спроектировать веб-приложение под браузерные движки Blink версия (1.0.1+), Gecko (версия 139.0+), WebKit (версия 139.0.7220.0. +) [1], предоставляющее обширную функциональность по поиску попутчиков и автобусных билетов городских автовокзалов. Веб-приложение должно обеспечивать возможность регистрации и авторизации, поиска опубликованных поездок и билетов автовокзалов, с возможностью бронирования, публикации собственных поездок в качестве водителя, просмотра истории поездок.

2. Функциональность

Основная функциональность спроектированного приложения представлена на диаграмме вариантов использования [2] (рис. 1).

3. Средства реализации

Для проектирования приложения использовались: язык программирования C# (13), фреймворк ASP .NET Core (8.0.16), IDE Visual Studio 2022 (17.4.0), СУБД PostgreSQL (17.4) с расши-

Рис. 1. Диаграмма вариантов использования

рением PostGIS, платформа контейнеризации Docker Desktop (4.42.0) и инструменты моделирования WhiteStarUML (5.5.8.0) и SAP PowerDesigner (16.6.10.0).

ASP .NET Core был выбран за высокую производительность, модульную архитектуру, кроссплатформенность и встроенные средства безопасности. В качестве СУБД, используется надёжная объектно-реляционная СУБД — PostgreSQL, так как она поддерживает хранение географических данных с помощью расширения PostGIS.

4. Проектирование веб-приложения

Веб-приложение спроектировано на основе микросервисной архитектуры [3]. В отличие от монолитной, она позволяет легко масштабировать приложение, также у отдельных микросервисов нет привязки к определенному языку программирования, что делает реализацию более гибкой. Чтобы сервисы составляли единый исполняемый файл, был использован механизм контейнеризации Docker.

Диаграмма классов, образующих сервис аутентификации представлена на рис. 2.

Сервис для взаимодействия со списком автобусов из API имеет иную структуру и представлен на рис. 3.

Каждый класс приложения включает ряд методов. Рассмотрим описание основных классов и методов (табл. 1):

Часть логической модели базы данных, связанная с бронированием поездок представлена на рис. 4.

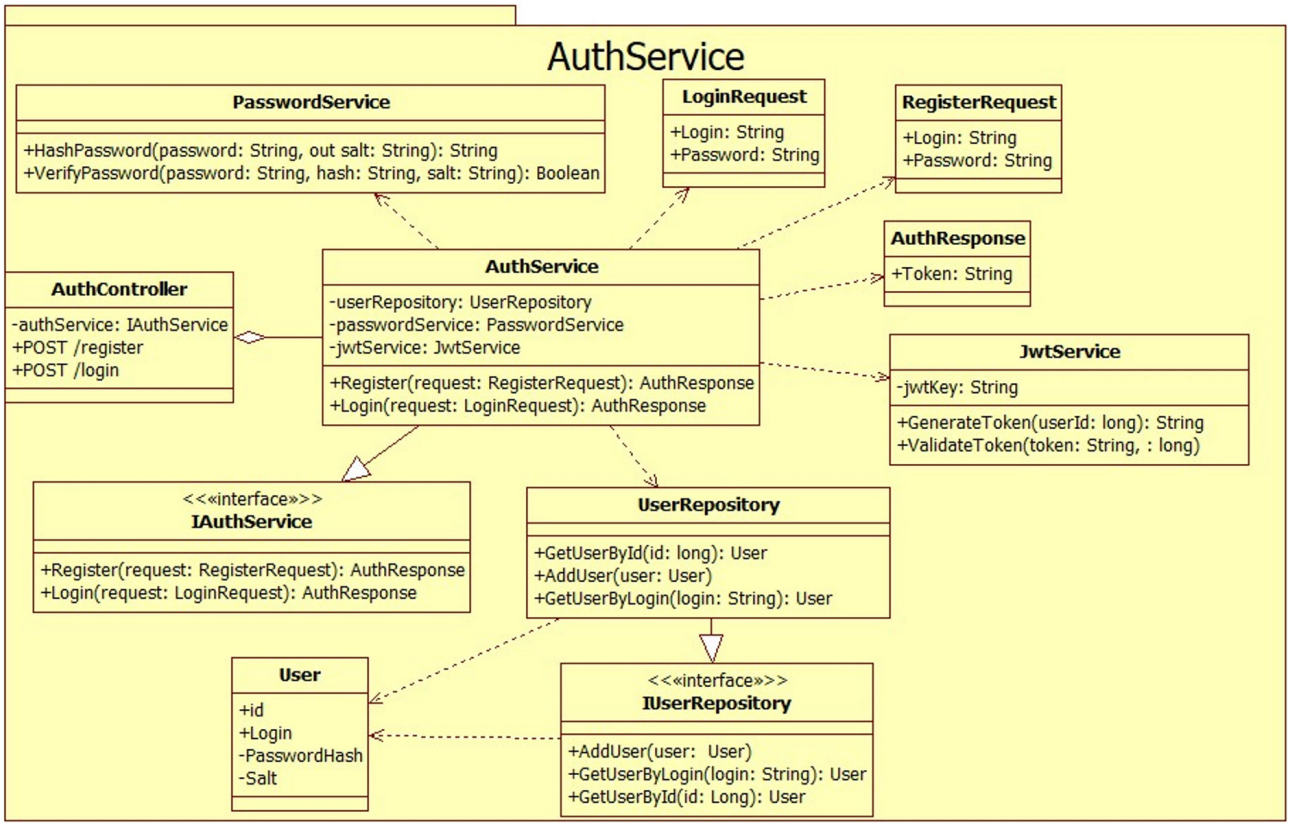

Рис. 2. Диаграмма классов сервиса аутентификации

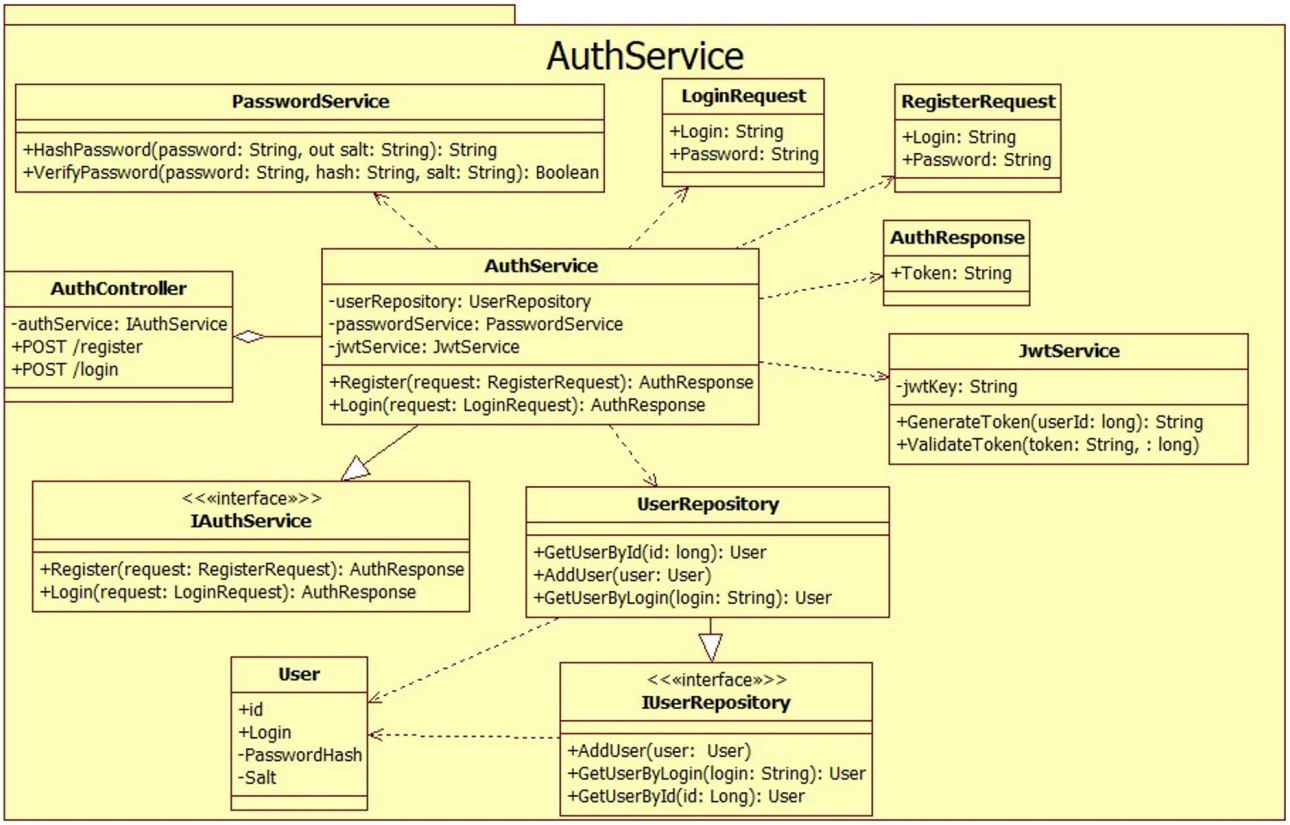

Рис. 3. Диаграмма классов сервиса взаимодействия с API

Таблица 1

Описание классов приложения

Класс	Метод	Описание
AuthService		Основной сервис, реализующий бизнес-логику регистрации и входа.
	`Register(RegisterRequest request)`	Регистрирует нового пользователя: хеширует пароль, сохраняет пользователя в базу, возвращает JWT токен.
	`Login(LoginRequest request)`	Проверяет логин и пароль, возвращает JWT токен, если данные верны.
AuthController		Обрабатывает HTTP-запросы /register и /login.
	`POST /register`	Обрабатывает HTTP-запрос на регистрацию, вызывает метод Register
	`string HashPassword(string password, out string salt)`	Создает соль и хеширует пароль с помощью неё.
PasswordService		Инкапсулирует логику хеширования и проверки пароля. Генерирует соль, хеширует пароль, проверяет его корректность.
	`bool VerifyPassword(string password, string hash, string salt)`	Проверяет пароль, сравнивая хеши.
	`string GenerateToken(long userId) bool`	Генерирует JWT токен с Id пользователя.
JwtService		Проверяет JWT токены. Включает в токен ID пользователя, валидирует подпись и срок действия токена.
	`bool ValidateToken(string token, out long userId)`	Проверяет JWT токен и извлекает Id пользователя.
RegisterRequest		DTO: содержит Login и Password для регистрации.
LoginRequest		DTO: содержит Login и Password для входа.
AuthResponse		DTO: содержит Token в ответ на регистрацию или вход.
User		Сущность: содержит Id, Login, PasswordHash, Salt.
YandexScheduleService		Основной сервис для работы с API Яндекс.Расписания. Делает HTTP-запросы, обрабатывает ответы.
	`SearchBusesAsync (BusSearchModel model)`	Делает запрос в API Яндекс.Расписания и возвращает список междугородних автобусов.
	`GetBusByIdAsync(string id)`	Получает расписание автобуса по ID через API Яндекса.

YandexScheduleController		Принимает параметры через BusSearchRequest, вызывает сервис, возвращает BusScheduleResponse
	`GET /buses`	Возвращает список автобусов по параметрам
	`GET /buses/{id}`	Возвращает автобус по ID.
	`SearchBusesAsync (BusSearchModel model)`	Интерфейс: поиск автобусов.

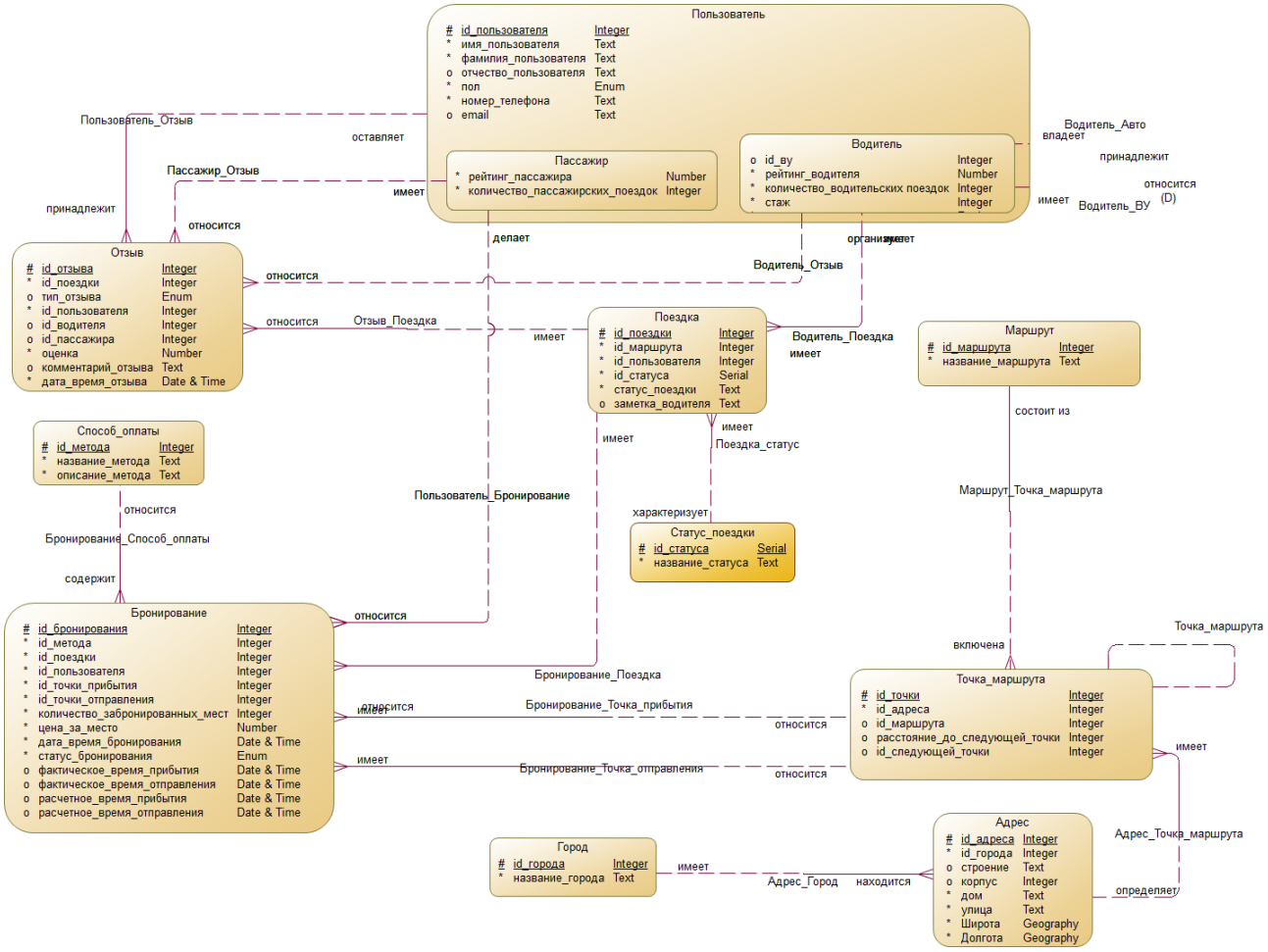

Рис. 4. Часть логической модели базы данных для хранения данных о поездках

Вторая часть логической модели связана с хранением пользовательских данных представлена на рис. 5.

Заключение

Спроектированное веб-мобильное приложение предоставляет функциональность для более гибкого и удобного перемещения между городами, что поможет не только иногородним студентам добираться на учебу, но и поможет стимулировать междугородний туризм.

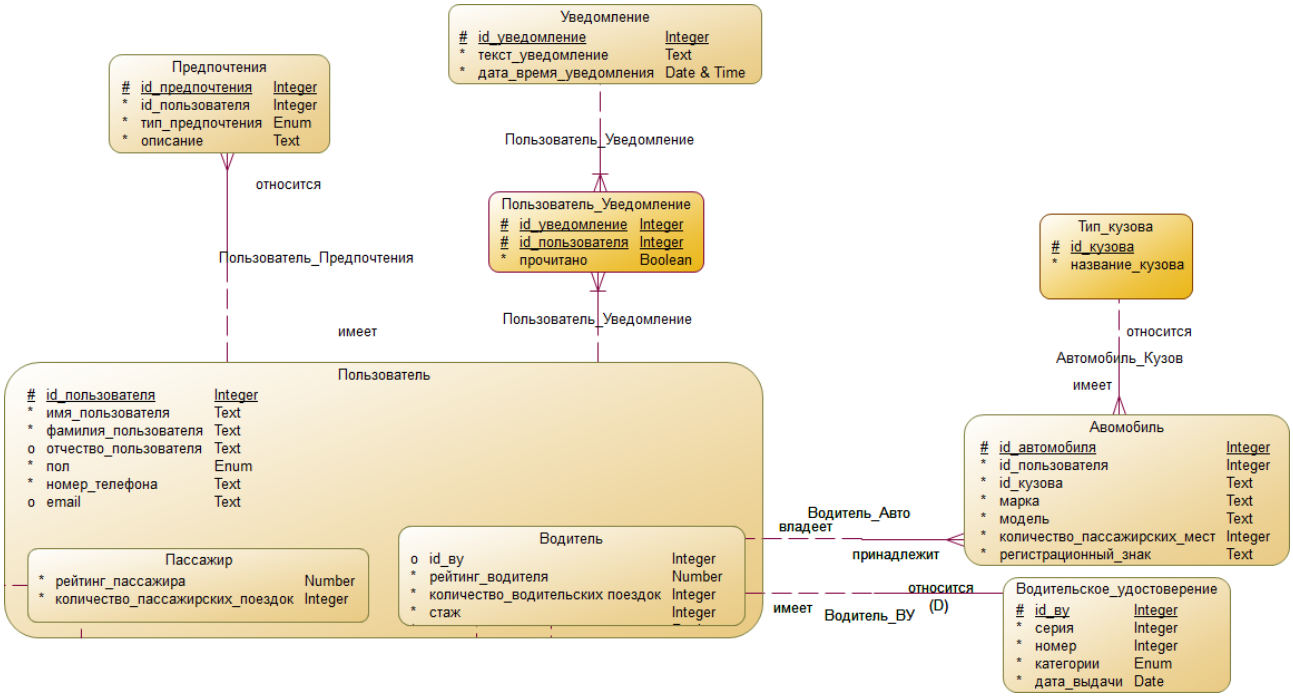

Рис. 5. Часть логической модели базы данных для хранения пользовательских данных

Литература

1. ASP.NET Core. Разработка приложений. – СПб.: Питер, 2018. – 464 с.

2. *Буч Г.* Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон ; пер. с англ. Н. Мухина. – 2-е изд. – Москва : ДМК Пресс, 2006. – 496 с.

3. Внедрение зависимостей .NET. – URL: https://learn.microsoft.com/ru-ru/dotnet/core/extensions/dependency-injection (дата обращения: 16.11.2025).

УДК 004.65

ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ АНАЛИЗА ДАННЫХ ИНТЕРНЕТ-ЗАКАЗОВ С ИСПОЛЬЗОВАНИЕМ NOSQL ХРАНИЛИЩА

В. Г. Булгаков, И. Ф. Астахова

Воронежский государственный университет

Аннотация. Данная работа посвящена проектированию приложения для анализа информации о заказах и путях их доставки. В статье описываются анализ задачи, доступных аналогов, а также методов и средств ее реализации. Результатом работы является готовый проект веб-приложения, позволяющего собирать и обрабатывать данные заказов. В качестве хранилища данных используется *MongoDB*, основной платформой является фреймворк *Django*. Готовый проект может использоваться как основа для дальнейшего развития аналитических систем в сфере электронной коммерции.
Ключевые слова: веб-приложение, программирование, программное обеспечение, СУБД, Python, Django, NoSQL, MongoDB.

Введение

Современный рынок электронной коммерции характеризуется стремительным ростом объёмов продаж и усложнением логистических процессов. По мере увеличения количества интернет-заказов возрастает потребность не только отслеживать динамику популярности товарных категорий по временным периодам, но и анализировать загруженность логистических маршрутов, оптимизировать доставку и повышать эффективность внутренних процессов. В этих условиях веб-приложения, объединяющие данные о заказах, товарах и транспортных узлах, приобретают особую актуальность.

В связи с этим возникает потребность в разработке системы, ориентированной на продуктовых аналитиков и менеджеров, для анализа интернет-заказов, которое позволит отслеживать популярность товарных категорий по временным периодам, а также оценивать загруженность путей доставки на основе координат складов и данных о маршрутах.

1. Постановка задачи

Необходимо разработать приложение-сервис, обеспечивающее комплексный анализ данных интернет-заказов, доступный в виде *SaaS*. Разрабатываемое решение должно обеспечивать обработку больших объёмов неструктурированных данных. Приложение должно обладать следующей функциональностью:

1. Возможность входа в существующий аккаунт и выхода из него с возвратом на главный экран. Новые аккаунты добавляются вне приложения.
2. Просмотр аналитики по выполненным заказам, включающей в себя:
   2.1. Информацию об общем числе заказов.
   2.2. Информацию о среднем чеке и количестве товаров в заказе.
   2.3. Диаграмму популярности товаров по категориям.
   2.4. Фильтрацию данных по временным периодам (кварталам) и статусам заказов.
3. Просмотр аналитики по логистике с использованием интерактивной карты. Доступна фильтрация данных по кварталам.

2. Анализ существующих решений

Для выявления закономерностей и общих принципов работы приложений для аналитики в ходе работы были изучены следующие существующие на рынке решения: *Tableau*, *Google Analytics* и *Power BI*.

Tableau предлагает многочисленные инструменты для визуализации и гибкой настройки вычислений вкупе с возможностью подключения к разнообразным источникам данных. Однако *Tableau* не предоставляет встроенных модулей обработки данных, специфичных для электронной коммерции. Для получения корректных аналитических показателей данные должны быть предварительно подготовлены. Кроме того, *Tableau* имеет сравнительно высокий порог стоимости и требует обучения для эффективного использования.

Google Analytics предоставляет возможность отслеживать продажи по категориям товаров, а также динамику заказов по периодам с визуализацией полученной статистики. Тем не менее в нем отсутствует аналитика логистики и доставок, их маршрутов и складов. Таким образом данное приложение специализируется в большей степени на маркетинговой аналитике.

Power BI предоставляет пользователям инструменты для построения отчётов, дашбордов, анализа продаж и прогнозирования, а также обладает удобной интеграцией с другими продуктами *Microsoft*. Одним из преимуществ платформы является сравнительно низкий порог входа и доступность для широкого круга пользователей. К минусам *Power BI* относится отсутствие автоматического вычисления *e-commerce* и логистических метрик (таких как загруженность складов, анализ маршрутов доставки).

Было выявлено, что каждое из рассмотренных приложений обладает как собственными преимуществами, так и недостатками. Во всех примерах отсутствовала работа с логистикой: ее визуализация и аналитика — что критически важно для бизнеса, связанного с доставками и перемещениями товаров внутри собственной системы складов. Соответственно требуется разработать оригинальную систему визуализации логистических данных. Также большинство платформ имеют переусложненный интерфейс, что создает потребность в дополнительном обучении персонала и настройке данных перед работой. Во избежание этого необходимо упростить интерфейс и обрабатывать данные непосредственно в приложении.

3. Анализ средств реализации

Для реализации приложения были выбраны следующие инструменты:

1. *HTML* — язык разметки страницы.
2. *CSS* — язык описания внешнего вида страниц.
3. *Python* — язык разработки.
4. *Django* — основной фреймворк разработки, нацеленных на удобную разработку веб-приложений на ЯП *Python*.
5. *MongoDB* — СУБД.
6. *Djongo* — библиотека для создания соединения между *MongoDB* и *Django*.
7. *Leaflet* — библиотека для отображения карт.
8. *PyCharm* — среда разработки.

3. Схема взаимодействия модулей

Приложение состоит из модулей, взаимодействующих между собой (рис. 1). Схема взаимодействия представляет из себя модель *MTV* (*Model-Template-View*), которая является логическим развитием модели *MVC* (*Model-View-Controller*) в рамках работы с фреймворком *Django* [2]:

• *Model* — отвечает за работу с данными и взаимодействие с базой данных: описывает структуру данных, правила хранения и предоставляет интерфейс для выполнения операций *CRUD* через *ORM*;

• *Template* — определяет внешний вид пользовательского интерфейса: содержит *HTML*-разметку и шаблонный язык *Django* на основе *Jinja* для отображения динамического контента [3];

• *View* — обрабатывает запросы и формирует ответы: получает данные из моделей, выполняет логику приложения и передаёт результаты в шаблоны для отображения пользователю [4];

• *MongoDB* — база данных, хранящая всю необходимую информацию.

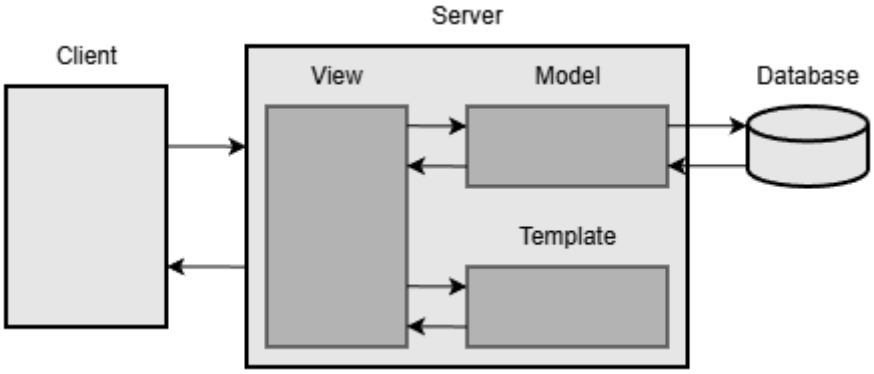

Рис. 1. Схема взаимодействия модулей приложения

Для визуализации порядка взаимодействия модулей приложения используется диаграмма последовательностей (рис. 2).

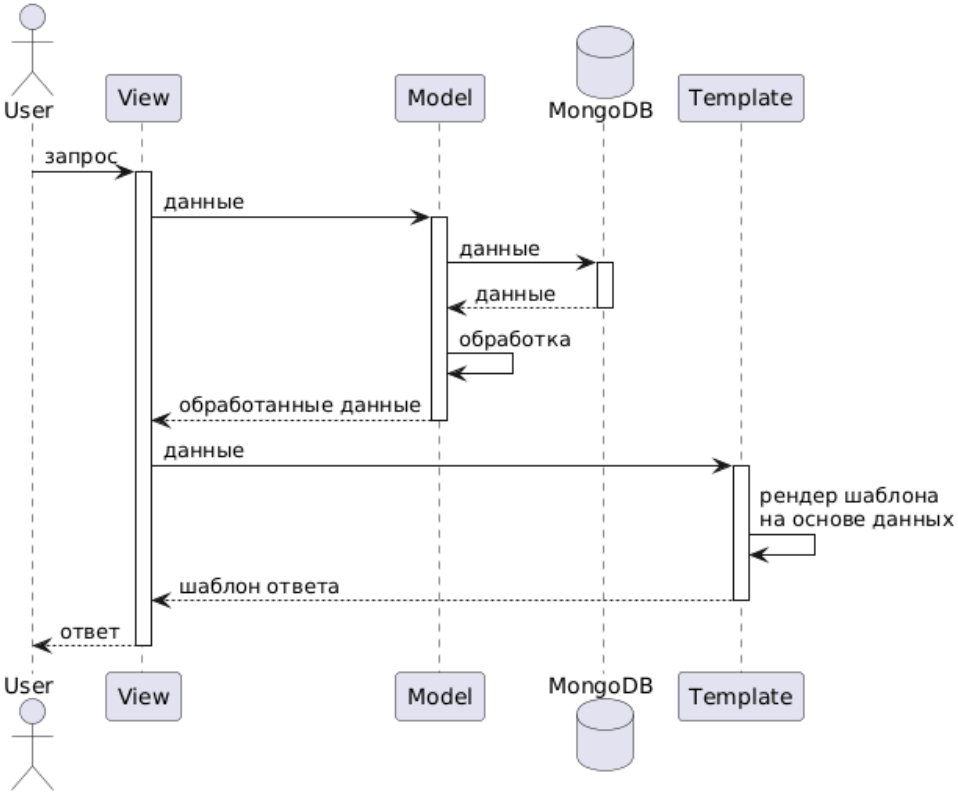

Рис. 2. Диаграмма последовательностей

Для обеспечения удобства использования и логичной навигации в разрабатываемом приложении была сформирована карта переходов между основными интерфейсными страницами (рис. 3). Поскольку система ориентирована на продуктовых аналитиков, ключевыми требова-

ниями к интерфейсу являются простота доступа к данным, минимизация количества лишних действий и чёткое разделение функциональных областей. Это особенно важно в условиях рабочей нагрузки, когда пользователю необходимо быстро получить нужную информацию о продажах или логистике.

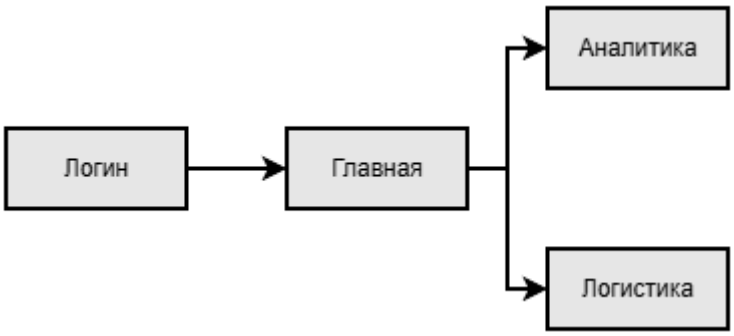

Рис. 3. Карта переходов приложения

Представленная карта переходов позволяет структурировать пользовательский интерфейс таким образом, чтобы доступ к аналитическим модулям был возможен только после авторизации, обеспечивая необходимый уровень безопасности. Разделение на страницы анализа по кварталам и логистики по кварталам отражает основные направления работы пользователей и делает навигацию интуитивно понятной. Данная архитектура интерфейса способствует повышению эффективности взаимодействия с системой и упрощает освоение приложения.

4. Структура данных

Для организации информации в *MongoDB* был выбран метод частичного встраивания информации, где для данных о пользователях в коллекции *users* встраивание не используется, в то время как для заказов в коллекции *orders* информация о содержимом и точках перемещения встраивается непосредственно в документ. Гибкость структуры данных в *MongoDB* позволяет хранить разнородные данные, не имеющими четко установленного списка и порядка атрибутов, однако для корректной работы приложения необходимо, чтобы определенные поля содержались в каждом документе коллекции [1].

Обязательные поля *users*:

- *id* — автоматически генерируемый при создании документа идентификатор пользователя;
- *login* — логин пользователя, отображаемый на главной странице. Тип *String*;
- *password* — хэшированный для повышения безопасности пароль пользователя. Тип *String*;
- *email* — электронная почта пользователя. Тип *String*.

Обязательные поля *orders*:

- *id* — автоматически генерируемый при создании документа идентификатор заказа;
- *price* — цена заказа. Тип *Number*;
- *customer* — идентификатор пользователя, совершившего заказ. Тип *String*;
- *status* — статус заказа. Тип *String*;
- *items* — список товаров в заказе. Каждый его элемент имеет следующие обязательные поля: *name* (название товара), *price* (собственная цена товара), *quantity* (количество экземпляров в заказе), *image* (ссылка на изображение).

Помимо вышеуказанных, в случае пересылки между несколькими точками на карте, в заказе может находиться поле *stocks* — массив координат со следующими полями: *lat* (широта), *lng* (долгота), *arrivalDate* (пометка о прибытии в виде даты формата дд.мм.гггг).

Заключение

В рамках работы было спроектировано веб-приложение для анализа и визуализации данных интернет-заказов с использованием *NoSQL*-хранилища. Предложенная архитектура на базе *MongoDB* и *Django* обеспечивает эффективную обработку неструктурированных данных, характерных для *e-commerce* (при условии наличия ключевых полей в документе), а также предоставляет возможности квартального анализа товарных категорий и логистических маршрутов. Разработка модели взаимодействия компонентов и карты переходов позволила сформировать самодостаточную архитектуру, ориентированную на продуктовых аналитиков и менеджеров.

Литература

1. *Бэнкер К.* MongoDB в действии / К. Бэнкер; пер. с англ. А. А. Слинкина. – М. : ДМК Пресс, 2020. – 394 с.

2. *Дронов В. А.* Django 4. Практика создания веб-сайтов на Python / В. А. Дронов. – СПБ. : БХВ, 2023. – 690 с.

3. *Меле А.* Django 4 в примерах / А. Меле; пер. с англ. А. В. Логунова. – М. : ДМК Пресс, 2023. – 800 с.

4. Официальная документация Django // Django Documentation – Режим доступа: https://www.djangoproject.com/

УДК 004.415

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ ДЛЯ ПОСТРОЕНИЯ ВЕБ-ПЛАТФОРМ УПРАВЛЕНИЯ УЧЕБНЫМИ ПРОЦЕССАМИ

Д. А. Вигерин, С. Ю. Болотова

Воронежский государственный университет

Аннотация. В данной работе представлен систематизированный обзор современных технологий и архитектурных подходов, применимых для разработки специализированных веб-платформ управления учебными задачами. Рассмотрены критерии выбора технологического стека, включая производительность, безопасность и масштабируемость. Актуальность темы обусловлена растущей сложностью образовательной экосистемы, распространением гибридных форматов обучения и наличием функционального разрыва между традиционными системами управления обучением (LMS) и системами управления проектами.

Ключевые слова: образовательные технологии, управление учебными процессами, веб-платформы, обзор технологий, Spring Boot, безопасность веб-приложений, микросервисная архитектура, системы хранения данных.

Введение

Современная образовательная экосистема характеризуется возрастающей сложностью и динамичностью. Студенты и преподаватели высших, средних и дополнительных учебных заведений сталкиваются с необходимостью эффективного управления большим объемом академических задач, проектов и дедлайнов. Традиционные методы организации, такие как бумажные ежедневники или разрозненные цифровые инструменты, зачастую не справляются с требованием обеспечения прозрачности, контроля прогресса и оперативного взаимодействия между всеми участниками образовательного процесса. В условиях роста популярности дистанционного и гибридного форматов обучения, а также увеличения доли проектной деятельности, потребность в специализированных, целостных платформах для управления учебными задачами становится особенно актуальной.

Существующие на рынке системы управления проектами зачастую избыточны и не адаптированы под специфику образовательного контекста. Они не учитывают ключевые роли студента и преподавателя, не предоставляют встроенных механизмов для проверки заданий, формирования отчетности по успеваемости и централизованного управления учебными группами. В свою очередь, многие академические системы фокусируются на контенте и тестировании, оставляя без внимания инструментарий для персонального таск-менеджмента и коллаборации между студентами. Таким образом, возникает ниша для разработки целевого решения, которое интегрирует в себе лучшие практики управления задачами и глубокое понимание образовательных процессов.

1. Обзор литературных источников

Современные вызовы в области образования, включая переход к дистанционным и гибридным форматам обучения, обострили потребность в специализированных системах управления учебными процессами. Традиционные системы управления обучением (LMS) зачастую не обеспечивают необходимой гибкости для управления индивидуальными образовательными траекториями и проектными заданиями, в то время как системы управления проектами не адаптированы под специфику образовательного контекста. Это создает предпосылки для разработки специализированных решений, сочетающих преимущества обоих подходов.

Анализ современных подходов к разработке веб-приложений

Исследование [5] предоставляет комплексный анализ фреймворков JVM для разработки enterprise-приложений. Авторы демонстрируют, что Spring Boot обеспечивает оптимальное сочетание производительности и функциональности для систем со сложной бизнес-логикой, подобных «TaskTracker». Важным преимуществом Spring Boot является встроенная поддержка модуля Spring Security, что критически важно для образовательных платформ с требованием надежного разграничения прав доступа.

Сравнительное исследование [6] подтверждает эффективность Spring Boot для построения RESTful API, отмечая его преимущества в контексте разработки SPA-приложений. Авторы подчеркивают, что правильный выбор бэкенд-фреймворка непосредственно влияет на производительность системы при работе с большим количеством одновременных подключений, что особенно актуально для образовательных платформ, используемых большим количеством студентов и преподавателей.

Исследование механизмов безопасности веб-приложений

Фундаментальные аспекты безопасности веб-приложений рассматриваются в монографии [7]. Автор детально анализирует современные угрозы и методы защиты, уделяя особое внимание механизмам аутентификации и авторизации. Исследование демонстрирует, что комбинированный подход с использованием JWT и Spring Security обеспечивает необходимый уровень защиты для образовательных платформ, обрабатывающих конфиденциальную академическую информацию.

В работе [8] глубоко исследуются возможности JSON Web Token для обеспечения безопасности современных веб-приложений. Авторы доказывают преимущества JWT перед традиционными сессионными механизмами в контексте масштабируемых SPA-приложений, отмечая их эффективность для систем с большим количеством пользователей и распределенной архитектурой.

Исследование [10] посвящено интеграции Spring Security Framework с современными протоколами аутентификации. Авторы демонстрируют, что комбинация Spring Security и JWT обеспечивает надежную основу для реализации сложных систем разграничения прав доступа, что особенно важно для образовательных платформ с многоуровневой ролевой моделью.

Анализ систем хранения данных для образовательных платформ

Систематический обзор [11] предоставляет комплексный анализ подходов к хранению данных в распределенных системах. Авторы обосновывают эффективность комбинированного подхода с использованием различных СУБД для решения специфических задач, что подтверждает правильность выбора PostgreSQL для структурированных данных, MongoDB для документоориентированного хранения и Redis для кэширования.

В контексте проектирования системы хранения данных для «TaskTracker» особую ценность представляет анализ возможностей PostgreSQL для работы со сложными структурами данных [1]. Исследование подтверждает, что поддержка JSONB и оконных функций делает PostgreSQL оптимальным выбором для систем, требующих гибкости в хранении и обработке разнородных данных.

Исследование архитектурных решений для распределенных систем

Работа [2] анализирует современные тенденции развития микросервисных архитектур. Авторы предлагают практические рекомендации по построению масштабируемых распределенных систем, что имеет непосредственное отношение к проектированию системы уведомлений «TaskTracker». Исследование подтверждает эффективность использования Apache Kafka для обеспечения надежной доставки сообщений в системах реального времени.

Исследование Зиминой К. И. и Лапониной О. Р. [3] посвящено механизмам межсервисной аутентификации в микросервисных архитектурах. Авторы рассматривают различные подходы к обеспечению безопасности взаимодействия между компонентами распределенной системы, что особенно актуально для «TaskTracker» в контексте интеграции системы уведомлений с основным приложением.

Анализ подходов к управлению доступом в веб-приложениях

В работе А.В. Беловодова и О.Р. Лапониной [4] рассматриваются современные подходы к управлению доступом на основе атрибутов. Авторы предлагают архитектурные решения для реализации сложных политик доступа, что имеет непосредственное практическое значение для «TaskTracker» с его требованием гибкого разграничения прав между студентами, преподавателями и администраторами.

Исследование [9] демонстрирует эффективность использования Keycloak для защиты API веб-приложений. Авторы подтверждают, что правильная реализация механизмов авторизации является критически важным аспектом безопасности образовательных платформ, обрабатывающих персональные данные пользователей.

Исследование принципов проектирования программного обеспечения

Работа [12] заложила основы современных подходов к проектированию программного обеспечения. Принципы, изложенные в этой монографии, были применены при проектировании модульной архитектуры «TaskTracker», обеспечивая возможность легкого расширения функциональности платформы.

В книге Robert C. Martin [13] рассматриваются принципы чистой архитектуры, которые были использованы при проектировании «TaskTracker». Автор подчеркивает важность разделения ответственности между компонентами системы, что особенно актуально для сложных веб-приложений с длительным жизненным циклом развития.

Обобщение результатов анализа литературы

Проведенный анализ литературы позволил сформировать комплексное представление о современных подходах к разработке образовательных платформ. Рассмотренные исследования демонстрируют эффективность выбранного технологического стека для решения специфических задач управления учебными процессами.

Особую ценность представляют работы, посвященные безопасности веб-приложений [7, 8, 10], которые подтвердили обоснованность выбора JWT и Spring Security для системы аутентификации «TaskTracker». Исследования в области микросервисных архитектур [2, 3] предоставили методологическую основу для проектирования масштабируемой системы уведомлений.

Анализ источников по системам хранения данных [1, 11] позволил обосновать выбор комбинированного подхода с использованием PostgreSQL, MongoDB и Redis, что обеспечивает оптимальное соотношение производительности и надежности для образовательной платформы.

Результаты проведенного анализа литературы были успешно применены при проектировании и реализации платформы «TaskTracker», что подтверждает практическую значимость рассмотренных исследований и их соответствие современным тенденциям разработки программного обеспечения для образовательных целей. Выбранные технологические решения обеспечивают необходимую масштабируемость, безопасность и производительность, позволяя эффективно решать задачи управления учебными процессами в современных образовательных учреждениях.

Заключение

В результате проведенного сравнительного анализа можно сделать вывод, что современный инструментарий для разработки веб-приложений предлагает мощные средства для создания специализированных образовательных платформ. Таким образом, на основе данного анализа можно сформировать архитектурный каркас для построения перспективных образовательных платформ, способных эффективно решать задачи управления учебными процессами в условиях современной динамичной образовательной среды.

Литература

1. *Градусов А. Б.* Базы данных: Введение в технологию баз данных : учебно-практическое пособие / А. Б. Градусов ; Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых. – Владимир : Изд-во ВлГУ, 2021. – 208 с.

2. *Стоянов Ж.* Направления будущих исследований и рекомендации по развитию микросервисной архитектуры / Ж. Стоянов, И. Христоски // Программные системы: теория и приложения. – 2024. – Т. 36, № 1. – С. 26.

3. *Зимина К. И.* Механизмы межсервисной аутентификации в приложениях с микросервисной архитектурой / К. И. Зимина, О. Р. Лапонина // Информация и безопасность. – 2023. – Т. 11, № 5. – С. 146–154.

4. *Беловодов А. В.* Использование управления доступом на основе атрибутов в протоколе OAuth 2.0 / А. В. Беловодов, О. Р. Лапонина // Труды института системного программирования РАН. – 2023. – Т. 11, № 6. – С. 182–189.

5. *Wyciślik Ł.* A comparative assessment of JVM frameworks to develop microservices / Ł. Wyciślik, Ł. Latusik, A. M. Kamińska // Applied Sciences. – 2023. – Vol. 13, No. 3. – P. 1343.

6. A Performance Comparison of Web Frameworks for Building RESTful Web Services / G. Antonio, J. R. Jose, S. A. Filipe, C. Filipe // Journal of Computer Science and Integration (JCSI). – 2025. – Vol. 35. – P. 121–128.

7. *McDonald M.* Web Security for Developers: Real Threats, Practical Defense / M. McDonald. – San Francisco : No Starch Press, 2020. – 312 p.

8. *Mahindraka P.* Insights of JSON Web Token / P. Mahindraka // International Journal of Recent Technology and Engineering (IJRTE). – 2020. – Vol. 8. – P. 1707–1710.

9. API Security: Protecting APIs With Keycloak / S. D. Danso, C. Yin // International Journal of Engineering Research & Technology (IJERT). – 2023. – Vol. 12, Issue 04. – P.561–564.

10. *Chatterjee A.* Applying Spring Security Framework with KeyCloak-based OAuth2 to Protect Microservice Architecture APIs: A Case Study / A. Chatterjee, A. Prinz // Sensors. – 2022. – Vol. 22, No. 5. – P. 1703.

11. *Al Maamari S. R. S.* A Comparative Analysis of NoSQL and SQL Databases: Performance, Consistency, and Suitability for Modern Applications with a Focus on IoT / S. R. S. Al Maamari, M. Nasar // East Journal of Computer Science. – 2025. – Vol. 1, No. 2. – P. 10-15.

12. *Gamma E.* Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. – Boston : Addison-Wesley Professional, 1995. – 416 p.

13. *Martin R. C.* Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. C. Martin. – Boston : Prentice Hall, 2017. – 432 p.

УДК 004.415.2

МАСШТАБИРУЕМАЯ АРХИТЕКТУРА ДЛЯ ОБЕСПЕЧЕНИЯ СТАБИЛЬНЫХ ПРОГНОЗОВ ПОКАЗАТЕЛЕЙ ФУНКЦИОНИРОВАНИЯ СТАНЦИЙ СОТОВОЙ СВЯЗИ

А. А. Герасимов

Санкт-Петербургский государственный технологический институт (технический университет)

Аннотация. Рассматривается подход к проектированию и реализации масштабируемой архитектуры на примере системы предназначенной для повышения эффективности мониторинга персоналом. Система позволяет строить прогнозы по выбранным показателям для прогнозирования временного ряда, сократить вероятность инцидентов. Анализ спрогнозированных данных позволит повысить общую стабильность системы, упростить работу операторов мониторинга.
Ключевые слова: автоматизированная система, микросервисы, масштабируемость, предиктивная аналитика, прогнозирование, временные ряды, мониторинг, отказоустойчивость, станции сотовой связи.

Введение

Развитие сотовой связи и увеличение объёмов передаваемых данных требуют внедрения современных инструментов мониторинга и прогнозирования функционирования сетевой инфраструктуры. Эффективное управление работой базовых станций [1] невозможно без автоматизированных систем, обеспечивающих сбор, хранение, анализ и предиктивную обработку большого количества показателей. Информационное обеспечение таких комплексных решений должно быть распределённым и высоко масштабируемым, соответствовать требованиям к безопасности, целостности и доступности данных так как данные характеристики напрямую влияют на сложность его поддержки, доработки и отвечают на вопрос о целесообразности использования.

1. Постановка требований к автоматизированной системе

Система должна предоставлять следующие возможности:

- аутентификация и авторизация;
- создание заявки на прогноз значения метрики;
- ввод параметров объекта мониторинга и метрики для расчёта;
- представление всех доступных метрик и объектов мониторинга программного продукта;
- представление результата расчёта в виде линейного графика;
- поддержка до пятидесяти активных прогнозов, по которым выполняются вычисления для каждого пользователя;
- автоматическое продление результатов с учетом новых значений.

2. Подходы к обеспечению масштабируемости системы

Существует широкий спектр подходов к масштабируемости [2], и выбор зависит от требований к нагрузке, скорости изменений и ограничений домена. Наиболее часто используемые группы решений:

- Вертикальное масштабирование: увеличение ресурсов отдельных узлов (CPU, RAM, диски более высокого класса). Упирается в физические и экономические пределы, повышает риск единичной точки отказа;

• Горизонтальное масштабирование: добавление однотипных узлов за балансировщиками. Требует продуманного распределения состояния (кэш, БД [3], очереди);

• Репликация и шардинг данных: масштабирование СУБД через реплики для чтения, шардинг для распределения записи. Повышает пропускную способность, но усложняет согласованность и маршрутизацию;

• Асинхронная обработка и очереди: вынос тяжёлых операций в фоновые воркеры (event-driven/streaming). Повышает отзывчивость фронтов, накладывает требования к идемпотентности и мониторингу;

• Контейнеризация и оркестрация [4]: Docker, Kubernetes/Nomad для автоскейлинга. Снижает операционные издержки при росте;

• Архитектурные стили: монолит, модульный монолит, микросервисы. Определяют границы развертывания и масштабирования.

Важно, что для автоматизированной системы прогнозирования временных рядов предварительная оценка профиля нагрузки не детерминирована, где намеренное завышение ресурсов системы может повлечь трату бюджетов на обеспечение, а занижение – невыполнение требований для обеспечения прогнозов.

Учитывая неравномерные профили нагрузки, а также потребность системы в независимом развитии отдельных доменных областей, высокую изменчивость требований целесообразно выбрать микросервисную архитектуру как базовый подход к масштабируемости.

Ключевые причины выбора:

• Независимое масштабирование: каждый сервис масштабируется по собственным метрикам (RPS, latency, queue depth), что экономит ресурсы по сравнению с масштабированием всего приложения целиком;

• Автоскейлинг [5]: возможность автоматического увеличения количества подов сервиса прогнозирования при преодолении порогового значения ресурсов и/или запросов;

• Изоляция отказов: сбои локализуются в границах сервиса; схемы circuit breaker и gracefull degradation предотвращают каскадные падения;

• Технологическая гибкость: подбор оптимальных стеков и хранилищ под конкретные задачи (в частности, заменяемая БД временных рядов).

3. Функциональная и сервисная архитектура системы

Система включает следующие подсистемы и модули:

• модуль аутентификации пользователей;

• модуль авторизации, логирования и аудита;

• модуль импорта актуальных временных рядов (периодическое получение актуальных значений метрик из внешних источников);

• модуль адаптер для получения или сохранения значений временных рядов в базу данных;

• модуль выполнения задач по расписанию для создания заявок на прогноз значения метрик и вызова процедуры прогноза;

• модуль прогнозирования значений для расчёта значений метрик по запросу и сохранения прогноза;

• БД источников данных для запроса актуальных значений;

• БД временных рядов для хранения действительных значений и прогнозов;

• БД расписания задач на прогнозирования.

Функциональная структура программного комплекса, в соответствии с которой должно осуществляться информационное взаимодействие компонентов программного комплекса, представлена на рис. 1.

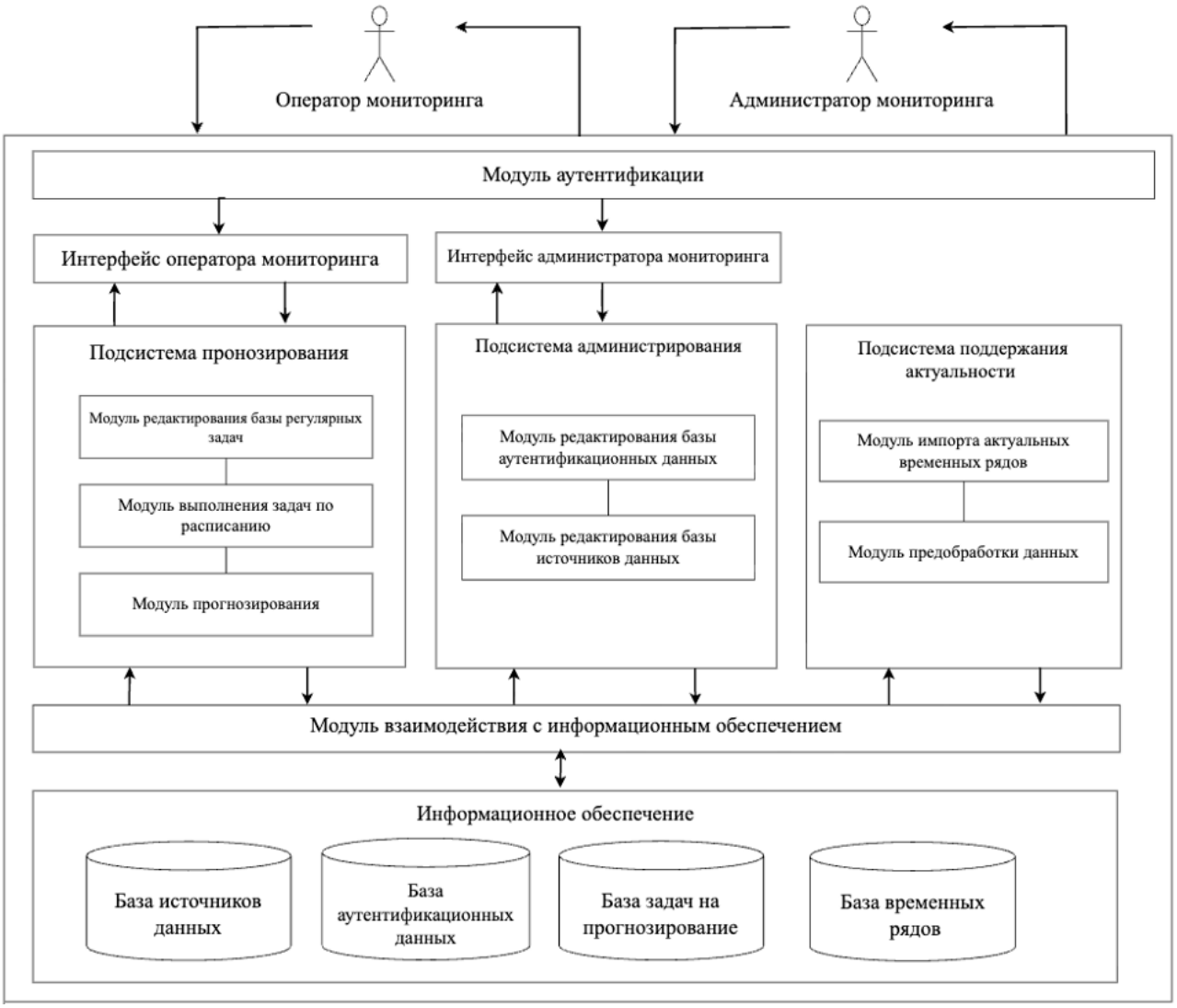

Рис. 1. Функциональная структура программного комплекса

В терминах составленного системного дизайна произведем отображение потоков данных для задачи получения и обработки первичных данных и прогнозирования значений функциональных показателей на рис. 2, 3.

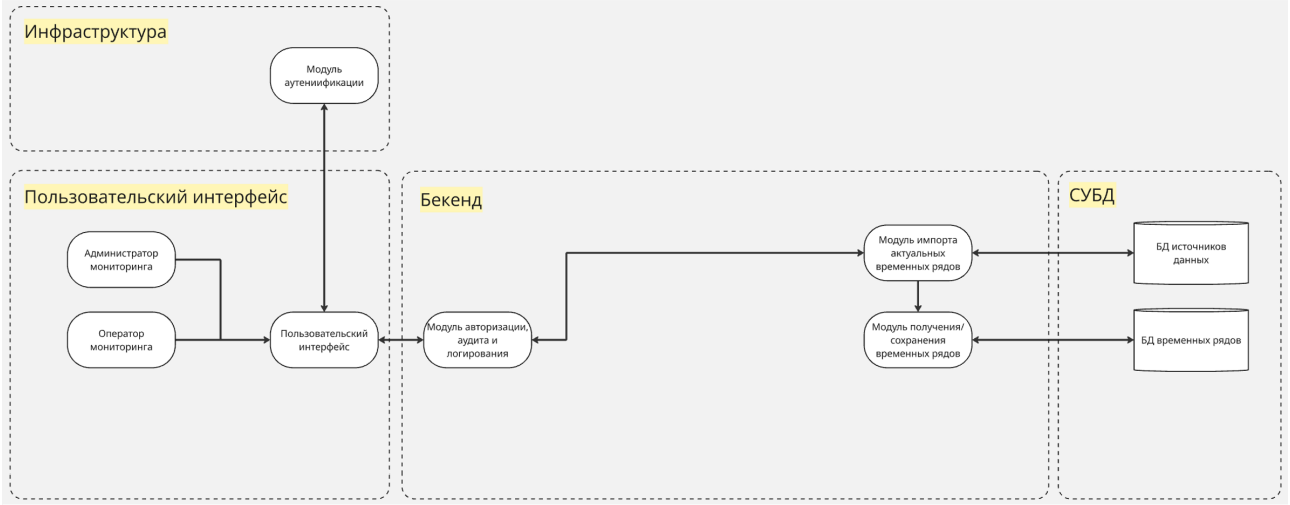

Рис. 2. Схема взаимодействия для задачи получения и обработки первичных данных

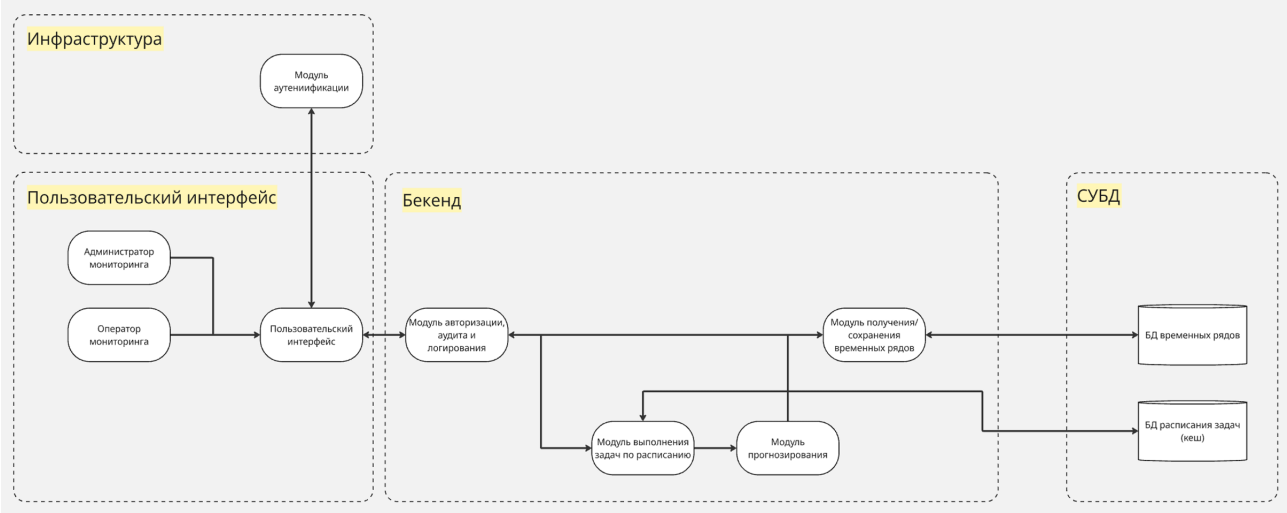

Рис. 3. Схема взаимодействия для задачи прогнозирования показателей

4. Масштабируемость архитектуры

Для обеспечения масштабируемости системы были приняты и имплементированы следующие подходы:

• Обеспечена полиглотная персистентность: выбор специализированных хранилищ под тип нагрузки (TSDB VictoriaMetrics для полученных метрик). Временные ряды в телекоммуникационных сетях характеризуются высокой частотой записей, сжатием по временной оси и запросами по окнам/агрегациям. TSDB оптимизированы под такие паттерны (эффективная компрессия, горизонтальный шардинг, быстрые range scan/aggregation), что снижает стоимость хранения и ускоряет аналитические запросы по сравнению с универсальными СУБД;

• Настроена реплика базы данных временных рядов. Реплики распределяют нагрузку на чтение (дашборды, алерты, обучение моделей), повышают отказоустойчивость (быстрое переключение при сбое), а также позволяют выполнять тяжёлые аналитические запросы на вторичных экземплярах без влияния на поток записи от коллектора метрик;

• Автоскейлинг сервиса расчёта прогнозов при достижении 100 rps от сервиса расписаний. Нагрузка на прогнозирование носит всплесковый характер (окна обучения/перепрогнозирования, пакетные задачи). Порог триггер на 100 rps позволяет эластично добавлять вычислительные ресурсы в момент пиков, сокращая латентность выдачи прогнозов и удерживая SLA, а в периоды спада — снижать расходы.

• Спроектирована микросервисаная архитектура, позволяющая в режиме реального времени изменять значения ресурсов и лимитов для каждого модуля автоматизированной системы, оптимизировать QoS под профиль каждого модуля, быстро выкатывать обновления и локализовать сбои, что критично для среды с высокими требованиями к доступности и времени реакции.

Заключение

В статье проанализированы подходы к масштабируемости, выбран курс на микросервисную архитектуру для решения задачи обеспечения надежности прогнозов показателей функционирования станций сотовой связи, поддержанной репликацией данных, асинхронным взаимодействием и автоскейлингом по метрикам потребления ресурсов. Принятые решения обеспечивают независимое масштабирование модулей, устойчивость к пиковым нагрузкам и локализацию отказов, а также ускоряют выпуск изменений за счет декомпозиции и автомати-

зации доставки. Благодаря полиглотной персистентности система достигает целевых показателей производительности без избыточных затрат в условиях недетерминированной нагрузки. В результате платформа готова к дальнейшему росту нагрузки и расширению функциональности при сохранении управляемости и предсказуемости эксплуатации.

Литература

1. 3GPP TS 28.550. Management and orchestration; Performance measurements collection. – 3GPP – URL: https://www.3gpp.org/ (дата обращения: 09.11.2025).

2. *Клеппман М.* Проектирование высоконагруженных приложений: надёжные распределённые системы. – М.: ДМК Пресс, 2022. – 608 с. (пер. англ. Kleppmann M. Designing Data-Intensive Applications).

3. *Новиков Б. А.* Основы технологий баз данных: учебное пособие. / Б. А. Новиков, Е. А. Горшкова, Н. Г. Графеева; под ред. Е. В. Рогова. – 2-е изд. – Москва : ДМК Пресс, 2020. – 582 с.

4. *Хайтауэр Б., Бёрнс Б., Беда Дж.* Kubernetes на практике, 3-е изд. – СПб. : Питер, 2024. – 384 с. (пер. англ. Hightower B., Burns B., Beda J. Kubernetes: Up & Running, 3rd ed.).

5. Kubernetes Documentation. Vertical/Cluster autoscaling – URL: https://kubernetes.io/docs/tasks/run-application/cluster-autoscale/ (дата обращения: 09.11.2025).

УДК 004.42

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ СОСТЫКОВКИ РАСПИСАНИЙ

А. А. Гонных, М. В. Матвеева

Воронежский государственный университет

Аннотация. В статье представлено приложение, позволяющее «состыковывать» расписания участников встреч. Проведен общий анализ задачи, рассмотрены представленные решения на рынке и алгоритм для нахождения свободных окон для проведения встреч, корректность вычислений которого была подтверждена тестированием. В заключение сформулированы выводы по результатам работы.
Ключевые слова: состыковка расписаний, веб приложение, алгоритм sweep-line.

Введение

В условиях интенсивного учебного и проектного процесса участникам приходится регулярно искать общее «окно» для встреч, консультаций и совместной работы.

Популярные сервисы упрощают задачу, однако:

– привязаны к экосистемам конкретных поставщиков;

– требуют коммерческих подписок.

Поэтому актуальна разработка независимого веб-приложения, которое:

– хранит расписания участников в едином формате;

– автоматически определяет пересечения свободных интервалов;

– предлагает удобный интерфейс взаимодействия.

Практическая значимость решения заключается в сокращении времени на организацию встреч.

1. Постановка задачи

Целью данной работы является создание веб-сервиса, обеспечивающего подбор свободных временных интервалов для встречи группы пользователей.

Программа должна выполнять следующие ключевые функции:

– регистрация и авторизация пользователей;

– ввод расписаний;

– хранение календарных данных в базе;

– алгоритмический поиск пересечений свободных временных интервалов с поддержкой маркеров приоритета и занятости участников;

– графическое отображение найденных «окон» и подтверждение выбранного слота в браузере пользователя.

Для этого необходимо решить следующие задачи:

1. Проектирование модели данных.

– определить требуемые сущности и связи между ними;

– выбрать СУБД и разработать физическую схему таблиц.

2. Разработка серверной части.

– реализовать REST-контроллеры для операций с пользователями и встречами;

– реализовать регистрацию, аутентификацию по JWT;

– обеспечить хранение расписаний и результатов подбора в базе данных;

– спроектировать и внедрить WebSocket-канал для мгновенной рассылки изменений клиентам;
– разработать и внедрить алгоритм для поиска пересечений свободных интервалов.
3. Создание клиентского интерфейса.
– разработать одностраничное веб-приложение на React и TypeScript;
– реализовать ввод индивидуальных расписаний и визуализацию найденных слотов на календаре;
– обеспечить подтверждение выбранного времени автором встречи.

2. Анализ задачи

2.1. Общий анализ задачи

Для решения поставленной задачи необходимо разработать серверную и клиентскую части приложения, которое позволит пользователям:
– создавать собственные расписания;
– в реальном времени получать список совместных «окон» для встречи;
– хранить календарные данные в базе данных;
– визуально оценивать доступность участников в браузере.
Чтобы обеспечить данную функциональность требуется создать несколько инструментов, выполняющих основные функции:
– ввод данных пользователем;
– алгоритмический поиск пересечений;
– сохранение и обработка данных;
– генерация ответа;
– графическое отображение.
Реализация перечисленного выше обеспечит эффективный инструмент для быстрого согласования встреч, пригодный как для учебных, так и для корпоративных проектов.

2.2. Анализ существующих решений

Doodle — онлайн-сервис групповых опросов для выбора времени [1].
Ключевые возможности:
– простой интерфейс;
– e-mail-уведомления;
– фильтрацию данных по меткам, исполнителям и временным интервалам.
Ограничения:
– нет алгоритмического поиска пересечений — только ручное голосование;
– данные хранятся на серверах Doodle, без on-prem-варианта;
– лимиты бесплатной версии.
When2Meet — бесплатный веб-опрос без регистрации [2].
Ключевые возможности:
– построение теплокарты занятости;
– минималистичный интерфейс;
– полностью бесплатен.
Ограничения:
– отсутствие полноценных учётных записей;
– нет алгоритмического поиска пересечений — только ручное голосование;
– хранения данных на серверах When2Meet.

Оба сервиса сводят подбор времени к ручному голосованию, не дают on-premise-развёртывания и расширенной авторизации. Предлагаемое приложение решает эти ограничения: хранит расписания локально, автоматически вычисляет общие «окна» алгоритмом и рассылает результаты через WebSocket [3], предоставляя защищённый REST-интерфейс для интеграций.

2.3. Алгоритм sweep-line

Для решения базовой задачи состыковки расписаний был выбран алгоритм sweep-line, который сводит задачу к сортировке двух «флажков» на каждое занятое окно и одному линейному проходу, что делает его оптимальным по асимптотике [4] и простым в реализации внутри сервиса.

Алгоритм:

1. Для каждого участника μ и каждого его свободного интервала $[s, e)$ создаём две метки $(s, +1), (e, -1)$:

– $+1$ — с этого мгновения μ становится свободен;

– -1 — становится занят.

2. Упорядочиваем получившийся массив по времени t и поддерживаем переменную $A(t) =$ кол-во свободных участников в момент t.

3. Каждый отрезок $[t_i, t_{i+1})$ сравниваем с текущим максимумом и фиксируем A_max.

3. Программный комплекс

3.1. Средства реализации

В качестве языка программирования выбран язык Java 21, фреймворк Spring Boot 3 [5]. Возможности данного языка и фреймворка предоставляют все необходимые инструменты для реализации поставленных в данной работе целей и задач. При реализации проекта использовалась среда разработки Intellij Idea 2024.1.4.

В качестве системы управления базами данных был выбран PostgreSQL версии 16.0. Она позволяет хранить и эффективно обрабатывать расписания участников, гарантируя целостность данных через поддерживаемые ограничения и транзакции [6]. Для работы с БД используется Spring Data JPA и JDBC-драйвер.

В качестве технологий для клиентской части были выбраны React 19.1.0 в связке с TypeScript [7]. Эти инструменты предоставляют все необходимые механизмы для реализации поставленных в данной работе целей и задач: TypeScript обеспечивает статическую типизацию и безопасность кода на этапе компиляции, а React отвечает за декларативное построение динамического и реактивного интерфейса.

3.2. Диаграмма классов

Сервис представляет собой монолитное приложение. Диаграмма классов сервера представлена на рис. 1.

3.3. Реализация серверной части

Методы класса EventController:

– `public ResponseEntity<?> getEventAvailability( @RequestHeader String authHeader, @PathVariable Long meetingId);`

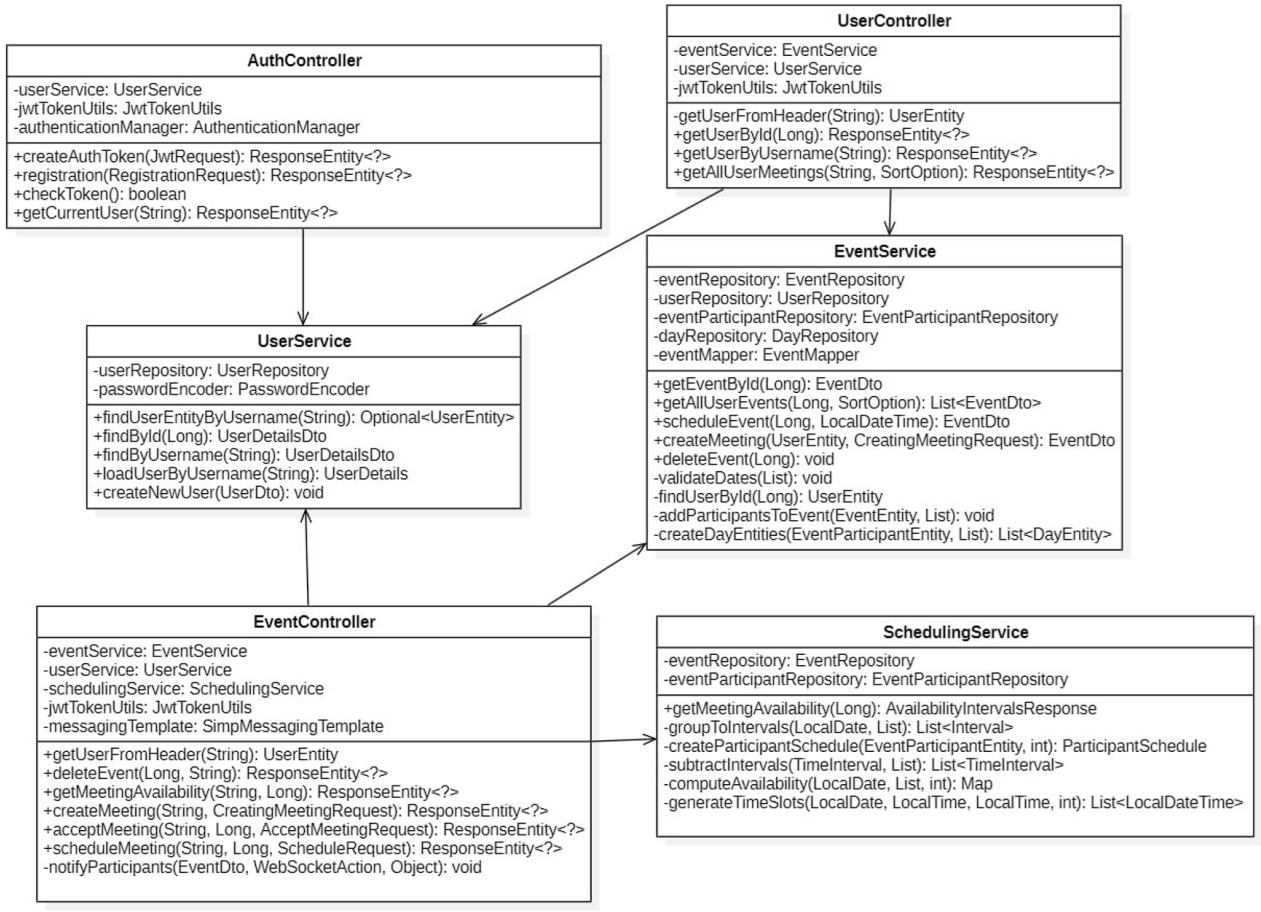

Рис. 1. Диаграмма классов модуля MeetingService

Метод вычисления свободных слотов расписания.

– `public ResponseEntity<?> acceptEvent(@RequestHeader String authHeader, @PathVariable Long meetingId, @RequestBody AcceptMeetingRequest request);`

Метод подтверждения участия во встрече.

– `public ResponseEntity<?> scheduleEvent(@RequestHeader String authHeader, @PathVariable Long meetingId, @RequestBody ScheduleRequest request).`

Метод закрепления встречи.

Методы класса EventService:

– `public List<EventDto> getAllUserEvents(Long id, SortOption sortOption);`

Метод получения списка событий заданного пользователя, с возможностью сортировки по дате, названию или статусу.

– `public EventDto scheduleEvent(Long eventId, LocalDateTime startTime);`

Метод назначения точного времени начала встречи: проверяет доступность автора и участников, обновляет статусы и сохраняет результат.

– `public EventDto createEvent(UserEntity author, CreatingMeetingRequest meeting);`

Метод создания новой встречи: сохраняет сущность Event, добавляет автора и участников, устанавливает возможные даты.

– `public void deleteEvent(Long eventId);`

Метод удаления встречи по её идентификатору, с каскадным удалением всех зависимостей.

– `private void addParticipantsToEvent(EventEntity eventEntity, List<Long> participantIds).`

Метод добавления участника во встречу.

Методы класса ShedulingService:

– `public AvailabilityIntervalsResponse getMeetingAvailability(Long meetingId);`

Метод получения списка доступных пользователю интервалов для указанной встречи с учётом занятости участников.

– `private List<Interval> groupToIntervals(LocalDate date, List<LocalDateTime> slots);`

Метод объединения последовательных точечных слотов в непрерывные временные интервалы одного дня.

– `private Map<LocalDateTime, Integer> computeAvailability(LocalDate day, List<ParticipantSchedule> schedules, int duration);`

Метод расчёта «теплокарты» доступности: превращает свободные интервалы участников в события, сортирует их и одним проходом подсчитывает число свободных в каждый момент.

3.4. Реализация клиентской части

Код клиентской части разбит на самостоятельные React-компоненты, каждый из которых располагается в отдельном .tsx-файле:

– AuthPage.tsx;

Компонент авторизации/регистрации пользователя, хранение JWT-токена в localStorage.

– Header.tsx;

Компонент для отображения шапки SPA: навигационные ссылки и кнопка выхода.

– MeetingForm.tsx;

Компонент для создания новой встречи в модальном окне с валидацией через React-Hook-Form.

– MeetingList.tsx;

Компонент для вывода таблицы встреч с возможностью фильтрации и сортировки.

– MeetingCard.tsx;

Компонент для визуального представления одной встречи с кнопками «Принять / Отклонить».

– Calendar.tsx;

Компонент для наглядного отображения событий в календаре на базе FullCalendar.

– MeetingConfirmation.tsx;

Компонент для выбора автором финального слота времени.

– ProtectedRoute.tsx;

Компонент-обёртка для ограничения доступа к маршрутам неаутентифицированных пользователей.

3.5. Интерфейс пользователя

Реализован интерфейс, представленный на рис. 2.

3.6. Тестирование

В рамках вычислительного эксперимента была проведена проверка корректности состыковки расписания на примере тестовых данных: приглашённый пользователь имеет занятость с 10:00 до 13:00 часов, а создатель встречи хочет провести собрание в промежутке с 9:00 до 18:00. Результат представлен на рис. 3.

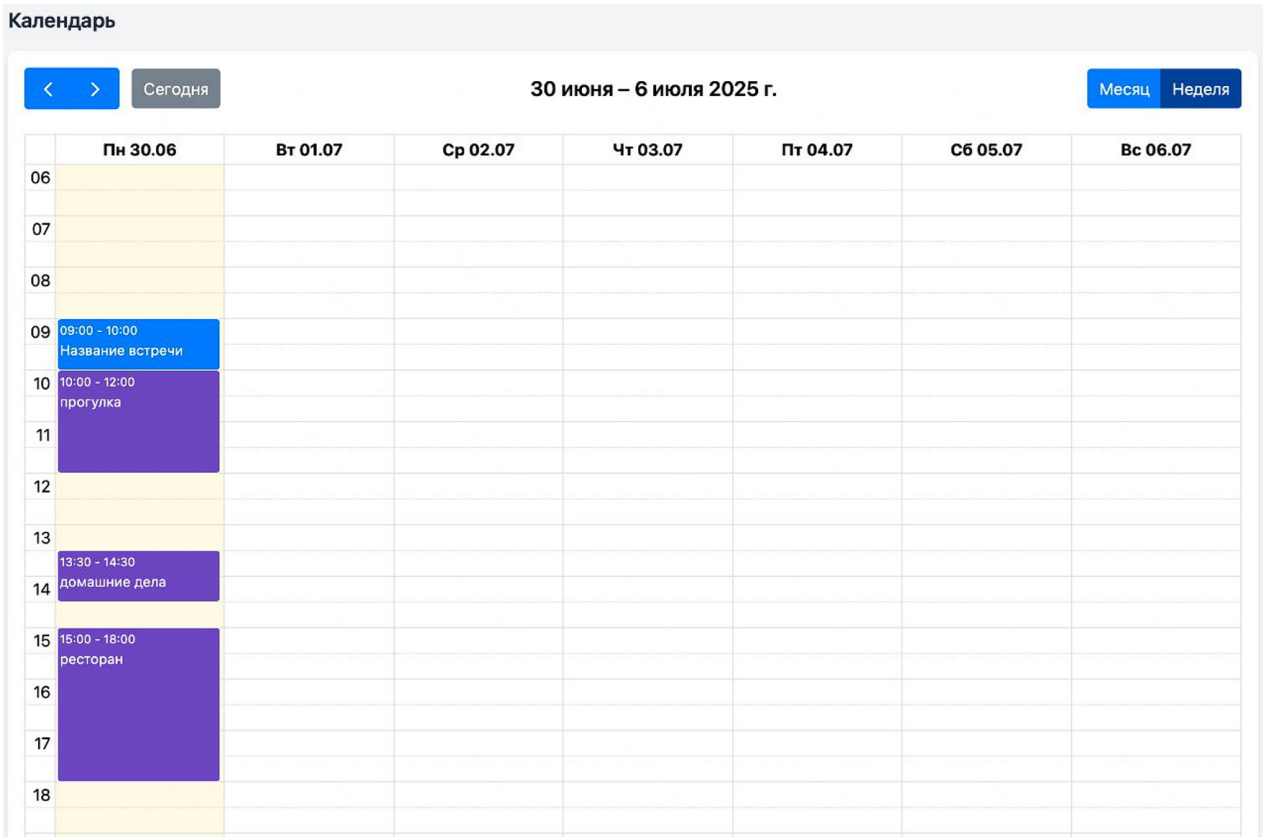

Рис. 2. Расписание пользователя

Рис. 3. Успешное получение свободных дат

Заключение

В ходе выполнения работы был спроектирован и реализован веб-сервис, позволяющий автоматически подбирать свободные временные интервалы для встречи группы пользователей. Разработанное приложение объединяет сервер на Java 21 и Spring Boot 3 с одностраничным клиентом на React 19 и TypeScript, что обеспечило высокую скорость отклика и удобство дальнейшего сопровождения.

Алгоритм сканирующей линии, применённый на сервере, успешно вычисляет пересечения свободных интервалов, тем самым существенно сокращая время согласования встреч.

Литература

1. Doodle AG. Doodle – Schedule Smarter Together. – Цюрих: Doodle AG, 2024. – URL: https://doodle.com (дата обращения: 25.03.2025).

2. When2Meet. Online Scheduling Tool. – [Б. м.], 2024. – URL: https://www.when2meet.com (дата обращения: 25.03.2025).

3. *Fette I., Melnikov A.* The WebSocket Protocol / I. Fette, A. Melnikov // IETF RFC 6455. – 2011. – 79 с. – URL: https://datatracker.ietf.org/doc/rfc6455 (дата обращения: 27.03.2025).

4. *Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.* Introduction to Algorithms. – 4th ed. – Cambridge: MIT Press, 2022. – 1 292 с.

5. Spring Boot 3.2 Reference Guide / Pivotal Software. – Palo Alto: Pivotal, 2023. – URL: https://docs.spring.io/spring-boot/docs/3.2.x/reference/ (дата обращения: 27.03.2025).

6. PostgreSQL Global Dev. Group, 2023. – URL: https://www.postgresql.org/docs/16/index.html (дата обращения: 10.04.2025).

7. React 19 Documentation / Meta Platforms, Inc. – Menlo Park, 2024. – URL: https://react.dev (дата обращения: 14.04.2025).

УДК 004.4

ПРОБЛЕМА СОЗДАНИЯ ЛЕГКОВЕСНЫХ АСИНХРОННЫХ ВЕБ-СЕРВИСОВ И ПОДХОДЫ К ИХ РЕШЕНИЮ С ИСПОЛЬЗОВАНИЕМ ФРЕЙМВОРКА KTOR

Н. В. Денисов, Н. А. Каплиева

Воронежский государственный университет

Аннотация. В статье рассматривается проблема создания легковесных и высокопроизводительных веб-сервисов в условиях ограниченных ресурсов и возрастающих требований к масштабируемости и асинхронности. Анализируются архитектурные особенности фреймворка Ktor — современного инструмента на языке Kotlin, построенного на корутинах и модульной системе плагинов. Описываются ключевые преимущества Ktor: поддержка неблокирующего ввода-вывода, гибкость конфигурации, кроссплатформенность и простота интеграции. Приводятся примеры практического применения фреймворка для разработки RESTful API, обработки событий в реальном времени и организации надежного взаимодействия между сервисами. Статья предназначена для разработчиков, стремящихся создавать эффективные, легко поддерживаемые и экономически целесообразные серверные решения.
Ключевые слова: Ktor, Kotlin, асинхронная разработка, корутины, веб-сервисы, микросервисы, REST API, неблокирующая архитектура, серверная разработка, кроссплатформенность, модульность, производительность, легковесные приложения, цифровая трансформация.

Введение

Современная серверная разработка все чаще сталкивается с необходимостью создания высоконагруженных, масштабируемых и ресурсно-эффективных веб-сервисов, способных обрабатывать тысячи одновременных запросов при минимальных затратах памяти и вычислительных ресурсов. Традиционные блокирующие фреймворки, построенные на поточной модели обработки запросов, зачастую оказываются избыточными, сложными в настройке и неэффективными в условиях ограниченной инфраструктуры — особенно для стартапов, микросервисов и нишевых отраслевых решений.

Фреймворк Ktor, разработанный компанией JetBrains на языке Kotlin, предлагает альтернативный подход: легкую, модульную и полностью асинхронную архитектуру, основанную на корутинах. Такая модель позволяет обрабатывать запросы неблокирующим образом, обеспечивая высокую производительность даже на скромном серверном оборудовании. Благодаря выразительному DSL, встроенной поддержке кроссплатформенности и гибкой системе расширений (плагинов), Ktor идеально подходит для быстрой разработки специализированных API, микросервисов и серверных компонентов современных распределенных систем.

Целью данной статьи является анализ проблемы создания легковесных асинхронных веб-сервисов в условиях растущих требований к производительности и экономичности, а также рассмотрение способов ее решения с использованием фреймворка Ktor. Мы охарактеризуем ключевые особенности архитектуры Ktor, продемонстрируем его преимущества перед традиционными подходами и проиллюстрируем практическое применение на примерах типовых сценариев серверной разработки.

1. Проблема создания легковесных асинхронных веб-сервисов в современной IT-практике

Современные требования к веб-сервисам предполагают не только корректную реализацию бизнес-логики, но и высокую производительность, масштабируемость, экономичность в экс-

плуатации и гибкость в адаптации под меняющиеся условия. Однако традиционные подходы к серверной разработке, основанные на блокирующих вызовах и потоковых моделях обработки запросов (*thread-per-request*), все чаще демонстрируют свою неэффективность. Каждый входящий запрос в такой архитектуре резервирует отдельный поток операционной системы, что при высокой нагрузке приводит к резкому росту потребления памяти, частым переключениям контекста и, как следствие, деградации производительности.

Эта проблема особенно остро стоит при разработке микросервисов, API для мобильных приложений, серверных компонентов для стартапов и нишевых отраслевых решений, где ресурсы ограничены, а требования к отзывчивости и стабильности высоки. В таких сценариях избыточность монолитных enterprise-фреймворков (например, *Spring Boot* с его зависимостями и стартовым временем) становится обременительной, а сложность настройки и сопровождения — непропорциональной масштабу задачи.

Таким образом, формируется ключевая проблема: как создать веб-сервис, который был бы одновременно:

1) *легковесным* — с минимальным объемом зависимостей и низким потреблением ресурсов;

2) *асинхронным* — способным эффективно обрабатывать тысячи одновременных подключений без блокировки потоков;

3) *гибким* — легко расширяемым и адаптируемым под специфику конкретного проекта.

Современные подходы к решению этой проблемы делают ставку на событийно-ориентированную архитектуру (*event-driven architecture*) и корутин-ориентированное программирование, которые позволяют писать асинхронный код в императивном стиле, избегая «ада колбэков» и сложной цепочки Promise/Future. Именно в этом контексте фреймворк Ktor становится релевантным инструментом: он разработан с нуля как асинхронный, модульный и кроссплатформенный фреймворк на языке Kotlin, в полной мере использующий возможности корутин и неблокирующего ввода-вывода.

Ktor не навязывает строгой структуры проекта, не требует тяжеловесного контейнера и позволяет запускать приложение как встроенный сервер (на *Netty*, *CIO*, *Jetty* и др.), так и в виде *serverless*-функции. Благодаря этому он идеально подходит для создания тех легковесных асинхронных веб-сервисов, чья разработка и эксплуатация остаются экономически и технически целесообразными даже для малых команд и ограниченной инфраструктуры.

2. Архитектурные особенности Ktor как ответ на современные вызовы серверной разработки

Фреймворк Ktor, разработанный компанией *JetBrains*, был создан в ответ на растущий спрос на инструменты, способные эффективно решать задачи высоконагруженной и ресурсоограниченной серверной разработки. Его архитектура строится на трех ключевых принципах: асинхронность на основе корутин, модульность через систему плагинов и поддержка кроссплатформенной разработки. Эти особенности позволяют Ktor напрямую устранять недостатки традиционных серверных решений и обеспечивать гибкость, необходимую для создания современных веб-сервисов.

2.1. Асинхронная модель на основе корутин Kotlin

В основе Ktor лежит неблокирующая архитектура, реализованная с использованием корутин Kotlin — легких, управляемых пользователем «нитей выполнения», которые не привязаны к потокам операционной системы. Это позволяет обрабатывать десятки тысяч одновремен-

ных подключений в рамках одного или нескольких потоков, что радикально снижает потребление памяти и исключает накладные расходы на переключение контекста.

В отличие от *callback*-подхода (например, в *Node.js*) или *Future*/*Promise* (в *Java*), корутины обеспечивают прямолинейную императивную структуру кода, сохраняя при этом асинхронную природу выполнения:

```
get("/users/{id}") {
    val userId = call.parameters["id"]!!
    val user = withContext(Dispatchers.IO) {
        userRepository.findById(userId)
    }
    call.respond(user)
}
```

Такой код легко читается, тестируется и поддерживается, что особенно важно в условиях быстрой итеративной разработки.

2.2. Модульность и расширяемость через систему плагинов

Ktor не навязывает монолитную структуру. Вместо этого он использует концепцию плагинов — независимых компонентов, которые подключаются по мере необходимости. Это позволяет разработчику собрать ровно ту функциональность, которая требуется в конкретном проекте:

1) *Routing* — маршрутизация HTTP-запросов;
2) *ContentNegotiation* — сериализация/десериализация JSON, XML и др.;
3) *Authentication* и *Authorization* — управление доступом;
4) *WebSockets* — поддержка двунаправленной связи в реальном времени;
5) *CORS*, *Compression*, *RateLimit* — настройка безопасности и производительности.

Такой подход минимизирует размер итогового приложения, ускоряет запуск и снижает поверхность атаки, что особенно ценно для микросервисов и *serverless*-архитектур.

2.3. Кроссплатформенность и единый стек разработки

Благодаря интеграции с *Kotlin Multiplatform*, *Ktor* позволяет использовать единый язык и логику как на сервере (*JVM*, *Native*), так и на клиенте (*Android*, *iOS*, *JavaScript*). Это открывает возможность для:

1) совместного использования моделей данных и валидационной логики;
2) написания интеграционных и end-to-end тестов на одном языке;
3) ускорения разработки в командах с ограниченными ресурсами.

Клиент Ktor, в свою очередь, предоставляет мощный, асинхронный *HTTP*-клиент с поддержкой тех же плагинов (например, сериализации, логирования, таймаутов), что делает взаимодействие между сервисами единообразным и надежным.

Таким образом, архитектурные особенности Ktor напрямую отвечают на ключевые вызовы современной веб-разработки: он обеспечивает высокую производительность при низком потреблении ресурсов, максимальную гибкость конфигурации и единый технологический стек — все это в совокупности делает его одним из наиболее перспективных инструментов для создания легковесных, асинхронных и масштабируемых веб-сервисов.

3. Практические сценарии применения Ktor в разработке современных веб-сервисов

Эффективность фреймворка Ktor проявляется не только в теоретических преимуществах его архитектуры, но и в реальных сценариях разработки. Его гибкость, легкость и поддерж-

ка асинхронного программирования делают его особенно подходящим для широкого спектра задач — от создания микросервисов и *RESTful API* до построения серверных компонентов для специализированных отраслевых приложений.

3.1. Разработка RESTful API для специализированных систем

Одной из наиболее частых задач при создании современных информационных систем является разработка надежного и производительного *API*. Ktor предоставляет выразительный *DSL*-синтаксис для маршрутизации, что позволяет описывать *эндпоинты* кратко и читаемо:

```
routing {
    route("/api/animals") {
        get {
            call.respond(animalService.getAll())
        }
        get("/{id}") {
            val id = call.parameters["id"] ?: throw
BadRequestException()
            call.respond(animalService.findById(id))
        }
        post {
            val animal = call.receive<AnimalCreateDto>()
            val saved = animalService.create(animal)
            call.respond(HttpStatusCode.Created, saved)
        }
    }
}
```

Такой подход особенно ценен при разработке отраслевых решений, где бизнес-логика требует кастомных моделей данных (например, *Animal*, *MedicalRecord*, *Volunteer*). Ktor легко интегрируется с библиотеками сериализации (*kotlinx.serialization*, *Jackson*), что упрощает обмен данными в формате *JSON* без избыточной конфигурации.

3.2. Поддержка веб-сокетов и событий в реальном времени

Для систем, требующих двунаправленной связи — например, оповещения о срочных ветеринарных случаях, мониторинга состояния животных в приюте или координации волонтерской деятельности — Ktor предоставляет встроенную поддержку *WebSockets*:

```
install(WebSockets)

webSocket("/notifications") {
    while (true) {
        val message = notificationService.awaitNext()
        outgoing.send(Frame.Text(message.toJson()))}}
```

Это позволяет реализовать push-уведомления без необходимости подключать сторонние брокеры сообщений или использовать поллинг, что снижает задержки и нагрузку на клиентские устройства.

3.3. Тестирование и сопровождение

Ktor предоставляет встроенные инструменты для интеграционного тестирования:

```
@Test
fun 'should create new animal'() = testApplication {
    val response = client.post("/api/animals") {
        contentType(ContentType.Application.Json)
        setBody("""{"name": "Барсик", "species": "Cat"}""")
    }
    assertEquals(HttpStatusCode.Created, response.status)
    assertNotNull(response.body<Animal>().id)
}
```

Такой подход позволяет быстро и надежно проверять корректность работы API, что особенно важно при итеративной разработке и частых изменениях требований.

Заключение

В данной работе была проанализирована проблема создания легковесных асинхронных веб-сервисов в условиях растущих требований к производительности, масштабируемости и экономичности серверных решений. Рассмотрены ограничения традиционных серверных фреймворков и предложен фреймворк Ktor как современный подход к их преодолению.

Ktor представлен как легковесное, модульное и полностью асинхронное решение, построенное на корутинах Kotlin. Его архитектура обеспечивает высокую производительность при низком потреблении ресурсов, а гибкая система плагинов позволяет подключать только необходимую функциональность, избегая избыточности. Ktor особенно эффективен для разработки микросервисов, специализированных API и серверных компонентов, где критичны простота развертывания, скорость и адаптивность под специфику

предметной области.

В дальнейшем результаты данного анализа будут использованы для написания магистерской диссертации, где Ktor будет применен в качестве основы серверной части. Это позволит обеспечить надежную, производительную и экономически целесообразную архитектуру, способную эффективно поддерживать отраслевые бизнес-процессы.

Литература

1. Ktor: Official Documentation // JetBrains : [сайт]. – 2025. – URL: https://ktor.io (дата обращения: 15.10.2025).

2. Kotlin Coroutines Guide // Kotlin : [сайт]. – 2025. – URL: https://kotlinlang.org/docs/coroutines-overview.html (дата обращения: 25.10.2025).

3. Kotlin Multiplatform // JetBrains : [сайт]. – 2025. – URL: https://kotlinlang.org/docs/multiplatform.html (дата обращения: 10.11.2025).

УДК 004

МОДЕЛИ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ПРОГРАММНОГО ОБОРУДОВАНИЯ

И. А. Добрыдень

Нижегородский государственный педагогический университет им. Козьмы Минина

Аннотация. В статье исследуются современные модели управления жизненным циклом программного оборудования, которые используются при создании и реализации информационных систем различных сфер. Рассматриваются этапы жизненного цикла программы, начиная от создания требований и заканчивая его сопровождением. Особое внимание уделено таким принципам как каскадная и спиральная модель, а также методики гибкого формата (Agile). Задокументированы факторы, которые влияют на эффективность внедрения каждой модели, включая их особенности среды разработки, уровень рисков и требования заказчика.
Ключевые слова: управление жизненным циклом, программное оборудование, каскадная модель, спиральная модель, Agile, разработка, проектирование, качество.

Введение

Управление жизненным циклом ПО представляет собой важным элементом при успешной разработки и использовании информационных систем. Подробно организованный процесс управления проектом помогает своевременно обнаружить и устранить риски, уменьшить сроки реализации, минимизировать затраты и создать высокое качество продукта. Несмотря на большое количество существующих моделей и подходов, важной задачей на данный момент становится выбор конкретной из методик для проектирования своего проекта разработчикам и заказчикам.

Актуальность данного исследования предопределена необходимостью постоянного совершенствования разработки ПО в условиях изменяющегося рынка и стремительно меняющихся технологий. Эффективность использованных моделей имеет огромное влияние на конкурентоспособность разработчика, удовлетворённость заказчика и всю экономическую выгоду для предприятия. Данная статья посвящена изучению наиболее распространенных моделей управления жизненным циклом ПО, поиску факторов, которые определяют успех или неудачу проектированного проекта, а также формированию рекомендаций для дальнейших поправок и создании наилучшей стратегии разработки.

1. История возникновения и эволюция моделей управления жизненным циклом

История моделей управления жизненным циклом ПО начинается ещё в начале XX века, ведь тогда изначально были использованы различные программы, которые стали создаваться уже в промышленных масштабах. Первоначально большинство проектов создавалось при помощи проб и ошибок, без какого-либо плана и разделения на этапы. При увеличении сложности проектов необходимо было использовать различные подходы к управлению жизненным циклом.

Каскадная модель — стала первой классическим методом проектирования проектов, предложенная в конце 1960-х годов Винером Роем. [1] На то время она предполагал строгую последовательность фаз проекта, где переход между этапами мог существовать только лишь после завершения предыдущего. Данная модель помогала лучше контролировать процесс разработки, люди могли четко отслеживать выполнения каждого этапа, допуская при этом минимальное количество ошибок. Этот метод имел также и недостатки, они были вполне очевидны —

невозможность вернуться на прошлый этап. Часто обнаруживались ошибки при проверки данных, но программа продолжала работать, из-за отсутствия возможности внесения изменений и невозможностью вернуться на прошлый этап.

В 1980-е годы начали появляться способы улучшения каскадной модели. V-модель стала одной из первых альтернатив, она была предложена немецкими специалистами. Особенность данной модели является внедрение тестов между этапами разработки, это повышло вероятность того, что ошибки будут найден раньше, чем разработчики перейдут на следующий этап. Однако данная модель всё равно имела недостатки линейности и не могла допускать существенных изменений требования при разработке проекта.

В то же время параллельно начали развиваться идеи итерационного подхода к разработке: появлялись концепции, которые позволяют возвращаться к предыдущим этапам проекта в случае доработок. Тогда же начала выявляться спиральная модель, впервые представленная в 1986 году Барри Боэмом. Она включала многократные попытки прохождения полного цикла разработки, каждый из этих циклов состоял из следующих шагов: определение целей, анализ рисков, разработка прототипа приложения, а также проверка работоспособности. Данная модель была по-своему революционной, ведь благодаря своей способности снижать риски и обеспечивать гибкость данных, но она требовала большое количество времени и ресурсов.

В конце XX века из-за развития компьютерных технологий и изменение бизнес-моделей требовались новые подходы, которые были бы ориентированы на быструю реакцию потребности клиента и уменьшению сроков подачи продукта на рынок. В то время возникали гибкие методы разработки (Agile). [4] Первая версия данного способа разработки появилась уже в 2001 году, и она объединила целый ряд принципов, основные из которых были: взаимодействие с заказчиком, быстрое реагирование на изменение, использование небольших команд над разработки разных частей проекта, а также регулярная поставка нового функционала. Данный подход стал прорывом в сфере разработки, особенно в крупные проекты с высоким уровнем гибкости и изменений требований.

2. Характеристика современных моделей управления жизненным циклом

Современные модели управления жизненным циклом программного оборудования включают разные подходы, каждый из которых предназначен для удовлетворения различных нужд и ситуаций при проектировании системы.

2.1. Каскадная модель (Waterfall)

Каскадная модель — одна из старейших способ управления жизненным циклом. В данном случае проект делится на некоторые фазы, которые последовательно переходят друг за другом. Пример данных этапов следующий: сбор требований, проектирование, реализация, тестирование, ввод в эксплуатацию. [5] Преимуществ данного подхода становится простота и предсказуемость, а также отсутствие возвратов к ранее пройденным этапам. Недостаток данной системы является невозможность быстро реагировать на изменения каких-либо требований, а также высокая стоимость исправления ошибок на последних этапах и низкая гибкость при изменении внешнего окружения (обновлений).

2.2. V-модель

Данная представляет собой развитие каскадной модели, но она отличается наличием обратной связи, а также здесь существуют параллельные процессы тестирования и разработки. Название V-модель определено её графическим изображение в виде буквы «V», левая сторо-

на — этапы анализа требований и проектирования, правая — этапы интеграции и тестирования. Следует выделить следующий характеристики данной методологии:

1) Последовательность этапов: каждый этап будет начинаться только после того, как будет завершен предыдущий

2) Тестирование параллельно разработке: каждый этап оснащен разработки в конце его проведения, после этого проводится перевод на соответствующий уровень (модульное тестирование и т. п.).

3) Четкое разделение фаз: Определенная структура проекта будет облегать управление рисками и координацию работ

2.3. Гибридная модель

Данный метод содержит в себе слияние нескольких моделей управления жизненным циклом. Примером может служить интеграция элементов итерационной и каскадной моделей, ведь это может повысить адаптивность и снизить риски при дальнейшей разработки программы. Одним из наиболее популярных способов слияния: Agile-методологии и Waterfall. Данная комбинация помогает объединить наилучшие качества каждого из подходов, при этом компенсируя ограничения друг друга.

3. Проблемы и перспективы развития моделей управления жизненным циклом

Современные проекты часто сталкиваются с огромным количеством серьезных вопросов, здесь можно выделить сложность бизнеса, растущие технологические потребности у заказчиков и учет глобализации рынка. К сожалению, традиционные методы управления жизненным циклом, такие как каскадная модель, оказываются все менее и менее эффективными перед быстрыми изменениями мира и высоких ожиданий заказчиков. Современные компании вынуждены искать способы организации собственной деятельности, которые будут способны обеспечить своевременное выполнение каких-либо задачи и высокое качество результата.

Выделяется ключевая проблема — несоответствие темпов появления новых технологий и возможности самих моделей перестраиваться под текущую текучку жизни. Быстро развивающиеся отрасли IT или же биотехнологии всегда требуют быстрой доставки каких-либо обновлений и улучшения функционала для клиентов. [2] Однако традиционные методы часто ухудшают и затормаживают этот процесс, а также заставляют компанию терять конкурентные преимущества.

Еще одной серьезной проблемой становится высокий рост угроз информационной безопасности. Часто большинство классических моделей управления жизненным циклом не в полной мере учитывают необходимость постоянной защиты данных и проверки на соответствие требованиям регуляторов. Компании приходится усиливать механизмы безопасности на каждом этапе проектирования жизненного цикла продукта.

Необходимо переходить к принципам непрерывной интеграции и доставки для преодоления нынешних трудностей. Данные подход гарантируют постоянную доставку небольших обновлений пользователям, при этом происходит снижение рисков сбоев системы и увеличивается общее число удовлетворенных клиентов. Компания Google и Amazon становится ярким примером успешного примером использования CI/CD принципов.

Будущее дальнейших разработок будут связаны с использованием интеллектуальных систем и машинного обучения, чтобы автоматизировать выбор оптимальных стратегий управления жизненным циклом. Данные случаи будут анализировать большие объёмы данных и автоматически заниматься выбором подходящих процессов и инструментов для конкретных

случаев, при этом будут повышать производительность системы и уменьшать вероятности совершения ошибок при проектировании проекта.

Наконец, необходимо упомянуть такое направление, как индивидуальная настройка управления жизненным циклами, анализируя специфику выбранного предприятия или же проекта. Данное нововведение позволит компания эффективно управлять ресурсами и достигать своих целей, несмотря на особенности своего бизнеса и уникальных условий.

Заключение

Исследование показало, что эффективное управление жизненным циклом играет важную роль при разработке и эксплуатации информационных систем. Используя информацию об истории эволюции моделей управления жизненным циклом, можно увидеть постепенный переход от простых подходов к уже более гибким и комплексным системам. Каждая современная модель предлагает улучшение отдельных аспектов управления проектов, это может быть контроль рисков, повышение производительности или же ускоренное предоставление готового продукта заказчику.

Наконец, современные условия предъявляют дополнительные требования к процессу управления жизненным циклом. [3] Меняющиеся рынки, высокие темпы технологических изменений, а также потребность в информационной безопасности необходимо делать постоянное совершенствование используемых подходов. Необходимо изучать основные характеристики современных моделей управления жизненным циклом для того чтобы выявить ключевые факторы успеха или же неудачи проектов, а также создать необходимые рекомендации для определения оптимальной методики для конкретного случая.

Литература

1. *Бургонов О. В.* Менеджмент жизненного цикла инновационных процессов в условиях цифровизации экономики / О. В. Бургонов, Н. П. Голубецкая // Цифровая экономика и финансы : Материалы Международной научно-практической конференции, Санкт-Петербург, 17–18 марта 2022 года. – Санкт-Петербург : Центр научно-производственных технологий «Астерион», 2022. – С. 205–209. – EDN EGSTSP.

2. *Барулина В. В.* Сравнительный анализ гибкой и каскадной методологии разработки программного обеспечения / В. В. Барулина // Вестник магистратуры. – 2020. – № 4-2(103). – С. 21–22. – EDN BEREEK.

3. *Дмитриев А. Г.* Исследование жизненного цикла проекта / А. Г. Дмитриев // Современные технологии в мировом научном пространстве: методы, модели, прогнозы. – Петрозаводск : Международный центр научного партнерства «Новая Наука», 2021. – С. 127–138. – EDN DVDHZO.

4. *Умеренков Д. И.* Метрики эффективности в Agile-проектах / Д. И. Умеренков, А. Г. Дмитриев // Прогрессивная экономика. – 2025. – № 2. – С. 45–56. – DOI 10.54861/27131211_2025_2_45. – EDN WGOOVN.

5. *Шинкарук А. С.* Киберфизическая модель управления жизненным циклом пассажирского вагона / А. С. Шинкарук // Международный научно-исследовательский журнал. – 2024. – № 12(150). – DOI 10.60797/IRJ.2024.150.105. – EDN MPVELA.

УДК 004.42

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ХРАНЕНИЯ И ПОДБОРА РЕЦЕПТОВ

Д. А. Долженков, М. В. Матвеева

Воронежский государственный университет

Аннотация. В статье представлено веб-приложение, предназначенное для хранения, поиска и подбора кулинарных рецептов. Рассмотрены архитектура сервиса, реализация серверной и клиентской частей, а также применение поискового движка Elasticsearch с поддержкой русской морфологии и синонимического анализа. Проведён обзор существующих решений и предложен подход, обеспечивающий более точный и гибкий поиск рецептов.
Ключевые слова: веб-приложение, рецепты, поиск, Elasticsearch, Spring Boot, PostgreSQL.

Введение

Современные пользователи сталкиваются с необходимостью быстро находить и систематизировать рецепты, подбирать блюда по ингредиентам, а также хранить свои собственные записи. Существующие решения зачастую привязаны к конкретным экосистемам, требуют регистрации или платных подписок, а также ограничены по функциональности.

Актуальной задачей является создание независимого веб-приложения, позволяющего:

– хранить рецепты и ингредиенты пользователей;

– выполнять интеллектуальный поиск по названию, составу и категориям блюд;

– учитывать морфологию русского языка и синонимы для повышения точности поиска;

– предоставлять удобный интерфейс взаимодействия с системой.

Практическая значимость проекта заключается в создании открытой платформы, которая может быть развёрнута локально и расширена дополнительными сервисами — например, подбором блюд по калорийности или автоматическим формированием списка покупок.

1. Постановка задачи

Целью работы является создание веб-сервиса, обеспечивающего хранение, поиск и подбор рецептов с учётом русской морфологии и синонимов.

Программа должна выполнять следующие ключевые функции:

– регистрация и авторизация пользователей;

– хранение данных о рецептах, ингредиентах и связях между ними;

– реализация REST API для работы с сущностями;

– подключение поискового движка Elasticsearch для морфологического поиска;

– визуализация данных и взаимодействие через Swagger UI.

Для этого необходимо решить следующие задачи:

1. Проектирование модели данных.

– определить требуемые сущности и связи между ними;

– выбрать СУБД и разработать физическую схему таблиц.

2. Разработка серверной части.

– реализовать REST-контроллеры и сервисный слой на Spring Boot;

– реализовать регистрацию, аутентификацию по JWT;

– интегрировать Elasticsearch с анализатором для русского языка;

– реализовать загрузку и индексирование данных рецептов;

– разработать и внедрить алгоритм для поиска подходящих рецептов.

3. Создание клиентского интерфейса.

– разработать одностраничное веб-приложение на React и TypeScript [3];

– реализовать добавление продуктов и рецептов;

– разработать алгоритм поиска наиболее подходящих рецептов на основе продуктов пользователя.

2. Анализ задачи

2.1. Общий анализ задачи

Для решения поставленной задачи необходимо разработать серверную и клиентскую части приложения, которое позволит пользователям:

– добавлять собственные рецепты и ингредиенты;

– формировать персональный список имеющихся продуктов;

– получать подборку рецептов, которые можно приготовить из доступных ингредиентов;

– хранить все данные в базе и обновлять их в реальном времени.

Для обеспечения этой функциональности требуется создание следующих компонентов:

– модуля хранения и управления данными о продуктах, ингредиентах и рецептах;

– поискового механизма на базе Elasticsearch для морфологического поиска по ключевым словам и синонимам;

– алгоритма подбора рецептов по доступным ингредиентам;

– REST API для взаимодействия между клиентом и сервером;

– графического интерфейса пользователя для работы с базой рецептов.

Реализация перечисленных элементов обеспечивает универсальный инструмент, который можно использовать как в бытовых, так и в профессиональных целях (например, для автоматизации меню или расчёта закупок в заведениях общественного питания).

2.2. Анализ существующих решений

Среди существующих аналогов можно выделить сервисы Povarenok.ru, Eda.ru и 1000.menu, предоставляющие поиск и сортировку рецептов по категориям.

Ключевые возможности:

– поиск по названию блюда;

– фильтрация по ингредиентам и времени приготовления;

– наличие фотографий и отзывов.

Ограничения:

– отсутствие персонализированного подбора под конкретные продукты пользователя;

– невозможность локального хранения данных (работа только через сайт);

– ограниченные возможности расширения или интеграции с другими системами.

Разрабатываемое приложение Recipe Service решает эти ограничения, предоставляя открытый REST API, возможность локального развёртывания и расширяемую архитектуру. Особое внимание уделено поиску: используется Elasticsearch [4, 5] с подключённой русской морфологией и пользовательским словарём синонимов, что обеспечивает корректную обработку запросов вроде «картошка» → «картофель», «томат» → «помидор».

2.3. Алгоритм подбора рецептов

Для реализации интеллектуального подбора блюд на стороне сервера используется алгоритм, сопоставляющий список продуктов пользователя с ингредиентами, необходимыми для

приготовления рецептов. Каждому пользователю соответствует множество его продуктов $U = \{p_1, p_2, ..., p_n\}$ с указанием количества. Каждому рецепту $R = \{i_1, i_2, ..., i_n\}$ соответствует набор ингредиентов и их требуемых количеств.

Для каждого рецепта вычисляется коэффициент совпадения C_R по формуле [7]:

$$C_R = \sum_{i \in R} \min\left(1, \frac{Q_{recipe}(i)}{Q_{user}(i)}\right),$$

где $Q_{user}(i)$ — количество ингредиента у пользователя, а $Q_{recipe}(i)$ — количество, требуемое рецепту. Рецепты сортируются по убыванию коэффициента совпадения C_R, после чего пользователю возвращается ограниченный список лучших совпадений.

3. Программный комплекс

3.1. Средства реализации

В качестве языка программирования выбран Java 21, фреймворк Spring Boot 3 [2]. Возможности данного стека обеспечивают быстрое создание надёжных REST-сервисов с авторизацией и документированием API. Разработка проекта велась в среде IntelliJ IDEA 2024.1.4.

В качестве системы управления базами данных использована PostgreSQL 16.0 [1]. Она позволяет хранить рецепты, ингредиенты и пользователей, обеспечивая целостность данных с помощью транзакций и ограничений. Для работы с базой применяется Spring Data JPA и Flyway для миграций.

Полнотекстовый поиск рецептов реализован через Elasticsearch 9.0.2, настроенный на работу с русской морфологией и словарём синонимов. Это обеспечивает корректный поиск даже при использовании различных форм слов.

Клиентская часть проекта создана на React 19.1.0 с использованием TypeScript, что обеспечивает статическую типизацию и декларативное построение интерфейса.

Сервис представляет собой монолитное приложение [6]. Архитектура сервера разделена на уровни:

– контроллеры — принимают запросы и формируют ответы клиенту;
– сервисы — реализуют бизнес-логику сервиса;
– репозитории — отвечают за взаимодействие с таблицами базы данных;
– сущности — представляют структуру хранимых объектов.

Диаграмма классов сервера представлена на рис. 1.

3.2. Реализация серверной части

RecipeController — контроллер, отвечающий за обработку HTTP-запросов, связанных с рецептами. Методы класса:

– `private UserEntity getUserFromHeader(String authHeader)`;

Выполняет извлечение и валидацию JWT-токена, определяя пользователя, который выполняет запрос.

– `public ResponseEntity<?> getAllRecipes(@RequestParam(required = false) String category)`;

Возвращает список рецептов. При указании категории выполняется фильтрация.

– `public ResponseEntity<?> addRecipe(@RequestBody RecipeInputDto recipeInputDto)`;

Добавляет новый рецепт, выполняя валидацию входных данных.

– `public ResponseEntity<?> searchRecipes()`.

Возвращает список рекомендаций на основе продуктов, находящихся у пользователя.

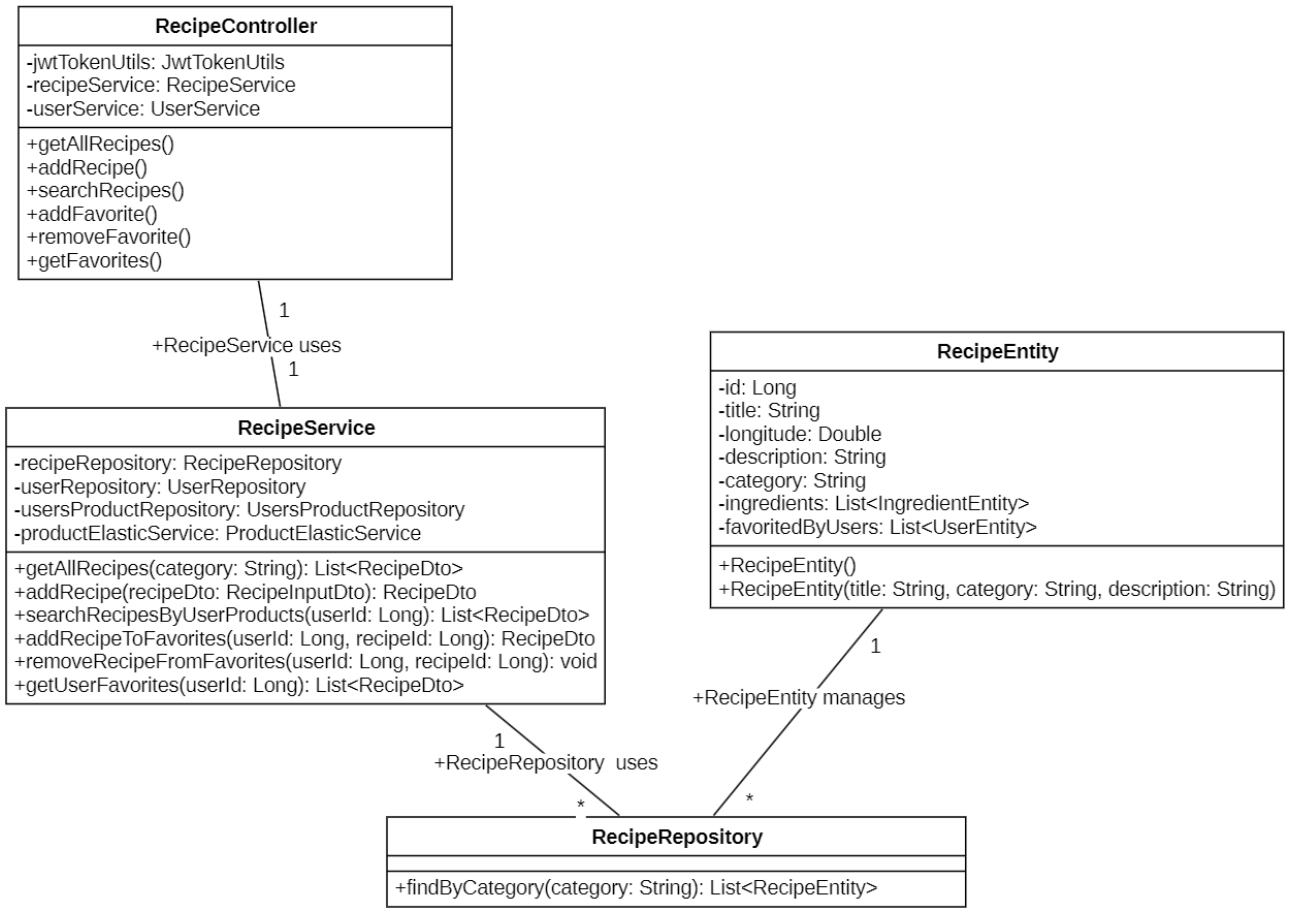

Рис. 1. Диаграмма классов модуля RecipeService

RecipeService содержит бизнес-логику обработки рецептов. Методы класса:

– `public List<RecipeDto> getAllRecipes(String category)`;

Загружает рецепты из базы и преобразует их в DTO.

– `public RecipeDto addRecipe(RecipeInputDto recipeDto)`;

Сохраняет новый рецепт, вместе с ингредиентами и связями между ними.

– `public List<RecipeDto> searchRecipesByUserProducts(Long userId)`;

Алгоритм подбора рецептов на основе холодильника пользователя. Для каждого рецепта вычисляется коэффициент совпадения. Рецепты сортируются по убыванию коэффициента.

– `public RecipeDto addRecipeToFavorites(Long userId, Long recipeId)`.

Добавляет рецепт в коллекцию избранных и сохраняет изменения.

RecipeRepository — интерфейс Spring Data JPA для работы с таблицей рецептов. Методы класса:

– `List<RecipeEntity> findByCategory(String category)`.

Выполняет поиск рецептов по категории с использованием автоматических запросов Spring Data.

3.3. Реализация клиентской части

Код клиентской части разбит на самостоятельные React-компоненты, каждый из которых располагается в отдельном .tsx-файле:

– LoginForm.tsx;

Форма авторизации и регистрации. Токен сохраняется в localStorage и автоматически подставляется в заголовки запросов.

– Navigation.tsx;

Верхнее меню приложения, содержащее ссылки на разделы и кнопку выхода.

– RecipeSection.tsx;

Основной компонент для просмотра рецептов. Поддерживаются фильтрация, сортировка и поиск по названию или ингредиентам.

– RecipeModal.tsx;

Модальное окно с подробной карточкой рецепта: изображение, описание, ингредиенты, время готовки, кнопка «Добавить в избранное».

– AddRecipeModal.tsx;

Форма создания нового рецепта: ввод названия, категории, ингредиентов и шагов приготовления. Выполняет клиентскую валидацию.

– FavoritesSection.tsx;

Отображает избранные рецепты пользователя, позволяет удалять рецепты из коллекции.

– FridgeSection.tsx;

Интерфейс управления «холодильником»: добавление, редактирование и удаление продуктов. Используется при подборе рецептов.

3.4. Интерфейс пользователя

Реализован интерфейс, представленный на рис. 2.

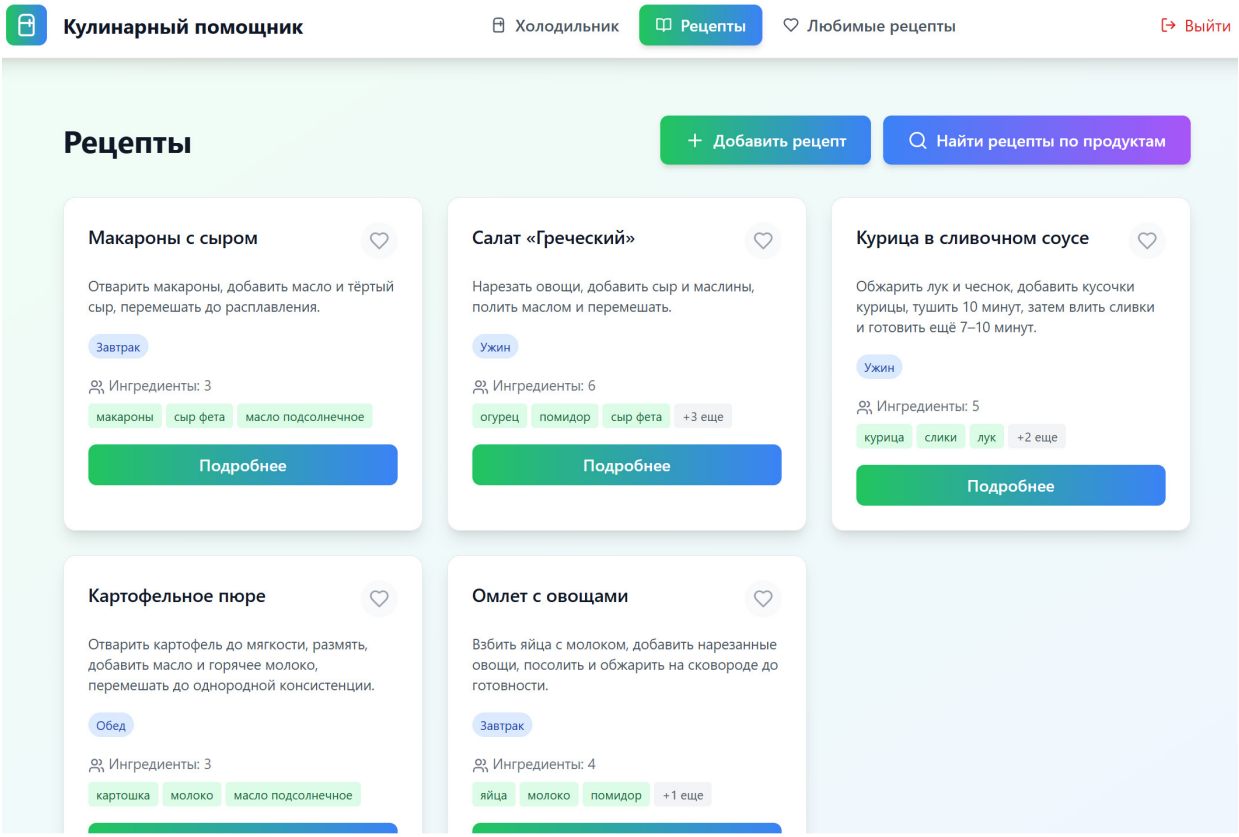

Рис. 2. Страница с рецептами

3.5. Тестирование

В рамках вычислительного эксперимента была проведена проверка корректности подборки рецептов на примере тестовых данных. Результат представлен на рис. 3.

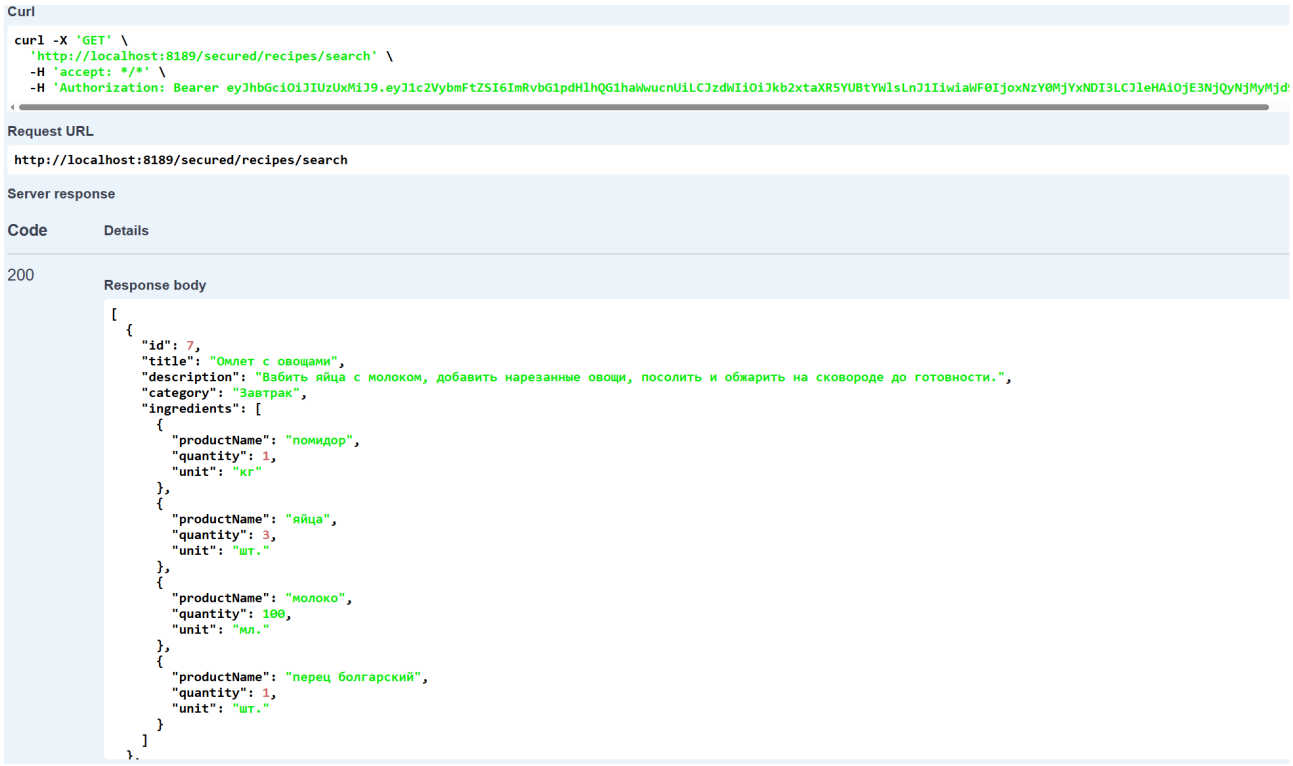

Рис. 3. Успешное получение рецептов

Заключение

В ходе выполнения работы был спроектирован и реализован веб-сервис, позволяющий хранить и подбирать рецепты с учётом доступных ингредиентов пользователя. Разработанное приложение объединяет сервер на Java 21 и Spring Boot 3 с клиентской частью на React 19 и TypeScript, что обеспечивает стабильную работу и удобный интерфейс.

Интеграция с Elasticsearch позволила реализовать поиск с поддержкой русской морфологии и синонимов, а алгоритм подбора рецептов по продуктам пользователя обеспечивает персонализированные результаты и расширяет практическое применение сервиса.

Литература

1. PostgreSQL Documentation, Global Development Group. PostgreSQL 16 Reference Manual. 2023. URL: https://www.postgresql.org/docs/16/index.html (дата обращения 01.06.2025).

2. Spring Boot Reference Guide, Pivotal Software. 2023. URL: https://docs.spring.io/spring-boot/docs/3.2.x/reference/ (дата обращения 28.06.2025).

3. React Documentation, Meta Platforms. 2024. URL: https://react.dev (дата обращения 27.06.2025).

4. Elasticsearch Reference, Elastic NV. 2024. URL: https://www.elastic.co/guide/index.html (дата обращения 20.06.2025)

5. *Gormley C., Tong Z.* Elasticsearch: The Definitive Guide. 2nd Edition. O'Reilly Media, 2020.

6. *Gamma E., Helm R., Johnson R., Vlissides J.* Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

7. *Knuth D. E.* The Art of Computer Programming. Vol. 1: Fundamental Algorithms. 3rd Edition. Addison-Wesley, 1997.

УДК 004.4

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ «ПАРКОВКА»

Г. А. Енокян, П. А. Серова

Воронежский государственный университет

Аннотация. Рассматривается разработка корпоративной системы управления бронированием парковочных мест сотрудниками компании. Система реализует авторизацию пользователей, просмотр доступности парковки в выбранную дату, создание заявок с указанием данных автомобиля, а также управление активными бронированиями.
Ключевые слова: мобильное приложение, серверная часть, микросервисная архитектура, Spring Boot, PostgreSQL, клиентская часть, платформа Android, реактивное программирование, Kotlin Coroutines.

Введение

Современные корпоративные структуры сталкиваются с необходимостью оптимизации внутренних процессов, особенно в управлении ограниченными ресурсами. Одним из таких ресурсов являются парковочные места, эффективное распределение которых требует надёжных, прозрачных и автоматизированных решений. Ручное бронирование приводит к избыточной административной нагрузке, снижает уровень организации, способствует возникновению конфликтов и снижает общее удобство использования территории.

Цифровизация процессов бронирования позволяет устранить эти проблемы, упростить контроль доступа, повысить эффективность использования инфраструктуры и обеспечить более высокий уровень комфорта для сотрудников.

Создание серверной части такой системы требует применения надёжных и масштабируемых технологий. Микросервисная архитектура в сочетании с современными средствами экосистемы Spring обеспечивает эффективное распределение ответственности между компонентами, упрощает сопровождение, масштабирование и внедрение новых функций. Spring Boot предоставляет стабильную и расширяемую платформу для создания микросервисов, а интеграция с модулями Spring Cloud позволяет реализовать автоматическую маршрутизацию, обнаружение микросервисов и их взаимодействие. Использование PostgreSQL в качестве СУБД гарантирует надёжное хранение информации и транзакционную целостность данных. Безопасность системы обеспечивается с помощью Spring Security и токенов JWT, валидация которых происходит централизованно, что снижает нагрузку на отдельные сервисы и упрощает контроль доступа.

Мобильные технологии, в частности платформа Android, предоставляют мощный инструментарий для создания интуитивной и доступной клиентской части рассматриваемой системы. Современные подходы к разработке, такие как Clean Architecture, Jetpack Compose и реактивное программирование с Kotlin Coroutines, позволяют создавать надежные, масштабируемые и удобные приложения, отвечающие высоким требованиям корпоративных пользователей. Интеграция локального хранения (Room) и сетевого взаимодействия (Retrofit) обеспечивает необходимую гибкость и отказоустойчивость.

Серверная часть мобильного приложения

Теоретические аспекты разработки серверных приложений

Микросервисная архитектура разделяет систему на небольшие независимые сервисы, которые можно развёртывать и изменять отдельно. Это повышает гибкость, снижает риски и упрощает масштабирование.

В основе проекта используется Spring Boot, который предоставляет встроенный сервер и минимизирует инфраструктурный код. Spring MVC обрабатывает HTTP-запросы и возвращает JSON-ответы. Дополняющие инструменты Spring Cloud — Gateway, Eureka и Feign — обеспечивают маршрутизацию, обнаружение сервисов и удобные декларативные HTTP-клиенты, что упрощает построение распределённой архитектуры.

Хранение данных реализовано с помощью PostgreSQL — надёжной СУБД с поддержкой транзакций и совместимостью со стандартом SQL. Flyway отвечает за миграции и синхронизацию схем базы данных, обеспечивая предсказуемость и контроль изменений в условиях множества сервисов.

Безопасность обеспечивается Spring Security и JWT: отдельный сервис формирует токены, а API Gateway проверяет их при каждом запросе. Такой подход централизует логику аутентификации и разгружает остальные микросервисы.

Межсервисное взаимодействие осуществляется через REST. Feign упрощает вызовы других сервисов, позволяя описывать их как обычные интерфейсы и автоматически интегрироваться с Eureka.

Совместное использование Spring Boot, Spring Cloud, Spring Security и PostgreSQL формирует компактную, модульную и масштабируемую архитектуру, в которой сервисы изолированы, маршруты и безопасность централизованы, а изменения можно внедрять быстро и локально, не затрагивая всю систему.

Проектирование системы

Разрабатываемая система управления бронированием парковочных мест построена на основе микросервисной архитектуры, обеспечивающей изоляцию бизнес-логики, масштабируемость и устойчивость к сбоям. Каждый сервис выполняет строго ограниченный набор функций и взаимодействует с другими компонентами исключительно через REST API.

Всего в систему входят пять микросервисов:

1. Auth-server — отвечает за авторизацию как по логину и паролю, так и по refresh-токену, а также выпуск и валидацию JWT-токенов.

2. User-service — управляет регистрацией, валидацией и хранением данных пользователей.

3. Booking-service — отвечает за создание, удаление и отображение бронирований.

4. Api-gateway — выполняет маршрутизацию запросов и фильтрацию по авторизации.

5. Discovery-server — служба обнаружения, позволяющая сервисам регистрироваться в системе и находить другие микросервисы.

Каждый микросервис реализован на базе Spring Boot с использованием Spring MVC.

Взаимодействие между клиентским приложением и серверной частью осуществляется через REST API, сгруппированные по зонам ответственности микросервисов. Все запросы защищены токеном доступа, передаваемым в заголовке Authorization, за исключением самой авторизации.

Примеры ключевых маршрутов:

• Авторизация:

POST /parking/auth/authorize — вход по логину и паролю,

POST /parking/auth/authorize/by_refresh — вход по refresh-токену.

• Бронирование:

POST /parking/booking_request — создание заявки на бронирование,

GET /parking/bookings/details — получение заявок на выбранную дату,

DELETE /parking/booking_request — удаление заявки.

• Пользователи:

GET /parking/personal_data — получение ФИО пользователя по access-токену,

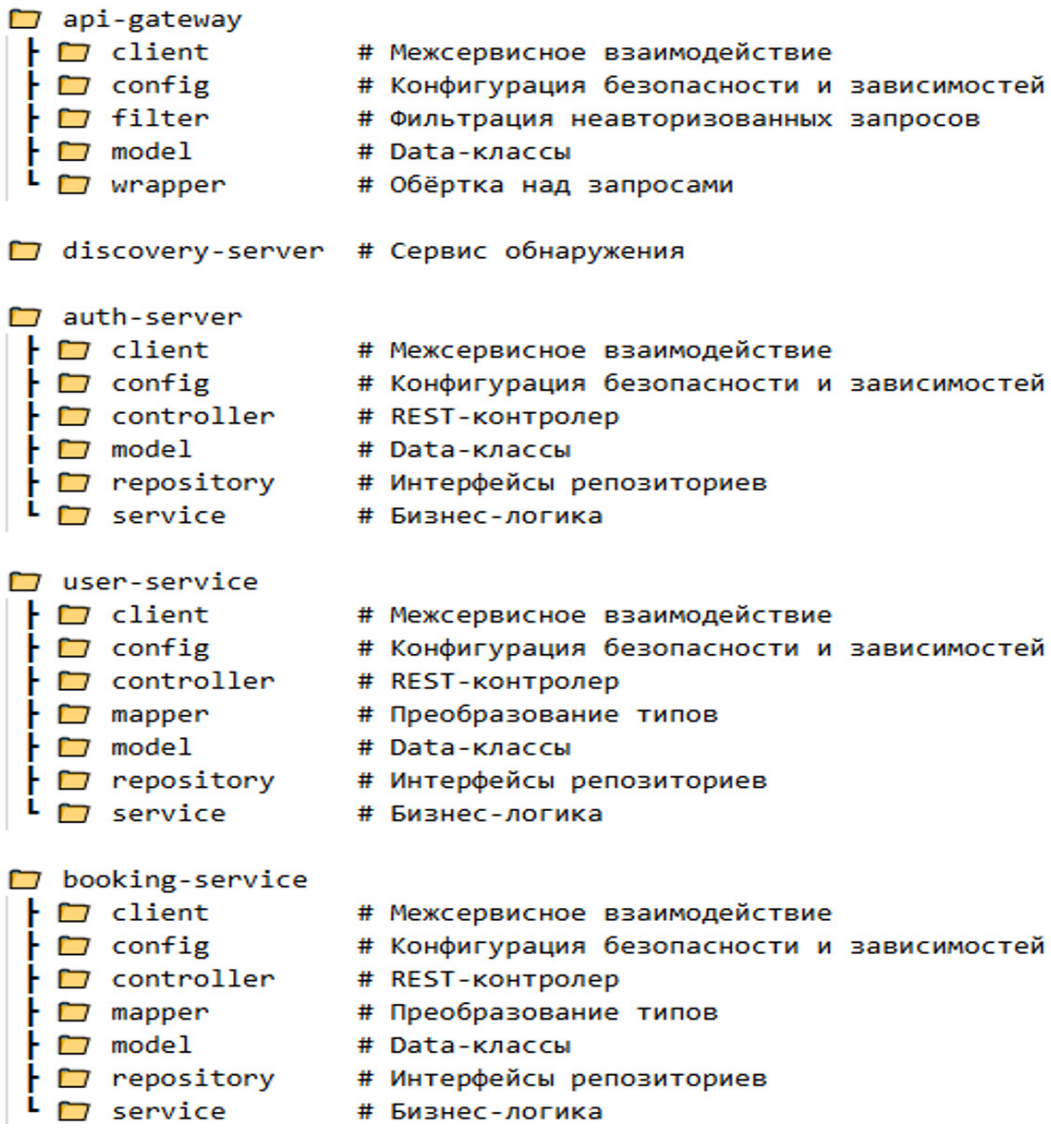

Рис. 1. Логическая архитектура приложения

POST /parking/users — регистрация пользователя (доступно только администратору).

Маршруты реализованы согласно REST-принципам, с разделением по HTTP-методам: GET — чтение, POST — создание, DELETE — удаление.

Ключевые микросервисы используют отдельные базы данных PostgreSQL, управляемые через Flyway. Миграции размещаются в директориях resources/db/migration и оформлены в SQL-формате.

Распределение по микросервисам:

• Auth-server: таблицы access_token и refresh_token с привязкой к username и времени действия.

• User-service: таблица user_entity, содержащая имя пользователя, пароль, ФИО и роли.

• Booking-service: таблица booking_entity, содержащая заявки пользователей на бронирование.

Таблицы спроектированы с учётом ограничений уникальности и первичных ключей. Например, в таблице заявок установлен уникальный составной ключ (created_by, date).

Аутентификация и проверка доступа в системе реализованы с использованием JWT (JSON Web Token) и Spring Security. Проверка валидности токенов осуществляется централизованно через auth-server.

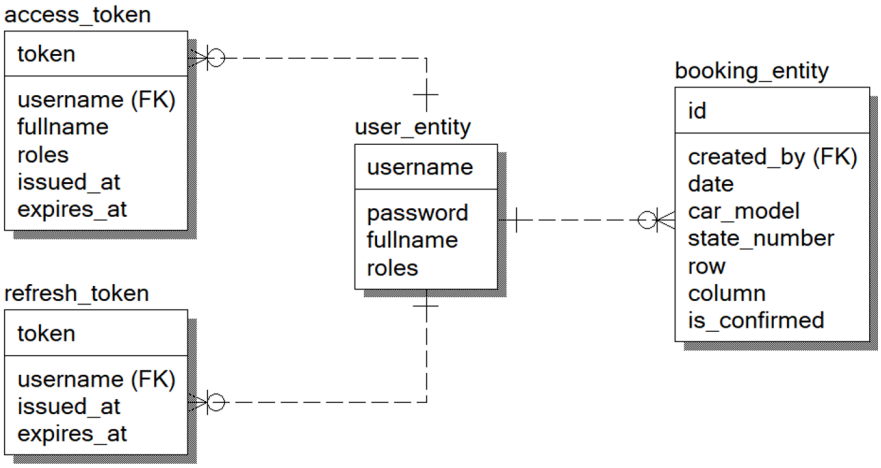

Рис. 2. Схема базы данных

После успешной авторизации пользователь получает пару токенов: accessToken (для доступа к защищённым ресурсам) и refreshToken (для обновления access-токена). Оба токена хранятся в базе данных auth-server и содержат время выдачи и истечения, а также имя пользователя.

При обращении к защищённому маршруту:

1. Клиент отправляет accessToken в заголовке Authorization.
2. Api-gateway направляет токен в auth-server для валидации.
3. В случае успеха auth-server возвращает имя пользователя.

Такой подход обеспечивает централизованный контроль доступа и снижает нагрузку на сервисы, исключая дублирование проверок. Токены подписываются и не могут быть подделаны, а их хранение в базе позволяет при необходимости отозвать права доступа (например, при выходе пользователя из системы).

Разделение системы на независимые микросервисы позволило чётко разграничить зоны ответственности и упростить сопровождение. Использование Spring Boot, Spring Cloud и централизованного шлюза обеспечило отказоустойчивость, масштабируемость и безопасность. Благодаря гибкой архитектуре систему можно легко расширять без риска для текущей функциональности.

Клиентская часть мобильного приложения

Теоретические аспекты разработки Android-приложений

Clean Architecture обеспечивает разделение ответственности и независимость кода от платформы. Архитектура строится вокруг трёх слоёв: Domain содержит бизнес-логику и интерфейсы, оставаясь полностью чистым; Data реализует работу с сетью и базой данных, опираясь только на абстракции Domain; Presentation обеспечивает UI и взаимодействие с пользователем. Инверсия зависимостей изолирует слои и делает код тестируемым.

Пользовательский интерфейс построен на Jetpack Compose — современном декларативном фреймворке, который описывает UI как функцию состояния и автоматически обновляет элементы при его изменении. Compose упрощает архитектуру, облегчает интеграцию с MVVM и значительно снижает объём шаблонного кода.

Внедрение зависимостей реализовано через Hilt, который автоматически генерирует необходимый DI-код, управляет жизненным циклом объектов и позволяет легко заменять реализации для тестов. Это упрощает связь между слоями и обеспечивает соблюдение принципа инверсии зависимостей.

Для локального хранения данных используется Room, предоставляющая типобезопасный доступ к SQLite, проверку запросов на этапе компиляции, поддержку корутин и механизм ми-

граций. Для сетевого взаимодействия применяется Retrofit, позволяющий декларативно описывать API, использовать Kotlin Serialization и выполнять запросы через suspend-функции.

В совокупности выбранный стек формирует стабильную и удобную в сопровождении архитектуру: Data-слой объединяет Retrofit, Room и корутины; Domain содержит чистые use case-классы; Presentation использует Compose, ViewModel и Hilt. Асинхронность, DI и сериализация реализованы единообразно и согласованно. Такой подход соответствует современным практикам Android-разработки, повышает тестируемость, надёжность и облегчает адаптацию приложения к новым требованиям.

Проектирование системы

Система имеет четкое разделение на слои через пакетную структуру:

Взаимодействие слоев:

1. Presentation → Domain: ViewModels вызывают Use Cases, получают доменные модели.

2. Domain → Data: Use Cases используют интерфейсы репозиториев. Реализации репозиториев находятся в Data слое.

3. Data → Domain: Репозитории возвращают доменные модели. Мапперы преобразуют DTO/Entity в Domain объекты.

Навигационная модель состоит из двух контроллеров:

1. Глобальный контроллер: содержит в себе логику навигации до авторизации и перехода к контроллеру с вкладками приложения.

```
data
├── api          # Retrofit, API сервисы
├── local        # Room DAO, база данных, локальный кэш, взаимодействие с AndroidKeyStorage
├── mapper       # Объекты для Mapping'а в Domain-Layer
├── repository   # Реализации репозиториев
└── ssl          # Интерфейс и реализация класса для использования SSL Pinning

di             # Модули Hilt для внедрения зависимостей

domain
├── model      # Data-классы
├── repository # Интерфейсы репозиториев
└── usecase    # Бизнес-логика

presentation
├── navigation # Навигация приложения
├── ui         # UI-компоненты Compose
├── screen     # Экранные composables
└── viewmodel  # ViewModels
```

Рис. 3. Пакетная структура приложения

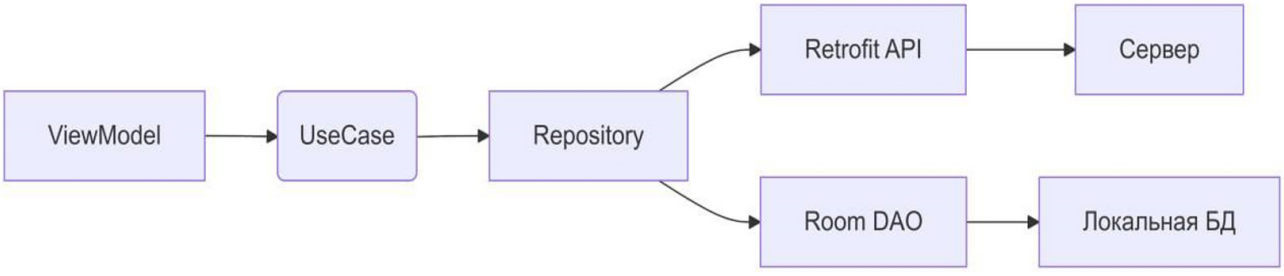

Рис 4. Схема зависимостей для сценария загрузки данных на экране списка бронирований

2. Контроллер вкладок приложения: содержит в себе логику навигации между вкладками после авторизации.

При этом контроллер вкладок приложения содержит в себе ссылку на глобальный контроллер, что позволяет при выходе из учетной записи навигироваться на экран авторизации.

Дизайн-система: используется MaterialTheme с заданными цветами для светлой и тёмной тем, реализован переход цвета от светлой темы к тёмной и обратно.

Принципы UX:

1. Индикация статуса: цветовая маркировка мест (свободно/занято/занято пользователем).

2. Однопоточность действий: линейный сценарий: Выбор даты → Выбор места → Подтверждение.

3. Обратная связь: AlertDialog для результатов операций.

4. Адаптивность: размеры элементов экрана указываются не в пикселях, а в аппаратно-независимых пикселях (Device independent pixel), что позволяет UI-элементам отображаться корректно на различных устройствах, вне зависимости от разрешения.

Макеты экранов:

1. Авторизация:

• Поле ввода логина;
• Поле ввода пароля;
• Кнопка действия с лоадером при ожидании ответа сервера;
• Полноэкранный лоадер при повторной авторизации, если не требуется ввод данных;
• Переход ко вкладкам приложения в случае успешной авторизации.

2. Вкладки приложения (навигационное меню):

• Карта парковки;
• Бронирование;
• Список броней;
• Настройки.

3. Карта парковки:

• Заголовок экрана;
• Виджет выбора даты;
• Схема парковки с кликабельными местами с шиммером при ожидании ответа;
• Боттомшит, открывающийся при нажатии на парковочное место, содержащий информацию о брони.

4. Экран бронирования:

• Заголовок экрана;
• Виджет выбора даты;
• Поле ввода модели авто;
• Поле ввода госномера;
• Кнопка действия с лоадером при ожидании ответа сервера;
• AlertDialog при получении успешного ответа от сервера.

5. Экран списка броней:

• Заголовок экрана;
• Поддержка Pull-to-Refresh;
• Вкладка, содержащая карточки с подтвержденным бронями;
• Вкладка, содержащая карточки с неподтвержденным бронями.

6. Экран настроек:

• Заголовок экрана;
• Блок с персональными данными пользователя (ФИО);
• Выпадающий список для выбора языка приложения;
• Выпадающий список для выбора темы приложения;

• Блок с информацией о приложении (название, версия);
• Кнопка действия (выхода из учетной записи) с лоадером при ожидании ответа сервера.

Все экраны должны показывать AlertDialog с описанием ошибки при получении ошибки в ответе сервера и отобразить на экране кэш, если он есть.

Полученная архитектура обеспечивает:
• Соблюдение принципа единственной ответственности через разделение пакетов;
• Тестируемость за счет инверсии зависимостей;
• Масштабируемость — возможность выделения модулей в будущем;
• Согласованность UI благодаря использованию единой дизайн системы.

Заключение

В рамках разработки была спроектирована и реализована корпоративная система бронирования парковочных мест, основанная на современных технологиях и подходах.

Разделение логики в серверной части по микросервисам позволило чётко разграничить функциональные зоны, упростить поддержку и сопровождение системы, а также заложить основу для дальнейшего масштабирования. Ключевыми компонентами стали сервисы аутентификации и авторизации, управления пользователями, приёма и обработки заявок на бронирование. Каждый из них разрабатывался как отдельное Spring Boot приложение с использованием Spring MVC и доступом к собственной базе данных PostgreSQL через Spring Data JPA. Для взаимодействия между сервисами использовались REST API и Feign-клиенты. Обнаружение и регистрация сервисов реализованы с помощью Eureka, а маршрутизация и проверка токенов — через Spring Cloud Gateway. Аутентификация пользователей осуществляется через JWT-механизм с централизованной валидацией токенов на уровне шлюза. Такой подход снижает нагрузку на микросервисы и обеспечивает единообразную политику доступа. Миграции баз данных выполняются с помощью Flyway, что позволяет автоматически применять изменения в схемах при развертывании.

Клиентская часть представляет собой комплексное Android-приложение, реализованное с помощью современных технологий разработки (Kotlin, Jetpack Compose, Clean Architecture), и позволяет сотрудникам эффективно бронировать места, отслеживать статус заявок и управлять своими бронированиями через интуитивно понятный интерфейс с поддержкой различных локализаций и тем оформления.

Литература

1. *Башарин И. В.* Spring в действии. – 6-е изд. – М. : ДМК Пресс, 2022. – 544 с.
2. *Walls C.* Spring Boot in Action. – Manning Publications, 2016. – 264 p.
3. *Richardson C.* Microservices Patterns: With examples in Java. – Manning Publications, 2019. – 520 p.
4. *Dehghani M.* Cloud-Native Patterns: Designing change-tolerant software. – O'Reilly Media, 2021. – 352 p.
5. PostgreSQL 15 Documentation. – PostgreSQL Global Development Group, 2023. – https://www.postgresql.org/docs/15
6. *Мартин Р.* Чистая архитектура. Искусство разработки программного обеспечения / Пер. с англ. – Москва : Питер, 2022. – 352 с.
7. *Гриффитс Дэвид, Гриффитс Дон*. Head First. Программирование для Android на Kotlin. 3-е изд. / Пер. с англ. – Е. П. Матвеев : Питер, 2023. – 912 с.
8. Retrofit. – Режим доступа: https://square.github.io/retrofit/.

УДК 531.2 + 004.4

РАЗРАБОТКА ОБУЧАЮЩЕЙ ПРОГРАММНОЙ СИСТЕМЫ ДЛЯ РЕШЕНИЯ ЗАДАЧ СТАТИКИ ТЕОРЕТИЧЕСКОЙ МЕХАНИКИ

В. М. Ермак, А. Е. Лазарев, У. Н. Пасько, Ю. Е. Иванова, А. А. Лаптева

Дальневосточный федеральный университет

Аннотация. В работе представлено описание программной системы для решения задач статики по теоретической механике. Приложение ориентировано на применение в учебном процессе и позволяет рассчитывать реакции опор в различных схемах из балок, опор и нагрузок. Основой вычислительного компонента служит теория графов, где конструкция представляется как неориентированный граф. Такой подход обеспечивает автоматическую генерацию уравнений равновесия независимо от сложности схемы. Решение осуществляется в символьном виде с использованием библиотеки SymPy. Программа реализована на языке Python и включает компоненты «Интерфейс» и «Решатель».
Ключевые слова: статика, балочные конструкции, теоретическая механика, реакции опор, уравнения равновесия, символьное решение, теория графов, неориентированный граф, алгоритм обхода графа, графический интерфейс.

Введение

При изучении раздела «Статика» учебной дисциплины «Теоретическая механика» в высших и средних учебных заведениях, когда речь идёт о равновесии балочных систем под нагрузками, одной лишь теоретической подготовки недостаточно. Необходимо выполнять значительное количество практических задач, чтобы выработать интуицию и методику решения задач определенного типа [1]. Здесь на помощь приходят специализированные программные продукты — они позволяют оперативно генерировать задания, выполнять вычисления и анализировать полученные результаты. При этом важно, чтобы приложением пользовался студент только как средством проверки конечного результата: само решение необходимо выполнять вручную. Такая форма работы позволит использовать программу как «чёрный ящик» и сосредоточиться на исследовании влияния геометрии конструкции и параметров нагрузок на реакции опор.

1. Анализ существующих решений

Наиболее популярными программными продуктами являются «Расчёт статически определимых рам и балок» [2], Балка-онлайн SOPROMATu.NET [3], СОПРОМАТ ГУРУ [4], Free Online Beam Calculator | Reactions, Shear Force, etc. [5]. В ходе их анализа выяснилось [6], что большинство решений имеют ограничения: не допускают произвольной постановки задач (например, составление собственной схемы с балками и нагрузками), либо часть функционала доступна лишь по платной подписке. Результат сравнения представлен в табл. 1. Учитывая эти ограничения, было принято решение разработать свою гибкую и визуально информативную программную систему, ориентированную на учебные задачи курса «Теоретическая механика». Основная цель разработки — обеспечить расчёт реакций опор для систем, моделируемых произвольным количеством тел (балок) с различными вариантами опор и нагрузок.

2. Архитектура программной системы

Архитектура программной системы предполагает наличие двух основных компонент: графический «Интерфейс» и вычислительный «Решатель» (рис. 1).

Таблица 1

Сравнительный анализ существующих решений

Критерий сравнения	Расчёт статически определимых рам и балок [2]	Балка-онлайн SOPROMATu.NET[3]	СОПРОМАТ ГУРУ [4]	Free Online Beam Calculator | Reactions, Shear Force, etc.[5]
Возможность добавить задачи	нет	нет	нет	нет
Печать	да	нет	да	нет
Доступ к интернету	необходим	необходим	необходим	необходим
Конструктор (возможность перетаскивать объекты)	да	нет	нет	нет
Русский язык	да	да	да	нет
Бесплатная версия	нет	да	да	нет

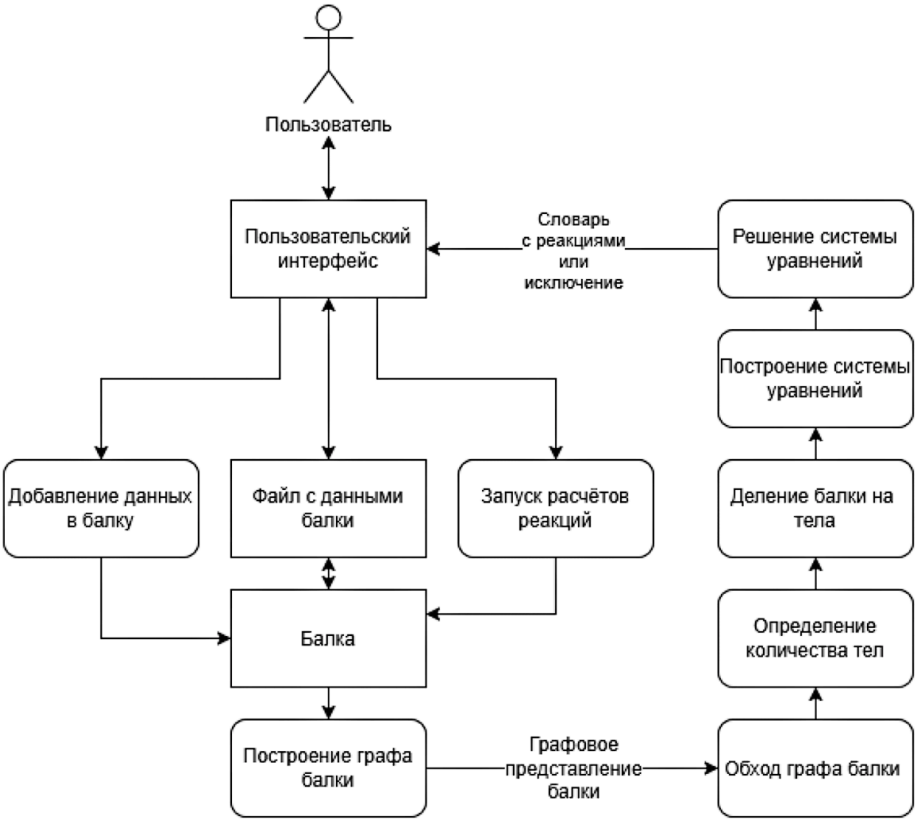

Рис. 1. Общая архитектурно-контекстная диаграмма программной системы для решения задач статики теоретической механики

Компонент «Интерфейс» обеспечивает взаимодействие пользователя с системой, позволяет задавать параметры расчётной схемы, добавлять тела, связи и нагрузки, а также отображать результат вычислений.

Компонент «Решатель» представляет собой вычислительный модуль, который на основе данных, полученных от «Интерфейса», формирует внутреннее представление конструкции в виде графа. Узлы графа соответствуют телам и точкам приложения связей, а рёбра — взаимодействиям между ними. Такая структура позволяет описывать системы с произвольным числом тел и связей. В «Решателе» проверяется корректность построенного графа, формируется и

решается система уравнений равновесия. Количество уравнений определяется конфигурацией системы, что дает возможность использовать «Решатель» для задач различной сложности (количество прямолинейных балок, сочетание видов опор и заделок). Результат расчёта — реакция опор.

Работа всей системы происходит последовательно: пользователь задаёт схему через «Интерфейс», информация передается в «Решатель», а результат возвращается обратно и отображается на экране. Такая архитектура обеспечивает модульность, расширяемость и независимость компонентов.

3. Представление балочной конструкции в виде графа

Задача компонента «Решатель» заключается в построении и решении системы уравнений равновесия в символьном виде. Число уравнений определяется количеством тел в системе: при увеличении числа балок пропорционально возрастает и число уравнений равновесия. Для обеспечения масштабируемости и универсальности в основе алгоритма лежат методы теории графов [7]. Механическую конструкцию можно формализовать как неориентированный граф: вершины этого графа — узлы, в которых соединяются балки, устанавливаются опоры или шарниры, а рёбра — сами балки с приложенными нагрузками. Такой подход позволяет автоматически формировать уравнения равновесия по всей структуре, независимо от геометрических особенностей (формы сечения, толщины элемента, гибкости соединения не учитываются, что соответствует упрощённой учебной постановке).

В модели вершина графа соответствует узловой точке конструкции, где может быть задано закрепление (опора, шарнир), либо соединение элементов. Ребро — это линейный элемент (балка) между двумя узлами, на котором могут быть приложены сосредоточенные силы, моменты или распределённая нагрузка. Для анализа системы используется алгоритм обхода графа в ширину, начиная с произвольно выбранного узла. При таком обходе каждое тело (балка) будет «посещено» ровно один раз, и топология системы будет последовательно учтена. В процессе обхода для каждой балки автоматически формируются три основных уравнения равновесия (сумма проекций сил по осям X и Y и сумма моментов). Дополнительно, для каждого шарнира, соединяющего тела, составляются два уравнения — сумма реакций по оси X и по оси Y. Эти уравнения выражают условие, что сумма сил в месте соединения тел равна нулю.

Предполагается, что для заданной конфигурации может быть построена полная система линейных алгебраических уравнений, достаточная для однозначного определения всех неизвестных реакций. Все внешние нагрузки считаются заранее известными по величине и направлению; неизвестными остаются только реакции, которые вычисляет «Решатель». Подобная схема допускает гибкую генерацию уравнений независимо от числа балок и сложности схемы. Сформированная система уравнений решается символьно, и результатом становится словарь значений реакций в формате, подходящем для отображения в «Интерфейсе».

Проверка корректности входных данных и обработка ошибок реализуется через собственную систему исключений.

Заключение

Реализована программная система, предназначенная для составления схем балочных конструкций в задачах статики курса «Теоретическая механика», и выполнения проверки полученных «вручную» студентами результатов. Проанализированы аналогичные программные решения, их преимущества и недостатки. Архитектура системы спроектирована с учетом разделения на два компонента «Интерфейс» и «Решатель». Для реализации выбран язык программирования Python.

Литература

1. *Тарг С.* М. Курс теоретической механики: учеб. пособие / С. М. Тарг. – Москва : Высшая школа, 1986. – 512 с.

2. Эпюры поперечных сил и изгибающих моментов // Sopromat.site: [сайт]. – URL: https://sopromat.site/epure/ (дата обращения: 20.10.2025).

3. Расчёт балки онлайн // Sopromatu.net: [сайт]. – URL: https://sopromatu.net/beam/ (дата обращения: 20.10.2025).

4. Сопротивление материалов: учебные материалы и калькуляторы // Sopromatguru.ru: [сайт]. – URL: https://sopromatguru.ru/ (дата обращения: 20.10.2025).

5. Free Beam Calculator // SkyCiv: [official website]. – Sydney, 2015. – URL: https://skyciv.com/free-beam-calculator/ (дата обращения: 20.10.2025).

6. *Ермак В. М.* Обзор программных средств для автоматизации разработки обучающих материалов в задачах статики теоретической механики [Электронный ресурс] / В. М. Ермак, А. Е. Лазарев, И. В. Лебединский, У. Н. Пасько // Региональная научно-практическая конференция студентов, аспирантов и молодых учёных по естественным наукам, Владивосток, 15–30 апреля 2024 г. – Владивосток : Дальневосточный федеральный университет, 2024. – С. 168. – URL: https://www.dvfu.ru/institute_of_high_technologies_and_advanced_materials/ Conferences/ (дата обращения: 20.10.2025).

7. *Буркатовская Ю. Б.* Теория графов. Часть 1: учеб. пособие / Ю. Б. Буркатовская ; Томский политехнический университет. – Томск : Изд-во Томского политехн. ун-та, 2014. – 200 с.

УДК 681.3.06

ШИФРОВАНИЕ ДАННЫХ ДЛЯ ПЕРЕДАЧИ МЕЖДУ СЕРВЕРОМ И ПОЛЬЗОВАТЕЛЕМ ПРИ РАЗРАБОТКЕ WEB-ПРИЛОЖЕНИЯ ASP.NET CORE

Д. А. Заложных

Воронежский государственный университет

Аннотация. Статья посвящена исследованию ключевых аспектов обеспечения безопасности передачи данных в современных веб-приложениях, в частности разработанных на платформе ASP.NET Core. Анализируются основные риски и угрозы, связанные с обменом информацией между сервером и клиентским браузером. Проводится обзор основных видов шифрования и протокол TLS, который является стандартом для защиты веб-трафика. Особое внимание уделяется встроенным инструментам и API защиты данных, предоставляемым платформой ASP.NET Core, которые позволяют разработчикам эффективно реализовывать шифрование, управление ключами и обеспечивать целостность передаваемой информации.
Ключевые слова: шифрование данных, ASP.NET Core, безопасность web-приложений, передача данных, TLS, симметричное шифрование, асимметричное шифрование, защита данных, информационная безопасность, криптография, веб-разработка.

Введение

В настоящее время в связи с повсеместным использованием web-приложений, вопрос обеспечения безопасности данных, передаваемых между сервером и конечным пользователем, особенно важен. Конфиденциальная информация, такая как учетные данные, платёжные реквизиты, личная переписка, становится целью злоумышленников.

Шифрование является фундаментальным механизмом защиты, гарантирующим конфиденциальность и целостность данных на всем пути их следования.

Разработка на платформе ASP.NET Core предоставляет мощные и гибкие инструменты для реализации надежных систем защиты информации.

1. Распространённые угрозы при передаче данных

Процесс передачи сообщений между клиентским приложением (браузером) и сервером уязвим для различных атак. Без надлежащих мер защиты информация передается в открытом виде, что создает серьезные угрозы безопасности. Для разработки надёжного web-приложения необходимо ознакомиться с основными типами атак.

Атака с использованием скомпрометированного ключа происходит, когда злоумышленник определяет какой-либо из ключей для шифрования, расшифровки или проверки секретной информации. Если злоумышленнику удаётся определить ключ, он может использовать его для расшифровки зашифрованных данных без ведома отправителя.

Атака типа «отказ в обслуживании» ставит своей целью нарушение нормальной работы сервера, сделав его недоступным для легитимных пользователей. Это происходит, когда злоумышленник заваливает сервис запросами, которые перегружают его так, что сервер не успевает обрабатывать запросы и переходит в режим «отказа в обслуживании».

Отличительной особенностью данного типа атак является то, что они обычно не приводят к краже, уничтожению или повреждению данных. Они наносят ущерб учреждениям, выводя их системы из строя.

При ***подслушивании (слежке)*** злоумышленник получает доступ к каналу передачи данных в сети и может отслеживать и считывать трафик. Если трафик передаётся в виде обычного текста, злоумышленник может считать его, получив доступ к каналу.

Подмена идентификационных данных происходит, когда злоумышленник определяет и использует номер телефона действительного пользователя (идентификатор абонента) или IP-адрес сети, компьютера или сетевого компонента без соответствующих полномочий. Успешная атака позволяет злоумышленнику действовать так, как если бы он был субъектом, обычно идентифицируемым по этому номеру телефона или IP-адресу.

Атака «человек посередине» происходит, когда злоумышленник перенаправляет соединение между двумя пользователями через свой компьютер. Злоумышленник может отслеживать и считывать трафик, прежде чем отправить его нужному получателю. Каждый пользователь, участвующий в обмене данными, неосознанно отправляет трафик злоумышленнику и получает от него трафик, при этом думая, что он общается только с нужным пользователем.

2. Виды и алгоритмы шифрования

Шифрование — это процесс преобразования читаемых данных (открытого текста) в зашифрованный формат (шифротекст) с помощью криптографического алгоритма и ключа. Расшифровать данные и вернуть их в исходный вид может только тот, кто обладает соответствующим ключом. Существует три основных вида шифрования.

2.1. Симметричное шифрование

При симметричном шифровании для зашифровки и расшифровки данных используется один и тот же секретный ключ. Этот метод отличается высокой скоростью работы, что делает его идеальным для шифрования больших объемов данных.

AES (Advanced Encryption Standard) на сегодняшний день является «золотым стандартом» симметричного шифрования. Он использует блоки размером 128 бит и ключи длиной 128, 192 или 256 бит. AES быстр, надежен и широко используется во всем мире, в том числе государственными структурами для защиты секретной информации.

Основная проблема симметричного шифрования заключается в безопасной передаче секретного ключа между сторонами.

2.2. Ассиметричное шифрование

Асимметричное шифрование (или криптография с открытым ключом) использует пару ключей: открытый (публичный) и закрытый (приватный). Открытый ключ свободно распространяется и используется для шифрования данных. Закрытый ключ хранится в секрете у владельца и является единственным ключом, способным расшифровать данные, зашифрованные соответствующим ему открытым ключом.

Этот подход решает проблему обмена ключами. Наиболее известный алгоритм асимметричного шифрования — RSA (Rivest-Shamir-Adleman).

2.3. Гибридное шифрование и протокол TLS

На практике для защиты веб-трафика используется гибридный подход, реализованный в протоколе TLS (Transport Layer Security). Протокол TLS обеспечивает безопасную передачу данных между узлами в сети Интернет. Асимметричное шифрование используется на началь-

ном этапе соединения для обмена ключами и аутентификации, а симметричное — для шифрования непосредственно сообщений.

В процессе установления защищенного соединения (TLS Handshake) клиент (браузер) подключается к серверу и запрашивает безопасное соединение, сервер отправляет в ответ свой SSL/TLS-сертификат, который содержит его открытый ключ. Клиент проверяет подлинность сертификата и, если все в порядке, генерирует случайный сеансовый ключ (для симметричного шифрования). Клиент шифрует этот сеансовый ключ с помощью открытого ключа сервера и отправляет его обратно. Сервер расшифровывает полученное сообщение своим закрытым ключом и получает сеансовый ключ.

После этого обе стороны обладают общим секретным ключом, который используется для симметричного шифрования (например, по алгоритму AES) всего последующего трафика в рамках данной сессии.

Использование обоих типов шифрования помогает протоколу TLS держать баланс между скоростью работы и безопасностью.

3. Защита данных в ASP.NET Core

Платформа ASP.NET Core предоставляет разработчикам встроенные механизмы для обеспечения безопасности данных, ключевым из которых является система защиты данных (Data Protection API).

3.1. ASP.NET Core Data Protection API

Эта система представляет собой набор криптографических API для защиты данных, включая шифрование и расшифровку. Она разработана для решения основной проблемы: как обеспечить безопасность данных, которые необходимо сохранить, а затем прочитать, но при этом не хранить в открытом виде. Можно выделить три основных сценария её использования.

Защита cookie-файлов аутентификации. Когда пользователь входит в систему, ASP.NET Core создает cookie, содержащий его удостоверение. Data Protection API шифрует этот cookie, чтобы его содержимое не могло быть прочитано или изменено на стороне клиента.

Защита токенов. Например, токены для сброса пароля или подтверждения электронной почты шифруются перед отправкой пользователю в виде URL-адреса.

Защита данных состояния (state). Например, в технологии Blazor Server эта система используется для защиты данных сеанса пользователя.

3.2. Основные пакеты ASP.NET Core Data Protection

Стек защиты данных состоит из пяти пакетов.

Microsoft.AspNetCore.DataProtection.Abstractions содержит IDataProtectionProvider и IDataProtector — интерфейсы для создания служб защиты данных и полезные методы расширения для работы с этими типами. например, IDataProtector.Protect.

Microsoft.AspNetCore.DataProtection содержит основную реализацию системы защиты данных, в том числе основные криптографические операции, управление ключами, конфигурация и расширяемость.

Microsoft.AspNetCore.DataProtection.Extensions содержит дополнительные API, которые разработчики могут найти полезными. Например, этот пакет содержит фабричные методы для создания экземпляра системы защиты данных для хранения ключей в определенном месте файловой системы без использования внедрения зависимостей, методы расширения для ограничения срока службы защищенной нагрузки.

Microsoft.AspNetCore.DataProtection.SystemWeb можно установить в существующее приложение ASP.NET 4.x для перенаправления <machineKey> операций для использования нового стека защиты данных ASP.NET Core.

Microsoft.AspNetCore.Cryptography.KeyDerivation предоставляет реализацию процедуры хэширования паролей PBKDF2 и может использоваться системами, которые должны безопасно обрабатывать пароли пользователей.

3.3. Настройка HTTPS

ASP.NET Core по умолчанию настроен на использование HTTPS, что является реализацией протокола HTTP поверх защищенного TLS-соединения. При создании нового проекта в Visual Studio автоматически генерируется самозаверенный сертификат для разработки, а в код запуска приложения добавляется промежуточное ПО (middleware) для перенаправления всех HTTP-запросов на HTTPS (app.UseHttpsRedirection()).

Для производственной среды (production) необходимо получить и настроить доверенный SSL/TLS-сертификат от центра сертификации (например, Let's Encrypt). Настройка выполняется на уровне веб-сервера (IIS, Nginx, Apache) или непосредственно в Kestrel — встроенном веб-сервере ASP.NET Core.

4. Использование ASP.NET Core Data Protection при разработке web-приложения

При использовании services.AddDataProtection() в приложении (делается неявно при работе с некоторыми компонентами, например Identity) по умолчанию включается ряд механизмов.

Используются **алгоритмы защиты данных** для сохранения их конфиденциальности и целостности. По умолчанию, алгоритм шифрования — AES-256-CBC, а алгоритм проверки — HMACSHA256. Вся полезная нагрузка сначала шифруется, затем подписывается.

Осуществляется **управление ключами**. Новые ключи генерируются автоматически каждые 90 дней (настраивается). Старые ключи сохраняются, чтобы можно было расшифровать данные, защищённые ранее. В Windows, если профиль пользователя доступен, ключи хранятся в папке по умолчанию (%LOCALAPPDATA%\ASP.NET\DataProtection-Keys) и защищаются с помощью Data Protection API, чтобы их мог прочитать только текущий пользователь или процесс.

По умолчанию, даже если два приложения используют общее хранилище ключей, они **изолированы** на основе их пути к контенту. Это предотвращает расшифровку данных одного приложения другим, даже если у них один и тот же ключ.

Хранение ключей по умолчанию подходит только для **среды разработки** или для приложения, работающего на **одном сервере**. Оно не подходит для сред с балансировкой нагрузки (web farms), Docker-контейнеров или облачных сервисов (Azure App Service и т.д.). В этих случаях необходима настройка хранения ключей в централизованном расположении.

Можно использовать **общее файловое хранилище** (services.AddDataProtection() .PersistKeysToFileSystem(newDirectoryInfo(@"\\server\share\directory\"));), пакеты расширения Azure Blob Storage или Redis для хранения ключей в **облачных службах**, хранение ключей в **базах данных** (services.AddDataProtection().PersistKeysToDbContext<MyDbContext> ();).

В производственной среде необходимо защитить ключи, хранящиеся в общем хранилище, от несанкционированного доступа. Например, использовать **шифрование с помощью другого ключа или защиту с помощью сертификата** (services.AddDataProtection() .PersistKeysToFileSystem(...).ProtectKeysWithCertificate(thumbprint);).

Используя интерфейсы **IDataProtector** и **IDataProtectionProvider** можно зашифровать/расшифровать собственные конфиденциальные данные.

Для получения протектора необходимо внедрить IDataProtectionProvider (через Dependency Injection) и вызывать метод CreateProtector() с уникальной строкой назначения. Критически важно использовать уникальную строку для каждого типа защищаемых данных. Это обеспечивает изоляцию: протектор, созданный с целью «A», не сможет расшифровать данные, защищённые протектором с целью «B». Далее можно производить защиту и расшифровку данных с помощью методов Protect и Unprotect (string protectedData = _protector.Protect(«Sensitive Info»); string originalData = _protector.Unprotect(protectedData);).

Самостоятельное использование **Protect** и **Unprotect** — это мощный инструмент, который можно использовать, когда вам нужно отправить данные за пределы вашего доверенного кода, а затем получить их обратно, будучи уверенным, что они не были просмотрены или изменены.

В рамках работы над собственным приложением данный подход был использован мною для **обеспечения безопасности параметров в URL-адресах** и сокрытия конфиденциальных идентификаторов (например, ID пользователя или заказа) в URL вместо передачи идентификатора в открытом виде (например, /invoices/view/123).

С помощью метода Protect (созданного с уникальной строкой назначения, например, «Invoices.View») идентификатор можно преобразовывается в непрозрачную, криптографически защищенную строку (например, «CfDJ8A...ZpA») и эту строка используется для формирования URL (/invoices/view/{protectedId}).

При обработке входящего запроса, в контроллере выполняется обратная операция Unprotect. Если защищенная строка была изменена, метод Unprotect сгенерирует исключение CryptographicException, что позволяет пресечь попытку несанкционированного доступа.

Механизм также можно использовать для **создания безопасных одноразовых токенов**, таких как ссылки для сброса пароля или подтверждения адреса электронной почты, чтобы гарантировать подлинность запроса при возвращении пользователя в приложение.

Вместо хранения токена в базе данных, формируется полезная нагрузка (payload), содержащая необходимую информацию (например, ID пользователя и метку времени истечения срока действия). Эта строка защищается методом Protect (с соответствующей строкой назначения, например, «Auth.PasswordReset»). Полученный токен включается в URL, отправляемый пользователю.

При переходе пользователя по ссылке, приложение извлекает токен, выполняет операцию Unprotect для восстановления исходной полезной нагрузки и проверки ее целостности. Затем приложение анализирует данные, проверяя, например, срок действия токена, перед предоставлением доступа к функции.

Заключение

Обеспечение безопасности передачи данных является неотъемлемой частью разработки современных веб-приложений. Риски, связанные с перехватом и модификацией информации, требуют от разработчиков применения надежных криптографических методов. Платформа ASP.NET Core предоставляет мощный и удобный инструментарий для решения этих задач. Использование протокола TLS для защиты транспортного уровня и встроенной системы защиты данных (Data Protection API) для шифрования информации на уровне приложения позволяет создавать по-настоящему безопасные и надежные веб-системы, защищающие конфиденциальность пользователей и целостность их данных.

Литература

1. *Никифоров С.* Методы защиты информации. Шифрование данных : учеб. пособие / С. Никифоров. – Санкт-Петербург : Лань, 2018. – 160 с.

2. *Прайс М.* C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Прайс. – 6-е изд. – Санкт-Петербург : Питер, 2025. – 848 с. – ISBN 978-5-4461-2249-3.

3. Введение в TLS: обзор принципов и основных характеристик // Yandex Cloud: [сайт]. – 2025. – URL: https://yandex.cloud/ru/docs/glossary/tls (дата обращения: 12.10.2025).

4. ASP.NET Core Data Protection // Microsoft Docs: [сайт] – 2024. – URL: https://learn.microsoft.com/en-us/aspnet/core/security/data-protection/introduction (дата обращения: 12.10.2025).

5. Common security threats in modern day computing // Microsoft Docs: [сайт] – 2024. – URL: https://learn.microsoft.com/en-us/skypeforbusiness/plan-your-deployment/security/common-threats (дата обращения: 12.10.2025).

УДК 004.421

ИСПОЛЬЗОВАНИЕ ДЕРЕВЬЕВ КВАДРАНТОВ ДЛЯ ИНДЕКСАЦИИ КАРТОГРАФИЧЕСКИХ ДАННЫХ

А. А. Золин

Воронежский государственный университет

Аннотация. В работе рассматриваются общие подходы к индексации двумерного пространства, а также их применение для решения задачи индексации картографических данных. В теоретической части дано общее введение в проблему быстрого пространственного поиска по коллекции объектов. Основное внимание уделено дереву квадрантов — древовидной структуре данных для организации коллекции выравненных по осям координат прямоугольников. Были рассмотрены его внутренняя структура, а также алгоритмы поиска и вставки. Приведена реализация системы хранения и индексации географических данных на языке программирования Java.
Ключевые слова: индексация, дерево квадрантов, дерево поиска, индексация в двумерном пространстве, индексация картографических данных, пространственный поиск, поисковое окно, разбиение пространства, проекция Меркатора, OpenStreetMap.

Введение

Выполнение над коллекцией объектов поисковых запросов, которые выбирают объекты на основании их формы и положения в пространстве, является одной из ключевых задач в геоинформационных системах и других предметных областях, где важно обрабатывать пространственные отношения объектов.

При больших объёмах хранимой информации особенно актуальна проблема индексации данных, над которыми будет осуществляться поиск: последовательный просмотр всей коллекции делает выполнение запросов слишком медленным. Специализированные структуры данных, такие как деревья пространственного индексирования, позволяют существенно сократить время поиска, исключая из рассмотрения области, заведомо не удовлетворяющие запросу.

Целью работы является исследование существующих методов индексации двумерного пространства. Рассматривается частная задача представления, хранения и индексации картографических данных.

1. Индексация в двумерном пространстве

В общем случае задача поиска в двумерном пространстве может быть сформулирована следующим образом: в системе хранится коллекция произвольных геометрических фигур, среди которых необходимо выделить такие фигуры, которые соответствуют заданным критериям. Критерии поиска могут быть различными: поиск всех объектов, содержащих заданную точку; поиск всех объектов, содержащихся в заданном подпространстве или имеющих с ним пересечение; поиск всех пар пересекающихся объектов.

Для ускорения поиска объекты дополнительно сохраняются в пространственном индексе. Стоит отметить, что нет необходимости хранить в индексе точную форму объекта. Достаточно хранить ссылку на сам объект и некое приближение его формы, например, минимальных охватывающий прямоугольник. В таком случае, поисковые запросы будут выполнять быстрые тесты на пересечение прямоугольников, и их результатом будет список кандидатов, которые потенциально могут удовлетворять условию из запроса. Дале будет производиться дополнительное тестирование на точное пересечение средствами аналитическое геометрии.

Существует множество способов организации пространственного индекса. В [1] описан один из простейших, но тем не менее эффективных, подходов, который заключается в наложении на пространство равномерной решётки. Решётка делит всё пространство на регионы одинакового размера, называемые ячейками. Затем каждый объект ассоциируется с ячейками, с которыми он пересекается. При поисковом запросе достаточно тестировать только те объекты, которые принадлежат ячейкам, пересекающимся с поисковым окном.

Более сложной структурой для пространственного индекса являются деревья квадрантов. Термин дерево квадрантов (*quadtree*) используется для обозначения целого класса иерархических структур данных, общим свойством которых является то, что они основываются на принципе рекурсивного разбиения пространства на четыре части. Впервые дерево квадрантов было описано в [2] как структура данных для хранения точек на плоскости. В таком дереве точки хранятся в узлах, каждый из которых имеет четыре дочерних. Корень дерева делит индексируемую область на четыре квадранта, каждому из которых соответствует дочерний узел. Область, занимаемая квадрантами, ограничивается двумя прямыми, которые проходят через хранящуюся в узле точку и параллельны осям координат. Далее каждый дочерний узел также делит ассоциированную с ним область на четыре части, и таким образом происходит рекурсивное разбиение всего пространства.

Стоит отметить, что при хранении объектов на внешнем накопителе требуется дополнительная оптимизация доступа к страницам памяти при выполнении поисковых запросов. Такими свойствами обладает R-дерево, описанное в [3]. Это сильно ветвистое дерево, в котором используются алгоритмы, подобные тем, что применяются в B-дереве.

2. MX-CIF Quadtree

Рассмотрим задачу индексации картографических данных, которые будут храниться в оперативной памяти. В индексе для каждого объекта на карте будем сохранять его минимальный охватывающий прямоугольник. Основным поисковым запросом будет нахождение всех объектов на карте, которые находятся в заданной области. Удобной структурой данных для такого сценария использования является разновидность дерева квадрантов *MX-CIF Quadtree*, определение которой дано в [4].

MX-CIF Quadtree — это дерево, которое хранит в себе прямоугольники со сторонами, параллельными осям координат. Каждый узел соответствует определённой области индексируемого пространства, корень дерева покрывает всё пространство целиком; дочерние узлы делят область родителя на четыре равные части, называемые квадрантами. Каждый прямоугольник R ассоциируется с узлом дерева, который соответствует наименьшей области, которая полностью содержит R. Пример MX-CIF Quadtree представлен на рис. 1.

Прямоугольники могут храниться как в листовых, так и в промежуточных узлах. Разбиение на квадранты прекращается, когда область текущего узла не содержит прямоугольников. Также разбиение пространства может быть остановлено, как только область текущего узла становится меньше некого порогового значения, или когда дерево достигает заранее заданной максимальной высоты. Часто размер минимального допустимого квадранта выбирают приблизительно равным ожидаемому размеру хранимых прямоугольников. Как только прямоугольник сохраняется в узле, считается, что он не принадлежит всем потомкам данного узла. Горизонтальная и вертикальная прямые, проходящие через центр квадранта, называются его осями. Альтернативно можно считать, что прямоугольник принадлежит определённому квадранту, если он пересекает одну из его осей.

Из описания MX-CIF Quadtree следует, что более чем один прямоугольник может быть ассоциирован с данным квадрантом. В [4] описан подход для реализации их хранения, в котором используется одномерный аналог дерева квадрантов для организации прямоугольников

внутри одного узла. Пусть P — это узел дерева квадрантов, а S — набор принадлежащих ему прямоугольников. Также будем считать, что область, занимаемая P, имеет размеры $2 \cdot LX \times 2 \cdot LY$, а её центр расположен в точке (CX, CY). Все члены S, которые пересекают прямую $x = CX$ формируют один набор, и все члены S, которые пересекают прямую $y = CY$ формируют другой набор. Иначе говоря, эти наборы соответствуют прямоугольникам, пересекающим оси x и y, проходящие через (CX, CY). Если прямоугольник пересекает обе оси (содержит центр квадранта), то считается, что он ассоциирован с осью y.

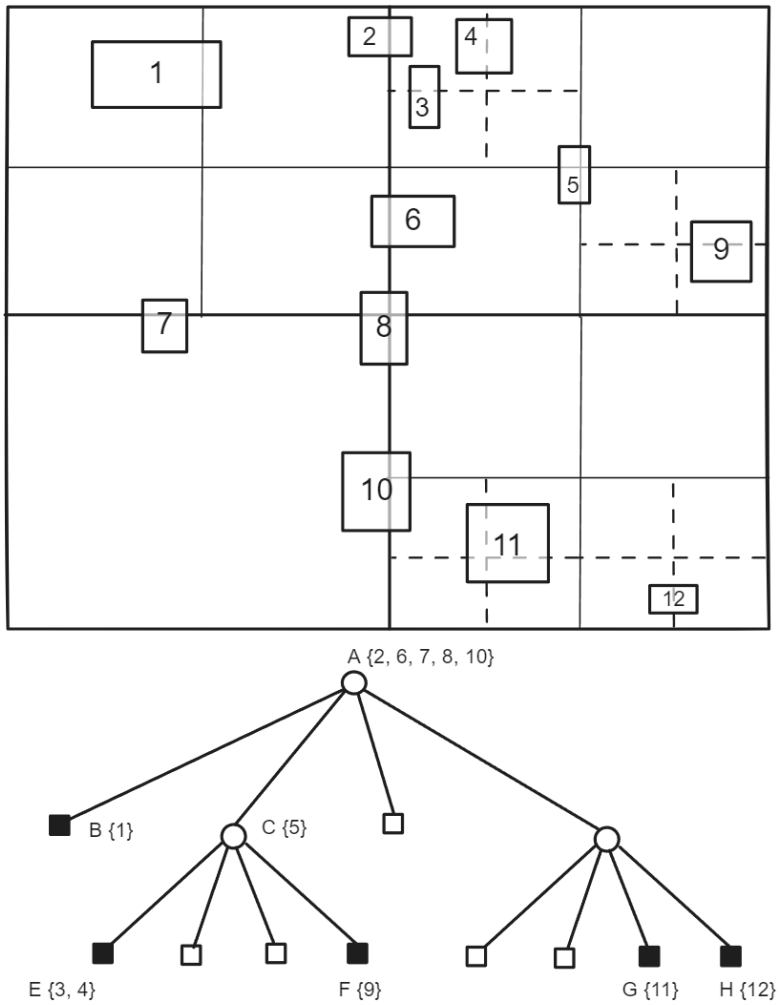

Рис. 1. MX-CIF Quadtree

Эти наборы реализуются как бинарные деревья, рекурсивно разбивающие ось квадранта пополам. Каждый узел отвечает за часть оси, и прямоугольник считается ассоциированным с данным узлом, если он полностью содержится в области этого узла. Нетрудно заметить, что такое дерево является одномерным аналогом описываемого MX-CIF Quadtree. Рис. 2 показывает, как прямоугольники хранятся в бинарном дереве, соответствующем оси x.

При вставке прямоугольника в MX-CIF Quadtree необходимо определить нужный квадрант и позицию в бинарном дереве этого квадранта. Для этого происходит спуск от корневого узла к листьям дерева. Этот поиск можно разделить на два основных этапа. На первом этапе необходимо определить первый квадрант, начиная с корня дерева, который имеет хотя бы одну ось, пересекающую границы вставляемого прямоугольника. После того как подходящий квадрант найден, начинается второй этап — рекурсивное деление выбранной оси квадранта пополам. Этот процесс продолжается до тех пор, пока не будет обнаружена точка разделения,

которая находится внутри границ вставляемого прямоугольника. Эта точка будет соответствовать узлу бинарного дерева, в который и будет вставлен прямоугольник.

Поиск всех прямоугольников, находящихся в заданной области, также начинается с корня дерева. Каждый прямоугольник, хранящийся в текущем узле, тестируется на пересечение с поисковым окном, и если пересечение найдено, то он добавляется в результирующий набор. Далее происходит рекурсивный спуск вниз по дереву. Оптимизация поиска достигается за счёт того, что перед тем, как перейти к обработке очередного узла, происходит проверка на пересечение соответствующего квадранта с поисковым окном. Если пересечения нет, поддерево полностью исключается из рассмотрения. Чтобы внутри конкретного узла определить прямоугольники, удовлетворяющие поисковому запросу, нужно применить аналогичный алгоритм к двум бинарным деревьям этого узла.

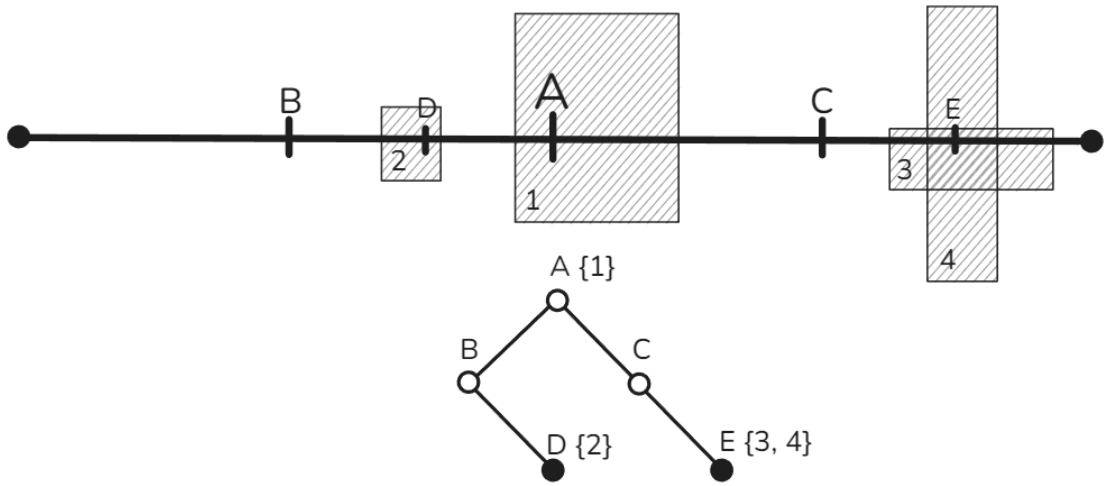

Рис. 2. Бинарное дерево прямоугольников

Построение MX-CIF Quadtree максимальной глубины n для N прямоугольников в худшем случае имеет время выполнения $O(n \cdot N)$. Эта ситуация возникает, когда каждый прямоугольник помещается в узел глубиной n. При этом, в большинстве случаев ожидаемое время работы будет лучше.

3. Хранение картографических данных

Для исследования и тестирования созданного решения важно использовать реальные картографические данные. Для этого можно воспользоваться открытыми географическими базами данных. Один из таких проектов — это OpenStreetMap (OSM). Внутри OSM все географические данные представлены с помощью трёх базовых элементов: точек, линий и отношений. Дополнительно каждый элемент может иметь один или несколько тегов.

Точки являются наименьшим элементом для представления данных в OSM. Они задают конкретное местоположение на земном шаре b определяются с помощью широты и долготы. Лини используются для описания линейных объектов на карте. Так, с помощью линий можно представить дорогу, границы здания или реку. Внутри OSM линии представляются в виде списка точек размером от 2 до 2000, который задаёт ломаную линию. В зависимости от того, совпадает ли начальная точка линии с конечной, их делят на открытые и замкнутые. Отношения описывают логические связи между объектами.

Рассмотренная структура данных OSM представляет координаты объектов через широту и долготу, т. е. задаёт их положение на поверхности земного шара. Для использования этих данных в методах индексирования, работающих с объектами на плоскости, необходимо приме-

нить к данным из OSM картографическую проекцию. Одна из самых часто используемых картографических проекций — проекция Меркатора. По заданным широте θ и долготе φ в проекции Меркатора используются следующие формулы для получения координат x, y на плоскости [5]:

$$\begin{cases} x = \varphi \\ y = \log(\tan\theta + \sec\theta) = \log\left(\tan\dfrac{\theta}{2} + \dfrac{\pi}{4}\right). \end{cases}$$

Процесс построения проекции Меркатора можно представить, как оборачивание цилиндра вокруг земного шара. При этом две поверхности будут касаться друг друга по линии экватора. Далее поверхность Земли проецируется на цилиндр таким образом, чтобы углы между кривыми сохранялись. После этого цилиндр разворачивается в плоскость, которая и будет являться картой.

4. Реализация

Для реализации рассмотренного выше подхода к хранению и индексации картографической информации был выбран язык программирования Java. В [4] алгоритмы вставки и поиска для MX-CIF Quadtree используют предположение, что все хранимые прямоугольники не пересекаются. Чтобы обойти это ограничение в предложенной реализации внутри каждого узла бинарного дерева дополнительно хранится список всех принадлежащих ему прямоугольников. Также была добавлена возможность ограничивать максимальную глубину дерева. Это ограничение применяется при вставке во время принятия решения о расщеплении текущего квадранта.

Определение класса дерева квадрантов приведено ниже:

```
public final class MxCifQuadtree {
    private final Rectangle area;
    private final Node root;
    /* … */
}
```

В нём хранится корень дерева и область, на которой происходит индексирование. Стоит отметить, что в самих узлах информация о занимаемой ими области не сохраняется. Вместо этого она каждый раз вычисляется по мере спуска по дереву. Узел имеет следующую структуру:

```
private static final class Node {
    private final EnumMap<QuadrantRegion, Node> quadrants
        = new EnumMap<>(QuadrantRegion.class);
    private final RectangleBinaryTree xAxisBinaryTree
        = new RectangleBinaryTree(Axis.X);
    private final RectangleBinaryTree yAxisBinaryTree
        = new RectangleBinaryTree(Axis.Y);
    private final int maxDepth;
    /* … */
}
```

В нём содержатся дочерние узлы для каждого из четырёх его квадрантов, два бинарных дерева для каждой оси, в которых хранятся прямоугольники, а также значение максимальной глубины дерева. Ниже приведены алгоритмы вставки и поиска для этого дерева. Поисковой запрос возвращает список всех прямоугольников, которые содержатся или пересекаются с поисковым окном, заданным параметром window:

```
public void insert(Rectangle rectangle, Rectangle area, int depth) {
    var positionX = xAxisBinaryTree.determineRectanglePosition(
        rectangle, area.center()
    );
    var positionY = yAxisBinaryTree.determineRectanglePosition(
        rectangle, area.center()
    );
    if (depth < maxDepth && positionX != AxisPosition.CENTER
        && positionY != AxisPosition.CENTER) {
        var region = determineContainingQuadrant(rectangle,area.center());
        getChild(region).insert(rectangle,getChildArea(region,area),depth+1);
        return;
    }
    if (positionX == AxisPosition.CENTER) {
        yAxisBinaryTree.insert(rectangle, area);
    } else {
        xAxisBinaryTree.insert(rectangle, area);
    }
}

public List<Rectangle> query(Rectangle window, Rectangle area) {
    if (!area.intersects(window)) {
        return Collections.emptyList();
    }
    var xAxisResult = xAxisBinaryTree.query(window, area);
    var yAxisResult = yAxisBinaryTree.query(window, area);
    var childrenResult = quadrants.entrySet().stream()
            .map(e -> {
                var c = e.getValue();
                return c.query(window,getChildArea(e.getKey(), area));
            })
            .flatMap(List::stream)
            .toList();
    return Stream.of(xAxisResult, yAxisResult, childrenResult)
            .flatMap(List::stream)
            .toList();
}
```

Класс RectangleBinaryTree, как уже было отмечено ранее, является одномерным аналогом MX-CIF Quadtree, поэтому его реализация аналогична реализации для двумерного пространства, которая приведена выше:

```
public final class RectangleBinaryTree {
    private RectangleBinaryTree left;
    private RectangleBinaryTree right;
    private final Axis axis;
    private final List<Rectangle> rectangles = new ArrayList<>();
    /* … */
}
```

В каждом узле бинарного дерева содержатся ссылки на правый и левый дочерние узлы, а также список прямоугольников. В алгоритме вставки в бинарное дерево определяется положение прямоугольника относительно центра отрезка, который ассоциирован с узлом дерева. Если центр содержится в прямоугольнике, то он сохраняется внутри узла. В противном слу-

чае — происходит рекурсивный спуск по дочерним узлам, до тех пор, пока не будет найден нужный отрезок. В алгоритме поиска все прямоугольники узла проверяются на пересечение с поисковым окном. Те, которые имеют пересечение, добавляются в результирующий набор. Далее он объединяется с результатами поиска либо в левом, либо в правом поддереве.

Данные из OSM представлены набором следующих классов:

```
private record Node(double x, double y) { }

private record Way(List<Long> nodes) { }

private record Map(
        List<Node> nodes,
        List<Node> standaloneNodes,
        List<Way> ways,
        Rectangle bounds
) { }
```

Для каждой точки хранятся её спроецированные координаты. Линия определяется списком идентификаторов составляющих её точек. Класс Map хранит информацию о всех объектах на карте, в данном случае отношения из OSM не рассматриваются, так как они описывают только логическую связь нескольких объектов. Для сохранения линии в индекс определяются её крайние точки, по координатам которых строится минимальный охватывающий прямоугольник. При импортировании данных карты дополнительно определяются точки, которые не являются частью ни одной линии. Они представляют отдельно стоящие объекты на карте, отметки и т. д. Чтобы сохранить такие точки в индекс, искусственно вводятся квадраты с центром в этих точках и сторонами малого размера. Для хранения в индексе информации о разнотипных объектах используется механизм наследования:

```
public final class NodeRectangle extends Rectangle {
    private final Long nodeId;
    /* … */
}

public final class WayRectangle extends Rectangle {
    private final Long wayId;
    /* … */
}
```

Заключение

В работе были рассмотрены ключевые подходы к индексации двумерного пространства, особое внимание было уделено одной из вариаций дерева квадрантов — MX-CIF Quadtree. Анализ структуры и принципов работы данного дерева продемонстрировал его пригодность для решения задачи индексации картографических данных. На примере реализованной на языке Java системы хранения и индексации географических данных было показано, как рассмотренные теоретические принципы могут быть применены на практике. Предложенная реализация может служить основой для дальнейшего развития систем геоинформационной обработки, включая интеграцию дополнительных пространственных структур и поддержку более сложных поисковых запросов.

Литература

1. *Ericson C.* Real-Time Collision Detection / C. Ericson. – Boca Raton, FL, United States : CRC Press, Inc., 2004. – 632 с.

2. *Finkel R. A.* Quad trees a data structure for retrieval on composite keys / R. A. Finkel, J. L. Bentley // Acta Informatica. – 1974. – Vol. 4, No 1. – P. 1–9. – DOI 10.1007/bf00288933. – EDN HYHQMW.

3. *Guttman A.* R-trees: a dynamic index structure for spatial searching / A Guttman // ACM SIGMOD Record. – 1984. – Т. 14, № 2. – С. 47–57.

4. *Samet H.* The design and analysis of spatial data structures / H. Samet. – Boston, MA, United States : Addison-Wesley Longman Publishing Co., Inc., 1990. – 493 с.

5. Mercator Projection // Wolfram MathWorld : сайт. – URL: https://mathworld.wolfram.com/MercatorProjection.html (дата обращения: 10.11.2025)

УДК 004.4’2

АНАЛИЗ ВЛИЯНИЯ KEY-VALUE ХРАНИЛИЩ НА ПРОИЗВОДИТЕЛЬНОСТЬ И ЭНЕРГОПОТРЕБЛЕНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

М. А. Зотьева

Воронежский государственный университет

Аннотация. В статье рассматривается влияние key-value хранилищ на две характеристики мобильных приложений: производительность и энергопотребление. Исследованы механизмы, посредством которых операции хранения данных влияют на общую эффективность мобильного приложения. Представлены рекомендации по оптимизации производительности и энергопотребления. Результаты исследования показывают, что грамотный выбор стратегии хранения данных и правильное использование key-value хранилищ могут значительно улучшить производительность и энергопотребление мобильных приложений.
Ключевые слова: key-value хранилище, данные, операции с данными, мобильные приложения, производительность, энергопотребление, оптимизация, хранение данных, SharedPreferences, SQLite, Realm, UserDefaults.

Введение

В современном мире мобильные приложения стали неотъемлемой частью повседневной жизни миллионов пользователей. С увеличением объемов обрабатываемых данных, вопросы производительности и энергопотребления становятся критически важными. Одним из ключевых аспектов, определяющих эффективность мобильного приложения, является выбор стратегии хранения данных.

Key-value хранилища представляют собой один из популярных подходов к локальному хранению данных в мобильных приложениях. В отличие от реляционных баз данных, они предлагают модель данных, основанную на парах ключ-значение, благодаря этому удаётся достичь высокой скорости доступа к данным.

В данной статье будут рассмотрены два основных аспекта влияния key-value хранилищ на мобильные приложения: производительность и энергопотребление, а также несколько наиболее типичных способов оптимизации этих показателей.

1. Влияние key-value хранилищ на производительность мобильных приложений

1.1. Роль хранения данных для производительности приложений

Производительность мобильного приложения определяется множеством факторов, среди которых операции с локальным хранилищем данных занимают особое место. Мобильные приложения часто выполняют значительное количество операций чтения и записи данных локально. Это связано с необходимостью кэширования данных, сохранения пользовательских настроек, работы в офлайн-режиме и обеспечения быстрого доступа к часто используемой информации.

Key-value хранилища, будучи одним из основных механизмов локального хранения, напрямую влияют на производительность через несколько каналов. Во-первых, скорость операций чтения определяет время отклика интерфейса при загрузке данных. Во-вторых, операции записи могут блокировать основной поток выполнения, вызывая задержки в пользовательском интерфейсе. В-третьих, объем хранимых данных и эффективность индексации влияют на время поиска и извлечения значений.

Таким образом, можно выделить основные метрики производительности, на которые может повлиять способ хранения данных:

1. Время чтения: среднее и максимальное время доступа к значению по ключу.
2. Время записи: длительность операций сохранения данных, включая синхронизацию.
3. Пропускная способность: количество операций в секунду при различных нагрузках.
4. Масштабируемость: изменение производительности при увеличении объема данных.

Важно помнить, что каждое key-value хранилище имеет свои преимущества и недостатки, в том числе и при влиянии на показатели производительности.

Рассмотрим примеры влияния современных key-value хранилищ на показатели производительности. SharedPreferences в Android оптимизирован для хранения небольших объемов простых данных и обеспечивает очень быстрый доступ к данным, однако становится неэффективным при работе с большими объемами данных или частыми операциями записи [1]. SQLite, в качестве key-value хранилища обеспечивает более высокую производительность при работе с большими наборами данных благодаря более эффективной индексации и оптимизации запросов, по сравнению с SharedPreferences, однако на данных более 1 ГБ могут наблюдаться проблемы [2].

В тестах на мобильном устройстве Android, при работе с небольшими данными (записи до 1 КБ) SharedPreferences обеспечивает время чтения порядка 0,1–0,5 мс на операцию, в то время как SQLite требует 1-3 мс для аналогичных операций. При увеличении объема данных до 100 КБ ситуация меняется: SharedPreferences демонстрирует время чтения 5–15 мс, а SQLite — 2–5 мс благодаря эффективной индексации.

При операциях записи различия еще более заметны: для пакетной записи 100 единиц данных SharedPreferences может требовать 50–100 мс, в то время как SQLite с оптимизированными транзакциями — 20–40 мс

Следует отметить, что влиять на производительность можно не только через способ хранения. Операции с локальным хранилищем данных, как правило, составляют 10–30 % от общего времени выполнения типичного мобильного приложения. Для сравнения, сетевые операции (HTTP-запросы, синхронизация с сервером) могут занимать 30–50 % времени выполнения, особенно при работе в условиях нестабильного интернет-соединения. Графические операции и рендеринг пользовательского интерфейса также оказывают значительное влияние на производительность, составляя 20-40% времени выполнения. CPU-интенсивные вычисления (обработка изображений, криптографические операции) могут занимать 15–25 % времени.

Таким образом, key-value хранилища являются важным фактором производительности, нет единственного лучшего инструмента хранения данных. При выборе инструмента важно понимать особенности разрабатываемого приложения и выбирать, то хранилище, которое будет давать наилучший результат при определённых особенностях продукта. Однако на производительность могут влиять и другие компоненты приложения, поэтому оптимизация должна проводиться комплексно с учетом всех факторов.

1.2. Способы оптимизации производительности

Оптимизация производительности key-value хранилищ требует комплексного подхода, так как не все способы могут быть возможны для приложения.

Одним из наиболее эффективных методов является минимизация размера хранимых данных, так как небольшие объёмы обрабатываются значительно быстрее. При разработке следует избегать хранения избыточной, часто используемой информации и использовать эффективные форматы сериализации. Например, использование бинарных форматов вместо JSON может сократить размер данных на 30–50 % и уменьшить время сериализации на 40–60 %. Для приложения, выполняющего 1000 операций чтения в минуту такая оптимизация может

сократить общее время обработки данных с 50 мс до 20–30 мс, что составляет улучшение производительности на 40–60 % [3].

Ещё одним важным механизмом оптимизации является кэширование часто используемых данных в памяти. Многие key-value хранилища предоставляют встроенные механизмы кэширования, но при необходимости можно реализовать дополнительные уровни кэширования для критически важных данных. Данный подход может ускорить повторные чтения в 10–100 раз по сравнению с чтением из постоянного хранилища. Например, если операция чтения из SharedPreferences занимает 0,5 мс, то чтение из кэша в памяти — 0,01–0,05 мс. Для приложения с высокой частотой повторных обращений к одним и тем же данным (например, настройки пользователя) кэширование может снизить время отклика с 10–15 мс до 1–2 мс, что составляет улучшение на 80–90 %.

Для того, чтобы не делать множество последовательных команд часто используется пакетная обработка операций записи. Такой подход позволяет значительно улучшить производительность, так как, данные накапливаются и записываются одним пакетом, что снижает накладные расходы на синхронизацию и операции ввода-вывода [4]. Пакетная обработка может увеличить скорость записи в 2–3 раза по сравнению с последовательными операциями. Например, запись 100 отдельных записей может занимать 100–150 мс, в то время как пакетная запись тех же 100 записей около 40–60 мс. Это особенно заметно при работе с SQLite, где использование транзакций для пакетной записи может дать улучшение производительности на 150–200 % [5].

Также, использование асинхронных операций для всех операций записи, которые не требуют немедленного подтверждения, позволяет избежать блокировок основного потока, тем самым могут улучшить производительность приложения на 30–50 %, так как пользовательский интерфейс остается отзывчивым во время выполнения операций записи. Это особенно важно для операций, которые могут занимать значительное время, таких как запись больших объемов данных. Для приложения, которое выполняет операции записи, занимающие 20–30 мс, использование асинхронного подхода может полностью устранить задержки в интерфейсе.

Таким образом, для оптимизации производительности можно применять разные подходы или комбинировать их вместе. Выбор конкретного варианта будет зависеть от особенностей разрабатываемого приложения. Например, для небольших объемов простых данных оптимальным выбором могут быть легковесные решения вроде SharedPreferences или UserDefaults, а для больших объёмов, структурированных данных более подходящими могут быть SQLite или специализированные решения вроде Realm.

2. Влияние key-value хранилищ на энергопотребление мобильных приложений

2.1. Понятие энергопотребления в контексте мобильных приложений

Энергопотребление мобильного приложения представляет собой количество энергии, потребляемого устройством при выполнении операций, связанных с этим приложением. В контексте мобильных устройств с ограниченной емкостью батареи, энергопотребление является критически важным параметром, напрямую влияющим на время автономной работы устройства и, следовательно, на пользовательский опыт.

Энергопотребление приложения складывается из нескольких компонентов: использования процессора (CPU), операций ввода-вывода (I/O), работы с памятью, сетевой активности и работе других компонентов устройства (GPS, камера, датчики).

Операции с локальным хранилищем данных, включая key-value хранилища, вносят значительный вклад в общее энергопотребление через несколько механизмов [6].

Измерение может проводиться двумя основными способами. Первый подход: измерение абсолютного падения заряда батареи при выполнении фиксированного набора операций. Например, выполнение 1000 операций записи в key-value хранилище может привести к падению заряда батареи на 0,3–0,7 % в зависимости от типа хранилища и размера данных. Второй подход: измерение скорости падения заряда в единицу времени при активной работе приложения. Например, приложение, активно работающее с локальным хранилищем (100 операций записи в минуту), может расходовать 1,5–2,5 % заряда батареи в час. В данной работе будет использоваться первый вариант. Для точных измерений используются специализированные инструменты: Battery Historian для Android, Instruments для iOS.

Аналогично с производительностью: на показатели энергопотребления влияет не только тип key-value хранилищ, но и другие факторы. Операции с локальным хранилищем данных, как правило, составляют 5–15 % от общего энергопотребления мобильного приложения. Для сравнения, сетевые операции (Wi-Fi, мобильный интернет) являются наиболее энергозатратными и могут составлять 30–50 % от общего энергопотребления, особенно при активной передаче данных. Использование GPS и других датчиков также оказывает значительное влияние, составляя 20–40 % энергопотребления для приложений, активно использующих геолокацию. Графические операции и рендеринг интерфейса могут занимать 15–25 % энергопотребления, особенно при работе со сложными визуальными эффектами.

Таким образом, хотя key-value хранилища вносят заметный вклад в энергопотребление, их влияние меньше, чем у сетевых операций и использования датчиков, но сопоставимо с другими компонентами приложения. Однако оптимизация операций с хранилищем может дать значительный эффект, особенно для приложений, которые активно работают с локальными данными.

2.2. Механизмы влияния хранения данных на энергопотребление

Операции с данными в key-value хранилища влияют на энергопотребление через несколько каналов:

1. Операции ввода-вывода требуют активации контроллера памяти и накопителя (флеш-памяти), что потребляет энергию.

2. Обработка данных требует использования процессора, который является одним из наиболее энергопотребляющих компонентов устройства.

3. Операции с данными могут вызывать пробуждение устройства из режима энергосбережения, что приводит к дополнительному расходу энергии.

4. Каждая операция записи требует синхронизации данных с постоянным хранилищем, что включает в себя активацию контроллера памяти, выполнение операций записи во флэш-память и возможную дефрагментацию. Эти операции являются энергозатратными и могут значительно влиять на общее энергопотребление приложения при частом вызове [7].

5. Запись больших объемов данных требует больше времени и энергии, чем запись небольших порций. Кроме того, операции с большими данными могут требовать больше операций с памятью и процессором для обработки.

Разные реализации key-value хранилищ демонстрируют различия в энергопотреблении. Легковесные решения, такие как SharedPreferences в Android или UserDefaults в iOS, оптимизированы для минимального энергопотребления при работе с небольшими объемами данных, так как используют простые механизмы хранения и минимизируют операции ввода-вывода. Измерения показывают, что для операций с небольшими данными (до 1 КБ) SharedPreferences потребляет примерно 0,1–0,3 мВт/ч на операцию записи, в то время как SQLite требует 0,3–0,6 мВт/ч для аналогичных операций. При работе с объемами данных 100 КБ и более разли-

чия становятся более заметными: SharedPreferences может потреблять 2–4 мВт/ч на операцию, а SQLite 1,5–3 мВт/ч благодаря более эффективным алгоритмам работы с большими данными.

SQLite, будучи более функциональным решением, потребляет больше энергии из-за более сложной структуры данных и дополнительных операций индексации. Однако при правильной настройке и оптимизации запросов, SQLite может быть достаточно эффективным с точки зрения энергопотребления. Использование транзакций для пакетной записи может снизить энергопотребление на 20–30 % по сравнению с отдельными операциями записи.

Realm, может демонстрировать различное энергопотребление в зависимости от паттернов использования. При частых операциях записи они могут потреблять больше энергии из-за сложных механизмов синхронизации и управления памятью. Однако при преимущественно операциях чтения они могут быть более эффективными благодаря оптимизированным алгоритмам доступа к данным. При операциях чтения Realm может потреблять на 15–25 % меньше энергии, чем SQLite, но при операциях записи — на 10–20 % больше.

Количественные оценки показывают, что оптимизация операций записи может снизить энергопотребление на 15–25 %. Минимизация размера данных также дает заметный эффект: сокращение размера записываемых данных на 50 % может снизить энергопотребление на 20–30 % за счет уменьшения количества операций ввода-вывода и времени работы процессора.

2.3. Способы оптимизации энергопотребления

Оптимизация энергопотребления при работе с key-value хранилищами требует внимания к нескольким аспектам. Важно отметить, что многие способы оптимизации аналогичны оптимизации производительности, так как эти два параметры очень взаимосвязаны. Рассмотрим способ, который не был разобран в первой части.

Отложенная запись (lazy writing), часто использующийся механизм. Данные записываются в постоянное хранилище только при необходимости, например, при закрытии приложения или при достижении определенного порога накопленных изменений. Это позволяет избежать ненужных операций записи и снизить энергопотребление.

Все остальные популярные способы аналогичны рассмотренным ранее методам оптимизации производительности: оптимизация размера хранимых данных, использование асинхронных операций, минимизация частоты операций.

Несмотря на то, что многие методы оптимизации одновременно улучшают и производительность, и энергопотребление, существуют ситуации, когда оптимизация одного параметра может негативно влиять на другой. Например, использование кэша, с одной стороны, позволяет снизить количество операций чтения из постоянного хранилища, что должно приводить к снижению энергопотребления, но при этом увеличивает использование оперативной памяти, что, напротив, повышает затраты энергии [7]. В таких случаях необходимо искать компромиссное решение, которое будет оптимальным для конкретного приложения и сценария использования.

Современные мобильные устройства предоставляют различные режимы работы, которые отражают компромисс между производительностью и энергопотреблением на системном уровне. Режим максимальной производительности приоритизирует скорость выполнения операций, используя максимальную частоту процессора и отключая ограничения энергосбережения. В этом режиме key-value хранилища могут работать с максимальной скоростью, но энергопотребление значительно возрастает. Режим экономии энергии снижает частоту процессора, ограничивает фоновые процессы и может замедлять операции с хранилищем для снижения энергопотребления.

Таким образом, производительность и энергопотребление тесно связаны друг с другом, есть как похожие способы их оптимизации, так и различия, между которыми необходимо на-

ходить баланс в процессе разработки. Стратегии поиска компромисса включают мониторинг и динамическую настройку параметров работы с хранилищем. Приложение может отслеживать уровень заряда батареи и автоматически переключаться между стратегиями: при высоком уровне заряда использовать более производительные, но энергозатратные методы, а при низком уровне заряда экономить энергию, снижая производительность. Такой адаптивный подход позволяет максимизировать производительность, когда это возможно, и минимизировать энергопотребление, когда это необходимо, обеспечивая оптимальный пользовательский опыт в различных условиях.

Заключение

Таким образом, проведенный анализ влияния key-value хранилищ на производительность и энергопотребление мобильных приложений позволяет сделать несколько важных выводов.

Во-первых, выбор стратегии хранения данных является критически важным решением, которое напрямую влияет на ключевые характеристики приложения. Способов к подходам оптимизации достаточно много и каждое из хранилищ может лучше работать с разными подходами.

Во-вторых, производительность и энергопотребление тесно взаимосвязаны. Оптимизация одного параметра может влиять на другой, что требует комплексного подхода.

В-третьих, не существует универсального решения, оптимального для всех случаев. Выбор key-value хранилища и способов оптимизации должен основываться на конкретных требованиях приложения: объеме данных, частоте операций чтения и записи и сложности структуры данных. Важно помнить, что подходы можно комбинировать между собой, для получения наилучшего результата.

Литература

1. Android Developers. SharedPreferences: официальный сайт – США, 2025. – URL: https://developer.android.com/reference/android/content/SharedPreferences (дата обращения: 06.11.25).

2. SQLite Documentation: официальный сайт – США, 2025. – URL: https://www.sqlite.org/ (дата обращения: 06.11.25).

3. *Баранов Ю. К.* NoSQL для всех. Как перестать бояться и начать применять / Ю. К. Баранов. – Санкт-Петербург : БХВ-Петербург, 2018. – 304 с.

4. *Ригин А. М.* Расширение СУБД SQLite для индексации данных / А. М. Ригин // Национальный исследовательский университет «Высшая школа экономики», 2020. – 14 с. – URL: https://cyberleninka.ru/article/n/sqlite-rdbms-extension-for-data-indexing-using-b-tree-modifications/viewer (Дата обращения 06.11.25).

5. Key-value для хранения метаданных в СХД. Тестируем встраиваемые базы данных // Блог компании RADIX: [сайт]. – 2017. – URL: https://habr.com/ru/companies/raidix/articles/345076/ (Дата обращения 20.10.25).

6. *Рябцев Я. В.* Разработка базы данных для мобильного приложения управления документами из облачных хранилищ / Я. В. Рябцев // Международный научный журнал «Символ науки» – 2020. – URL: https://cyberleninka.ru/article/n/razrabotka-bazy-dannyh-dlya-mobilnogo-prilozheniya-upravleniya-dokumentami-iz-oblachnyh-hranilisch (Дата обращения 20.10.25).

7. *Гилязова Л. М.* Обзор и анализ методов обеспечения безопасности в различных СУБД / Л. М. Гилязова // Международный научный журнал «Инновационная наука» – 2023. – URL: https://cyberleninka.ru/article/n/obzor-i-analiz-metodov-obespecheniya-bezopasnosti-v-razlichnyh-subd (Дата обращения 15.10.25).

УДК 004.4

ПРОЕКТИРОВАНИЕ СИСТЕМЫ ОТОБРАЖЕНИЯ РАСПИСАНИЯ ЗАНЯТИЙ СТУДЕНТОВ И ЗАНЯТОСТИ ПРЕПОДАВАТЕЛЕЙ НА ПРИМЕРЕ РАСПИСАНИЯ В ВГПУ

Л. А. Иванникова

Воронежский государственный педагогический университет

Аннотация. В статье рассматривается проектирование веб-системы для отображения расписания занятий студентов и занятости преподавателей на примере существующего сервиса электронного расписания Воронежского государственного педагогического университета. Проект возник в связи с ограничением доступа к локальному серверу вуза, что усложнило получение расписания. Для реализации выбран фреймворк Django на Python, обеспечивающий безопасность, масштабируемость и простоту интеграции с данными в формате JSON. Анализируется структура исходных данных, проектируется реляционная база данных с использованием MySQL и Django ORM. Разработан пользовательский интерфейс с формами выбора групп и преподавателей, динамическими таблицами расписания и страницей занятости аудиторий. Проект демонстрирует потенциал для дальнейшего развития, включая фильтрацию по факультетам и сохранение избранных аудиторий.
Ключевые слова: расписание занятий, Django, Python, JSON, база данных, интерфейс, сервиса электронного расписания, сервер, ВГПУ, студенты, преподаватели.

Введение

Начало нового учебного года 2024/2025 началось с изменения политики информационной безопасности относительно сервера физико-математического факультета Воронежского Государственного Педагогического Университета. По ряду причин сервер стал доступен только из локальной сети вуза. На нем функционировало, да и в настоящий момент работает, достаточно большое число сервисов. Одним из наиболее используемых был сервис электронного расписания. Для студентов доступ к расписанию был через личный кабинет, а также с официального сайта вуза в формате электронных таблиц. Для преподавателей же все превратилось в захватывающий квест. Им приходилось сначала скачивать файлы с расписанием тех факультетов и курсов, где они ведут занятия, а затем поиском по нескольким электронным таблицам искать себя, чтобы составить свое личное расписание. С учетом изменений, которые всегда есть в расписании в начале семестра, это становилось достаточно неудобной задачей. Тем более, что в предыдущие годы, используя сервис электронного расписания, можно было просто выбрать свою фамилию и посмотреть его. Все эти возникшие неудобства и сподвигли на идею создания новой системы отображения расписания для студентов и преподавателей.

1. Подходы к решению: выбор фреймворка Django

Для решения проблемы был выбран веб-фреймворк Django на языке программирования Python. Этот выбор обусловлен некоторыми ключевыми преимуществами, которые идеально подходят для проекта подобного типа:

– Готовые инструменты для работы с данными и веб-интерфейсами: Django предоставляет встроенную ORM (Object-Relational Mapping) для взаимодействия с базами данных, что упрощает моделирование сложных структур.

– Безопасность и масштабируемость: Учитывая, что проект предназначен для внешнего доступа в Интернете, Django предлагает встроенные механизмы защиты от распространенных угроз, таких как SQL-инъекции и XSS-атаки. Фреймворк также легко масштабируется, позволяя развернуть приложение на облачных серверах.

– Простота разработки и сообщество: Большое сообщество и обширная документация облегчают решение возникающих проблем. В сравнении с альтернативами, такими как Flask (более минималистичный, но требующий дополнительных настроек), Django оказался оптимальным балансом между функциональностью и удобством.

– Интеграция с данными: Поскольку исходные данные выгружались в формате JSON, Django легко сочетается с библиотеками вроде json для парсинга и импорта, что позволило автоматизировать и обновление расписания.

В целом, выбор Django обеспечил быструю разработку, сосредоточив усилия на логике проекта, а не на инфраструктуре.

2. Анализ структуры выгружаемых данных

Исходные данные с локального сервера вуза выгружались в формате JSON, представляющем собой иерархическую структуру расписания. Анализ показал, что файл содержит массив «timetable», где каждый элемент соответствует неделе семестра. Компоненты структуры выглядят следующим образом:

– Верхний уровень («timetable»): Список объектов, каждый из которых описывает одну неделю.

Поля включают «week_number» (номер недели), «date_start» и «date_end» (диапазон дат в формате «ДД-ММ-ГГГГ»).

– Группы («groups»): Внутри каждой недели — массив групп студентов. Каждая группа имеет поля: «group_name» (название, например, «-_a241»), «course» (курс), «faculty» (факультет), «changes» (количество изменений).

– Дни («days»): Для каждой группы - массив дней недели («weekday» от 1 для понедельника до 6 для субботы). Если для дня нет занятий, массив «lessons» может отсутствовать или быть пустым.

– Занятия («lessons»): Внутри дней - массив занятий. Каждое занятие описывается полями:
«subject» (название дисциплины).
«type» (тип занятия, для лекции или практики).
«subgroup» (подгруппа).
«time_start» и «time_end» (время начала и окончания занятия в формате «ЧЧ:ММ»).
«time» (номер пары, от 1 до 4 или больше).
«week» (номер недели, повторяет верхний уровень).
«date» (конкретная дата дня).
«teachers» (массив преподавателей : «teacher_name» — ФИО, «teacher_id» — идентификатор).
«auditories» (массив аудиторий: «auditory_name» — номер или название).
«Lesson_ID_Num» (уникальный идентификатор урока, например, «2_9_1»).

Структура реляционная: занятия связаны с группами, днями, преподавателями и аудиториями. Для анализа использовались скрипты на Python для парсинга JSON.

3. Проектирование собственной базы данных

На основе анализа была создана независимая реляционная база данных с использованием Django ORM и MySQL.

В целях простоты и быстрой реализации была выбрана ненормализованная структура с одной моделью (таблицей), что позволило избежать сложностей с реляционными связями и сосредоточиться на хранении данных в «плоском» виде.

Основная модель (Schedule): Единственная модель, содержащая все необходимые атрибуты напрямую:

– weekday: День недели (VARCHAR(12)).
– date: Дата занятия (DATE(10)).
– teacher_name: Имя преподавателя(VARCHAR(100).
– subject: Название дисциплины(VARCHAR(120).
– type_subject: Тип занятия(VARCHAR(5).
– auditory_name: Название аудитории(VARCHAR(10).
– time: Время начала и окончания пары(VARCHAR(50).
– group_name: Название группы(VARCHAR(20).
– subgroup: Подгруппа (INT(1)).

Все данные хранятся в одной таблице без foreign keys и отдельных моделей для групп, преподавателей или аудиторий. Это приводит к возможным дубликатам, но упрощает запросы и импорт.

Эта простая структура независима, но синхронизируема, что эффективно решает проблему доступности при минимальных затратах на разработку.

id	weekday	date	teacher_name	subject	type_subject	auditory_name	time	group_name	subgroup
41	1	2025-02-24	Григорьев О.А.	Теория и методика физического воспитания	л.	2-402	09:00-10:35	ФК-ФК_Об231	0
42	1	2025-02-24	Лобачев В.В.	Гимнастика с методикой преподавания	пр.	с.з.1 (1)	10:45-12:20	ФК-ФК_Об231	0
43	1	2025-02-24	Добросоцкая А.В.	Технология и организация воспитательных …	пр.	2-404	12:40-14:15	ФК-ФК_Об231	0
44	1	2025-02-24	Сумин А.В.	Великая Отечественная война: без срока да…	пр.	2-404	14:40-16:15	ФК-ФК_Об231	0
45	2	2025-02-25	Григорьева И.П.	Плавание с методикой преподавания	л.	2-402	09:00-10:35	ФК-ФК_Об231	0
46	2	2025-02-25	Арцыбашев А.А.	Лыжная подготовка с методикой преподава…	пр.	лыж.база	10:45-12:20	ФК-ФК_Об231	0
47	2	2025-02-25	Костина И.Б.	Философия	пр.	4-520	12:40-14:15	ФК-ФК_Об231	0
48	2	2025-02-25	Орлова Г.В.	Учебная практика (технологическая) 'Психо…	пр.	2-403	14:40-16:15	ФК-ФК_Об231	0
49	2	2025-02-25	Коровин Н.Р.	Элективные курсы по физической культуре …	пр.	УСК(1)	18:10-19:45	ФК-ФК_Об231	0
50	3	2025-02-26	Григорьева И.П.	Гимнастика с методикой преподавания	л.	2-405	09:00-10:35	ФК-ФК_Об231	0
51	3	2025-02-26	Фирстова Е.В.	Технология и организация воспитательных …	л.	208	10:45-12:20	ФК-ФК_Об231	0

Рис. 1. Структура таблицы базы данных расписания занятий

4. Построение интерфейса для отображения типичных страниц

Интерфейс разработан с акцентом на простоту и удобство, используя Django templates, HTML/CSS, JavaScript.

Главная страница: Содержит две формы с выпадающими списками (dropdowns). Первая — для выбора группы. Вторая — для преподавателя. После выбора и нажатия кнопки «Показать расписание» генерируется запрос.

Страница расписания: Динамическая таблица, отображающая данные с понедельника текущей недели до последней доступной даты. Столбцы — дата, день недели, время, подгруппы. Ячейки заполняются информацией о занятиях: предмет, тип, преподаватель/группа, аудитория.

Интерфейс минимизирован, чтобы фокус был на данных, без лишних элементов.

Страница занятости аудиторий: Дополнительная страница включает форму выбора аудитории и даты. После нажатия кнопки «Показать» отображается таблица с занятостью выбранной аудитории на указанный день. В ячейках используется обозначение «+» для занятого слота и «–» для свободного. При наведении курсора на ячейку появляется всплывающая подсказка (tooltip), показывающая, кем именно занята аудитория (группа, дисциплина и преподаватель).

5. Перспективы развития проекта

Проект обладает значительным потенциалом для дальнейшего развития, особенно в направлении работы со списками, которая могла бы радикально повысить удобство. Работа со списками может оказаться очень удобной. Добавление фильтрации групп по факультетам. Например, сначала пользователь выбирает факультет из списка, после чего в выпадающем меню отображаются только группы, относящиеся к этому факультету. А также, добавление возможности сохранения любимых аудиторий для пользователя, чтобы ему не приходилось каждый раз выбирать из полного списка. Избранные аудитории можно отображать в отдельном разделе или предлагать первыми в выпадающем меню.

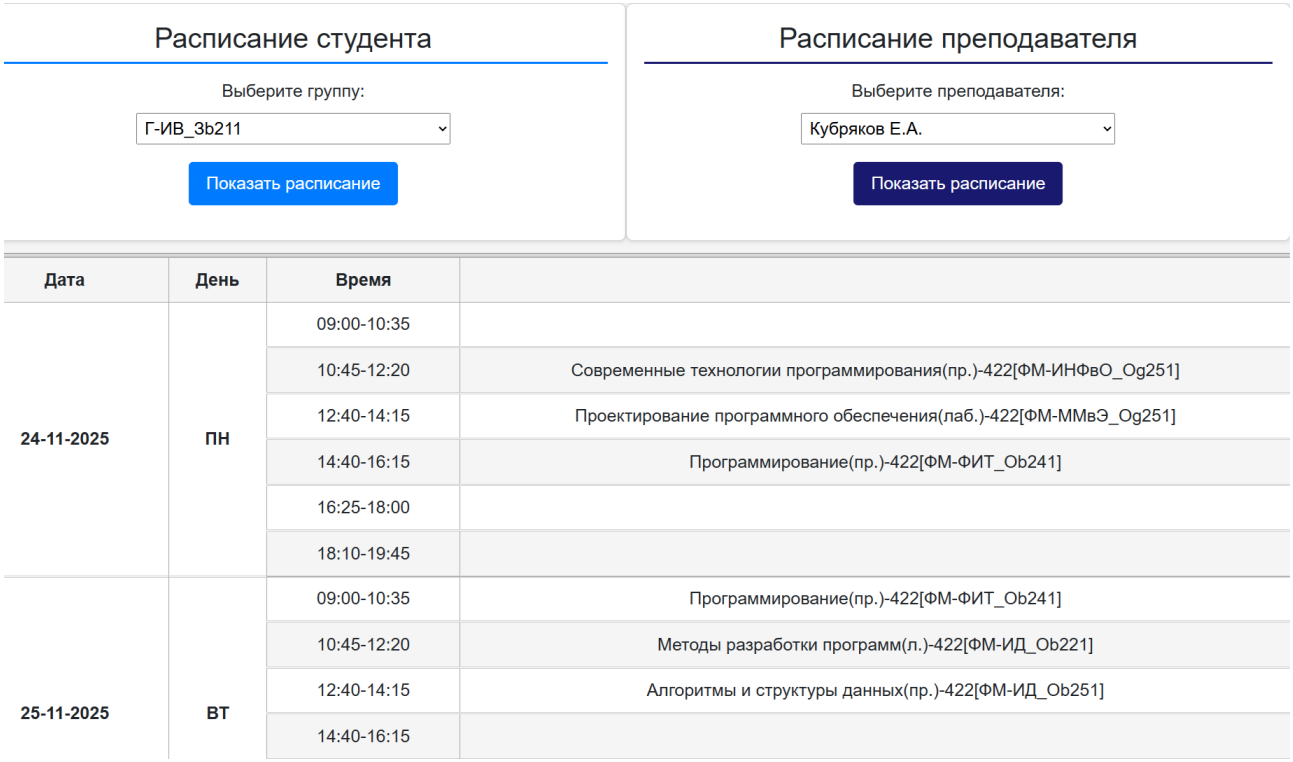

Рис. 2. Основная страница с расписанием

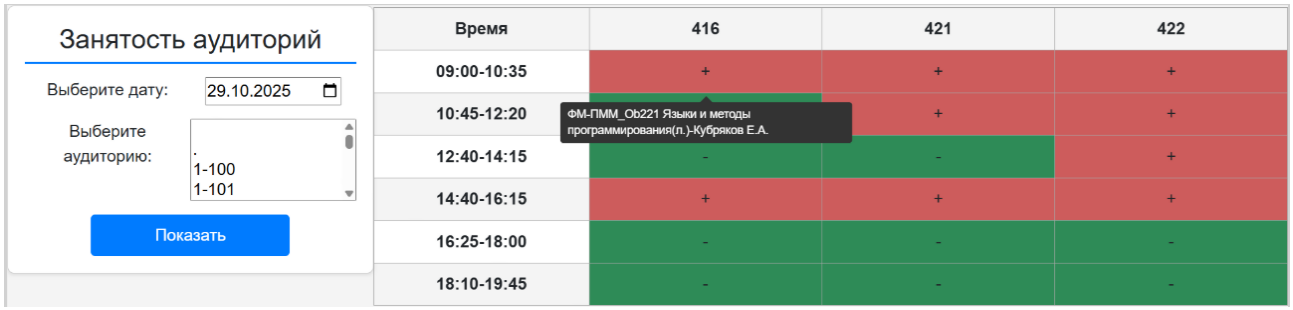

Рис. 3. Страница занятости аудиторий

Заключение

Проект демонстрирует практическую ценность, решая реальные проблемы доступности информации в образовательной среде. Он подчеркивает преимущества Django в контексте веб-разработки для образовательных сервисов, а также открывает возможности для дальнейшего расширения, такого как фильтрация по факультетам и сохранение пользовательских предпочтений. В целом, разработанная система не только восполняет пробел в доступе к расписанию, но и служит основой для эволюции подобных сервисов, способствуя повышению эффективности учебного процесса в вузе. Разработанная система отображения расписания занятий доступна для ознакомления и использования по адресу: https://rsp-vspu.online

Литература

1. Django : официальная документация [Электронный ресурс]. – URL: https://docs.djangoproject.com/en/stable/ (дата обращения: 15.10.2023).

2. *Рубцов В. С.* Современные базы данных : учеб. пособие / В. С. Рубцов. – Москва : Интернет-Университет Информационных Технологий ; БИНОМ. Лаборатория знаний, 2013. – 432 с.

3. *Лутц М.* Изучаем Python : пер. с англ. / М. Лутц. – 4-е изд. – Санкт-Петербург : Символ-Плюс, 2011. – 1280 с.

4. ГОСТ Р 7.0.100-2018. Библиографическая запись. Библиографическое описание. Общие требования и правила составления : национальный стандарт Российской Федерации : дата введения 2019-07-01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2018. – 124 с.

5. *Бек К.* Экстремальное программирование : разработка через тестирование : пер. с англ. / К. Бек. – Санкт-Петербург : Питер, 2003. – 224 с.

6. *Иванов И. И.* Проектирование информационных систем : учеб. пособие / И. И. Иванов. – Москва : Высшая школа, 2009. – 336 с.

7. JSON : официальная спецификация [Электронный ресурс]. – URL: https://www.json.org/json-ru.html (дата обращения: 15.10.2023).

8. *Сидоров А. А.* Веб-разработка на Python с использованием Django / А. А. Сидоров // Информатика и образование. – 2018. – № 3. – С. 45–52.

9. *Петрова Е. В.* Базы данных в образовательных системах : монография / Е. В. Петрова. – Воронеж : Воронежский государственный университет, 2015. – 198 с.

10. MySQL : официальная документация [Электронный ресурс]. – URL: https://dev.mysql.com/doc/ (дата обращения: 15.10.2023).

УДК 004.9

ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ «ПОМОЩНИК ТУРИСТА» ПОД ОПЕРАЦИОННУЮ СИСТЕМУ ANDROID

К. И. Илларионова, Т. В. Курченкова

Воронежский государственный университет

Аннотация. В данной работе рассматривается процесс проектирования мобильного приложения «Помощник туриста», предназначенного для поиска интересных мест, построения маршрутов и получения информации в офлайн-режиме. Приложение спроектировано под операционную систему Android с использованием языка программирования Kotlin, СУБД PostgreSQL и фреймворка Spring Boot. В работе приведены основные этапы проектирования, структура базы данных и ключевые элементы интерфейса пользователя.
Ключевые слова: мобильное приложение, Android, Kotlin, PostgreSQL, Spring Boot, туризм.

Введение

Туризм является одной из наиболее популярных сфер деятельности, способствующих культурному обмену и расширению кругозора. Однако успешное планирование поездки требует значительных временных затрат на поиск информации о достопримечательностях, маршрутах и услугах. Современные мобильные технологии позволяют упростить этот процесс, предоставляя пользователям удобные инструменты для планирования путешествий.

Мобильное приложение «Помощник туриста» предназначено для создания маршрутов, поиска интересных мест, просмотра отзывов. Приложение предусматривает возможность работы в офлайн-режиме. В отличие от существующих решений, данное приложение объединяет функции офлайн-доступа, системы отзывов, интерактивных карт и фильтрации мест и маршрутов.

1. Постановка задачи

Необходимо спроектировать мобильное приложение под операционную систему Android, предоставляющее удобный инструмент для туристов [1]. Приложение должно обеспечивать возможность регистрации и авторизации, поиска и фильтрации мест и маршрутов, отображения интерактивной карты и работы в офлайн-режиме.

2. Функциональность

Основная функциональность спроектированного приложения представлена на диаграмме вариантов использования [2] (рис. 1). Рассмотрим подробно каждый из прецедентов.

Использовать готовый маршрут. Пользователь после регистрации и авторизации может просматривать уже готовые маршруты, созданные другими пользователями или рекомендованные системой.

Просмотреть информацию о месте. Пользователь имеет возможность ознакомиться с описанием интересных мест, их категориями (музеи, кафе, парки, достопримечательности и др.), фотографиями и отзывами других туристов.

Искать место. Приложение предоставляет функцию поиска мест по названию, категории или расположению, а также возможность фильтрации по типу и рейтингу.

Поделиться маршрутом. Созданный маршрут можно сохранить и поделиться им с другими пользователями через встроенные средства обмена.

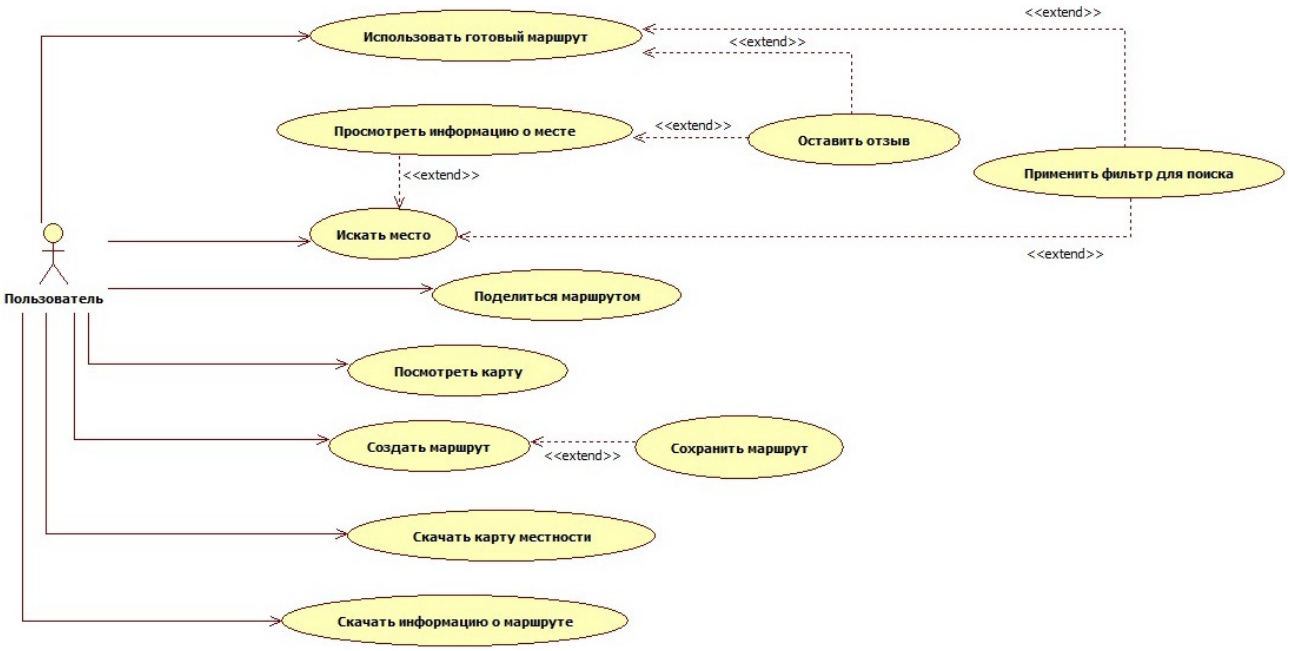

Рис. 1. Диаграмма вариантов использования

Посмотреть карту. Приложение интегрировано с сервисом Яндекс.Карт, что позволяет просматривать интересные места и маршруты на интерактивной карте, а также отслеживать текущее местоположение пользователя.

Создать маршрут. Пользователь может создавать собственные маршруты, добавляя выбранные места, задавая их последовательность и время посещения.

Скачать карту местности. Для работы без доступа к сети Интернет реализована возможность скачивания карт выбранного региона, обеспечивающая навигацию в офлайн-режиме.

Скачать информацию о маршруте. Пользователь может заранее сохранить данные о маршруте, включая список мест, описания и фотографии, чтобы использовать их при отсутствии интернет-соединения.

3. Средства реализации

Для проектирования приложения использовались современные технологии: язык программирования Kotlin, среда разработки Android Studio, СУБД PostgreSQL, фреймворк Spring Boot, язык разметки XML и инструменты моделирования WhiteStarUML и PowerDesigner.

Kotlin был выбран как современный и безопасный язык, поддерживаемый Android Studio. Spring Boot использовался для реализации серверной части, обеспечивающей хранение и обработку данных. PostgreSQL обеспечивает надежное хранение информации о пользователях, местах и маршрутах.

4. Проектирование мобильного приложения

Приложение спроектировано с использованием архитектуры клиент–сервер. Клиентская часть создается на Kotlin [3], а серверная — на Spring Boot. Взаимодействие между ними осуществляется через REST API. Основные функциональные возможности приложения включают регистрацию, авторизацию, поиск и фильтрацию мест, создание маршрутов, добавление отзывов и просмотр интерактивной карты.

Диаграмма классов представлена на рис. 2.

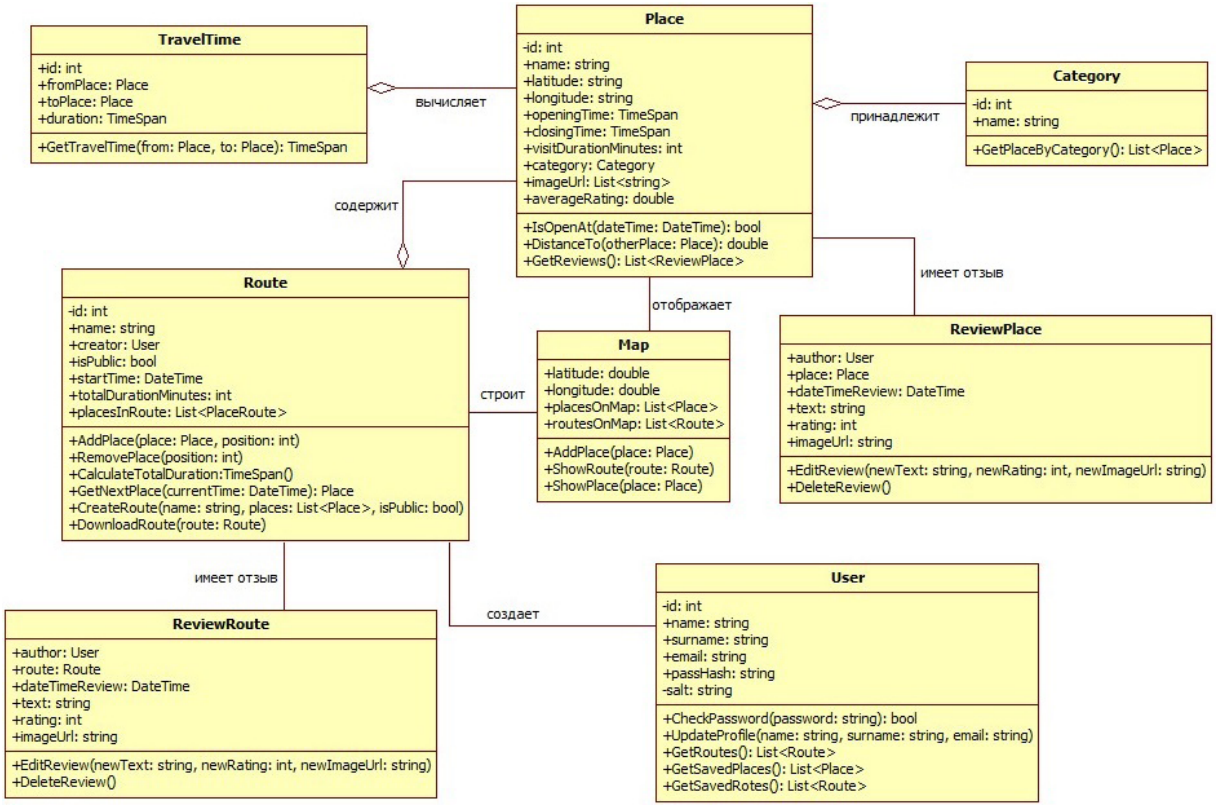

Рис. 2. Диаграмма классов

Основные классы приложения:

1. *User* — хранит сведения о пользователе: имя, фамилию, логин, адрес электронной почты, хэш пароля и соль. Также содержит списки маршрутов и сохранённых мест пользователя.

2. *Route* — представляет маршрут, созданный пользователем. Содержит название, автора, дату начала, общую продолжительность, список мест и флаг публичности.

3. *Place* — описывает место: название, адрес, координаты, время открытия и закрытия, категорию, список тегов и средний рейтинг.

4. *Category* — хранит информацию о категории места.

5. *Map* — отображает места и маршруты на карте, содержит координаты и список доступных точек.

6. *ReviewPlace* — хранит отзыв о месте: автора, место, дату, рейтинг, текст и изображение.

7. *ReviewRoute* — хранит отзыв о маршруте: автора, маршрут, дату, текст, рейтинг и изображение.

8. *TravelTime* — хранит данные о времени перемещения между двумя местами: место отправления, место назначения и длительность пути.

Каждый класс приложения включает ряд методов. Рассмотрим основные методы, их параметры и возвращаемые значения, а также опишем их работу (табл. 1).

Таблица 1

Описание классов приложения

Класс	Метод	Описание
User	`CheckPassword(password: string): bool`	Проверяет соответствие переданного пароля хешу, сохранённому у пользователя, с учётом соли. Возвращает true, если пароль верный.

	UpdateProfile(name: string, surname: string, email: string)	Обновляет личную информацию пользователя: имя, фамилию и адрес электронной почты.
	GetCreatedRoutes():List<Route>	Возвращает список маршрутов, созданных данным пользователем.
	GetSavedPlaces():List<Place>	Возвращает список мест, сохранённых пользователем.
	GetSavedRoutes():List<Route>	Возвращает список маршрутов, сохранённых пользователем.
Route	AddPlace(place: Place, position: int)	Добавляет место вмаршрут на указанную позицию.
	RemovePlace(position: int)	Удаляет место из маршрута по указанной позиции.
	CalculateTotalDuration: TimeSpan()	Вычисляет общую продолжительность маршрута, включая время на дорогу и посещение мест.
	GetNextPlace(currentTime: DateTime): Place	Возвращает следующее место маршрута, которое должно быть посещено, исходя из текущего времени.
	CreateRoute(name: string, places:List<Place>, isPublic: bool)	Создаёт и возвращает новый маршрут с заданными параметрами: названием, списком мест и флагом публичности.
	DownloadRoute(route: Route)	Загружает (сохраняет локально) выбранный маршрут, включая все связанные места, время и последовательность, чтобы использовать оффлайн.
Place	IsOpenAt(dateTime: DateTime): bool	Проверяет, открыто ли место в заданное время.
	DistanceTo(otherPlace: Place):double	Вычисляет расстояние до другого места по координатам.
	GetReviews():List<ReviewPlace>	Возвращает список отзывов о данном месте.
Category	GetPlaceByCategory():List<Place>	Возвращает список мест, относящихся к данной категории.
TravelTime	GetTravelTime(from: Place, to: Place): TimeSpan	Возвращает время в пути между двумя местами.
ReviewPlace	EditReview(newText: string, newRating: int, newImageUrl: string)	Обновляет содержимое отзыва: текст, рейтинг и изображение.
	DeleteReview()	Удаляет отзыв.
ReviewRoute	EditReview(newText: string, newRating: int, newImageUrl: string)	Обновляет содержимое отзыва о маршруте: текст, рейтинг и изображение.
	DeleteReview()	Удаляет отзыв.

Map	`AddPlace(place: Place)`	Добавляет метку на карте для одного места.
	`ShowRoute(route: Route)`	Отображает весь маршрут целиком на карте.
	`ShowPlace(place: Place)`	Отображает конкретное место на карте

Пользовательский интерфейс состоит из нескольких экранов: экран входа, регистрации, главная страница с рекомендованными маршрутами и местами, а также интерактивная карта с возможностью отображения точек интереса.

Для проектирования интерфейса был использован онлайн-сервис для разработки интерфейсов и прототипирования — Figma.

На главном экране есть основные элементы:

– горизонтальный свайп — интересные места;

– рекомендованные — топ маршрутов с высокими рейтингами;

– мои маршруты — сохраненные пользовательские маршруты;

– недавние — последние просмотренные маршруты;

– создать маршрут — кнопка для построения нового маршрута.

Нижняя навигационная панель:

– карта — переход к интерактивной карте;

– поиск — поиск мест и маршрутов;

– профиль — информация о пользователе.

Нажимая на карту, пользователь переходит на страницу интерактивной карты, где ему предлагается использовать текущее местоположение (рис. 4) и (рис. 5).

Рис. 3. Главный экран

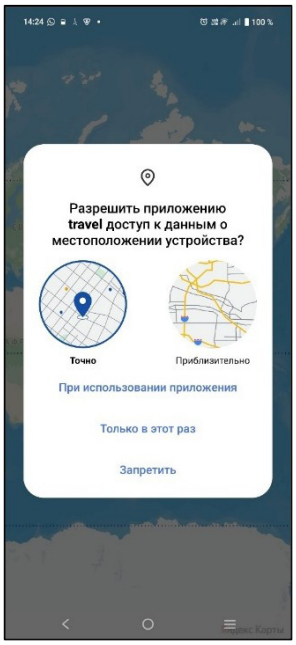

Рис. 4. Страница разрешения доступа

Рис. 5. Страница интерактивной карты

Заключение

Спроектированное мобильное приложение «Помощник туриста» реализует функциональность, необходимую современному пользователю для планирования путешествий. Результаты могут быть полезны как туристам, так и студентам, изучающим проектирование мобильных приложений.

Литература

1. *Форрестер А., Буджнах Э., Думбраван А., Тигкал Д.* Как создать Android-приложения с Kotlin / А. Форрестер, Э. Буджнах, А. Думбрава́н, Д. Тигкал. – 2-е изд. – Москва : Диалектика, 2023. – 368 с.

2. *Буч Г.* Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон ; пер. с англ. Н. Мухина. – 2-е изд. – Москва : ДМК Пресс, 2006. – 496 с.

3. *Сошин А.* Шаблоны проектирования Kotlin и лучшие практики / А. Сошин. – 3-е изд. – СПб. : Питер, 2024. – 420 с.

УДК 004.657

ПРАКТИЧЕСКИЙ ПРИМЕР ОПТИМИЗАЦИИ ЗАПРОСА С АГРЕГАЦИЕЙ И ПОИСКОМ ЭКСТРЕМАЛЬНОГО ЗНАЧЕНИЯ В СУБД POSTGRESQL

М. А. Канатников, М. В. Матвеева

Воронежский государственный университет

Аннотация. В статье проводится сравнительный анализ методов оптимизации SQL-запросов в PostgreSQL на примере задачи агрегации данных о спортивных клубах и наградах спортсменов. Анализируется эффективность использования индексов, оконных функций, общих табличных выражений, временных таблиц и партиционирования. На основе метрик стоимости и времени выполнения оцениваются преимущества и недостатки каждого метода и сформулированы практические рекомендации, основанные на этих метриках, а также на периодичности выполнения запроса.
Ключевые слова: базы данных, SQL, СУБД PostgreSQL, оптимизация запросов, языки манипулирования данными, индексы, оконные функции, CTE, временные таблицы, партиционирование, план выполнения запроса.

Введение

Эффективность работы с реляционными базами данных напрямую связана с качеством SQL-запросов. Низкопроизводительные запросы, особенно при обработке больших объемов данных, приводят к существенным временным затратам и повышенной нагрузке на вычислительные ресурсы. В статье на примере конкретной задачи рассматривается процесс оптимизации запроса в PostgreSQL и проводится сравнительный анализ различных подходов для повышения производительности выборки данных.

Основной целью работы является выявление и сравнение методов оптимизации для класса запросов, требующих агрегации данных и поиска экстремальной записи внутри каждой группы.

1. Постановка задачи

Условие задачи на выборку данных состоит в следующем. Выбрать данные о спортивных клубах (название, дата создания клуба, адрес), спортсмены которых были удостоены наград за заданный временной интервал. Для каждого клуба требуется вычислить суммарное количество полученных наград и определить фамилию последнего награждённого спортсмена в рамках указанного периода. Итоговые результаты должны быть упорядочены по убыванию числа наград, а при совпадении этого значения — в лексикографическом порядке по названию клуба.

На рис. 1 представлен фрагмент ER-модели с необходимыми для решения задачи таблицами.

Таблица *Person* — супертип для таблиц, связанных с людьми. Таблица *Athlete* — подтип *Person*, содержащий антропометрию спортсмена, его адрес проживания и внешний ключ, связывающий спортсмена с клубом (таблица *SportClub*) связью один-ко-многим. Таблица *Award* является справочной и хранит существующие спортивные награды, она связана с таблицей *Athlete* многие-ко-многим через таблицу *Award Athlete*, хранящую дату получения спортсменом награды.

Для сравнительной оценки вариантов решения поставленной задачи были выбраны две ключевые метрики: время выполнения запроса и нагрузка на центральный процессор (CPU). Эти показатели напрямую влияют на время отклика системы и общую производительность сервера. Другие характеристики, такие как количество операций I/O и потребление оперативной памяти, в рамках данного анализа не рассматриваются.

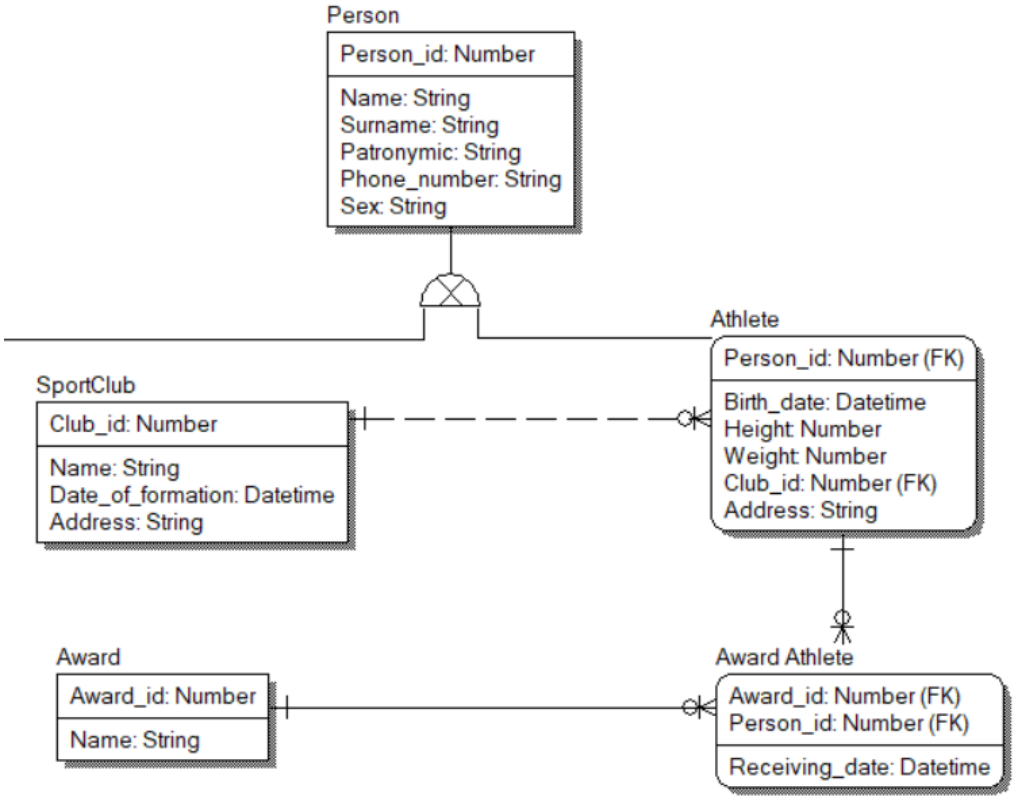

Рис 1. Фрагмент ER-модели базы данных спортивных клубов

Анализ проводится на СУБД PostgreSQL 17-й версии. Объем тестовых данных составляет приблизительно 200 тысяч записей в каждой из задействованных таблиц, за исключением таблицы *Athlete*, заполненной одним миллионом данных.

Технические параметры компьютера, на котором проводилось исследование: процессор AMD Ryzen 5 2600H; 16 ГБ оперативной памяти; SSD-накопитель Intel SSDPEKNW512G8.

2. Анализ исходного запроса

В качестве исходного метода решения поставленной задачи рассмотрим подход с использованием коррелирующего подзапроса. Под временным периодом в условии задачи подразумевается календарный год. Следует отметить, что в представленных решениях не рассматриваются связанные переменные. В качестве конкретного анализируемого периода выбран 2024 год.

Первый вариант реализации использует коррелирующий подзапрос в секции *SELECT* для определения спортсмена, получившего последнюю награду в каждом клубе за указанный период. Подзапрос выбирает соответствующую строку путём сортировки дат вручения наград и ограничения вывода первой записью (*LIMIT 1*).

```
SELECT sc.Club_Name,
       sc.Date_Of_Formation,
       sc.Address,
       COUNT(aa.Award_id) AS total_awards,
       (SELECT a2.Surname
         FROM Award_Athlete aa2
           JOIN Athlete a2
            ON aa2.Person_id = a2.Person_id
         WHERE a2.Club_id = sc.Club_id
               AND aa2.Receiving_Date >= '2024-01-01'
```

```
            AND aa2.Receiving_Date < '2025-01-01'
        ORDER BY aa2.Receiving_Date DESC, aa2.Award_id DESC
         LIMIT 1
        ) AS last_surname
    FROM Sport_Club sc
     JOIN Athlete a
      ON sc.Club_id = a.Club_id
     JOIN Award_Athlete aa
      ON a.Person_id = aa.Person_id
  WHERE aa.Receiving_Date >= '2024-01-01' AND aa.Receiving_Date < '2025-01-01'
  GROUP BY sc.Club_id,
           sc.Club_Name,
           sc.Date_Of_Formation,
           sc.Address
  ORDER BY COUNT(aa.Award_id) DESC,.Club_Name;
```

Данный подход представляет собой интуитивно понятное решение поставленной задачи, однако, обладает существенными недостатками в отношении производительности.

Используем команду *EXPLAIN ANALYZE*, чтобы проанализировать план выполнения запроса [1]. Часть результата выполнения команды представлена ниже:

```
-> Hash Join (cost=11964.34..12802.61 rows=1 width=29)
             (actual time=28.855..31.651 rows=70 loops=209)
      Hash Cond: (aa2_person_id = a2_person_id)
      -> Seg Scan on award_athlete aa2 (cost=0.00..747.87 rows=34440 width=12)
                                      (actual time=0.003..2.479 rows=34589 loops=209)
           Filter: ((receiving_date <= '2024-01-01'::date) AND
                    (receiving_date <= '2025-01-01'::date))
            Rows Removed by Filter: 2
      -> Hash (cost=11964.31..11964.31 rows=2 width=25)
              (actual time=27.659..27.659 rows=99 loops=209)
           Buckets: 1024 Batches: 1 Memory Usage: 14kB
      -> Seg Scan on athlete a2 (cost=0.00..11964.31 rows=2 width=25)
                                (actual time=4.703..27.644 rows=99 loops=209)
            Filter: (club_id = sc.club_id)
            Rows Removed by Filter: 225606
```

План показал высокую стоимость (*cost=433253068.80*) и время выполнения (6506 мс). Причины низкой производительности в неоптимальном запросе:

1. Последовательное сканирование больших таблиц (*Seq Scan on award_athlete, Seq Scan on athlete*).

2. Коррелирующий подзапрос во фразе *SELECT* порождает циклы (*loops=209*).

3. Оптимизация запроса

В главе рассматриваются различные пути оптимизации исходного запроса при помощи создания новых объектов базы данных или изменения первоначального текста запроса.

3.1. Создание индексов

В плане исходного запроса таблицы *Athlete* и *Award_Athlete* просматривались полностью, что крайне затратно в случае большого объёма данных. Последовательное сканирование про-

исходит при выполнении операции *JOIN*. Чтобы ускорить соединение таблиц, создадим индексы [2] на столбцы, используемые в условиях соединения.

```
CREATE INDEX I_Athlete_Club_id ON Athlete(Club_id);
CREATE INDEX I_Award_Athlete_Person_id ON Award_Athlete(Person_id);
```

После создания индексов ещё раз вызовем EXPLAIN ANALYZE для скрипта исходного запроса. Теперь вместо последовательного сканирования (Seq Scan) используется *Index Scan*, благодаря чему сложность снижена с линейной до логарифмической в следствие структуры индексов (B-дерево). Фрагмент плана, показывающий использование индексов:

```
-> Index Scan using i_athlete_club_id on athlete a2
     Index Cond: (club id = sc.club id)
```

В результате стоимость запроса (cost) снизилась до 1224079.14, а время выполнения — до 149 мс.

3.2. Применение оконных функций

Коррелирующий вопрос во фразе *SELECT* значительно снижает производительность из-за необходимости несколько раз читать таблицы, используемые в этом подзапросе. Для избавления от корреляции вместо подзапроса будем использовать оконные функции для выявления спортсмена, получившего последнюю в клубе награду в 2024 году. Для этого используем функцию *ROW_NUMBER()* [3], при помощи которой пронумеруем строки для каждого клуба таким образом, что номер (award_rank) 1 будет присвоен самой последней награде в 2024 году для этого клуба. Теперь для определения спортсмена, получившего последнюю в клубе награду в 2024 году, нужно найти запись с *award_rank = 1*, а не выполнять коррелирующий подзапрос.

```
SELECT Club_Name,
       Date_Of_Formation,
       Address,
       COUNT(Award_id) AS total_awards,
       MAX(CASE WHEN award_rank = 1 THEN Surname END) AS last_surname
   FROM (
         SELECT  sc.Club_id,
                 sc.Club_Name,
                 sc.Date_Of_Formation,
                 sc.Address,
                 aa.Award_id,
                 aa.Receiving_Date,
                 a.Surname,
                 ROW_NUMBER() OVER (
                       PARTITION BY sc.Club_id
                       ORDER BY aa.Receiving_Date DESC, aa.Award_id
                 ) AS award_rank
            FROM Sport_Club sc
             JOIN Athlete a
              ON sc.Club_id = a.Club_id
             JOIN Award_Athlete aa
              ON a.Person_id = aa.Person_id
            WHERE aa.Receiving_Date >= ‘2024-01-01’
               AND aa.Receiving_Date < ‘2025-01-01’
        )
```

```
  GROUP BY Club_id, Club_Name, Date_Of_Formation, Address
  ORDER BY total_awards DESC, Club_Name;
```

Проанализируем фрагмент плана выполнения этого запроса:

```
-> Subquery Scan on unnamed_subquery (cost=21200.12..26244.43 rows=34440 width=84)
                                    (actual time=78.505..87.165 rows=14589 loops=1)
     -> WindowAgg (cost=21200.12..25900.03 rows=34440 width=88)
                  (actual time=78.504..86.003 rows=14589 loops=1)
```

Теперь вместо подзапроса используется оконная функция WindowAgg, в следствие чего имеем значительный прирост производительности в сравнении с первоначальным вариантом. Теперь стоимость сокращена до 31336.6, а время выполнения — до 85 мс.

3.3. Использование общих табличных выражений

В первоначальном запросе в подзапросе во фразе *SELECT* происходит внутреннее соединение таблиц *Athlete* и *Award_Athlete*, аналогичное соединение происходит и во внешнем запросе. Для того, чтобы не выполнять одну и ту же операцию дважды, вынесем соединение этих таблиц в CTE, также отберём в нём только записи за 2024 год. Тогда запрос примет вид:

```
WITH Recieved_At_2024 AS (
SELECT aa.award_id AS id_award,
       a.Club_id AS id_club,
       aa.Receiving_Date AS rdate,
       a.Surname AS surname
   FROM Athlete a
     JOIN Award_Athlete aa
      ON a.Person_id = aa.Person_id
WHERE aa.Receiving_Date >= '2024-01-01' AND aa.Receiving_Date < '2025-01-01')

SELECT sc.Club_Name,
       sc.Date_Of_Formation,
       sc.Address,
       COUNT(ra.id_award) AS total_awards,
       (SELECT ra.surname
          FROM Recieved_At_2024 ra
        WHERE ra.id_club = sc.Club_id
        ORDER BY ra.rdate DESC, ra.id_award DESC
         LIMIT 1
       ) AS last_surname
    FROM Sport_Club sc
      JOIN Recieved_At_2024 ra
       ON sc.Club_id = ra.id_club
  GROUP BY sc.Club_id, sc.Club_Name,
           sc.Date_Of_Formation, sc.Address
 ORDER BY total_awards DESC, sc.Club_Name;
```

Проанализируем план выполнения этого запроса. Вместо сканирования таблиц *Athlete* и *Award_Athlete* используется сканирование общего табличного выражения (*CTE Scan on recieved_at_2024 ra*). Таким образом убраны лишние соединения таблиц. Несмотря на это, запрос стал выполняться медленнее, чем первоначальный вариант с индексами (*cost=26726405.88, actual time=238.851.*). Причина в том, что CTE материализовалось и в плане выполнения CTE снова последовательное сканирование (*Seq Scan on athlete a*), то есть индексы не используются.

```
CTE received_at_2024
-> Hash Join (cost=1178.37..17154.78 rows=34440 width=33)
             (actual time=6.847..48.346 rows=14589 loops=1)
      Hash Cond: (a.person_id = aa.person_id)
      -> Seg Scan on athlete a (cost=0.00..11400.05 rows=225705 width=29)
                               (actual time=0.011..18.542 rows=225705 loops=1)
```

Чтобы использовать общее табличное выражение, но при этом использовать индексы, необходимо явно указать СУБД, что его не надо материализовать, добавив указание оптимизатору *AS NOT MATERIALIZED*. Тогда оптимизатор свернёт CTE в подзапрос и план выполнения будет идентичен способу из главы 3.1.

3.4. Временные таблицы

Если запрос будет многократно выполняться в течение одной сессии, имеет смысл создать временную таблицу, которая будет отбирать данные, аналогичные CTE из предыдущей главы.

```
CREATE TEMPORARY TABLE IF NOT EXISTS temp_received_2024 AS
       SELECT aa.award_id AS id_award,
              a.Club_id AS id_club,
              aa.Receiving_Date AS rdate,
              a.Surname AS surname
         FROM Athlete a
          JOIN Award_Athlete aa
           ON a.Person_id = aa.Person_id
         WHERE aa.Receiving_Date >= '2024-01-01' AND
               aa.Receiving_Date < '2025-01-01';
```

Также необходимо создать индекс для эффективного обращения к временной таблице.

```
CREATE INDEX I_Temp_Club_id ON temp_received_2024(id_club);
```

Теперь запрос будет выглядеть подобно запросу из предыдущей главы, только вместо CTE будет использоваться временная таблицы.

Фактическое время выполнения запроса сильно уменьшилось (10 мс) даже в сравнении с вариантом с оконной функцией, хотя стоимость запроса осталась высокой (1722769.22).

Также при использовании временной таблицы необходимо учитывать накладные расходы на хранение этой временной таблицы.

3.5. Анализ эффективности партиционирования

Если в задействованных таблицах предполагается хранение значительных объёмов данных, эффективным решением станет применение партиционирования таблиц. Данный подход предполагает разделение большой таблицы на отдельные сегменты (партиции) по какому-либо критерию. Разобьём таблицу *Award_Athlete* по годам вручения награды. Это позволит ограничить область поиска наград за конкретный год лишь соответствующей партицией, что резко сократит объём обрабатываемых данных и повысит общую производительность запроса.

Также важно не потерять данные, если таблица уже была создана и использовалась, для этого придётся использовать промежуточную таблицу.

```
CREATE TABLE Award_Athlete_new (
    Award_Athlete_id SERIAL,
    Award_id         INT NOT NULL,
```

```
    Person_id        INT NOT NULL,
    Receiving_Date   DATE NOT NULL,
    PRIMARY KEY (Award_Athlete_id, Receiving_Date)
) PARTITION BY RANGE (Receiving_Date);

CREATE TABLE Award_Athlete_historical PARTITION OF Award_Athlete_new
                FOR VALUES     FROM ('0001-01-01') TO ('2022-01-01');
CREATE TABLE Award_Athlete_2022 PARTITION OF Award_Athlete_new
                               FOR VALUES FROM ('2022-01-01') TO ('2023-01-01');
CREATE TABLE Award_Athlete_2023 PARTITION OF Award_Athlete_new
                               FOR VALUES FROM ('2023-01-01') TO ('2024-01-01');
CREATE TABLE Award_Athlete_2024 PARTITION OF Award_Athlete_new
                               FOR VALUES FROM ('2024-01-01') TO ('2025-01-01');
CREATE TABLE Award_Athlete_future PARTITION OF Award_Athlete_new
                               FOR VALUES FROM ('2025-01-01') TO (MAXVALUE);

INSERT INTO Award_Athlete_new (Award_Athlete_id, Award_id, Person_id, Receiv-ing_
Date)
       SELECT Award_Athlete_id, Award_id, Person_id, Receiving_Date
       FROM Award_Athlete;

ALTER TABLE Award_Athlete RENAME TO Award_Athlete_old;
ALTER TABLE Award_Athlete_new RENAME TO Award_Athlete;

DROP TABLE Award_Athlete_old;
```

После реализации партиционирования таблицы Award_Athlete повторно проанализируем план выполнения запроса. Теперь вместо сканирования всей таблицы используется сканирование только необходимой партиции (Parallel Seq Scan on award_athlete_2024 aa). Хотя на тестовых данных прирост производительности оказался не столь значительным, при увеличении обьема данных эффект от такого подхода будет усиливаться за счёт сокращения области сканирования исключительно до необходимой партиции.

4. Сравнительный анализ

Результаты анализа эффективности всех методов приведены в табл. 1. Перед каждым запуском кэш очищался, чтобы обеспечить одинаковые условия выполнения.

Таблица 1

Сравнение методов оптимизации

Метод	Стоимость (cost)	Время (мс)
Исходный запрос	433253068.80	6506.186
Исходный запрос с индексами	1224079.14	148.896
Оконные функции	31336.69	84.704
CTE (материализованное)	26726405.88	238.851
Временная таблица	1722805.69	10.328
Партиционирование + исходный запрос	1221304.18	131.268

Для поставленной задачи наилучшие результаты показали временные таблицы и оконные функции. Во всех методах стоит создать индексы на столбцы, которые будут в условиях соединения таблиц. Временные таблицы целесообразно использовать при многократном выполне-

нии запроса в рамках одной сессии, тогда как оконные функции эффективны для однократных запросов. Партиционирование полезно при работе с большими объемами данных, отбираемыми в запросах с фильтрацией по дате получения награды.

Заключение

В ходе исследования был проведён сравнительных анализ различных способов оптимизации SQL-запроса с агрегацией и поиском экстремального значения в СУБД PostgreSQL. Основными проблемами исходного решения были последовательное сканирование крупных таблиц и использование ресурсоемкого коррелирующего подзапроса. В процессе работы были применены и проанализированы следующие методы оптимизации: создание индексов для ускорения операций JOIN, замена коррелирующего подзапроса оконной функцией, вынос повторяющихся операций в CTE, создание временных таблиц и партиционированиие. Результатом стало сокращение времени выполнения запроса более чем в 75 раз и существенное уменьшение нагрузки на центральный процессор.

Литература

1. *Домбровская Г.* Оптимизация запросов в PostgreSQL / Г. Домбровская, Б. Новиков, А. Бейликова ; пер. с англ. Д. А. Беликова. – Москва : ДМК Пресс, 2022. – 278 с.

2. PostgreSQL: Documentation : официальный сайт. URL: https://www.postgresql.org/docs/15/indexes-intro.html (дата обращения: 17.11.2025).

3. Postgres Professional : официальный сайт. URL: https://postgrespro.ru/docs/postgrespro/current/functions-window (дата обращения: 18.11.2025).

УДК 004.056.53

РАЗРАБОТКА ПРОГРАММЫ ДЛЯ ФАЗЗИНГ-ТЕСТИРОВАНИЯ ПРОМЫШЛЕННЫХ ПРОТОКОЛОВ АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМ ПРОЦЕССОМ

А. В. Комнатный, Д. В. Филонов

Воронежский государственный университет

Аннотация. Фаззинг-тестирование является одним из самых популярных видов тестирования на проникновение программного обеспечения, получившее широкое распространение благодаря своей эффективности и широте покрытия. Однако традиционные методы, включающие в себя фаззинг с обратной связью, могут показывать низкую эффективность в контексте использования с четко структурированными и сессионными промышленными протоколами. Привычные фаззеры затрачивают значительную часть времени на синтаксически неверные мутации, которые будут отброшены на этапе проверки соединения. В данной работе рассматриваются виды фаззеров, применимые для тестирования промышленных протоколов передачи данных, а также приводится реализация фаззера.

Ключевые слова: фаззинг-тестирование, автоматизированные системы управления, АСУ ТП, промышленные протоколы, Modbus TCP, структурный фаззинг, мутационный фаззинг, тестирование сетевых протоколов, boofuzz, AFLNet, уязвимости программного обеспечения, Semi-Stateful Fuzzer, парсинг пакетов, анализ сетевого трафика, безопасность критической инфраструктуры.

Введение

Фаззинг-тестирование стало общепризнанным стандартом, используемым для поиска уязвимостей в программном обеспечении. Оно прошло длинный путь, эволюционируя от простого перебора всех возможных вариантов до фаззинга с обратной связью по покрытию [1].

Использование фаззинг-тестирования в сфере промышленных систем долгое время оставалось невостребованным в связи с изолированной средой их функционирования. Однако современные тенденции показывают, что несмотря на отсутствие прямого доступа к атакуемой системе, злоумышленникам удается проведение атак на промышленный сегмент, используя для перехода обычные офисные компьютеры. Также не исключена возможность доступа к промышленным контроллерам через глобальную сеть Интернет. В связи с этим исследования методов фаззинг-тестрировая АСУ ТП становится как никогда актуально, особенно в связи с угрозами атаки на объекты критической информационной инфраструктуры (КИИ).

В данной работе будут рассмотрены самые передовые средства фаззинг-тестирования, проведен анализ их применимости к АСУ ТП и разработана программа для проведения тестирования.

1. Особенности тестирования промышленных протоколов

В состав АСУ ТП входят аппаратные средства (датчики, программируемые логические контроллеры, исполнительные механизмы) и программное обеспечение (зачастую различные SCADA системы). Для обеспечения взаимодействия между подсистемами и устройствами используют промышленные протоколы передачи данных. Широкое распространение получил протокол Modbus TCP, который инкапсулирует сообщения Modbus в пакеты TCP[2].

В связи с этим фаззинг-тестирование промышленного сегмента, является нестандартной задачей. Появляется необходимость учитывать особенности работы промышленных протоколов передачи данных:

– жесткая структура и синтаксис пакетов. Промышленные протоколы, такие как Modbus имеют строго определенные поля (ID транзакции, ID протокола, длина остатка пакета, адрес слейва, информационная часть пакета). Данных факт сделает заведомо неэффективными значительный процент случайных битовых мутаций и приведет лишь к ошибкам, связанным с неверным форматом пакета.

– внутреннее состояние сессии. Это указывает на необходимость отправлять не только адекватные по содержанию пакеты, но и поддерживающие или изменяющие внутренние состояние сервера (отправлять набор пакетов) для продвижения в глубь логики.

– закрытость программного обеспечения. Данных факт делает невозможным тестирование в режиме «серого ящика», поскольку зачастую внутренний код программ промышленного сегмента остается закрытым и трудно извлекаемым из устройства.

– ограниченные вычислительной мощности аппаратных средств. Необходимо учитывать скорость обработки пакетов, чтобы исключить переход в состояние отказ в обслуживании.

2. Фаззинг-тестирование

Самым первым способом фаззинг-тестирования был метод Dumb Fuzzing. При таком подходе на вход подаются случайно сформированные данные без учета структуры и очередности пакетов. Такой подход является абсолютно неэффективным и малоприменимым для промышленных протоколов передачи данных. Большинство пакетов будет отброшено из-за невозможности корректно распарсить их и пройти первичную проверку. Однако данный метод легко реализуем и может быть улучшен до Network replay fuzzing за счет формирования вручную первичного набора данных, необходимого для установки соединения и определения служебных полей управления соединением. При получении ответных пакетов возможно обнаружение переходов состояний внутренней логики сервера, что дает новые пути для фаззинга. Таким образом, возможно ручное формирование структуры пакета и очередности отправки.

Сейчас наиболее используемыми и результативными являются AFLNet (Coverage-Guided Fuzzing), используемый для тестирования серого ящика, и boofuzz (Structure-Based fuzzing), используемый для тестирования черного ящика. Далее будут описаны ключевые особенности работы этих фаззеров.

AFLNet — это фаззер сетевых протоколов, использующий мутационный подход и обратную связь по состоянию[3]. На вход фаззера подаются захваченные сетевые пакеты pcap включающие запросы и ответы между сервером и клиентом. Затем создает начальный набор последовательностей сообщений. Выполняется фильтрация ответов, чтобы получить трассировки запросов клиента, затем происходит разбиение отфильтрованных трассировок, чтобы определить начало и конец каждого сообщения. Также реализован конечный автомат состояний, который принимает ответы сервера и дополняется интересными состояниями протокола. Затем происходит выбор наиболее перспективного состояния конечного автомата на основе эвристик, полученных в результате анализа статистических данных. Наиболее перспективными являются направления, которые внесли вклад в увеличение покрытия кода или изменения состояний. Для корректной работы данного инструмента необходим доступ к исходному коду программы. Без него теряется главная особенность, связь по покрытию кода.

Boofuzz — фреймворк для фаззинга сетевых протоколов, поддерживающий сложные протоколы по состояниям. Для начала работы фаззеру подается описание структуры пакетов — это необходимо для возможности мутации только валидных частей. Затем определяется последовательность сообщений и формируется дерево состояний, в которых возможна отправка пограничных пакетов. Также в нем реализован детектор ошибок, позволяющий отслеживать ответы сервера или логические ошибки. Все эти особенности позволяют проводить глубокое тестирование в режиме черного ящика.

3. Реализация программы для фаззинг-тестирования

В ходе анализа программных решений были выбраны ключевые аспекты, необходимые для реализации программы осуществляющей фаззинг-тестирования. Программа будет создавать корректные пакеты за счет анализа реального общения устройства по выбранному протоколу. Реализована возможность определения наиболее перспективных пакетов из набора мутаций и формирования очереди для переходов в логике устройства. Ниже будут представлены основные функции программы.

Была реализована функция parse_modbus для парсинга Modbus пакета по заголовку. Это необходимо для дальнейшей корректной структурной мутации.

```
def parse_modbus(payload: bytes):

    if not payload or len(payload) < 8:
        return None

    return {
        "tid":  payload[0:2],
        "pid":  payload[2:4],
        "len":  payload[4:6],
        "uid":  payload[6],
        "fc":   payload[7],
        "data": payload[8:]
    }
```

Была реализована функция mutate для выполнения неструктурированного фаззинга. Она позволяет выполнять инверсию случайного байта, вставку случайных байт, удаление байта, дублирование и перемешивание.

```
def mutate(payload: bytes) -> bytes:

    if not payload:
        return bytes([random.randint(0, 255) for _ in range(8)])

    data = bytearray(payload)
    op = random.choice(["flip", "insert", "delete", "dup", "shuffle"])

    if op == "flip":
        idx = random.randint(0, len(data) - 1)
        data[idx] ^= random.randint(1, 255)
    elif op == "insert":
        idx = random.randint(0, len(data))
           data[idx:idx] = bytes([random.randint(0, 255) for _ in range(random.
randint(1, 8))])
    elif op == "delete" and len(data) > 1:
        idx = random.randint(0, len(data) - 1)
        del data[idx]
    elif op == "dup":
        idx = random.randint(0, len(data) - 1)
        data.extend(data[idx:idx + 1] * random.randint(1, 5))
    elif op == "shuffle" and len(data) > 2:
        a = list(data)
        random.shuffle(a)
```

```
        data = bytearray(a)

    return bytes(data)
```

Была реализована функция mutate_struct, позволяющая производить направленные изменения в конкретных полях пакета или полностью менять его структуру.

```
def mutate_struct(payload: bytes) -> bytes:
    parsed = parse_modbus(payload)
    if not parsed:
        return mutate(payload)  # fallback в байтовый мутатор

    data = bytearray(payload)

    mutation_type = random.choice([
        "fc_flip",
        "uid_rand",
        "len_corrupt",
        "data_expand",
        "data_flip",
        "truncate"
    ])
    if mutation_type == "fc_flip":
        idx = 7
       choices = [0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x80, random.randint(0, 255)]
        data[idx] = random.choice(choices)

    elif mutation_type == "uid_rand":
        data[6] = random.randint(0, 255)

    elif mutation_type == "len_corrupt":
        new_len = random.randint(0, 255)
        data[4] = 0
        data[5] = new_len

    elif mutation_type == "data_expand":
        data.extend(b"A" * random.randint(1, 50))

    elif mutation_type == "data_flip":
        if len(parsed["data"]) > 0:
            idx = random.randint(8, len(data) - 1)
            data[idx] ^= random.randint(1, 255)

    elif mutation_type == "truncate":
        if len(data) > 2:
            cut = random.randint(1, max(1, len(data) // 2))
            data = data[:-cut]

    return bytes(data)
```

Для отправки и анализа ответов сервера была реализована функция send_and_classify, позволяющая определять какой пакет вызвал интересную реакцию сервера.

```
def send_and_classify(ip: str, port: int, payload: bytes, timeout: float = TIMEOUT):
    s = None
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(timeout)
        s.connect((ip, port))
        s.send(payload)
        try:
            data = s.recv(2048)
            return classify_response(data), data
        except socket.timeout:
            return "TIME", None
    except ConnectionRefusedError:
        return "REFUSED", None
    except Exception as e:
        return "ERR", None
    finally:
        if s:
            s.close()
```

Функция main является функцией основного цикла и выполняет выбор seed из набора pcap, мутацию, отправку, классификацию, сохранение интересных пакетов для их дальнейшей мутации и отправки. Учтена необходимость выполнения time.sleep для сохранения соединения.

```
def main():
    seeds = extract_payloads(PCAP_FILE, limit=20)
    if not seeds:
        print("Не найдено seed пакетов и не удалось создать fallback seeds.")
        return
    print("[*] Запуск улучшенного Semi-Stateful Guided Fuzzer")
    counter = 0
    try:
        while True:
            seed = choose_seed(seeds, interesting)
            if seed is None:
                print("[!] Нет seed для отправки, завершаем.")
                break
            if random.random() < 0.7:
                mutated = mutate_struct(seed)
            else:
                mutated = mutate(seed)
            status, resp = send_and_classify(TARGET_IP, TARGET_PORT, mutated)
            counter += 1
            if status in ["EXC", "TIME", "OK"]:
                        interesting[status].append(mutated)
         print(f"[+] Добавлен пакет ({status}), всего = {len(interesting[status])}")
            if status == "REFUSED":
                print("\n[!!!] CONNECTION REFUSED ")
                print("Payload (hex):", mutated.hex())
                break
            if status == "ERR":
                print("[!] Ошибка при отправке/получении (ERR).")
```

```
            if counter % 100 == 0:
                print(f"[~] Отправлено {counter} пакетов. Статус: {status}")

            time.sleep(0.01)

    except KeyboardInterrupt:
        print("\n[!] Остановлено пользователем (KeyboardInterrupt).")
    except Exception as e:
        print(f"[!] Непредвиденная ошибка: {e}")
```

Заключение

В результате проведенного исследования была разработана программа для выполнения фаззинг-тестирования автоматизированных систем управления технологическим процессом, взаимодействующих по промышленным протоколам. Эта модель учитывает особенность тестирования и позволяет решать основные проблемы, которые возникают при использовании стандартных методов.

Литература

1. *Саттон М., Грин А., Амини П.* Fuzzing: Исследование уязвимостей методом грубой силы, «Символ Плюс». – СПб. – Москва, 2009. – 560 с.

2. *Klasen F., Oestreich V., Volz M.* Industrial Communication with Fieldbus and Ethernet / F. Klasen, V. Oestreich, M. Volz.–Berlin : VDE Verlag, 2011. – 330 с.

3. *Pham V.-T., Böhme M., Roychoudhury A.* AFLNET: A Greybox Fuzzer for Network Protocols // thuanpv.github.io: [сайт] URL https://thuanpv.github.io/publications/AFLNet_ICST20.pdf (дата обращения: 20.11.2025)

УДК 004.85

РИСКИ И ОГРАНИЧЕНИЯ ИСПОЛЬЗОВАНИЯ МОКИРОВАННЫХ ВНЕШНИХ СИСТЕМ ПРИ ИНТЕГРАЦИОННОМ ТЕСТИРОВАНИИ

С. С. Корнеева

Воронежский государственный университет

Аннотация. Статья посвящена анализу рисков и ограничений, возникающих при использовании мокированных внешних систем на этапе интеграционного тестирования распределённых программных комплексов. Рассматривается место мок-сервисов в общей стратегии тестирования, выделяются основные группы рисков, связанные с семантическим разрывом между моделью и реальным поведением, дрейфом API, искажением нефункциональных характеристик, недоучётом отказов и нестабильности, а также организационными и когнитивными эффектами. Обосновывается необходимость сочетания мок-окружений с другими типами тестовых стендов и формулируются подходы к снижению указанных рисков без отказа от мокирования как инструмента.
Ключевые слова: интеграционное тестирование, мок-сервис, тестовые дублёры, распределённые системы, дрейф API, отказоустойчивость, контрактное тестирование.

Введение

Современные программные системы всё чаще строятся как распределённые, опирающиеся на взаимодействие с внешними сервисами: платёжными шлюзами, системами аутентификации, логистическими платформами, сторонними API. От корректности и предсказуемости этих интеграций напрямую зависит устойчивость бизнес-процессов. В то же время доступ к реальным внешним системам для тестирования часто ограничен: боевые контуры закрыты по соображениям безопасности, тестовые стенды нестабильны или недоступны, публичные API имеют квоты и тарифные ограничения. В качестве решения широко применяется мокирование внешних систем: вместо реального сервиса используется эмулируемый, развёрнутый, например, в отдельном Docker-контейнере и воспроизводящий интерфейс (HTTP, gRPC и др.) в контролируемой форме. Это даёт очевидные преимущества: ускорение прогона тестов, повышение воспроизводимости и независимости от внешней инфраструктуры. Однако мок-сервис является лишь моделью поведения внешней системы, а любая модель неизбежно упрощает реальность. При избыточной опоре на моки формируется устойчивое, но потенциально ложное ощущение качества: интеграционные тесты демонстрируют стабильный успех, однако при переходе к взаимодействию с реальными сервисами обнаруживаются критичные расхождения.

Целью настоящей работы является систематизация и углублённый анализ рисков и ограничений, связанных с использованием мокированных внешних систем при интеграционном тестировании, а также рассмотрение подходов к снижению этих рисков в рамках общей стратегии качества программного обеспечения.

1. Место мокированных внешних систем в стратегии тестирования

В типовой практике тестирования распределённых систем принято выделять несколько уровней:

– юнит-тесты, ориентированные на проверку отдельных модулей с использованием внутриязыковых тестовых дублёров (моки, стабы, фейки);

– интеграционные тесты, проверяющие взаимодействие компонентов и инфраструктурных зависимостей;

– системные и сквозные (end-to-end) испытания, максимально приближенные к боевому окружению.

Мокированный внешний сервис занимает промежуточное положение между внутренними дублёрами и реальными внешними системами. С одной стороны, сохраняется сетевой характер взаимодействия и используются реальные протоколы; с другой — поведение сервиса полностью контролируется разработчиком тестового стенда.

Именно на интеграционном уровне возникает соблазн подменить большинство внешних зависимостей моками и ориентировать основную часть тестов на такое «контролируемое» окружение. В этом случае повышается скорость обратной связи и снижается стоимость развёртывания стендов, но одновременно возрастает зависимость качества от корректности и актуальности самой модели внешнего мира.

2. Семантический разрыв между моделью и реальным поведением

Ключевой риск мокирования связан с семантическим разрывом между поведением мок-сервиса и реальной внешней системы. Модель формируется на основе доступной документации, ограниченного набора примеров и текущих потребностей бизнес-логики. Такое представление:

– неполно, поскольку охватывает только наиболее очевидные и востребованные сценарии;

– склонно к устареванию из-за эволюции внешнего API;

– упрощено, так как сложные, редкие и «неудобные» случаи зачастую не моделируются.

В результате интеграционные тесты проверяют соответствие кода не внешней системе как таковой, а локальному представлению о ней. Неожиданные комбинации входных данных, допустимые, но редко документируемые форматы ответов, особенности обработки ошибок остаются вне поля зрения.

Практическим проявлением данного риска являются ситуации, при которых реальный внешний сервис вводит новые коды ошибок, дополнительные поля в структуре данных или изменяет семантику существующих полей. Мок-сервис, основанный на прежнем контракте, продолжает возвращать «идеальные» ответы, и интеграционные тесты не фиксируют расхождений вплоть до выхода обновлённой системы в эксплуатацию.

2.1. Дрейф API и контрактов

Даже при использовании формальных описаний интерфейсов (OpenAPI, protobuf и др.) мок-сервис подвержен дрейфу относительно реального API. Изменения на стороне поставщика внешнего сервиса могут происходить с иной частотой и приоритетом, чем обновления тестового окружения. В ряде случаев изменения считаются обратноссовместимыми, но на практике приводят к сбоям у конкретного потребителя.

Дрейф API проявляется в нескольких формах:

– в структуре данных (добавление или изменение полей, типов, обязательности);

– в наборе и семантике кодов ответов;

– в поведенческих особенностях (изменение бизнес-логики при тех же сигнатурах).

Поскольку мок-сервис развивается независимо, несоответствие может накапливаться незаметно. При этом устойчивая «зелёная» картина по интеграционным тестам создаёт эффект видимого благополучия.

Средством снижения этого риска выступает контрактное тестирование, ориентированное на автоматическую проверку согласованности потребителей и провайдеров интерфейса. Однако даже формальные контракты нуждаются в дисциплинированном сопровождении; при его отсутствии дрейф переносится с уровня кода на уровень метаданных и не устраняется полностью.

2.2. Искажение нефункциональных характеристик

Мокированные внешние системы существенно искажают восприятие нефункциональных характеристик интеграций. В тестовом окружении ответы мок-сервисов, как правило, быстры и стабильны, отсутствуют реальные ограничения по нагрузке, квоты и эффекты конкуренции за ресурсы.

На этом фоне возникает ряд характерных искажений:

– оптимистичная оценка производительности: система демонстрирует хорошие показатели при взаимодействии с быстрым мок-сервисом, но теряет устойчивость при подключении к более медленному и нестабильному реальному API;

– недооценка механизмов отказоустойчивости: логика повторных попыток, таймаутов, ограничений частоты запросов формально присутствует в коде, но не проходит испытание в условиях, приближенных к реальной эксплуатации.

Таким образом, при доминировании мок-окружений создаётся «лабораторный» профиль нагрузки, существенно отличающийся от реального. В особенности это критично для систем, обрабатывающих большие объёмы данных или работающих в условиях пиковых нагрузок, где задержки и отказоустойчивость имеют ключевое значение.

2.3. Недоучет отказов и нестабильности внешней среды

Реальные внешние сервисы характеризуются не только корректной работой, но и отказами, кратковременными недоступностями, сетевыми сбоями, непредсказуемыми задержками. Эти аспекты зачастую оказываются слабо отражёнными в мок-сервисах, ориентированных преимущественно на «позитивные» сценарии.

Негативные сценарии либо отсутствуют, либо моделируются одним-двумя тестами с искусственными ошибками. В результате остаются недостаточно исследованными важнейшие вопросы:

– как система реагирует на длительные или повторяющиеся сбои внешнего сервиса;

– корректно ли работают защитные механизмы (ограничение повторов, размыкание цепочки вызовов, деградация функциональности);

– как ведут себя цепочки микросервисов при частичной недоступности одного из внешних компонентов.

2.4. Организационные и когнитивные эффекты использования моков

Использование мокированных внешних систем оказывает влияние не только на техническую сторону разработки, но и на организационную культуру, а также на восприятие качества.

Во-первых, формируется устойчивое ощущение завершённости. Наличие набора интеграционных тестов, стабильно проходящих на мок-окружении, интерпретируется как признак достаточного покрытия интеграций. Это снижает мотивацию к построению полноценных стендов с реальными внешними системами и может приводить к смещению приоритетов в планировании работ по качеству.

Во-вторых, изменяется фокус инженерного мышления. Обсуждение поведения внешних систем всё чаще ведётся в терминах реализации мок-сервисов, а не реального API. Последнее воспринимается как нечто стороннее и в меньшей степени определяющее ежедневную работу команды.

В-третьих, развивается собственный техдолг вокруг мок-инфраструктуры. По мере усложнения мок-сервисов возрастает стоимость их сопровождения, миграции на новые версии технологий, обновления в соответствии с изменяющимися контрактами. В условиях ограничен-

ных ресурсов это приводит к выборочным обновлениям, локальным обходным решениям и, как следствие, к снижению доверия к интеграционным тестам.

3. Факторы, усиливающие влияния рисков

Выделенные риски проявляются с различной интенсивностью в зависимости от контекста. Наиболее уязвимыми оказываются ситуации, когда сочетаются следующие факторы:

– высокая динамика развития внешнего сервиса (частые изменения API, добавление новых функций);

– жёсткие требования к надёжности и регуляторное давление (банковский, медицинский, телекоммуникационный домены);

– микросервисная архитектура с большим числом взаимозависимых компонент;

– организационная установка на минимизацию затрат на тестовую инфраструктуру.

В таких условиях ставка исключительно на мок-окружения приводит к накоплению системных рисков, проявляющихся при интеграции с реальными внешними системами на поздних этапах жизненного цикла.

4. Подходы к снижению рисков при сохранении мок-окружений

Полный отказ от мокированных внешних систем не представляется целесообразным, поскольку они решают важные практические задачи. Вместо этого целесообразно рассматривать их как часть многоуровневой стратегии тестирования, предусматривающей компенсацию присущих мокам ограничений.

К числу таких подходов относятся:

– комбинирование мок-окружений с системными и сквозными тестами на стендах, приближенных к реальным условиям эксплуатации;

– формализация контрактов взаимодействия (OpenAPI, protobuf) и внедрение контрактного тестирования для контроля дрейфа API;

– расширение сценариев поведения мок-сервисов за счёт моделирования ошибок, задержек, лимитов, нестабильности;

– периодическая сверка ответов мок-сервисов с реальными стендами на ограниченном наборе запросов;

– корректная интерпретация роли моков в инженерной культуре: понимание их как удобного, но заведомо неполного инструмента ранней проверки.

Реализация указанных подходов позволяет сохранить преимущества мокирования — скорость, предсказуемость, удобство локальной разработки — при одновременном снижении вероятности критичных расхождений между тестовым и эксплуатационным окружениями.

Заключение

Мокированные внешние системы являются неотъемлемым элементом современного интеграционного тестирования распределённых программных комплексов. Они позволяют снизить зависимость от внешней инфраструктуры, ускорить разработку и обеспечить воспроизводимость тестов. Однако использование мок-сервисов несёт в себе ряд рисков, связанных с семантическим разрывом между моделью и реальным поведением внешних систем, дрейфом API, искажением нефункциональных характеристик, недоучётом отказов и нестабильности, а также организационными и когнитивными эффектами.

Проведённый анализ показывает, что эти риски не являются следствием «ошибочности» самого подхода мокирования, а возникают при его некритичном и изолированном примене-

нии. Мокированные внешние системы необходимо рассматривать как один из инструментов в составе комплексной стратегии тестирования, дополненный контрактным тестированием, системными и сквозными испытаниями на реальных стендах, а также сознательной работой с инженерной культурой.

Такой подход позволяет использовать моки для решения их естественных задач — обеспечения быстрой обратной связи и контроля регрессий — без формирования ложного ощущения завершённости и полноты проверки интеграций.

Литература

1. *Ньюман С.* Построение микросервисов. Разработка, тестирование и эксплуатация распределённых систем / С. Ньюман ; пер. с англ. – Санкт-Петербург : Питер, 2015. – С. 280.

2. *Мезарос Г.* xUnit Test Patterns: Рефакторинг кода тестов / Г. Мезарос ; пер. с англ. – Москва : Вильямс, 2008. – 944 с.

3. *Нигард М. Т.* Release It! — Делай надёжно: проектирование и развёртывание продакшен готового ПО / М. Т. Нигард ; пер. с англ. – Москва : ДМК Пресс, 2018. – 350 с.

УДК 004.031.43

ИНТЕГРАЦИЯ ПРИКЛАДНОГО РЕШЕНИЯ НА ПЛАТФОРМЕ 1С:ПРЕДПРИЯТИЕ С ВЕБ-ПРИЛОЖЕНИЕМ

Н. М. Крейдун

Воронежский государственный университет

Аннотация. Статья посвящена разработке интегрированной информационной системы для автоматизации аренды квартир на платформе 1С:Предприятие 8.3 с веб-приложением. Рассматриваются модели интеграции: SOAP, HTTP REST, OData и WebSocket. Описывается архитектура интеграции, реализация механизмов обмена данными и результаты тестирования системы.
Ключевые слова: 1С:Предприятие, веб-приложение, аренда недвижимости, интеграция, SOAP, REST API, OData, WebSocket, Avito API.

Введение

Для повышения эффективности бизнеса в сфере аренды и управления недвижимостью компании стремятся использовать информационные системы, позволяющие автоматизировать процессы по ведению реестрового учета объектов недвижимости, управлению договорами аренды и расчетами с арендаторами, эксплуатации объектов недвижимости. Использование платформы 1С:Предприятие позволяет построить комплексное решение, которое гарантирует гибкость настройки, удобство использования и подходит для использование на предприятиях любой численности.

Интеграция 1С:Предприятие с веб-приложениями расширяет функциональность системы, обеспечив доступ к информации через интернет и автоматизацию работы с внешними сервисами.

В рамках работы была создана интегрированная система на основе конфигурации 1С, взаимодействующей с веб-приложением через протоколы SOAP, HTTP REST, OData и WebSocket. Разработанная система обеспечивает учет объектов недвижимости, ведение договоров, учет платежей и интеграцию с внешним веб-приложением (API Avito для загрузки объявлений). Архитектура интеграции использует различные протоколы обмена данными и механизмы веб-взаимодействия.

1. Архитектура интеграции

Архитектура интеграции представляет собой многоуровневую структуру, обеспечивающую взаимодействие между платформой 1С:Предприятие, веб-приложением и внешними сервисами (API Avito). Архитектура включает основные компоненты: сервер 1С, база данных, веб-приложение, API Avito и клиентский интерфейс, связанные через стандартизированные протоколы обмена данными (рис. 1).

Основные компоненты:

• *Сервер 1С:Предприятие* — ядро системы, обеспечивающее бизнес-логику, хранение и обработку данных о квартирах, арендаторах, собственниках, договорах и платежах.

• *Веб-приложение* — внешний интерфейс для пользователей, обеспечивающий доступ к данным через интернет.

• *Внешние сервисы (API Avito)* — сторонние сервисы для получения объявлений о сдаче квартир.

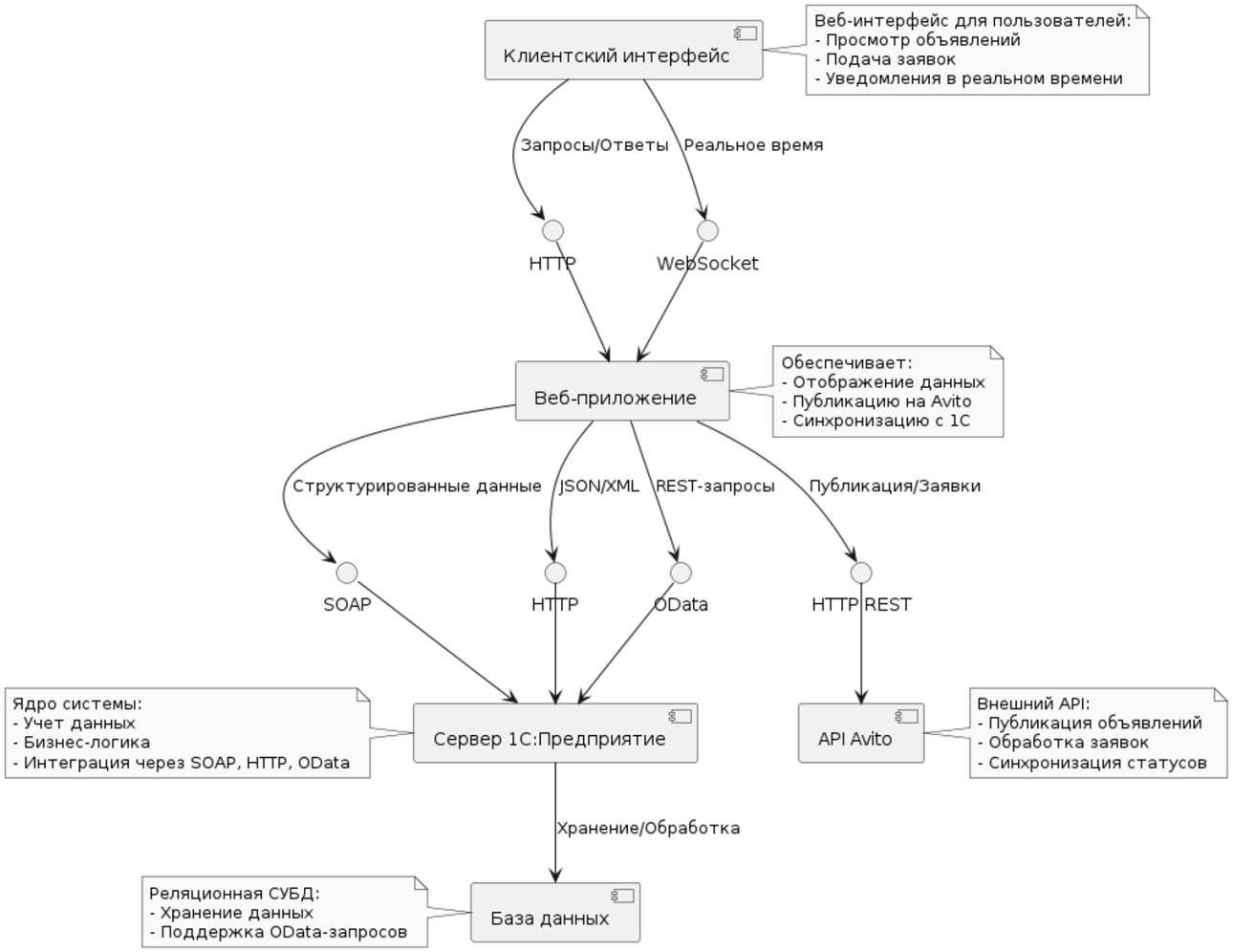

Рис. 1. Схема архитектуры интеграции

• *Клиентский интерфейс* — пользовательские приложения для просмотра объявлений и работы с системой.

2. Модели интеграции

В системе реализованы четыре модели интеграции, каждая из которых применяется для решения различных задач.

2.1. Интеграция на основе SOAP

SOAP (Simple Object Access Protocol) — протокол обмена структурированными данными на основе XML, используемый для передачи данных между 1С:Предприятие и веб-приложением. Обеспечивает надежность и стандартизацию, подходит для обмена справочниками.

В системе реализован веб-сервис SOAP_ОбменКвартирами, предоставляющий методы:

• flatlist — получение списка квартир;

• updateflat — обновление данных о квартирах.

Методы используют XDTO для сериализации данных в XML-формат, что обеспечивает строгую типизацию и валидность передаваемой информации.

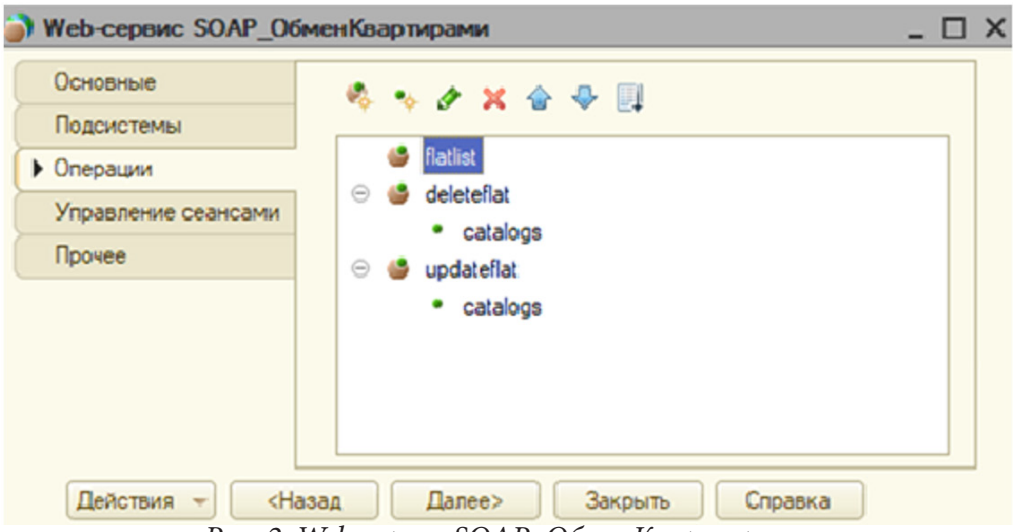

Рис. 2. Web-сервис SOAP_ОбменКвартирами

2.2. Интеграция на основе HTTP REST

HTTP используется для реализации REST API, обеспечивающего легковесный обмен данными между 1С:Предприятие, веб-приложением и Avito. В системе применяются HTTP-сервисы для передачи данных в формате JSON.

- Реализован HTTP-сервис HTTPСотрудники для обмена данными о сотрудниках:
- СотрудникиПолучитьСотрудников — получение списка сотрудников;
- СотрудникиИзменитьСотрудников — обновление данных о сотрудниках;
- СотрудникиУдалитьСотрудника — удаление сотрудника.

Интеграция с API Avito реализована через HTTP POST-запросы по HTTPS для получения информации об объявлениях в формате JSON.

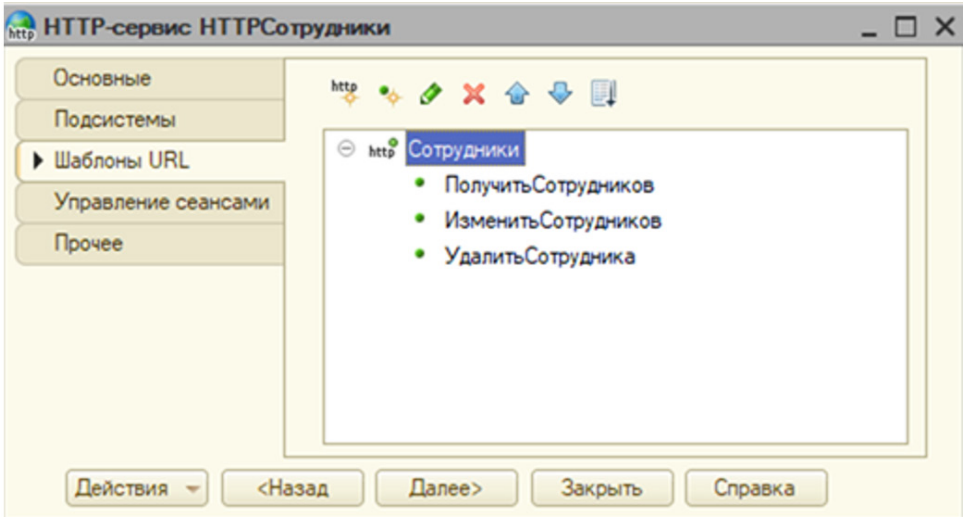

Рис. 3. HTTP-сервис HTTPСотрудники

2.3. Интеграция на основе OData

OData (Open Data Protocol) — стандартизированный протокол на основе REST, предоставляющий доступ к данным 1С:Предприятие через HTTP-запросы. Позволяет внешним приложениям запрашивать данные в удобном формате с возможностью фильтрации и сортировки.

В системе настроен встроенный OData-сервис платформы 1С:Предприятие для предоставления доступа к документам объявлений. Веб-приложение использует стандартизированные запросы OData ($filter, $orderby) для выборки данных с фильтрацией по дате и другим параметрам.

Пример запроса:

```
/ArendApartaments/odata/standard.odata/Catalog_Квартиры?$format=json&$filter=Date ge 2025-06-01 and Date le 2025-06-30
```

2.4. Интеграция на основе WebSocket

WebSocket обеспечивает двунаправленное соединение для передачи данных в реальном времени между клиентом и сервером. В системе используется для чата между агентами и обмена справочником агентств недвижимости.

Реализован WebSocket-клиент WS_WebSocketTest для:

- Синхронизации справочника агентств недвижимости в реальном времени;
- Реализации чата между агентами недвижимости.

Установление постоянного соединения позволяет мгновенно передавать изменения данных и сообщения между клиентами без необходимости периодических запросов.

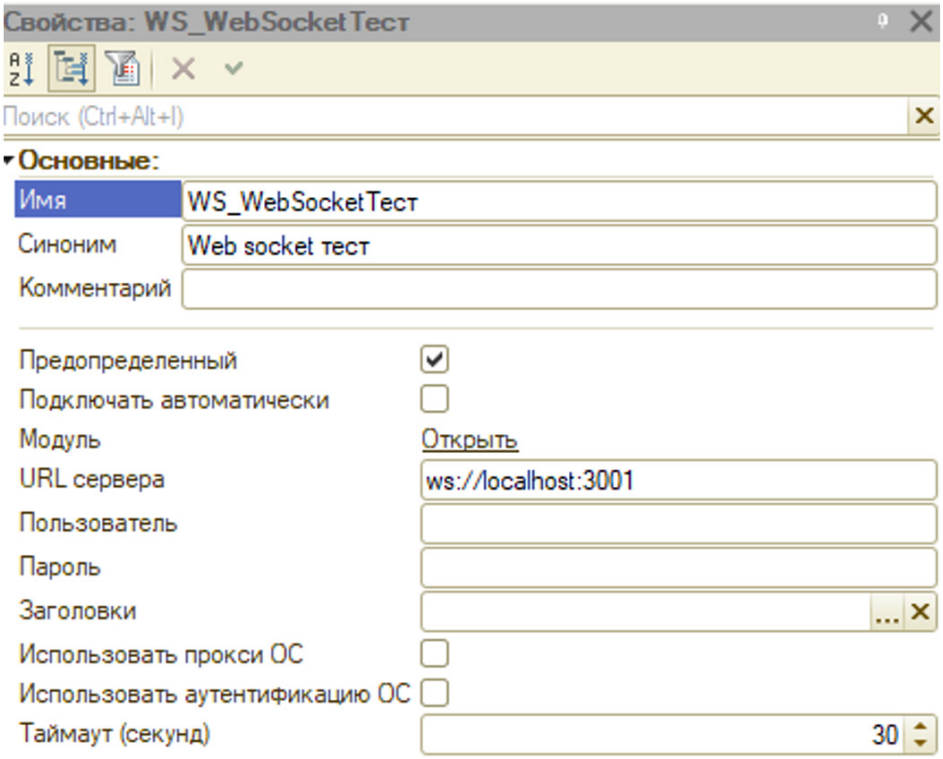

Рис. 4. Web-socket клиент WS_WebSocketTest

3. Реализация интеграции

Для обеспечения интеграции конфигурации «ArendApartaments» с внешними системами разработаны серверные модули и обработки, поддерживающие различные протоколы веб-обмена.

Реализация веб-обмена разделена на несколько направлений:

- Публикация объявлений о сдаче квартир;
- Синхронизация данных о сотрудниках через HTTP и OData;
- Обмен данными о квартирах через SOAP;
- Взаимодействие через WebSocket для чата и синхронизации справочников.

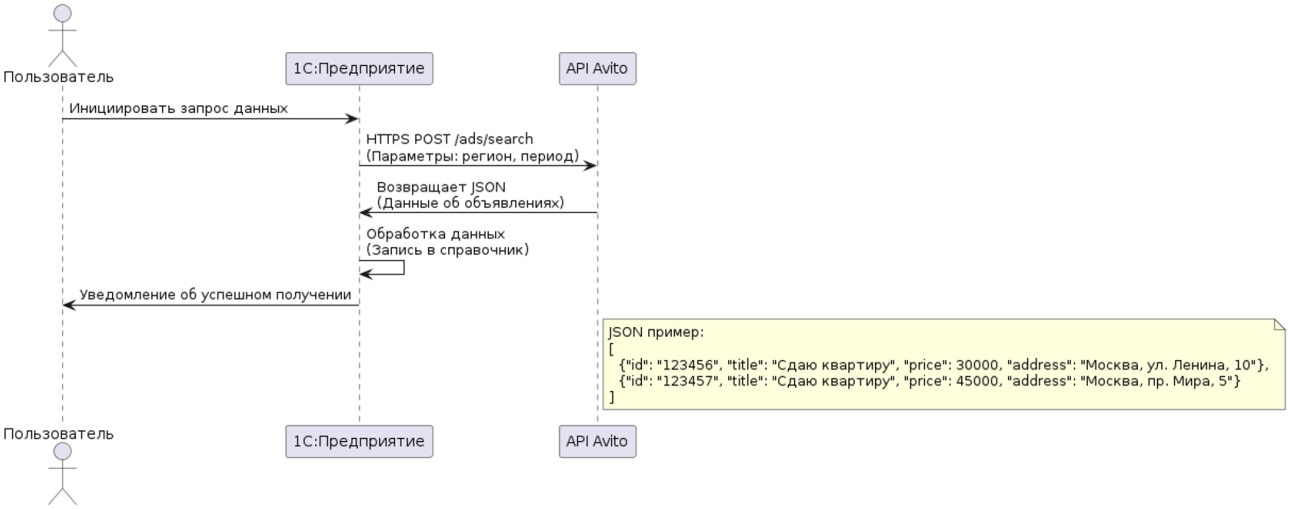

Рис. 5. Схема взаимодействия пользователя, 1С:Предприятия и API Avito

4. Тестирование системы

Были проведены следующие тестовые сценарии:

- Получение объявлений с Avito через HTTP REST API.
- Отправка и получение данных через WebSocket.
- Обмен квартирами через SOAP-сервис.
- Получение данных через OData-интерфейс.
- Проверка работы регистров накопления и сведений.

Тестовые сценарии подтвердили стабильность работы системы, включая корректную синхронизацию данных и обработку запросов.

В перспективе возможно расширение функциональных возможностей: подключение других сервисов объявлений (ЦИАН), углубленная аналитика, автоматическая генерация договоров, внедрение мобильного приложения.

Заключение

Платформа «1С:Предприятие» является открытой системой. Она предоставляет возможность для интеграции практически с любыми внешними программами и оборудованием на основе общепризнанных открытых стандартов и протоколов передачи данных. Она поддерживает в прикладных решениях возможность создания web- и HTTP-сервисов и работу с внешними web- и HTTP-сервисами. Поддерживается доступ внешних систем к данным приложений 1С по протоколу OData.

Универсальный механизм обмена данными позволяет организовать взаимодействие с различными информационными системами, в том числе реализованными не на платформе «1С:Предприятие».

Литература

1. *Гаранин М. А.* Разработка и администрирование 1С:Предприятие 8.3. – М. : ДМК Пресс, 2022.

2. *Козырев А. В.* Интеграция 1С:Предприятие с веб-сервисами. – СПб. : Питер, 2021.

3. Документация 1С:Предприятие 8.3. – https://v8.1c.ru

4. Avito API Documentation. – https://www.avito.ru/api

5. OData Version 4.0 — https://www.odata.org

УДК 004.9:159.9

СИСТЕМА АВТОМАТИЗИРОВАННОГО СОПРОВОЖДЕНИЯ ПСИХОЛОГИЧЕСКОГО КОНСУЛЬТИРОВАНИЯ НА ОСНОВЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ С ФУНКЦИЕЙ АНАЛИТИКИ

С. С. Лавлинская

Воронежский государственный университет

Аннотация. Статья представляет проект разработки мобильного приложения для автоматизации работы психолога. Рассматриваются перспективы создания iOS-приложения с использованием SwiftUI и облачной платформы Firebase. Особое внимание уделяется проектированию архитектуры системы, включающей модули психодиагностики с применением валидных методик (шкала Бека, опросник Спилбергера — Ханина) и интеллектуального формирования рекомендаций. Определены направления дальнейшего развития системы, включающие расширение банка диагностических методик, создание комплексной аналитической платформы для отслеживания динамики психологического состояния клиентов.

Ключевые слова: мобильное приложение, психологическое консультирование, iOS разработка, облачные технологии, психодиагностика, автоматизация, персонализированный подход, доказательный подход, этические принципы разработки, социальное значение, терапевтическая интервенция.

Введение

В современном мире люди всё чаще сталкиваются с необходимостью получения психологической помощи. Для удобства клиентов создано множество различных платформ по поиску специалиста. Внутри этих платформ у самих психологов есть средства, которые могут упрощать ведение сессий и отслеживание динамики их клиентов. Но далеко не все психологи работают в подобных компаниях. Многие работающие в обычных клиниках или работающие «на себя» не могут с такой же эффективностью анализировать свой график и состояния клиентов. Поэтому существует необходимость создания приложения, которое позволило бы любому психологу получить такие преимущества.

Психологическое консультирование включает в себя множество аспектов. Специалисту приходится иметь дело с большим количеством информации от множества людей. Каждому из них необходимо качественно оказывать услуги. Для более четкого понимания проблемы, с которой обращается человек, нужно проводить тестирования, которые в настоящее время, чаще всего приходится проводить вручную: печатать вопросы теста, подсчитывать и анализировать результаты, а также хранить в дальнейшем эти данные, чтобы воспользоваться ими для оценки динамики. Также этот способ не подразумевает мгновенного результата, что продляет срок терапии и усложняет процесс консультирование, а также занимает личное время психолога. Данный подход является устаревшим и требует автоматизации. Реализовать подобную систему удобнее всего в виде мобильного приложения, так как это то устройство, которое всегда есть и просто в использовании.

Целью данной работы является создание системы автоматизированного анализа результатов психологических тестирований и отслеживание динамики состояний клиентов.

1. Проектирование системы

1.1. Психологическая основа системы

Разработка цифрового инструмента для психологической практики требует строгого соблюдения методологических принципов психодиагностики. Основополагающим требованием является использование стандартизированного диагностического инструментария, обеспечивающего валидность и надежность измерений. Валидность понимается как соответствие теста измеряемому психологическому конструкту, а надежность — как устойчивость результатов к воздействию случайных факторов.

В качестве диагностического ядра системы были выбраны следующие методики:

1. *Шкала депрессии Бека (Beck Depression Inventory)* — стандартизированный опросник для количественной оценки выраженности аффективной симптоматики.

2. *Шкала тревожности Спилбергера — Ханина* — методика, реализующая диспозиционный подход к измерению тревожности через дифференциацию личностной и ситуативной составляющих.

3. *Опросник выгорания MBI (Maslach Burnout Inventory)* — инструмент, основанный на трехфакторной модели профессионального выгорания.

Ключевым аспектом проектирования стала операционализация психологических конструктов — перевод абстрактных понятий в измеримые показатели. Каждый тест транслирует сырые баллы в стандартизированные диагностические профили, например: «умеренный уровень депрессивной симптоматики» или «высокая реактивная тревожность». Формирование рекомендаций базируется на принципе доказательности (evidence-based practice) в психологии. Это предполагает, что связь между диагностированным конструктом и рекомендуемым ресурсом должна быть обоснована эмпирическими исследованиями и соответствовать параметрам научной обоснованности.

1.2. Архитектурные решения и техническая реализация

Проектом предусматривается создание системы с многоуровневой архитектурой, где клиентская часть будет реализована на платформе iOS с использованием фреймворка SwiftUI. Архитектура MVVM (Model-View-ViewModel) позволит обеспечить четкое разделение между логикой представления и бизнес-логикой.

На серверной стороне планируется использование облачной платформы Firebase, которая предоставляет возможности для:

- хранения структурированных данных в NoSQL базе Cloud Firestore;
- реализации бизнес-логики через бессерверные функции (Cloud Functions);
- обеспечения безопасности данных через встроенные механизмы аутентификации и авторизации;

Проектом предусматривается создание модуля аналитики, способного обрабатывать данные в нескольких разрезах: индивидуальная динамика психологических показателей, статистические закономерности в группах клиентов, эффективность различных типов рекомендаций.

Для реализации этих возможностей может быть использована событийно-ориентированная архитектура (Event-Driven Architecture), где каждое взаимодействие с системой инициирует каскад событий обработки данных.

1.3. Интеллектуальный модуль рекомендаций: методология и реализация

Психологическая составляющая модуля базируется на принципах персонализированного подхода в психологическом консультировании. Алгоритм функционирует как система под-

держки принятия решений (Decision Support System), обеспечивая релевантность предлагаемых материалов через установление семантических связей между диагностическими конструктами и терапевтическими ресурсами.

Техническая реализация использует концепцию событийно-ориентированной архитектуры (Event-Driven Architecture). Сохранение результатов тестирования инициирует каскад событий: *триггеринг* облачной функции при изменении состояния данных, параллельный запрос к базе рекомендаций с использованием предикатов совпадения, ранжирование результатов по метрике релевантности. Процесс ранжирования реализует алгоритм TF-IDF (Term Frequency — Inverse Document Frequency), адаптированный для психологической предметной области. Вес рекомендации вычисляется как функция от частоты встречаемости диагностических маркеров и их специфичности для конкретного психологического состояния.

Для формализации функциональных требований к системе была разработана Use-Case диаграмма (рис. 1), которая наглядно демонстрирует ключевые сценарии взаимодействия психолога с приложением.

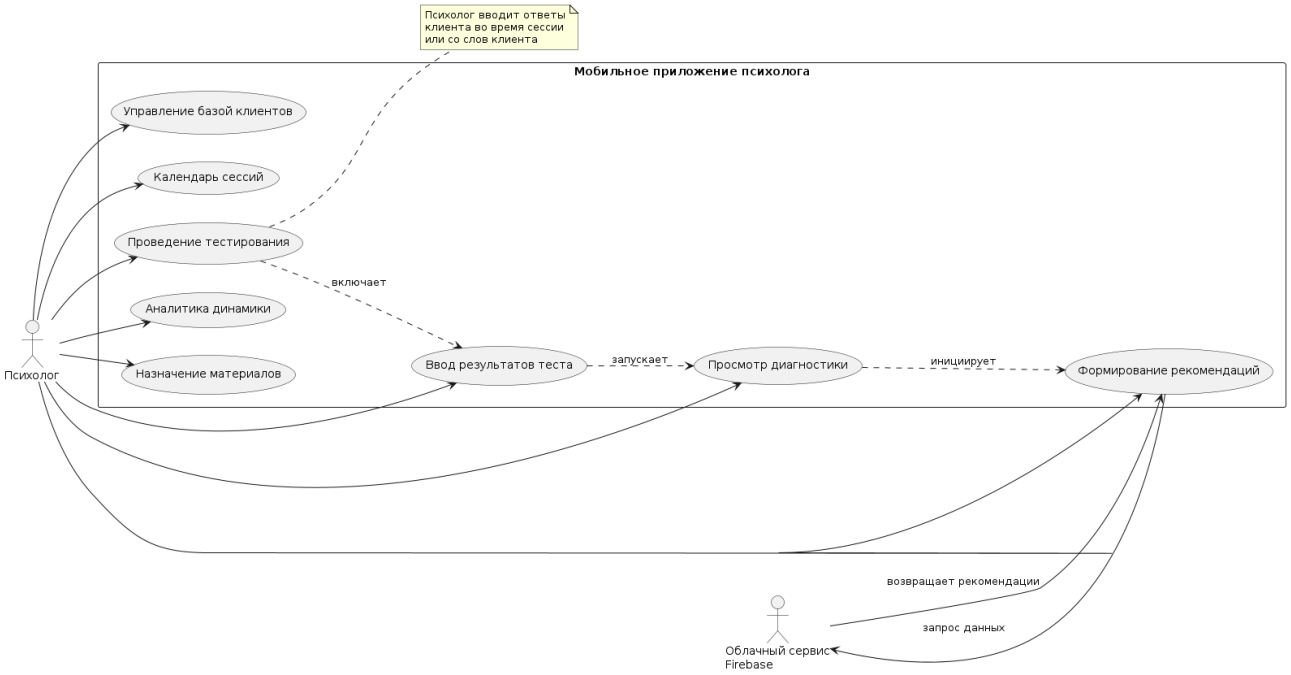

Рис. 1. Use-Case диаграмма мобильного приложения для психолога

Как видно из диаграммы, система поддерживает полный цикл работы специалиста: от управления клиентской базой и планирования сессий до проведения диагностики и анализа результатов. Особенностью архитектуры является разделение процесса тестирования на два последовательных прецедента: непосредственное проведение тестирования и ввод результатов, что отражает реальную практику, когда психолог сначала проводит сессию, а затем фиксирует данные в системе.

Для обеспечения психолога инструментами отслеживания динамики состояния клиентов был разработан специализированный модуль аналитики (рис. 2).

Как демонстрирует диаграмма, процесс аналитики начинается с выбора психологом конкретного клиента в мобильном приложении. Система автоматически запрашивает из облачной базы данных всю историю тестирования, включая результаты различных психодиагностических методик, применявшихся в ходе терапии.

Ключевой особенностью модуля является возможность интерактивного выбора временного периода для анализа. Это позволяет психологу изучать как общую динамику состояния

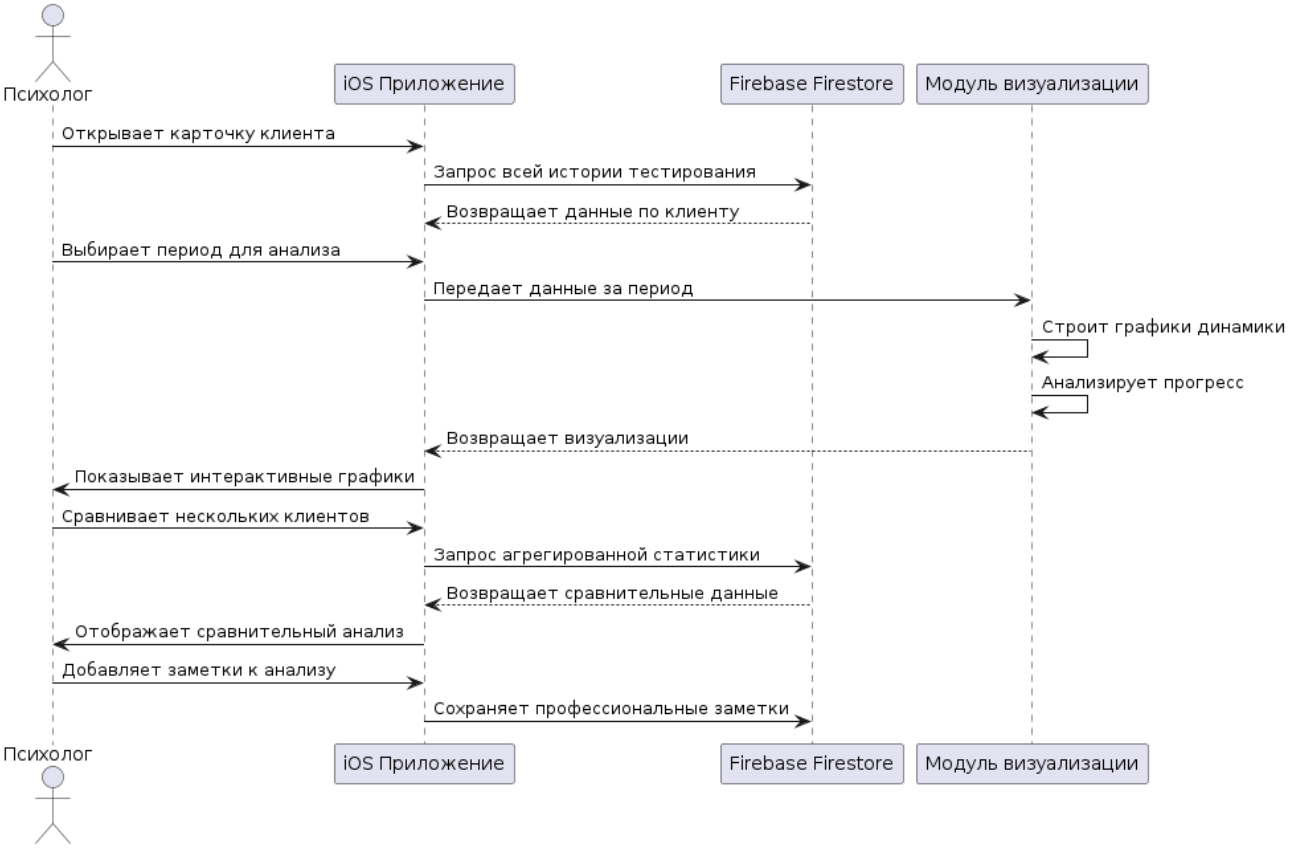

Рис. 2. Диаграмма последовательности процесса аналитики для психолога

клиента за весь период терапии, так и анализировать конкретные временные отрезки, что особенно важно для оценки эффективности отдельных терапевтических интервенций.

Диаграмма также показывает возможность сравнительного анализа нескольких клиентов. Эта функция обеспечивает конфиденциальность данных — психолог видит только агрегированную статистику без персональных идентификаторов. Такой подход позволяет специалисту выявлять общие закономерности в группе клиентов, оценивать эффективность различных терапевтических подходов, формировать статистическую базу для супервизии и профессионального роста.

Завершающим этапом процесса является возможность добавления профессиональных заметок к аналитическому отчету, которые сохраняются в облачной базе данных и могут быть использованы для последующего анализа.

2. Потенциальная эффективность и этические аспекты

Разрабатываемое приложение оптимизирует работу психолога, так как его использование экономит время за счет автоматизации рутинных операций, позволяя специалисту сосредоточиться на непосредственно работе с клиентом. Также стандартизация диагностики обеспечивает сопоставимость результатов на разных этапах терапии. Визуализация динамики состояния клиента помогает отслеживать эффективность терапии, что является наглядной аналитикой.

При создании приложения учитываются важнейшие *этические принципы разработки*:

• ответственное использование данных - система разрабатывается с соблюдением принципа «не навреди». Все персональные данные клиентов защищаются многоуровневой системой шифрования;

• профессиональный контроль — приложение позиционируется как помощник психолога, а не замена специалисту. Окончательные решения по интерпретации результатов и выбору методик остаются за человеком;

• прозрачность алгоритмов — психолог всегда может понять логику формирования рекомендаций и при необходимости скорректировать их;

• конфиденциальность — реализуется строгое разграничение доступа к информации, исключающее возможность несанкционированного доступа к данным клиентов.

Проект имеет важное *социальное значение*, поскольку способствует:

• повышению доступности качественной психологической помощи;

• снижению нагрузки на специалистов;

• развитию цифровой культуры в сфере психического здоровья;

• система создает основу для формирования нового стандарта работы в психологической практике, сочетающего профессиональную экспертизу и современные технологические возможности.

Заключение

В ходе проведенного исследования был спроектирован и всесторонне обоснован проект мобильной системы автоматизированного сопровождения психологического консультирования. Предложенное решение интегрирует в единый цифровой инструмент ключевые аспекты работы практикующего психолога: от административных функций управления расписанием и клиентской базой до сложных задач психодиагностики и аналитики.

Архитектурные решения на основе iOS-платформы и облачных технологий Firebase обеспечивают не только техническую реализуемость проекта, но и соответствие методологическим требованиям современной психодиагностики. Использование действенного диагностического инструментария в сочетании с алгоритмами формирования рекомендаций создает основу для внедрения доказательного подхода в повседневную практику психолога.

Важнейшим результатом работы стало проектирование модуля аналитики, предоставляющего специалисту инструменты для объективной оценки динамики состояния клиентов и эффективности терапевтических интервенций. Внедрение такой системы позволит не только оптимизировать рабочее время психолога за счет автоматизации рутинных операций, но и повысить качество оказываемой помощи через использование стандартизированных протоколов и персонализированных рекомендаций.

Предложенная система открывает возможности для формирования нового стандарта цифрового инструментария психолога, сочетающего профессиональную экспертизу и современные технологические решения.

Литература

1. *Усов В. А.* Swift. Основы разработки приложений под iOS и macOS. – Санкт-Петербург : Питер, 2018. – 448 с.

2. *Исаев Г. Н.* Проектирование информационных систем. Учебное пособие. – Москва : Омега–Л, 2015. – 424 с.

3. *Мак-Вильямс Н.* Психоаналитическая диагностика: Понимание структуры личности в клиническом процессе. – Москва : Независимая фирма «Класс», 1998. – 480 с.

4. *Истратова О. Н.* Психодиагностика. Коллекция лучших тестов. – Ростов: Феникс, 2006. – 375 с.

5. *Шапарь В. Б.* Практическая психология. – Ростов : Феникс, 2010. – 672 с.

6. Firebase for Apple platforms [Электронный ресурс] // Firebase Documentation / Google. – URL: https://firebase.google.com/docs/ios/setup (дата обращения: 14.11.2025)

УДК 004.42

РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ УЧЕТА ТРЕНИРОВОК И СТРЕЛЬБ СПОРТСМЕНОВ

А. Н. Леденев

Воронежский государственный университет

Аннотация. В работе описан процесс разработки клиентской части мобильного приложения для учёта тренировок и стрельб спортсменов. Рассмотрены выбор платформы Flutter и обоснование кроссплатформенного подхода. Представлено проектирование пользовательского интерфейса с учётом адаптивности, интуитивности и поддержки светлой и тёмной темы. Описана реализация ключевых модулей: управление профилем пользователя, учёт спортивного оружия, интерактивный аудиотренажёр и просмотр PDF-документов. Рассмотрена архитектура приложения, включая модульную структуру и систему валидации данных. Приложение обеспечивает автономную работу и удобство взаимодействия. Полученный результат может служить основой для дальнейшего расширения функциональности и интеграции с серверной частью.
Ключевые слова: мобильное приложение, спортивная стрельба, кроссплатформенная разработка, пользовательский интерфейс, цифровизация спорта.

Введение

Современный спортивный процесс требует систематизации данных о тренировках и результатах стрельбы. Ведение статистики позволяет отслеживать динамику подготовленности спортсмена, выявлять слабые стороны и повышать эффективность тренировочного процесса.

Для решения этой задачи было разработано мобильное приложение, предназначенное для учёта тренировок и стрельб спортсменов. Клиентская часть приложения обеспечивает взаимодействие пользователя с системой: ввод и отображение данных, управление записями, а также доступ к учебным материалам и тренировочным сценариям.

Разработка клиентской части требует не только продуманного интерфейса, но и корректной архитектуры, обеспечивающей стабильность и удобство работы. В приложении реализованы ключевые функции: просмотр документов в формате PDF, звуковой тренажёр для воспроизведения сценариев тренировок.

Создание такого приложения имеет прикладное значение, так как способствует цифровизации спортивной подготовки и формированию единой системы анализа тренировочных данных. Технические решения, архитектура и функциональные особенности приложения рассматриваются в последующих разделах работы.

1. Средства реализации и выбор платформы разработки

Разработка клиентской части мобильного приложения велась с ориентацией на кроссплатформенную архитектуру. Такой подход позволяет использовать единый программный код для операционных систем Android и iOS, что существенно снижает трудозатраты и упрощает поддержку проекта.

При выборе технологической платформы учитывались следующие критерии: удобство проектирования пользовательского интерфейса, поддержка адаптивной вёрстки, наличие инструментов для реализации тем оформления, а также доступность документации и активного сообщества разработчиков.

На этапе анализа рассматривались три основных решения: React Native, Xamarin и Flutter. Фреймворк React Native [6], основанный на JavaScript, был отклонён из-за необходимости дополнительного изучения синтаксиса и архитектуры, что увеличило бы сроки реализации. Платформа Xamarin [8], использующая язык C# и фреймворк .NET, показала ограничения в части интерфейсной унификации — разработка визуальной части для каждой платформы выполняется отдельно.

Наиболее подходящим решением признан фреймворк Flutter [2], основанный на языке Dart. Он обеспечивает высокую производительность, широкие возможности интерфейсного проектирования, поддержку «горячей перезагрузки» и встроенные средства адаптивной вёрстки. Простота освоения языка и развитая экосистема инструментов сделали Flutter оптимальным выбором для реализации клиентской части приложения.

2. Проектирование пользовательского интерфейса

Проектирование интерфейса мобильного приложения ориентировано на принципы интуитивности и лаконичности. Особое внимание уделено формированию визуальной иерархии элементов, обеспечивающей быстрое освоение интерфейса и комфортное взаимодействие даже при первом запуске. Интерфейс адаптируется под различные размеры экранов и поддерживает смену светлой и тёмной тем, что повышает доступность приложения в разных условиях освещённости.

В качестве одного из элементов повышения удобства использования применена система тактильной отдачи: все ключевые пользовательские действия — нажатия кнопок, подтверждения, ошибки ввода — сопровождаются лёгкой вибрацией устройства. Это решение обеспечивает дополнительную обратную связь и улучшает пользовательский опыт.

2.1. Структура интерфейса и основные экраны

Пользовательское взаимодействие начинается с загрузочного экрана, включающего анимацию логотипа приложения, после которой выполняется переход на стартовый экран. Здесь пользователю предлагается создать профиль и ознакомиться с лицензионным соглашением (рис. 1). Реализована проверка корректности заполнения формы: при отсутствии обязательных данных выводится сообщение об ошибке, что обеспечивает надёжность ввода информации.

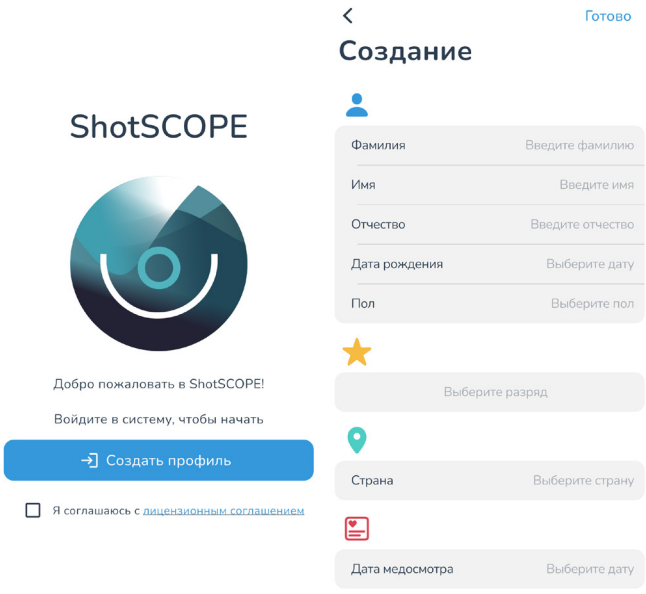

Рис. 1. Стартовый экран и экран создания профиля

После создания профиля открывается главный экран, являющийся центральным узлом навигации по разделам приложения. На нём представлены карточки перехода к основным функциям: «Моё оружие», «Электронная мишень», «Всё о стрельбе», «Тренажёр» и «Настройки». В верхней части экрана размещена иконка редактирования профиля, а также система уведомлений о сроках действия медицинского осмотра. Цветовая индикация (синяя и красная) позволяет пользователю быстро оценить актуальность данных (рис. 2).

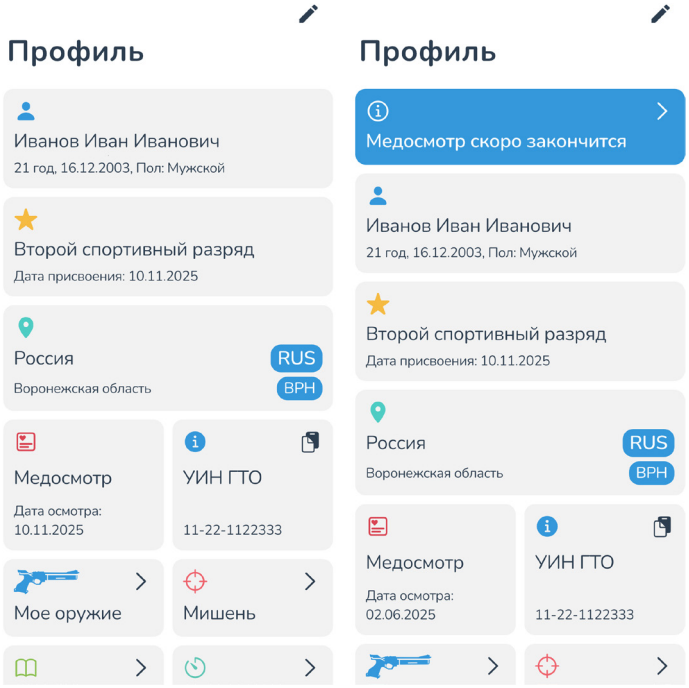

Рис. 2. Главный экран и уведомление об окончании медосмотра

Раздел «Моё оружие» предназначен для учёта спортивного инвентаря. Каждая карточка содержит сведения о типе оружия, названии, серийном номере, калибре и связанном упражнении. Добавление, редактирование и удаление данных выполняется при помощи интуитивных жестов и подтверждений. Дополнительно предусмотрен экран выбора приоритетного оружия по упражнениям (рис. 3).

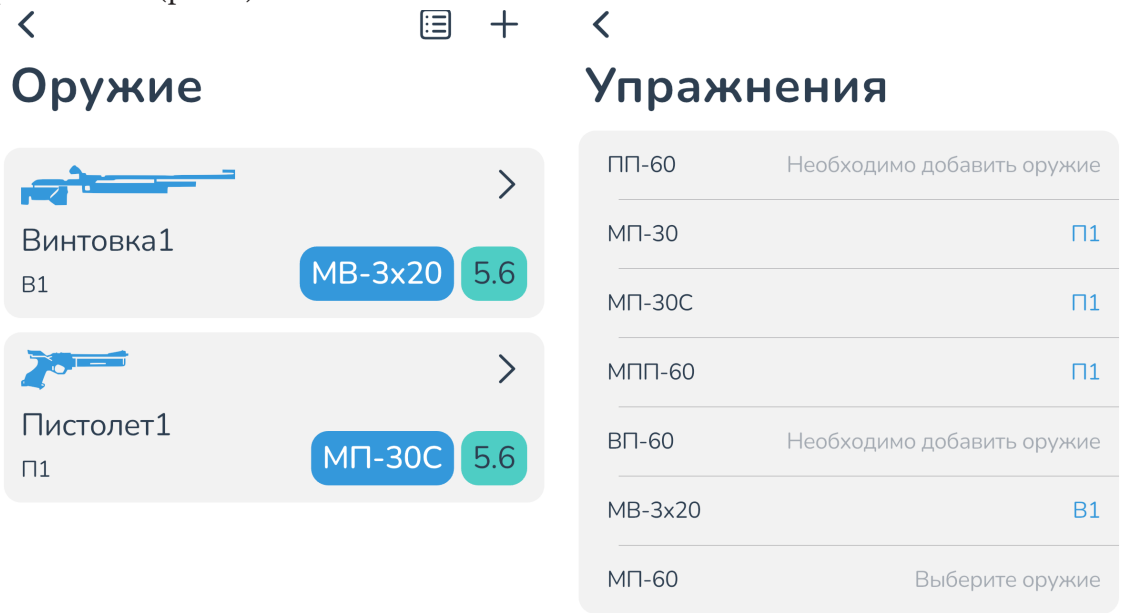

Рис. 3. Страница «Моё оружие» и выбор приоритетного оружия

На странице «Всё о стрельбе» пользователь получает доступ к методическим материалам и нормативным документам. Встроенный просмотрщик PDF-файлов обеспечивает удобное чтение без использования сторонних приложений.

Особое место занимает раздел «Тренажёр», реализующий аудиоплеер для тренировки по звуковым сигналам (рис. 4). В течении воспроизведения аудиофайла цвет индикатора изменяется в соответствии с звуковыми сигналами. Пользователь может выбрать упражнение и режим отображения: круговой индикатор или полноэкранный фон. Поддерживаются функции перемотки, паузы, повтора и запуска с момента «внимание».

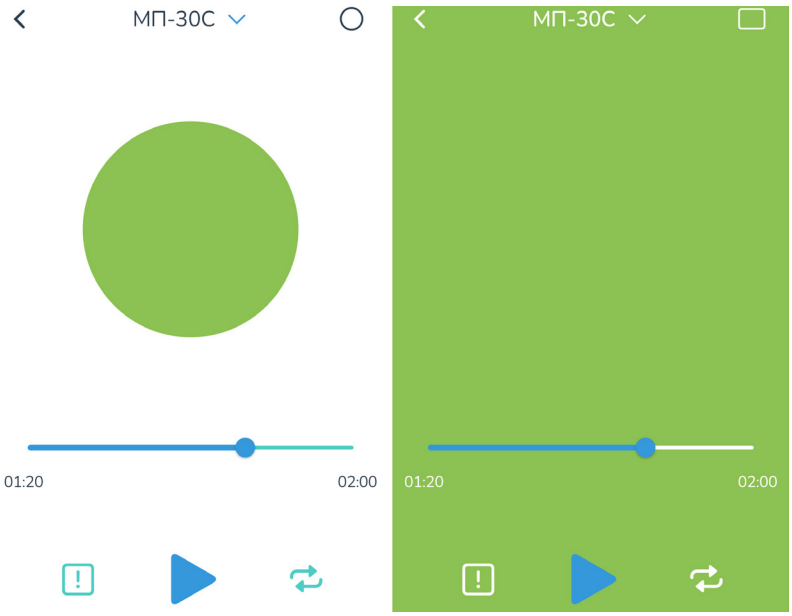

Рис. 4. Интерфейс звукового тренажёра

Раздел «Настройки» обеспечивает выбор темы оформления (системная, светлая, тёмная) и доступ к информации «О приложении». Также реализована функция выхода из профиля с переходом на экран выбора аккаунта, где можно активировать, удалить или создать новый профиль.

2.2. Принципы проектирования

В процессе проектирования интерфейса использовались принципы визуальной последовательности и минимализма. Каждая страница приложения содержит не более одного основного действия, что снижает когнитивную нагрузку и повышает скорость ориентации пользователя. Цветовые акценты применяются умеренно и функционально, подчёркивая активные элементы и состояния.

Интерфейс приложения сочетает эстетическую целостность с функциональной завершённостью. Адаптивный дизайн, визуальная и тактильная обратная связь, а также структурированная навигация делают систему удобной, надёжной и ориентированной на практическое использование в спортивной подготовке.

3. Реализация

Разработка клиентской части мобильного приложения велась с использованием модульного подхода, что обеспечило логическую структуризацию кода, повышение читаемости и упрощение последующей поддержки. Каждая функциональная часть приложения реализована в виде отдельного модуля, включающего три ключевых элемента: services, views и components.

Папка *services* содержит классы, реализующие бизнес-логику экранов и взаимодействие с источниками данных — как локальными, так и удалёнными. В частности, в данном модуле выполняется загрузка профиля пользователя, управление данными об оружии и работа с аудиофайлами.

В каталоге *views* размещаются основные файлы экранов, определяющие структуру, навигацию и визуальное отображение интерфейса. Каждый экран оформлен в виде отдельного виджета и имеет собственный файл, что способствует изоляции логики.

Папка *components* включает переиспользуемые элементы интерфейса — карточки, кнопки, поля ввода, иконки и другие визуальные компоненты, адаптированные под дизайн и логику конкретных экранов.

3.1. Реализация светлой и темной темы интерфейса

В приложении реализована поддержка светлой и тёмной темы, что обеспечивает комфортное взаимодействие в различных условиях освещённости и соответствует современным требованиям к дизайну мобильных интерфейсов. Переключение темы выполняется посредством специального класса *ThemeManager*, использующего механизм *ValueNotifier*. Это позволяет динамически перестраивать интерфейс через *ValueListenableBuilder* без перезапуска приложения.

Все цвета, шрифты и параметры стиля определены централизованно, что обеспечивает единообразие внешнего вида и облегчает модификацию оформления при расширении функциональности.

3.2. Формы и валидация данных

Для работы с пользовательским вводом реализована система форм, основанная на стандартных средствах Flutter с расширенной логикой валидации. Основным элементом является компонент *RowField*, построенный на основе *CupertinoTextFormFieldRow*. Он поддерживает настройку маски ввода, ограничение пробелов, тип клавиатуры, начальные значения и логику проверки.

Валидация выполняется при потере фокуса и контролируется через *GlobalKey<FormState>*. Метод *validate()* обеспечивает централизованную проверку всех полей формы перед сохранением данных. Форматирование и фильтрация выполняются при помощи *MaskTextInputFormatter* [5] и *FilteringTextInputFormatter*.

3.3. Главный экран приложения

Главный экран реализован на основе *StatefulWidget*, что обеспечивает возможность динамического обновления содержимого при изменении данных профиля. Получение информации выполняется асинхронно через *ProfileService*, который взаимодействует с локальным репозиторием данных.

Информация о пользователе представлена в виде карточек (*ProfileCard*), а при необходимости — сопровождается предупреждающими сообщениями (*PushCard*). Для ручного обновления данных реализован механизм *CupertinoSliverRefreshControl*, обеспечивающий интерактивное обновление содержимого при жесте «потянуть для обновления».

Все визуальные компоненты автоматически адаптируются к активной теме оформления, что создаёт целостный и согласованный внешний вид приложения.

3.4. Модуль «Мое оружие»

Модуль «Моё оружие» предназначен для учёта, отображения и редактирования информации об используемом спортсменом оружии. Он реализован с применением асинхронных запросов и компонент *FutureBuilder* и *SliverList*.

Пока данные загружаются, пользователю отображается индикатор состояния (*LoadingSample*). После загрузки формируется список карточек (*WeaponCard*), каждая из которых содержит основные сведения: название, серийный номер, тип и дисциплину. Поддерживаются действия свайпа для удаления и перехода на детальную страницу (*DetailsWeaponPage*).

Добавление нового оружия выполняется через экран *CreateWeaponPage*, после чего происходит обновление списка посредством *setState*. Для удобства реализовано обновление данных с помощью жеста «потянуть вниз» (*CupertinoSliverRefreshControl*).

3.5. Модуль «Тренажёр»

Модуль «Тренажёр» представляет собой интерактивный аудиоплеер, предназначенный для воспроизведения тренировочных сценариев стрельбы. Для обработки аудиофайлов используется библиотека *just_audio* [4], обеспечивающая управление воспроизведением, паузой, перемоткой и отслеживанием текущей позиции.

Каждому аудиофайлу соответствует JSON-файл с временными метками событий. В процессе воспроизведения приложение синхронизирует визуальные элементы с аудиопотоком: в зависимости от текущего времени изменяется цвет круга-индикатора — зелёный, красный или нейтральный, в зависимости от контекста сценария.

Реализованы функции начала воспроизведения с заданной метки, повтора сценария и выбора аудиофайла из списка. Для предотвращения автоматической блокировки экрана при активном воспроизведении используется *WakelockPlus*.

3.6. Работа с PDF-документами

Модуль просмотра PDF-документов (AboutShootingPage) предоставляет пользователю доступ к учебным материалам и методическим рекомендациям. При инициализации страницы выполняется асинхронный запрос через AboutShootingService, возвращающий список доступных файлов.

Загруженные документы отображаются в виде карточек (DocumentCard) с названием и иконкой перехода. При выборе элемента осуществляется переход на страницу PdfViewPage, где документ открывается во встроенном просмотрщике.

Заключение

В ходе реализации проекта была достигнута поставленная цель — разработана клиентская часть мобильного приложения для учёта тренировок и стрельб спортсменов.

Реализация основных модулей — управления профилями пользователей, учёта используемого оружия, навигации по разделам, а также поддержки светлой и тёмной темы — позволила сформировать функциональный и удобный инструмент для спортсменов. Дополнительно внедрены встроенный просмотр PDF-документов и страница справочной информации, оформленная в формате Markdown, что повышает информативность и удобство использования приложения.

Разработка велась с учётом принципов стабильности, расширяемости и эргономики интерфейса. Полученный результат может быть использован как основа для дальнейшего раз-

вития — в частности, интеграции с серверной частью и реализацией синхронизации данных между устройствами.

Работа выполнена под научным руководством старшего преподавателя кафедры программного обеспечения и администрирования информационных систем Воронежского государственного университета, Матвеевой Марии Валерьевной.

Литература

1. *Алеев А*. Быстрый старт Flutter-разработчика. – М. : Ridero, 2020. — 242 с.

2. Flutter [Электронный ресурс]. – Режим доступа: https://flutter.dev (дата обращения: 03.04.2025).

3. flutter_pdfview | Flutter Package [Электронный ресурс]. – Режим доступа: https://pub.dev/packages/flutter_pdfview (дата обращения: 08.05.2025).

4. just_audio | Flutter Package [Электронный ресурс]. – Режим доступа: https://pub.dev/packages/just_audio (дата обращения: 08.05.2025).

5. mask_input_formatter | Flutter Package [Электронный ресурс]. – Режим доступа: https://pub.dev/packages/mask_input_formatter (дата обращения: 08.05.2025).

6. React Native [Электронный ресурс]. – Режим доступа: https://reactnative.dev (дата обращения: 03.04.2025).

7. wakelock_plus | Flutter Package [Электронный ресурс]. – Режим доступа: https://pub.dev/packages/wakelock_plus (дата обращения: 08.05.2025).

8. Xamarin [Электронный ресурс]. – Режим доступа: https://dotnet.microsoft.com/ru-ru/apps/xamarin (дата обращения: 03.04.2025).

9. Android Developers. SpeechRecognizer. – Режим доступа: https://developer.android.com/reference/android/speech/SpeechRecognizer (дата обращения: 17.04.2025).

УДК 004.5

РЕАЛИЗАЦИЯ МАСШТАБИРУЕМОЙ МИШЕНИ В МОБИЛЬНОМ ПРИЛОЖЕНИИ FLUTTER С СОХРАНЕНИЕМ ТОЧНОСТИ КООРДИНАТ

А. А. Мальцев

Воронежский государственный университет

Аннотация. В статье рассматривается разработка механизма масштабирования и отображения мишени в мобильном приложении для анализа результатов пулевой стрельбы. Предложено архитектурное решение на основе комбинации виджетов OverflowBox и Transform с использованием кастомного контроллера масштабирования. Особое внимание уделено обеспечению точности координатной системы при выполнении жестов масштабирования и панорамирования. Описаны алгоритмы обработки пользовательского ввода и ограничения смещения мишени. Приведены результаты тестирования точности позиционирования и производительности системы.
Ключевые слова: мобильное приложение, Flutter, пулевая стрельба.

Введение

Современные технологии стремительно проникают во все сферы человеческой деятельности, включая спорт. Особое внимание уделяется цифровизации тренировочного процесса, где программные решения позволяют не только фиксировать результаты, но и проводить глубокий анализ показателей эффективности. В пулевой стрельбе, где каждая доля миллиметра имеет значение, использование мобильных приложений становится важным элементом подготовки спортсменов.

Традиционные методы ведения учета результатов – бумажные таблицы и простые электронные формы — оказываются недостаточно удобными и информативными. Они не позволяют оперативно визуализировать распределение попаданий и проводить детальный анализ техники стрельбы. В этих условиях разработка мобильного приложения, обеспечивающего наглядное отображение мишени, фиксацию выстрелов и анализ попаданий, является актуальной задачей.

Основное внимание в работе уделено решению проблемы корректного отображения мишени на экране мобильного устройства. Реальные размеры спортивных мишеней значительно превышают диагональ дисплея смартфона, что требует внедрения механизмов масштабирования и панорамирования без потери точности координат. Корректная реализация этих функций обеспечивает реалистичное соответствие между виртуальной моделью и физическими параметрами мишени [1].

Разрабатываемое программное решение направлено на создание интерактивного инструмента для стрелков и тренеров, позволяющего анализировать структуру попаданий, отслеживать прогресс и выявлять закономерности в выполнении серий.

Целью данной работы является разработка и реализация алгоритмов масштабирования, отображения и взаимодействия с мишенью в кроссплатформенном мобильном приложении на основе Flutter [2].

Для достижения поставленной цели решаются следующие задачи:

- обеспечение точного соответствия экранных координат физическим параметрам мишени;
- исключение искажений при масштабировании и перемещении элементов;
- обеспечение высокой производительности и отзывчивости интерфейса;
- создание архитектурной основы для последующей интеграции алгоритмов анализа попаданий.

1. Анализ существующих решений и выбор архитектурного подхода

Разработка масштабируемых графических интерфейсов представляет собой одну из наиболее распространенных задач в современной мобильной разработке. Во Flutter для решения этой задачи предусмотрен целый ряд стандартных виджетов, включая InteractiveViewer, Transform.scale, FittedBox и SingleChildScrollView. Однако практическое применение этих решений в задачах, требующих метрической точности координат, выявило существенные ограничения и недостатки [1, 3].

Проведенный детальный анализ существующих решений позволил выявить следующие ключевые проблемы:

InteractiveViewer, несмотря на кажущуюся простоту реализации жестов масштабирования и панорамирования, обладает фундаментальным недостатком — он трансформирует всю систему координат потомков. Это приводит к неизбежному рассогласованию модельных и экранных координат точек попаданий, что абсолютно недопустимо в задачах точной визуализации спортивных результатов. Дополнительным ограничением является сложность контроля за поведением компонента при экстремальных значениях масштаба.

Transform.scale и Transform.translate, предоставляя разработчику полный контроль над преобразованиями, требуют ручного пересчета координат для каждого дочернего элемента. Такой подход не только значительно увеличивает сложность кода, но и многократно повышает вероятность ошибок при позиционировании графических элементов. Особенно остро эта проблема проявляется при работе с динамически изменяющимся набором данных.

FittedBox, обеспечивающий автоматическое масштабирование контента под размеры контейнера, демонстрирует свою несостоятельность в условиях необходимости интерактивного изменения масштаба и панорамирования. Его статичная природа делает невозможным создание полноценного интерактивного интерфейса.

SingleChildScrollView, решая задачу прокрутки крупной мишени, оставляет без внимания критически важную функциональность масштабирования, что существенно ограничивает его применение в рассматриваемой предметной области.

Учитывая строгие требования к точности координат и производительности, а также необходимость обеспечения бесперебойной работы в условиях интенсивного пользовательского взаимодействия, было принято решение о разработке комбинированного решения на основе виджетов OverflowBox и Transform с использованием кастомного ScaleController.

Выбранный архитектурный подход обладает рядом принципиальных преимуществ:

- сохранение исходной системы координат мишени независимо от применяемых преобразований;
- обеспечение точного и предсказуемого управления масштабом и смещением;
- эффективный контроль за поведением графических элементов при выходе за границы видимой области;
- возможность тонкой настройки параметров преобразований в соответствии с требованиями конкретной спортивной дисциплины [4].

2. Реализация механизма масштабирования и отображения

2.1. Архитектурное решение на основе OverflowBox

Основой реализованного подхода является виджет OverflowBox, который обеспечивает принципиально важную возможность — сохранение дочерним элементом своих физических размеров, даже если они превышают габариты родительского контейнера. Это свойство обеспечивает независимость координатной системы мишени от преобразований, применяемых к

внешним слоям, что является ключевым требованием для обеспечения точности позиционирования.

Для отображения мишени используется виджет OverflowBox, который сохраняет физические размеры мишени независимо от преобразований. Параметры ширины и высоты задаются на основе модели Target, содержащей характеристики мишени для конкретной спортивной дисциплины. Модель включает диаметры зон, толщину линий, цветовую схему и другие визуальные параметры, соответствующие стандартам ISSF [4]. В качестве дочернего элемента используется CustomPaint с отрисовщиком TargetPainter, который визуализирует мишень и точки попаданий на основе переданных данных. Данный подход обеспечивает независимость координатной системы мишени от применяемых преобразований масштабирования и перемещения.

Важным аспектом реализации является обеспечение точного соответствия масштаба отображения. Алгоритм вычисления начального масштаба строится следующим образом: ширина экрана мобильного устройства делится на диаметр седьмой зоны мишени, который представляет собой одну из стандартных метрик в стрелковых дисциплинах. Седьмая зона мишени выбрана в качестве базового ориентира, поскольку ее размеры являются репрезентативными для обеспечения сбалансированного начального отображения всей мишени на экране устройства.

Данный метод расчета обеспечивает автоматическую адаптацию под различные размеры экранов мобильных устройств — от компактных смартфонов до крупноформатных планшетов. В результате мишень изначально отображается в таком масштабе, который позволяет пользователю одновременно охватить взглядом всю целевую область, сохраняя при этом достаточный уровень детализации для идентификации основных зон и элементов мишени.

2.2. Разработка ScaleController для управления преобразованиями

Для обработки пользовательских жестов масштабирования и панорамирования был разработан специализированный ScaleController, который инкапсулирует всю логику преобразований и обеспечивает согласованное состояние системы. Архитектура контроллера построена на принципах реактивного программирования и обеспечивает эффективное управление состоянием интерфейса [1, 3].

Контроллер реализует три основных метода обработки жестов:

• onScaleStart — фиксирует начальное состояние системы, запоминая текущий масштаб и позицию фокальной точки;

• onScaleUpdate — выполняет расчёт нового масштаба и смещения на основе жеста пользователя;

• onScaleEnd — завершает жест и обновляет конечное состояние системы.

Принцип работы системы заключается в последовательном выполнении нескольких взаимосвязанных операций.

На первом этапе вычисляется новое значение масштаба путем умножения предыдущего значения масштаба на коэффициент жеста пользователя. Полученное значение автоматически ограничивается заранее определенными границами минимального и максимального масштаба, что предотвращает чрезмерное увеличение или уменьшение мишени до нерабочих размеров.

Далее система рассчитывает коэффициент соотношения между новым и текущим масштабом. Этот коэффициент используется для корректировки текущего смещения мишени — значение смещения умножается на вычисленный коэффициент, что обеспечивает сохранение относительного положения видимой области мишени относительно ее центра при изменении масштаба.

После обновления значения масштаба происходит обработка смещения мишени. К текущему смещению добавляется величина изменения фокальной точки жеста, что позволяет ре-

ализовать функцию панорамирования — перемещения мишени вслед за движением пальцев пользователя.

Завершающим этапом является применение функции ограничения смещения, которая гарантирует, что мишень не может быть перемещена за пределы видимой области экрана. Алгоритм ограничения учитывает текущий масштаб отображения и физические размеры мишени, вычисляя допустимые границы смещения по осям X и Y, что обеспечивает стабильность интерфейса и предотвращает возникновение визуальных артефактов.

Данный многоступенчатый подход обеспечивает естественное и интуитивно понятное поведение мишени при жестах масштабирования и панорамирования, сохраняя при этом точность координатной системы и плавность анимации

Функция ограничения смещения обеспечивает стабильность работы интерфейса, предотвращая выход мишени за границы видимой области. Алгоритм ограничения строится на расчете допустимых границ перемещения с учетом текущего масштаба отображения и физических размеров мишени.

Вычисление граничных значений происходит по осям X и Y отдельно. Для каждой оси определяются минимальное и максимальное допустимые значения смещения, которые зависят от соотношения размеров экрана устройства и размеров мишени с учетом применяемого масштаба.

Функция принимает текущее значение смещения и возвращает скорректированное значение, в котором компоненты смещения по осям X и Y ограничены вычисленными границами. Это гарантирует, что мишень всегда остается в пределах области просмотра и не может быть смещена в положение, где пользователь теряет визуальный контакт с целевой областью.

2.3. Производительность

Обеспечение плавности интерфейса при частоте 60 кадров в секунду потребовало реализации комплекса мер по оптимизации производительности. В условиях интенсивного пользовательского взаимодействия и работы с большими объемами графических данных особое внимание было уделено следующим аспекта.

Изоляция слоёв через механизм RepaintBoundary позволила существенно снизить нагрузку на систему рендеринга. Статичные элементы мишени были заключены в отдельные области перерисовки, что ограничило зону обновления только динамическими компонентами (точками попаданий, индикаторами кластеров) [5]. Такой подход позволил сократить время перерисовки на 40–60 % в зависимости от сложности сцены.

Кэширование графических ресурсов было реализовано через специализированный SvgRepository, который обеспечивает однократную загрузку и парсинг SVG-изображений мишеней с последующим многократным использованием. Паттерн Singleton, положенный в основу репозитория, гарантирует единообразие данных и исключает дублирование ресурсов в памяти.

Использование ValueNotifier для управления состоянием интерфейса позволило минимизировать вызовы setState и обеспечить точечное обновление только тех компонентов, которые действительно требуют перерисовки. Реактивная архитектура обеспечила отзывчивость интерфейса даже в условиях интенсивной обработки жестов.

Дополнительной мерой оптимизации стало ленивое создание объектов, когда ресурсоемкие компоненты инициализируются только в момент первого обращения к ним. Это позволило сократить время запуска приложения и снизить потребление памяти на 15–20 %.

3. Тестирование и анализ результатов

Для всесторонней оценки эффективности разработанного решения был проведен комплекс тестов, включавший проверку точности координат, адаптивности интерфейса и производительности системы в различных условиях эксплуатации.

3.1. Оценка точности координатной системы

Основным критерием эффективности разработанного решения являлась точность позиционирования графических элементов. В рамках тестирования производилась серия операций масштабирования и панорамирования мишени с последующим добавлением тестовых выстрелов в заранее известные координаты.

Методика тестирования включала:

- последовательное масштабирование от минимального до максимального значения;
- панорамирование мишени по всем направлениям;
- добавление контрольных точек в узловые позиции мишени;
- сравнение фактических позиций с эталонными значениями после возврата к исходному масштабу.

Результаты тестирования продемонстрировали, что максимальная погрешность позиционирования не превышает 1 пикселя, что эквивалентно менее 0.2 мм на реальной мишени. Такой уровень точности полностью удовлетворяет требованиям спортивной аналитики и обеспечивает достоверность визуализации результатов стрельбы.

3.2. Тестирование адаптивности интерфейса

Адаптивность пользовательского интерфейса проверялась на широком спектре устройств — от компактных смартфонов с диагональю 5.5 дюймов до современных планшетов с диагональю 11 дюймов. В качестве тестовых устройств использовались Google Pixel 9 Pro XL, Pixel Tablet.

Критериями оценки адаптивности являлись:

- сохранение читаемости текстовых меток при всех разрешениях экрана;
- доступность элементов управления для взаимодействия;
- пропорциональность масштабирования всех графических элементов;
- сохранение функциональности при изменении ориентации устройства.

Результаты тестирования подтвердили корректное отображение всех элементов интерфейса на всех тестовых устройствах. Особое внимание было уделено работе на устройствах с нестандартными соотношениями сторон, где также не было выявлено критических проблем (рис. 1).

3.3. Анализ производительности системы

Производительность системы оценивалась в условиях, максимально приближенных к реальным сценариям использования. Тестирование включало измерение ключевых метрик при различных типах нагрузки.

- Время отклика на жесты — измерялась задержка между жестом пользователя и визуальной реакцией интерфейса. Среднее значение составило менее 80 мс, что соответствует требованиям к интерактивным приложениям [6].
- Время перерисовки при добавлении выстрела — оценивалась длительность процесса обновления интерфейса при добавлении новой точки попадания. Результат в 12 мс обеспечивает запас производительности для поддержания стабильных 60 FPS.

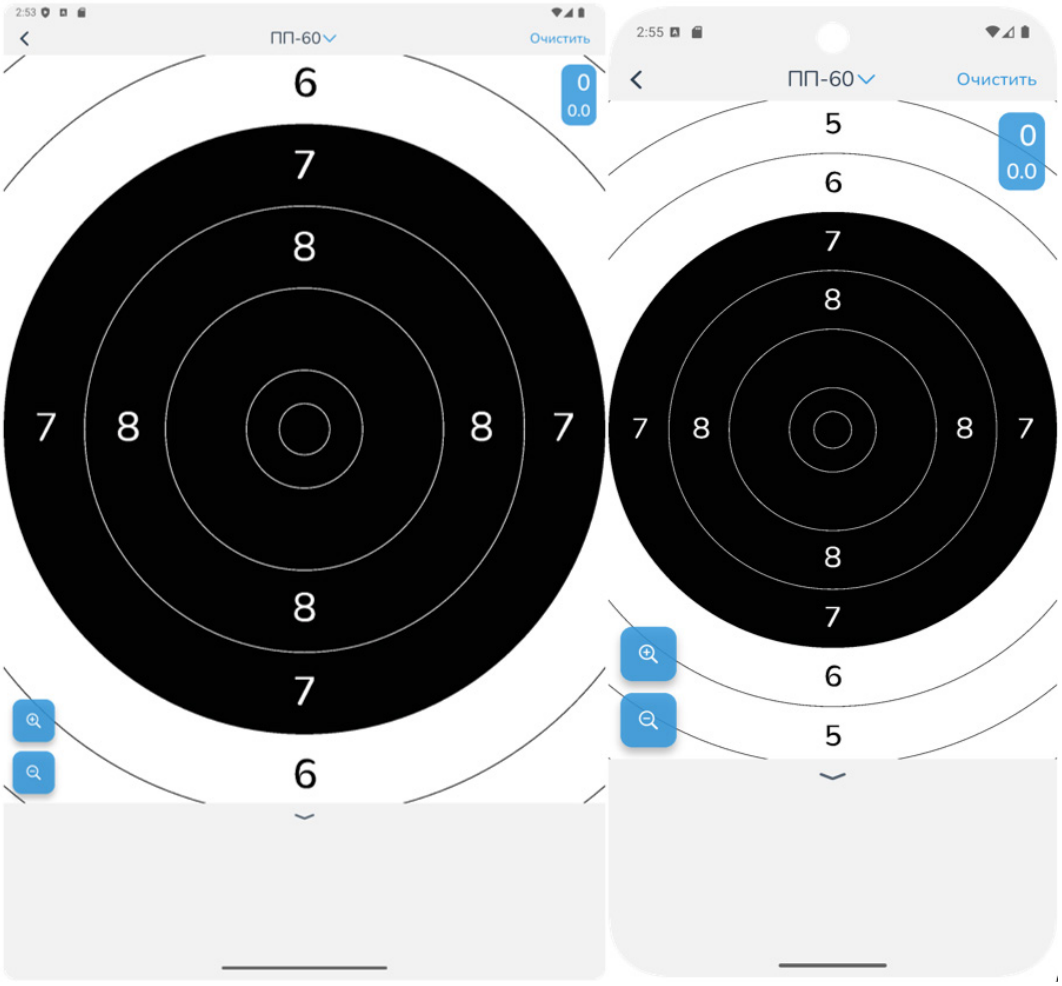

Рис 1. Отображение на устройствах Pixel Tablet и Pixel 9 Pro XL

• Стабильность частоты кадров — контролировалась в процессе активного масштабирования и работы с сериями до 100 выстрелов. Система демонстрировала стабильные 55–60 FPS даже в условиях интенсивной нагрузки.

Заключение

В данной работе была рассмотрена комплексная реализация механизма масштабирования и отображения мишени в мобильном приложении для анализа результатов пулевой стрельбы. Проведенный детальный анализ существующих решений для работы с графическими интерфейсами во Flutter позволил выбрать и обосновать оптимальный архитектурный подход, полностью соответствующий требованиям сохранения точности координат и обеспечения высокой производительности.

Использование комбинации OverflowBox, Transform и кастомного ScaleController обеспечило необходимую гибкость управления преобразованиями и гарантировало точность отображения попаданий при сохранении высокого уровня пользовательского опыта. Разработанная архитектура демонстрирует устойчивость к различным сценариям использования и эффективно масштабируется при увеличении объема данных.

Разработанный пользовательский интерфейс работы с мишенью спроектирован с учетом принципов эргономики и реальных сценариев использования тренерами и спортсменами. Особое внимание было уделено обеспечению точности координат попаданий при любых

трансформациях мишени и минимизации задержек при взаимодействии. Реализованная система жестового управления обеспечивает интуитивно понятное масштабирование и панорамирование, а изоляция слоёв отрисовки гарантирует стабильную производительность даже при работе с большими сериями выстрелов.

Результаты экспериментального тестирования подтвердили высокую эффективность реализованного решения: точность позиционирования составила менее 0,2 мм в пересчёте на реальные размеры мишени, а частота обновления интерфейса сохранялась на уровне 55–60 кадров в секунду при активной работе с жестами масштабирования. Тестирование на различных устройствах продемонстрировало корректную адаптацию интерфейса к различным размерам и разрешениям экранов.

Интеграция механизма точного масштабирования в приложение для анализа стрелковых результатов расширяет возможности взаимодействия с данными тренировок, делает анализ наглядным и точным, а также способствует повышению эффективности тренировочного процесса. Разработанное решение успешно интегрируется с алгоритмами кластеризации попаданий, обеспечивая полноценную аналитическую платформу для спортивной подготовки.

Работа выполнена под научным руководством старшего преподавателя кафедры программного обеспечения и администрирования информационных систем Воронежского государственного университета, Матвеевой Марии Валерьевной.

Литература

1. Flutter Documentation: official site [Электронный ресурс]. – URL: https://docs.flutter.dev (дата обращения: 13.09.2025).

2. Teale P. Building Games with Flutter / P. Teale. – Birmingham: Packt Publishing, 2022. – 412 p.

3. *Windmill E.* Flutter in Action / E. Windmill. – Shelter Island: Manning Publications, 2019. – 368 p.

4. Сайт ISSF: Международная федерация спортивной стрельбы. – URL: https://www.issf-sports.org (дата обращения: 11.05.2025).

5. Miola A. Flutter Complete Reference 2.0: The Ultimate Reference for Dart and Flutter / A. Miola. – 2023. – 826 p.

УДК 004.021

МЕТОД АДАПТАЦИИ КВАНТОВО-ИНСПИРИРОВАННЫХ АЛГОРИТМОВ МНОГОКРИТЕРИАЛЬНОЙ ОПТИМИЗАЦИИ НА ОСНОВЕ КУДИТОВ ПОД ГИБРИДНЫЕ КВАНТОВО-КЛАССИЧЕСКИЕ СИСТЕМЫ

В. В. Масленников

МИРЭА – Российский технологический университет

Аннотация. В работе представлен подход к адаптации квантово-инспирированных многокритериальных алгоритмов оптимизации, использующих кудиты, в формат, пригодный для выполнения на гибридных квантово-классических платформах. Основу метода составляют отображение многоуровневых квантовых состояний на кубитные регистры минимальной длины, строгая схема фазового представления целевых функций и тесное взаимодействие квантовых схем с классическими модулями анализа и отбора недоминируемых решений. Предложенная архитектура сохраняет ключевые квантовые эффекты, такие как параллелизм и усиление амплитуд, при этом обеспечивая совместимость как с физическими квантовыми процессорами, так и с классическими симуляторами, что предоставляет возможность плавного перехода от классических реализаций к полностью квантовым реализациям без перестройки логики алгоритма.

Ключевые слова: квантово-инспирированная оптимизация, многокритериальная оптимизация, кудит, гибридная квантово-классическая система, квантовая логическая схема, фазовое кодирование, оператор диффузии Гровера, квантовый оракул, фронт Парето.

Введение

Ценность существующих квантово-инспирированных алгоритмов оптимизации заключается в их практической применимости и доступности любому производству [1] ввиду способности обходиться без использования квантовых вычислительных устройств и систем. Такие алгоритмы эффективно имитируют ключевые принципы квантовой механики, в частности, суперпозицию и запутанность, на классических вычислительных архитектурах, обеспечивая при этом заметное ускорение по сравнению с традиционными методами оптимизации.

В будущем при появлении на рынке квантового оборудования с необходимой мощностью и устойчивостью кубитов к стороннему шуму, исходящему от окружающей среды [2], появится возможность бесшовного переноса квантово-инспирированных алгоритмов оптимизации на реальные квантовые платформы [1]. Однако исследования в области реализации квантово-инспирированных алгоритмов на основе квантовых логических схем [3–6] указывают на необходимость не просто «переноса», а глубокой адаптации квантово-инспирированных алгоритмов оптимизации под квантово-классические вычислительные системы.

Гибридный подход, сочетающий классические и квантовые компоненты, позволяет рационально распределять вычислительную нагрузку: задачи, требующие интенсивной обработки и логического анализа, выполняются на классических устройствах, тогда как квантовые схемы используются для операций, в которых проявляется их принципиальное преимущество.

В связи с этим становится актуальной задача переосмысления архитектуры квантово-инспирированных алгоритмов оптимизации: их необходимо формализовать не только как последовательность математических преобразований, но и как совокупность взаимодействующих квантовых и классических модулей. Это включает проектирование квантовых логических схем для инициализации квантовых состояний, кодирования параметров оптимизационной задачи в фазы кубитов, реализации оракулов для выявления недоминируемых решений и применения операторов амплитудного усиления. Одновременно требуется разработка интер-

фейсов между квантовыми и классическими компонентами, обеспечивающих согласованную передачу данных и синхронизацию вычислительных этапов.

В данной работе предлагается метод адаптации квантово-инспирированных алгоритмов многокритериальной оптимизации на основе кудитов под гибридные квантово-классические вычислительные системы, основанный на формализации алгоритмических шагов в виде квантовых логических схем и их интеграции с классическими процедурами обработки решений. Такой подход позволяет эффективно использовать преимущества обеих парадигм: квантового параллелизма и классической гибкости для решения сложных многокритериальных задач в условиях современных гетерогенных вычислительных платформ.

Описание метода

Поскольку в рассматриваемом подходе в качестве основной логической единицы используется кудит, а визуализация и работа с квантовыми состояниями размерности $d \geq 3$ с помощью сферы Блоха невозможны [7], реализация кудита на квантовой логической схеме предлагается через набор кубитов. Для определения минимального числа кубитов, необходимых для представления одного кудита, применяется формула:

$$n = \lceil \log_2 d \rceil, \tag{1}$$

где n — количество кубитов; d — количество базисных состояний кудита.

Тогда состояние кудита $|j\rangle$ (где $j \in \{0,1,\ldots,d-1\}$) кодируется в базисных состояниях n кубитов:

$$|j\rangle \mapsto |b_{n-1} b_{n-2} \ldots b_0\rangle, \tag{2}$$

где b_k — классический бит (при $b_k = \lfloor j / 2^k \rfloor \bmod 2$); k — позиция классического бита (при $k = \overline{0, n-1}$).

Так, например, если нужно использовать кудит с размерностью $d = 4$, потребуется 2 кубита, кодирующих базисные состояния кудита следующим образом:

$$|0\rangle_d \equiv |00\rangle,\ |1\rangle_d \equiv |01\rangle,\ |2\rangle_d \equiv |10\rangle,\ |3\rangle_d \equiv |11\rangle, \tag{3}$$

где индекс d указывает на принадлежность к состоянию кудита.

Важно отметить, что в случае, когда d не является степенью двойки, также требуется $n = \lceil \log_2 d \rceil$ кубитов. Однако в этом случае часть состояний, доступных в пространстве кубитов, будет неиспользуемой. Например, для $d = 3$ также необходимо 2 кубита, но состояние $|11\rangle$ будет игнорироваться.

Для создания суперпозиции всех возможных состояний в системе $|\psi\rangle$ из n кубитов применяется тензорное произведение n вентилей Адамара:

$$H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle, \tag{4}$$

где H — вентиль Адамара (при $H|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$); $|j\rangle$ — состояние кудита, закодированное в базисных состояниях n кубитов.

После создания суперпозиции состояний в системе $|\psi\rangle$ необходимо установить только те состояния, с которыми допустимо работать согласно количеству d базисных состояний кудита. Для этого в систему $|\psi\rangle$ добавляется вспомогательный кубит в состоянии $|0\rangle$. Теперь система принимает вид:

$$|\psi'\rangle = |\psi\rangle \otimes |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle \otimes |0\rangle. \tag{5}$$

Между первым основным кубитом и вспомогательным кубитом в системе $|\psi'\rangle$ применяется вентиль $CNOT$. Данное действие позволяет преобразовать первые n значений в состояниях системы $|\psi'\rangle$ в значения, соответствующие базисным состояниям n кубитов. В общем виде это можно записать как:

$$CNOT|c\rangle|t\rangle = \left(|0\rangle\langle 0|\otimes I + |1\rangle\langle 1|\otimes X\right)|c\rangle|t\rangle = \begin{cases}|c\rangle|t\rangle, & c=0\\ |c\rangle X|t\rangle, & c=1\end{cases}, \tag{6}$$

где $|c\rangle$ — управляющий кубит, $|t\rangle$ — управляемый (целевой) кубит; $|0\rangle\langle 0|$ и $|1\rangle\langle 1|$ — проекторы на состояния управляющего кубита; I — единичный оператор; X — оператор Паули-X.

К вспомогательному кубиту применяется оператор Паули-X, который инвертирует состояние вспомогательного кубита на противоположное.

Теперь вентиль $CNOT$ повторно применяется к первому основному кубиту и вспомогательному кубиту, но с тем учётом, что основной кубит является управляемым, а вспомогательный кубит — управляющим. Это действие меняет состояние вспомогательного кубита на $|1\rangle$ для всех состояний системы $|\psi'\rangle$.

Затем определяется множество допустимых состояний кудита $\mathcal{B} = \{|j\rangle : j = \overline{0, d-1}\}$ и применяется квантовый оракул $U_{\mathcal{B}}$, который переводит вспомогательный кубит в состояние $|1\rangle$, если основные кубиты находятся в одном из допустимых состояний:

$$U_{\mathcal{B}} : |j\rangle\otimes|0\rangle \to \begin{cases}|j\rangle\otimes|1\rangle, & |j\rangle\in\mathcal{B}\\ |j\rangle\otimes|0\rangle, & |j\rangle\notin\mathcal{B}\end{cases}. \tag{7}$$

После применения квантового оракула $U_{\mathcal{B}}$ получается состояние системы:

$$|\psi''\rangle = \frac{1}{\sqrt{2^n}}\left(\sum_{|j\rangle\in\mathcal{B}}|j\rangle\otimes|1\rangle + \sum_{|j\rangle\notin\mathcal{B}}|j\rangle\otimes|0\rangle\right). \tag{8}$$

Далее измеряется вспомогательный кубит. В том случае, если его состояние является $|1\rangle$, то основные кубиты коллапсируют в подпространство допустимых состояний [8]:

$$|\psi_s\rangle = \frac{1}{\sqrt{|\mathcal{B}|}}\sum_{|j\rangle\in\mathcal{B}}|j\rangle, \tag{9}$$

где $|\mathcal{B}|$ — количество допустимых состояний.

С учётом (9) для N кудитов общее состояние системы кудитов является тензорным произведением состояний всех кудитов:

$$|\psi_s^{(N)}\rangle = \frac{1}{\sqrt{|\mathcal{B}|^N}}\sum_{(|j_1\rangle,|j_2\rangle,\ldots,|j_N\rangle)\in\mathcal{B}}|j_1\rangle\otimes|j_2\rangle\otimes\ldots\otimes|j_N\rangle = \frac{1}{\sqrt{|\mathcal{B}|^N}}\sum_{(|j_1\rangle,|j_2\rangle,\ldots,|j_N\rangle)\in\mathcal{B}}|j_1,j_2,\ldots,j_N\rangle. \tag{10}$$

Кодирование значений целевых функций $\mathbf{f}_{\mathbf{x}_\sigma}$ можно осуществить через фазовый сдвиг. Пусть $\mathbf{f}_{\mathbf{x}_\sigma} = [f_1(\mathbf{x}_\sigma), f_2(\mathbf{x}_\sigma),\ldots, f_h(\mathbf{x}_\sigma)]$, $\mathbf{x}_\sigma = (x_{\sigma,1}, x_{\sigma,2},\ldots, x_{\sigma,\eta})$, η — количество переменных, h — количество целевых функций. Для каждого состояния $|j_1, j_2,\ldots, j_N\rangle_\delta$ $(\delta = \overline{1,|\mathcal{B}|^N})$ из суперпозиции состояний $|\psi_s^{(N)}\rangle$ вычисляются значения целевых функций в $\mathbf{f}_{\mathbf{x}_\sigma}$ на основе заданных в условии ограничений значений переменных в $\mathbf{x}_\sigma$, причём значения переменных в $\mathbf{x}_\sigma$ изменяются для каждого состояния $|j_1, j_2,\ldots, j_N\rangle_\delta$. Тогда фазовое кодирование значений для всех целевых функций в $\mathbf{f}_{\mathbf{x}_\sigma}$ для состояния $|j_1, j_2,\ldots, j_N\rangle_\delta$ можно записать как:

$$\phi_\delta = \sum_{l=1}^{h} o_l f_l\left(x_{\sigma,1}, x_{\sigma,2},\ldots, x_{\sigma,\eta}\right), \tag{11}$$

где o_l — весовой коэффициент для l-й целевой функции, $o_l \in \mathbb{R}$.

Кодирование выполняется с использованием вентиля RZ, который применяется к каждому кубиту состояния $|j_1, j_2,\ldots, j_N\rangle_\delta$ следующим образом:

$$RZ(\phi_\delta)|j_1, j_2, \ldots, j_N\rangle_\delta = e^{-i\phi_\delta/2}|j_1, j_2, \ldots, j_N\rangle_\delta = e^{-i\phi_\delta/2}\left(|j_1\rangle_\delta \otimes |j_2\rangle_\delta \otimes \ldots \otimes |j_N\rangle_\delta\right) =$$

$$= e^{-i\phi_\delta/2}\left(\left(|q\rangle^{\otimes n}\right)_1 \otimes \left(|q\rangle^{\otimes n}\right)_2 \otimes \ldots \otimes \left(|q\rangle^{\otimes n}\right)_N\right) = \left(e^{-i\phi_\delta/2}|q_1\rangle \otimes e^{-i\phi_\delta/2}|q_2\rangle \otimes \ldots \otimes e^{-i\phi_\delta/2}|q_n\rangle\right)_1 \otimes \quad (12)$$

$$\otimes \left(e^{-i\phi_\delta/2}|q_1\rangle \otimes e^{-i\phi_\delta/2}|q_2\rangle \otimes \ldots \otimes e^{-i\phi_\delta/2}|q_n\rangle\right)_2 \otimes \left(e^{-i\phi_\delta/2}|q_1\rangle \otimes e^{-i\phi_\delta/2}|q_2\rangle \otimes \ldots \otimes e^{-i\phi_\delta/2}|q_n\rangle\right)_N,$$

где $|q\rangle$ — состояние кубита.

Состояние системы кудитов после фазового кодирования можно записать в следующем виде:

$$|\psi_f^{(N)}\rangle = \frac{1}{\sqrt{|\mathcal{B}|^N}} \sum_{(|j_1\rangle, |j_2\rangle, \ldots, |j_N\rangle) \in \mathcal{B}} \left(e^{-i\phi/2}|j_1, j_2, \ldots, j_N\rangle\right). \quad (13)$$

Далее на систему $|\psi_f^{(N)}\rangle$ действует квантовый оракул U_P, который инвертирует фазу состояния $|j_1, j_2, \ldots, j_N\rangle_\delta$, если оно соответствует недоминируемому решению, а другие состояния оставляет без изменений. Квантовый оракул U_P работает по правилу:

$$U_P : |j_1, j_2, \ldots, j_N\rangle_\delta \to \begin{cases} -|j_1, j_2, \ldots, j_N\rangle_\delta, & |j_1, j_2, \ldots, j_N\rangle_\delta \Rightarrow \mathcal{L} \\ |j_1, j_2, \ldots, j_N\rangle_\delta, & |j_1, j_2, \ldots, j_N\rangle_\delta \not\Rightarrow \mathcal{L} \end{cases}, \quad (14)$$

где $\mathcal{L}$ — недоминируемое решение; оператор $\Rightarrow$ означает отношение соответствия.

Оператор диффузии Гровера или «инверсия относительно среднего» [9, 10] усиливает амплитуды «помеченных» состояний, что позволяет выделить идентифицированные в (14) решения, определяемые «помеченными» состояниями, среди всех возможных состояний системы.

Согласно методу «инверсии относительно среднего», вычисляется среднее значение амплитуд $\bar{\alpha}$ вероятности существования квантовых состояний системы $|\psi_f^{(N)}\rangle$ по формуле:

$$\bar{\alpha} = \frac{1}{|\mathcal{B}|^N} \sum_{\delta=1}^{|\mathcal{B}|^N} \alpha_\delta, \quad (15)$$

где α_δ — амплитуда вероятности состояния $|j_1, j_2, \ldots, j_N\rangle_\delta$.

Каждая амплитуда α_δ преобразуется под действием оператора диффузии Гровера следующим образом:

$$\alpha'_\delta = 2\bar{\alpha} - \alpha_\delta. \quad (16)$$

Преобразование амплитуд состояний системы $|\psi_f^{(N)}\rangle$ оператором диффузии Гровера на квантовой логической схеме в соответствии с [10] реализуется путём применения к каждому кубиту, формирующему многокудитное состояние $|j_1, j_2, \ldots, j_N\rangle_\delta$, вентиля Адамара, оператора Паули-X и многокубитного оператора $C\ldots CNOT$.

Локальная оптимизация решений из фронта Парето $F = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \ldots, \mathbf{s}_v\}$ реализуется путём создания окрестности возможных решений для каждого решения $\mathbf{s}_b \in F$, $b = \overline{1, v}$ и выбора среди новых решений тех, которые соответствуют недоминируемым. С этой целью выполняется обновление фазы кудитов в системе $|\psi_f^{(N)}\rangle$ последовательным применением к каждому кубиту многокудитного состояния $|j_1, j_2, \ldots, j_N\rangle_\delta$ вентиля RZ. В этом случае новая фаза ϕ'_δ формируется за счёт добавления к фазе ϕ_δ из (11) малого возмущения [11, 12]:

$$\phi'_\delta = \phi_\delta + \delta\phi, \quad (17)$$

где $\delta\phi \sim \mathcal{U}(-\epsilon, \epsilon)$ — случайное возмущение из равномерного распределения на интервале $[-\epsilon, \epsilon]$, $\epsilon \in \mathbb{R}$ — параметр, контролирующий масштаб возмущения.

Тогда применение вентиля RZ к каждому кубиту состояния $|j_1, j_2, \ldots, j_N\rangle_\delta$ записывается аналогично (12) с заменой ϕ_δ на ϕ'_δ .

Затем на систему $\left|\psi_f^{(N)}\right\rangle$ снова оказывает действие квантовый оракул U_P, инвертирующий фазу состояния $\left|j_1, j_2, \ldots, j_N\right\rangle_{\delta}$, если оно соответствует недоминируемому решению, а также применяется оператор диффузии Гровера, усиливающий амплитуды «помеченных» состояний.

Общий вид квантово-классической алгоритмической системы, реализующей квантово-инспирированный алгоритм многокритериальной оптимизации, изображён на рис. 1.

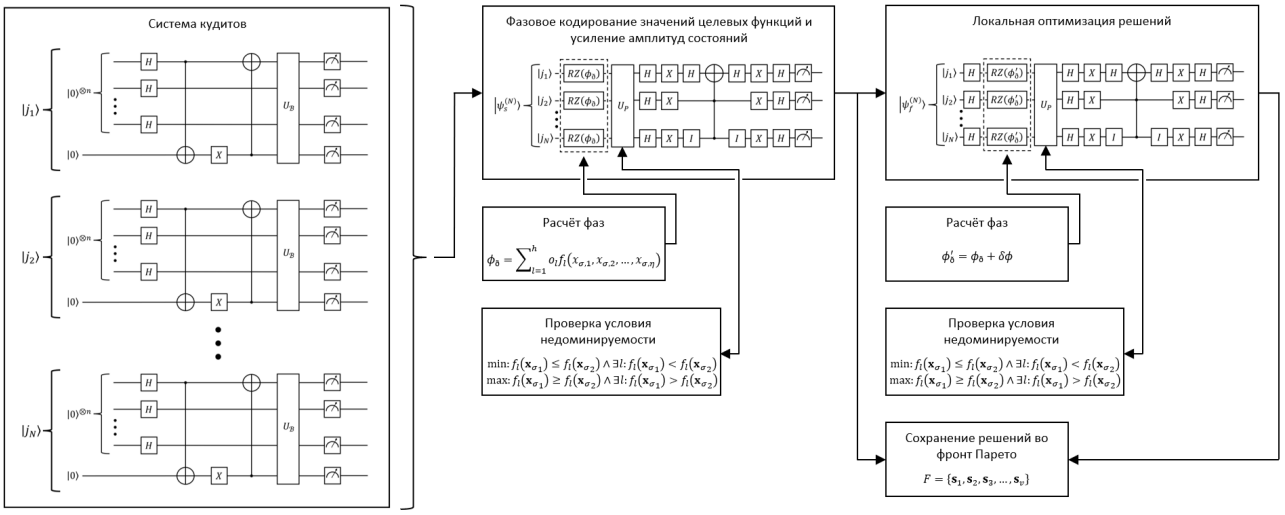

Рис. 1. Схема квантово-классической алгоритмической системы, реализующей квантово-инспирированный алгоритм многокритериальной оптимизации

Квантово-классическая алгоритмическая система выполняется в следующем порядке.

Первоначально строится система из N квантовых логических схем, каждая из которых реализует кудит через n кубитов. При этом определяются допустимые базисные состояния в пространстве кубитов. Затем получившаяся система кудитов $\left|\psi_s^{(N)}\right\rangle$ подвергается воздействию вентилей RZ, которые кодируют значения целевых функций $\mathbf{f}_{\mathbf{x}_\sigma}$, вычисляемые на классическом вычислительном устройстве, в фазы ϕ_{δ} (при $\delta = \overline{1, \left|\mathcal{B}\right|^N}$). Далее на классическом вычислительном устройстве выполняется проверка решений на недоминируемость. В том случае, если решение является недоминируемым, квантовый оракул U_P инвертирует фазу соответствующего данному решению многокудитного квантового состояния $\left|j_1, j_2, \ldots, j_N\right\rangle_{\delta}$, а оператор диффузии Гровера усиливает амплитуду этого состояния. Недоминируемые решения сохраняются во фронте Парето F на классическом вычислительном устройстве.

На втором этапе к получившейся системе $\left|\psi_f^{(N)}\right\rangle$ применяются вентили Адамара, создающие суперпозицию всех возможных состояний. Воздействие вентилей RZ на систему $\left|\psi_f^{(N)}\right\rangle$ обновляет фазы ϕ_{δ} на ϕ'_{δ}, вычисляемые для каждого состояния $\left|j_1, j_2, \ldots, j_N\right\rangle_{\delta}$ на классическом вычислительном устройстве. В результате вокруг каждого решения из фронта Парето F создаётся окрестность возможных решений. После этого новые решения из окрестности проверяются на классическом вычислительном устройстве на недоминируемость, а квантовый оракул U_P и оператор диффузии Гровера определяют и усиливают каждое многокудитное квантовое состояние $\left|j_1, j_2, \ldots, j_N\right\rangle_{\delta}$, соответствующее недоминируемому решению из окрестности. Затем новые решения обновляют фронт Парето F.

Заключение

Предложенный метод обеспечивает строгую и воспроизводимую процедуру адаптации квантово-инспирированных алгоритмов многокритериальной оптимизации, основанных на

кудитах, под современные гибридные квантово-классические вычислительные архитектуры. За счёт представления кудитов через минимально необходимое число кубитов, формализации фазового кодирования целевых функций и применения квантовых оракулов совместно с оператором диффузии Гровера метод позволяет эффективно интегрировать квантовые вычислительные примитивы в классический оптимизационный процесс.

Важно отметить, что разработанная схема сохраняет вычислительные преимущества квантового параллелизма и амплитудного усиления, при этом оставаясь совместимой как с физическими квантовыми устройствами, так и с классическими симуляторами. Это предоставляет возможность к постепенной миграции алгоритмов от полностью классических реализаций к полноценным квантовым реализациям без потери логической целостности и без необходимости полной переработки алгоритмической структуры.

Представленный подход расширяет инструментарий гибридных квантово-классических систем и формирует основу для построения масштабируемых, устойчивых и эффективных методов решения сложных многокритериальных задач в реальных прикладных областях, включая инженерное проектирование, логистику, управление сложными технологическими процессами и принятие решений в условиях неопределённости.

Литература

1. *Bova F., Goldfarb A., Melko R.* Commercial applications of quantum computing // EPJ Quantum Technology. – 2021. – 8. – P. 1–13.

2. *Maslennikov V. V., Demidova L. A.* Modification of a Quantum-inspired Genetic Algorithm for Numerical Optimization Using Qudit under Conditions of Simulating Quantum Decoherence // Computational Nanotechnology. – 2024. – 11. – P. 58–85.

3. *Joe M., Parkavi K.* Integration of quantum-inspired algorithms in circuit technologies for enhanced computational efficiency // ICTACT Journal on Microelectronics. – 2025. – 10. – P. 1952–1956.

4. *Brizuela D., Uria S.* Hybrid classical-quantum systems in terms of moments // Physical Review A. – 2024. – 109. – P. 1–21.

5. *Rallis K., Liliopoulos I., Tsipas E., Varsamis G., Melissourgos N., Karafyllidis I., Sirakoulis G., Dimitrakis P.* Hardware-level Interfaces for Hybrid Quantum-Classical Computing Systems. arXiv 2025, submitted.

6. *Nikoloska I., Simeone O.* Training Hybrid Classical-Quantum Classifiers via Stochastic Variational Optimization. arXiv 2022, submitted.

7. *Демидова Л. А.* Применение многоуровневых квантовых систем для параллельной оценки решений в задачах многокритериальной оптимизации / Л. А. Демидова, В. В. Масленников // Вестник Рязанского государственного радиотехнического университета. – 2025. – № 92. – С. 57–76. – DOI 10.21667/1995-4565-2025-92-57-76. – EDN ZVIXFT.

8. *Pagni V., Huber S., Epping M., Felderer M.* Fast Quantum Amplitude Encoding of Typical Classical Data. arXiv 2025, submitted.

9. *Goscinski A.* The parallel Grover as dynamic system. arXiv 2018, submitted.

10. *Al-Bayaty A., Perkowski M.* A concept of controlling Grover diffusion operator: a new approach to solve arbitrary Boolean-based problems // Scientific Reports. – 2024. – 14. – P. 1–16.

11. *Facchi P., Ligabò M., Viesti V.* Robustness of quantum symmetries against perturbations // Journal of Physics A: Mathematical and Theoretical. – 2025. – 58. – P. 1–23.

12. *Melo A., Beugnot G., Minganti F.* Variational Perturbation Theory in Open Quantum Systems for Efficient Steady State Computation. arXiv 2025, submitted.

УДК 004.657

ОПТИМИЗАЦИЯ АГРЕГИРУЮЩЕГО ЗАПРОСА С РАСЧЕТОМ ОТКЛОНЕНИЙ ОТ СРЕДНЕГО ЗНАЧЕНИЯ ДЛЯ КАЖДОЙ ГРУППЫ

М. В. Матвеева, В. Ю. Шипилова

Воронежский государственный университет

Аннотация. Статья посвящена исследованию методов оптимизации SQL-запросов в среде СУБД PostgreSQL для выборки данных с последующей агрегацией и расчетом отклонений значений от средних показателей. В качестве метрик производительности используются стоимость запроса, оценивающая нагрузку на CPU, и время выполнения. Установлено, что использование партиционирования таблиц обеспечивает наименьшую стоимость, а комбинация индексов и оконных функций является наиболее сбалансированным подходом для несекционированных таблиц.
Ключевые слова: базы данных, СУБД PostgreSQL, оптимизация запросов, план выполнения запроса, SQL, время выполнения запроса, стоимость запроса, индексы, оконные функции, язык запросов, CTE, общие табличные выражения.

Введение

Стремительный рост объема данных, наблюдаемый последние годы, повышает требования к скорости и эффективности их обработки. В этом контексте производительность запросов непосредственно влияет на время отклика информационных систем, делая задачу оптимизации запросов одной из ключевых.

Дисциплина «Оптимизация SQL-запросов», читаемая на факультете ПММ Воронежского государственного университета для студентов бакалавриата по направлению «Математическое обеспечение и администрирование информационных систем», призвана сформировать у обучающихся компетенции в области анализа планов выполнения запросов и применения методов их оптимизации.

В статье представлены результаты исследования, выполненного в рамках вышеуказанной дисциплины. Основной целью работы является выявление и анализ методов оптимизации агрегирующего запроса средствами преобразования самого запроса либо посредством внедрения дополнительных объектов базы данных.

1. Постановка задачи

Предметная область принадлежит сектору здравоохранения.

База данных включает в себя таблицы содержащие данные о врачах (Doctors), пациентах (Patients) и приемах (Appointments). Структура таблиц и ограничения первичных и внешних таблиц представлены ниже.

```
CREATE TABLE Doctors (
   DoctorID INT PRIMARY KEY,
   LastName TEXT,
   FirstName TEXT,
   MiddleName TEXT,
   DateOfBirth DATE,
   Gender TEXT,
   PhoneNumber VARCHAR(50));
```

```
CREATE TABLE Patients (
   PatientID INT PRIMARY KEY,
   LastName TEXT,
   FirstName TEXT,
   MiddleName TEXT,
   DateOfBirth DATE,
   Gender TEXT,
   Address TEXT,
   PhoneNumber VARCHAR(50),
   InsurancePolicy TEXT);
CREATE TABLE Appointments (
   AppointmentID INT PRIMARY KEY,
   AppointmentDate DATE,
   Symptoms TEXT,
   Prescriptions TEXT,
   Notes TEXT,
   DoctorID INT REFERENCES Doctors(DoctorID),
   PatientID INT REFERENCES Patients(PatientID));
```

Задача состоит в выборе данных о врачах (идентификатор DoctorID, фамилию и имя врача), расчете среднего количества приемов в день для каждого врача за указанный год, а также определении отклонения этого показателя от средней величины по всему массиву врачей. Полученные результаты сортируются по общему числу проведенных врачом приемов за исследуемый период. Рассматриваемый временной интервал охватывает 2017 год.

Требуется предложить варианты решения поставленной задачи и провести анализ производительности запросов в среде СУБД PostgreSQL 16. Эффективность запросов будет проводиться по двум ключевым метрикам: стоимость — нагрузка на центральный процессор (CPU) и время выполнения запроса. Стоимость всего запроса включает стоимость выполнения всех его узлов. В свою очередь стоимость узла зависит от типа узла (чтение данных, соединение таблиц, сортировка и т. п.) и от объема обрабатываемых узлом данных. Нагрузку на дисковую подсистему (количество I/O операций), потребление оперативной памяти и другие параметры в данной статье рассматриваться не будут.

Тестовая база имеет следующий объем данных. Таблица Doctors содержит данные о пятидесяти врачах. Таблица Patients включает 100000 записей о пациентах. Количество строк в таблице Appointments соответствует 500000 приемам за несколько лет.

Все варианты решений задачи выборки данных проводятся на одних и тех же данных, на единой аппаратной платформе и в одинаковых условиях. Под условиями понимается состояние кэша данных, актуальная статистика и т.д.

Параметры компьютера, на котором проводятся измерения следующие: процессор 12th Gen Intel(R) Core(TM) i5-12450H 2.00 GHz (8 ядер и 12 потоков), оперативная память 16 ГБ.

2. Анализ выборки с подзапросом

Одним из вариантов решения поставленной задачи является запрос с подзапросом для подсчета среднего количество приемов на одного врача. Код данного решения выглядит следующим образом.

```
SELECT
   d.DoctorID, d.LastName, d.FirstName,
   COUNT(a.AppointmentID) AS total_appointments,
   COUNT(a.AppointmentID) / COUNT(DISTINCT a.AppointmentDate) AS avg_appointments_
per_day,
```

```
   (COUNT(a.AppointmentID) / COUNT(DISTINCT a.AppointmentDate)) -
     (SELECT AVG(daily_avg)
      FROM (
      SELECT COUNT(AppointmentID) / COUNT(DISTINCT AppointmentDate) AS daily_avg
      FROM Appointments
      WHERE EXTRACT(YEAR FROM Appointments.AppointmentDate) = 2017
      GROUP BY DoctorID
     ) AS subq) AS diff_from_avg
FROM Doctors d
LEFT JOIN Appointments a ON d.DoctorID = a.DoctorID
     AND EXTRACT(YEAR FROM a.AppointmentDate) = 2017
GROUP BY d.DoctorID, d.LastName, d.FirstName
ORDER BY total_appointments DESC;
```

План запроса:

```
Sort  (cost=20403.19..20403.31 rows=50 width=87) …
  InitPlan 1 (returns $1)
   -> Aggregate  (cost=10169.42..10169.43 rows=1 width=32)
      -> GroupAggregate  (cost=9858.25..10168.80 rows=50 width=12) …
         -> Gather Merge  (cost=9858.25..10149.42 rows=2500 width=12) …
            -> Sort  (cost=8858.23..8860.84 rows=1042 width=12) …
               -> Parallel Seq Scan on appointments (cost=0.00..8806.00 rows=1042
                  width=12)
                Filter: (EXTRACT(year FROM appointmentdate) = ‘2017’::numeric)
  -> GroupAggregate  (cost=10206.34..10232.34 rows=50 width=87) …
     -> Sort  (cost=10206.34..10212.59 rows=2500 width=47) …
        -> Hash Right Join  (cost=1002.12..10065.24 rows=2500 width=47) …
           -> Gather  (cost=1000.00..10056.00 rows=2500 width=12) …
              -> Parallel Seq Scan on appointments a  (cost=0.00..8806.00
                 rows=1042 width=12)
                 Filter: (EXTRACT(year FROM appointmentdate) = ‘2017’::numeric)
                 -> Hash  (cost=1.50..1.50 rows=50 width=39)
                    ->  Seq Scan on doctors d  (cost=0.00..1.50 rows=50 width=39)
```

Анализ плана выполнения выявляет существенные проблемы производительности. Как в подзапросе, так и во внешнем запросе осуществляется полное последовательное сканирование таблицы приёмов Appointments с фильтрацией по году. Стоимость выполнения составляет 20403 условных единиц, при этом оптимизатор некорректно оценивает количество обрабатываемых строк при актуальной статистике — ожидается 2500, фактически обрабатывается более 44000, а при ожидании тысячи — 15 тысяч записей.

3. Устранение полного сканирования таблицы

Для того, чтобы уйти от полного сканирования таблиц добавим индекс по полю даты приёма AppointmentDate и модифицируем условие фильтрации по году таким образом, чтобы индекс мог быть использован в плане [1–3]. Для этого откажемся от использования функции EXTRACT и перейдём к условию вхождения в диапазон дат.

```
SELECT
    d.DoctorID, d.LastName, d.FirstName,
    COUNT(a.AppointmentID) AS total_appointments,
    COUNT(a.AppointmentID) / COUNT(DISTINCT a.AppointmentDate) AS avg_appointments_
per_day,
```

```
    (COUNT(a.AppointmentID) / COUNT(DISTINCT a.AppointmentDate)) -
      (SELECT AVG(daily_avg)
       FROM (
        SELECT COUNT(AppointmentID) / COUNT(DISTINCT AppointmentDate) AS daily_avg
        FROM Appointments
        WHERE Appointments.AppointmentDate >= '2017-01-01'
              AND Appointments.AppointmentDate < '2018-01-01'
        GROUP BY DoctorID
       ) AS subq) AS diff_from_avg
FROM Doctors d
LEFT JOIN Appointments a ON d.DoctorID = a.DoctorID
        AND a.AppointmentDate >= '2017-01-01'
        AND a.AppointmentDate < '2018-01-01'
GROUP BY d.DoctorID, d.LastName, d.FirstName
ORDER BY total_appointments DESC;
```

План выполнения запроса после изменений:

```
Sort  (cost=21714.68..21714.80 rows=50 width=87) …
  InitPlan 1 (returns $0)
  ->  Aggregate  (cost=10791.81..10791.82 rows=1 width=32)
      ->  GroupAggregate  (cost=10341.81..10791.18 rows=50 width=12) …
          ->  Sort  (cost=10341.81..10454.00 rows=44874 width=12) …
              ->  Bitmap Heap Scan on appointments  (cost=520.38..6874.49
                  rows=44874 width=12)
                  Recheck Cond: ((appointmentdate >= '2017-01-01'::date) AND
                                (appointmentdate < '2018-01-01'::date))
                  ->  Bitmap Index Scan on idx_appointments_date
                      (cost=0.00..509.16 rows=44874 width=0) …
  ->  GroupAggregate  (cost=10471.71..10921.45 rows=50 width=87) …
      ->  Sort  (cost=10471.71..10583.90 rows=44874 width=47) …
          ->  Hash Right Join  (cost=522.51..7004.39 rows=44874 width=47) …
              ->  Bitmap Heap Scan on appointments a  (cost=520.38..6874.49
                  rows=44874 width=12)
                  Recheck Cond: ((appointmentdate >= '2017-01-01'::date) AND
                                 (appointmentdate < '2018-01-01'::date))
                   -> Bitmap Index Scan on idx_appointments_date (cost=0.00..509.16
                      rows=44874 width=0)…
                  ->  Hash  (cost=1.50..1.50 rows=50 width=39) …
                      ->  Seq Scan on doctors d  (cost=0.00..1.50 rows=50 width=39)
```

Переход от полного сканированию таблиц к сканированию по индексу снизил стоимость прохода по таблице в 17 раз. Одновременно устранилась ошибка оценки количества строк оптимизатором. Конечно, при этом надо учитывать, что появились затраты на поддержание индекса, но введу того, что приоритетной задачей является повышение производительности выборки данных, оценивать затраты на индекс не будем. Еще одним недостатком обновлённого запроса является повторный проход по таблице Appointments.

4. Применение общего табличного выражения

Дальнейшее улучшение производительности достигается за счёт реструктуризации запроса с применением Common Table Expression (CTE).

```
WITH doctor_stats AS (
    SELECT
        DoctorID,
        COUNT(AppointmentID) AS total_appts,
        COUNT(DISTINCT AppointmentDate) AS working_days,
           COUNT(AppointmentID) / NULLIF(COUNT(DISTINCT AppointmentDate), 0) AS
daily_avg
    FROM Appointments
    WHERE AppointmentDate >= '2017-01-01' AND AppointmentDate < '2018-01-01'
    GROUP BY DoctorID)
SELECT
    d.DoctorID, d.LastName, d.FirstName,
    COALESCE(ds.total_appts, 0) AS total_appointments,
    COALESCE(ds.daily_avg, 0) AS avg_appointments_per_day,
    COALESCE(ds.daily_avg, 0) -
        (SELECT AVG(daily_avg) FROM doctor_stats) AS diff_from_avg
FROM Doctors d LEFT JOIN doctor_stats ds ON d.DoctorID = ds.DoctorID
ORDER BY total_appointments DESC;
```

Такой подход обеспечивает однократное выполнение сканирования таблицы и фильтрации с последующим использованием результатов в подзапросе и во внешнем запросе. План запроса с общим табличным выражением:

```
Sort  (cost=10797.37..10797.49 rows=50 width=87) …
  CTE doctor_stats
    ->  GroupAggregate  (cost=10341.81..10791.30 rows=50 width=28) …
          ->  Sort  (cost=10341.81..10454.00 rows=44874 width=12) …
             ->  Bitmap Heap Scan on appointments  (cost=520.38..6874.49 rows=44874
                 width=12) …
                 ->  Bitmap Index Scan on idx_appointments_date  (cost=0.00..509.16
                     rows=44874 width=0)
                     Index Cond: ((appointmentdate >= '2017-01-01'::date) AND
                                  (appointmentdate < '2018-01-01'::date)) …
  ->  Hash Right Join  (cost=2.12..3.52 rows=50 width=87) …
        ->  CTE Scan on doctor_stats ds  (cost=0.00..1.00 rows=50 width=20)
        ->  Hash  (cost=1.50..1.50 rows=50 width=39) …
             ->  Seq Scan on doctors d  (cost=0.00..1.50 rows=50 width=39)
```

В итоге стоимость уменьшается примерно вдвое, а время выполнения сокращается в 6 раз по сравнению с вариантом без использования CTE, но при этом потребление памяти на время выполнения запроса возрастет из-за хранения результатов выборки CTE [2].

5. Запрос с оконными функциями

Альтернативный вариант модификации запроса заключается в использовании оконных функций для того, чтобы предотвратить повторный проход по таблице Appointments.

```
SELECT
   d.DoctorID, d.LastName, d.FirstName,
   COUNT(a.AppointmentID) AS total_appointments,
   COUNT(a.AppointmentID) / COUNT(DISTINCT a.AppointmentDate) AS avg_appointments_
per_day,
   (COUNT(a.AppointmentID) / COUNT(DISTINCT a.AppointmentDate)) -
       AVG(COUNT(a.AppointmentID) / COUNT(DISTINCT a.AppointmentDate)) OVER () AS
```

```
        diff_from_avg
FROM Doctors d
LEFT JOIN Appointments a ON d.DoctorID = a.DoctorID
     AND a.AppointmentDate >= '2017-01-01' AND a.AppointmentDate < '2018-01-01'
GROUP BY d.DoctorID, d.LastName, d.FirstName
ORDER BY total_appointments DESC;
План запроса с оконными функциями:
 Sort  (cost=10923.61..10923.74 rows=56 width=87) …
  ->  WindowAgg  (cost=10471.71..10922.20 rows=56 width=87)
        ->  GroupAggregate  (cost=10471.71..10920.95 rows=56 width=55) …
              ->  Sort  (cost=10471.71..10583.90 rows=44874 width=47) …
                  ->  Hash Right Join  (cost=522.51..7004.39 rows=44874 width=47) …
                       ->  Bitmap Heap Scan on appointments a  (cost=520.38..6874.49
                          rows=44874 width=12)
                       Recheck Cond: ((appointmentdate >= '2017-01-01'::date) AND
                                      (appointmentdate < '2018-01-01'::date))
                                 ->  Bitmap Index Scan on idx_appointments_date
                                     (cost=0.00..509.16 rows=44874  width=0) …
                            ->  Hash  (cost=1.50..1.50 rows=50 width=39)
                                 ->  Seq Scan on doctors d  (cost=0.00..1.50 rows=50
                                    width=39)
```

Стоимость незначительно превышает стоимость из предыдущего подхода, а время выполнения превышает примерно в два раза.

6. Секционирование таблицы

В качестве эксперимента рассмотрим варианты решений с секционированной таблицей Appointments. Поскольку стоимость выполнения запроса с оконными функциями незначительно отличается от запроса с CTE, то будут проанализированы оба подхода в сочетании с партиционированием. Создание партиционированной таблицы приёмов Appointments_partitioned и индексов для неё показано ниже.

```
CREATE TABLE Appointments_partitioned (
    AppointmentID INT,
    AppointmentDate DATE,
    Symptoms TEXT,
    Prescriptions TEXT,
    Notes TEXT,
    DoctorID INT,
    PatientID INT,
    PRIMARY KEY (AppointmentID, AppointmentDate)
) PARTITION BY RANGE (AppointmentDate);
CREATE TABLE appointments_before_2015 PARTITION OF Appointments_partitioned
FOR VALUES FROM (MINVALUE) TO ('2015-01-01');
CREATE TABLE appointments_y2015 PARTITION OF Appointments_partitioned
FOR VALUES FROM ('2015-01-01') TO ('2016-01-01');
CREATE TABLE appointments_y2016 PARTITION OF Appointments_partitioned
FOR VALUES FROM ('2016-01-01') TO ('2017-01-01');
CREATE TABLE appointments_y2017 PARTITION OF Appointments_partitioned
FOR VALUES FROM ('2017-01-01') TO ('2018-01-01');
……
INSERT INTO Appointments_partitioned
```

```
SELECT * FROM Appointments;
CREATE INDEX idx_appointments_part_doctor_id ON Appointments_
partitioned(DoctorID);
CREATE INDEX idx_appointments_part_date ON Appointments_
partitioned(AppointmentDate);
CREATE INDEX idx_appointments_part_patient_id ON Appointments_
partitioned(PatientID);
CREATE INDEX idx_appointments_y2017_doctor ON appointments_y2017(DoctorID);
CREATE INDEX idx_appointments_y2017_patient ON appointments_y2017(PatientID);
CREATE INDEX idx_appointments_y2017_date ON appointments_
y2017(AppointmentDate);
```

Код запросов с оконными функциями и с CTE к партиционированной таблице здесь не приводится, так как отличие состоит только в имени одной таблицы (Appointments заменяем на Appointments_partitioned).

Проанализируем планы запросов, обращающихся к партиционированной таблицей с использованием оконной функций и общего табличного. План запроса с использованием партиционирования и оконных функций:

```
Sort  (cost=5222.66..5222.79 rows=50 width=87) …
  ->  WindowAgg  (cost=4771.79..5221.25 rows=50 width=87)
      ->  GroupAggregate  (cost=4771.79..5220.00 rows=50 width=55) …
          ->  Sort  (cost=4771.79..4883.72 rows=44771 width=47) …
              ->  Hash Right Join  (cost=2.12..1313.17 rows=44771 width=47) …
                  ->  Seq Scan on appointments_y2017 a  (cost=0.00..1183.57
                      rows=44771 width=12) …
                      ->  Hash  (cost=1.50..1.50 rows=50 width=39) …
                          ->  Seq Scan on doctors d  (cost=0.00..1.50 rows=50
                              width=39)
```

План запроса с использованием партиционирования и CTE:

```
Sort  (cost=2138.22..2138.35 rows=58 width=87) …
  CTE doctor_stats
    ->  GroupAggregate  (cost=0.29..2132.16 rows=58 width=28)…
        ->  Index Scan using idx_appointments_y2017_doctor_date on appointments_
            y2017
            appointments_partitioned  (cost=0.29..1795.62 rows=44771 width=12)
                Index Cond: ((appointmentdate >= ‘2017-01-01’::date) AND
                             (appointmentdate < ‘2018-01-01’::date)) …
  ->  Hash Right Join  (cost=2.12..3.52 rows=50 width=87) …
      ->  CTE Scan on doctor_stats ds  (cost=0.00..1.00 rows=50 width=20)
          ->  Hash  (cost=1.50..1.50 rows=50 width=39) …
              ->  Seq Scan on doctors d  (cost=0.00..1.50 rows=50 width=39)
```

Что интересно, несмотря на наличие индексов, по плану выполнения запроса с партиционированием и оконными функциями видно, что оптимизатор использует последовательное сканирование по партиции, что замедляет выполнение. В варианте с CTE запрос выполняет сканирование по индексу только в пределах одной партиции, что обеспечивает стоимость 2138 и время выполнения 19 миллисекунд.

7. Сравнительный анализ подходов оптимизации

Сравнительный анализ производительности рассмотренных запросов (результаты представлены в таблице) позволяет сформулировать рекомендации по выбору стратегии оптими-

зации. Критическими факторами при этом являются характер данных (статические или обновляемые), их объем и частота выполнения запроса.

Таблица

Стоимость и время выполнения запросов

Тип запроса	Стоимость (cost)	Время (мс)
Запрос с подзапросом	20403.31	322.145
Запрос с подзапросом и индексом	21714.80	244.356
Запрос с CTE	10797.39	39.244
Запрос с оконной функцией	10923.74	85.941
Запрос с партиционированной таблицей и оконной функцией	5222.79	62.976
Запрос с партиционированной таблицей и CTE	2138.35	19.492

Для исторических данных, изменение которых маловероятно (например, записи за 2017 год), создание материализованного представления является оптимальным решением. Оно обеспечивает минимальную стоимость (~10) и время выполнения менее 1 мс. Однако данный подход приводит к дублированию данных и увеличению затрат на хранение [1]. Следовательно, его применение оправдано только в сценариях с частыми обращениями к данным. В противном случае дополнительные издержки не компенсируются выигрышем в производительности, поэтому в данном исследовании этот вариант детально не анализируется.

При работе с часто обновляемыми данными использование материализованного представления может оказаться нецелесообразно, поскольку требует его регулярного обновления, что нивелирует плюсы в его использовании. Исключение могут составить случаи, когда частые запросы к данным сочетаются с приемлемой частотой обновления представления.

Для таблиц с большим объемом данных эффективным методом является партиционирование, которое позволяет значительно повысить производительность за счет сокращения числа просматриваемых строк. Недостатком метода являются затраты на поддержание секционированной таблицы, а также потенциальное снижение эффективности запросов, выполняемых не по ключу партиционирования.

При сравнительно небольшом объеме данных применение оконных функции, CTE и индексов демонстрирует высокую эффективность. При этом необходимо учитывать дополнительные расходы на поддержание индексов и потребление оперативной памятью при использовании CTE.

Заключение

В результате проведенного исследования был проведен сравнительный анализ методов оптимизации SQL-запроса, предназначенного для выборки данных с агрегацией и расчетом отклонения для каждой группы от среднего значения по всем группам. Основные выводы работы заключаются в следующем. Использование секционированных таблиц обеспечивает существенное снижение стоимости выполнения запросов благодаря уменьшению объёма обрабатываемых данных. Однако секционирование целесообразно применять лишь тогда, когда объём данных действительно велик. Применение индексов по полям фильтрации совместно с оконными функциями или CTE для снижения количества проходов по таблицам демонстрирует значительное преимущество перед использованием подзапросов в сложных агрегатных операциях.

Работа выполнена под научным руководством канд. техн. наук, доц., доцента кафедры программного обеспечения и администрирования информационных систем Воронежского государственного университета, Селезнева Константина Егоровича.

Литература

1. Домбровская Г. Оптимизация запросов в PostgreSQL / Г. Домбровская, Б. Новиков, А. Бейликова ; пер. с англ. Д. А. Беликова. – Москва : ДМК Пресс, 2022. – 278 с.

2. Рогов Е. В. PostgreSQL 16 изнутри / Е. В. Рогов – М.: ДМК Пресс, 2024. – 664 с. – URL: https://ibooks.ru/bookshelf/399520/reading (дата обращения: 27.11.2025).

3. PostgreSQL: Documentation : официальный сайт. URL: https://postgrespro.ru/docs/postgresql/16/sql (дата обращения: 20.11.2025).

УДК 004

ПРОБЛЕМЫ РЕАЛИЗАЦИИ НИЗКОЛАТЕНТНОЙ RTSP ТРАНСЛЯЦИИ НА ПЛАТФОРМЕ ANDROID: АРХИТЕКТУРНЫЕ ОГРАНИЧЕНИЯ И ПУТИ РЕШЕНИЯ

С. Н. Медведев, Ю. С. Синика

Воронежский государственный университет

Аннотация. Статья посвящена анализу причин высокой задержки при воспроизведении RTSP-потоков с IP-камер на мобильных устройствах под управлением операционной системы Android. Описываются архитектурные особенности мультимедийной подсистемы Android, включая изоляцию процессов, механизм безопасности SELinux, модель работы медиасервера и аппаратных декодеров. Отдельное внимание уделено сравнению с подходами, применяемыми на desktop-системах с использованием библиотек VLC, FFmpeg и GStreamer. Проанализировано влияние фрагментации реализаций MediaCodec и особенностей работы механизма Low Memory Killer на работу мобильных устройств. Предложены и классифицированы возможные пути снижения латентности. Показано, что достижение малой задержки требует сочетания архитектурных, программных и инфраструктурных решений, а также учёта ограничений безопасности платформы.
Ключевые слова: видеостриминг, RTSP, IP-камера, Android, MediaCodec, Stagefright, Low Memory Killer, SELinux, WebRTC, SRT, буферизация.

Введение

Потоковая передача видеоданных в реальном времени представляет собой критически важную задачу в ряде технических и прикладных областей, включая системы видеонаблюдения, телеметрию беспилотных летательных аппаратов, дистанционное управление технологическими процессами и промышленную автоматизацию. Для организации потоков широко применяются протоколы прикладного уровня, такие как RTSP (Real Time Streaming Protocol), разработанные для обеспечения минимальных задержек при передаче мультимедийных данных.

На настольных платформах такие решения позволяют достигать приемлемой латентности при воспроизведении видеопотоков. В то же время при переносе аналогичных решений на мобильные устройства, в частности на платформу Android, наблюдается значительное увеличение задержки, что затрудняет использование RTSP-потоков в сценариях, критичных ко времени реакции. Более того, стандартные методы оптимизации, эффективные на desktop-системах, зачастую оказываются недостаточными или неприменимыми на мобильных.

В данной статье приводится комплексный анализ технических причин высокой латентности RTSP воспроизведения на операционной системе Android, включающий рассмотрение архитектурных особенностей платформы, препятствующих достижению показателей, сопоставимых с настольными платформами, а также формулирование практических рекомендаций по разработке RTSP-клиентов для платформы Android с низкой задержкой.

1. Архитектурные особенности мультимедийной подсистемы Android

Мультимедийная подсистема Android с момента появления платформы претерпела существенные изменения, главным образом обусловленные требованиями безопасности, а не производительности. Анализ особенностей её архитектуры необходим для понимания причин высокой задержки при воспроизведении потокового видео.

В ранних версиях операционной системы Android вся мультимедийная обработка осуществлялась единым монолитным процессом mediaserver, обладавшим широким набором приви-

легий, включая доступ к камере, аудиоподсистеме, видеодрайверам, файловой системе и сети. Такая архитектура обеспечивала минимальные расходы на межпроцессное взаимодействие и позволяла быстро обрабатывать медиаданные. Однако критические уязвимости, обнаруженные в 2015 году, продемонстрировали фундаментальную небезопасность данного подхода, поскольку компрометация процесса mediaserver предоставляла доступ ко всем медиафункциям устройства.

Начиная с Android версии 7.0, корпорация Google произвела радикальную реструктуризацию мультимедийной архитектуры, разделив монолитный процесс mediaserver на множество изолированных процессов, каждый из которых получил строго ограниченный набор привилегий. Внутренние процессы начали функционировать независимо друг от друга и были изолированы посредством механизмов SELinux (Security-Enhanced Linux) [1], обеспечивающих обязательный контроль доступа (Mandatory Access Control, MAC) на уровне ядра Linux. Механизм SELinux в Android работает в режиме enforcing, где любое несанкционированное действие не только логируется, но и активно блокируется, что означает невозможность обхода политик безопасности.

Согласно модели безопасности, платформа Android структурирована таким образом, чтобы минимизировать доверие к самой себе и содержать множественные механизмы изоляции компонентов друг от друга [2]. В контексте воспроизведения медиафайлов это означает, что даже при непосредственном вызове медиакодека приложением, он не может напрямую получить доступ к нижележащим ресурсам без соответствующего разрешения от медиасервера в силу действующей MAC-политики в ядре Linux (рис. 1).

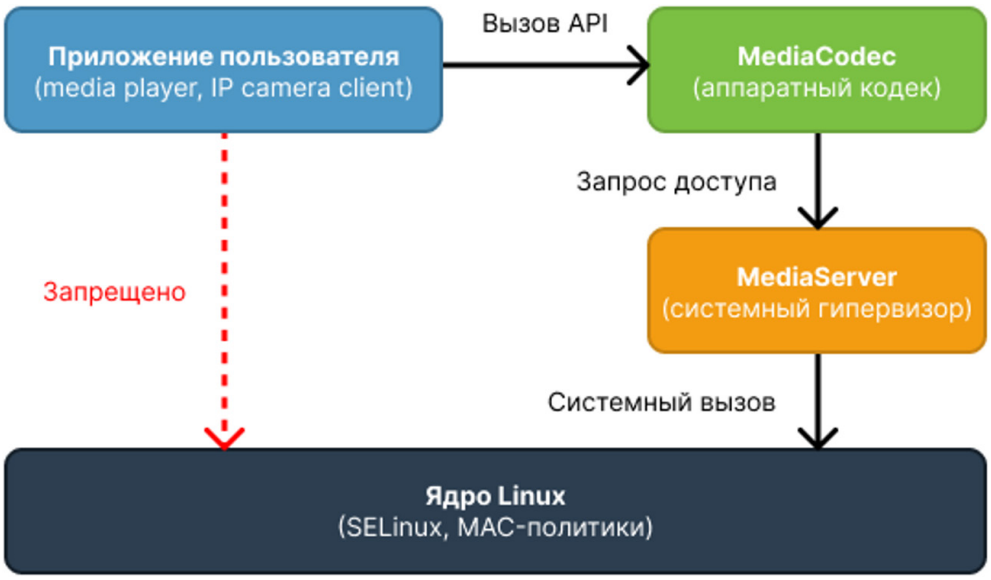

Рис. 1. Изоляция доступа к медиа ресурсам платформы Android

Центральным компонентом мультимедийной архитектуры Android является движок Stagefright, который с момента своего появления в Android версии 2.0 отвечает за воспроизведение и запись аудио и видео. Движок Stagefright реализует сложную систему управления буферами, основанную на концепции пулов буферов (buffer pools). В начале воспроизведения создается пул буферов, которые циклически запрашиваются, заполняются декодированными данными и возвращаются обратно для повторного использования. Критически важным аспектом является то, что память в буфере Stagefright выделяется из драйвера ядра Linux, что делает ее доступной для отображения и использования между процессами. Эта межпроцессная коммуникация создает дополнительные точки синхронизации, которые увеличивают общую латентность обработки медиаданных [3].

В отличие от монолитной архитектуры, где все компоненты находились в едином адресном пространстве, современная изолированная архитектура требует копирования данных между процессами или использования разделяемой памяти с соответствующими механизмами синхронизации.

Низкоуровневый интерфейс MediaCodec, введенный в Android 4.1, предоставляет унифицированный интерфейс доступа к аппаратным и программным кодекам через уровень абстракции OpenMAX IL (Integration Layer). Интерфейс OpenMAX IL является отраслевым стандартом, который позволяет производителям чипсетов предоставлять свои аппаратные кодеки в виде OpenMAX-компонентов, доступных через единый API. Движок Stagefright взаимодействует с этими компонентами через стандарт ACodec, который транслирует вызовы MediaCodec в команды OpenMAX.

С выходом новой версии Android Q (10.0) в 2019 году корпорация Google представила Codec 2.0 как замену интерфейса OpenMAX с целью упростить реализацию кодеков производителями и снизить фрагментацию экосистемы. Codec 2.0 реализует механизм управления буферами с возможностью нулевого копирования для минимизации накладных расходов на копирование данных и повышения производительности. Однако переход на Codec 2.0 происходит медленно, и многие устройства продолжают использовать устаревшую инфраструктуру OpenMAX.

Фундаментальная проблема низкоуровневого API MediaCodec заключается в крайне высокой степени фрагментации реализаций на различных устройствах. Каждый производитель чипсетов реализует свой набор аппаратных кодеков с уникальными особенностями поведения, проблемами и ограничениями. Это создает значительные сложности для разработчиков, которым приходится реализовывать обходные пути для каждой модели устройства.

Отдельно следует учитывать ограничения модели управления памятью операционной системы Android, которая фундаментально отличается от desktop-систем и оказывает существенное влияние на производительность мультимедийных приложений. Android, являясь многопользовательской системой Linux, где каждое приложение представляет отдельного пользователя с уникальным UID, работает в условиях жестких ограничений оперативной памяти. Мобильные устройства имеют относительно малый объем оперативной памяти и должны поддерживать энергоэффективность.

Для управления нагрузкой платформа Android реализует механизм Low Memory Killer (LMK), работающий в связке с процессом kswapd. Когда процесс kswapd не может освободить достаточно памяти через стандартные механизмы, система вызывает сигнал onTrimMemory(), уведомляя приложение о необходимости уменьшить потребление памяти. Если это недостаточно, механизм LMK начинает принудительно завершать процессы на основе показателя oom_adj_score, при этом в первую очередь завершаются фоновые приложения, затем предыдущее приложение, альтернативный лаунчер, сервисы, перцептивные приложения и только в последнюю очередь foreground-приложение и системные процессы. Подробное распределение процессов в зависимости от показателя oom_adj_score отображено на рис. 2.

Критически важным аспектом является то, что операционная система Android уничтожает процессы целиком. Это означает, что, если мультимедийное приложение RTSP-клиента уходит в фоновый режим, система может в любой момент завершить его процесс для освобождения памяти. При возвращении в приложение потребуется полное повторное установление RTSP-соединения, что добавит задержку в несколько секунд.

Производители устройств могут изменять поведение механизма LMK, что создает дополнительную фрагментацию. На устройствах с ограниченной памятью LMK-механизм активируется значительно чаще, приводя к постоянному завершению фоновых процессов и деградации пользовательского опыта.

Рис. 2. Распределение приоритетов завершения процессов LMK

2. Существующие методы вывода RTSP видеопотока на персональных компьютерах и их несовместимость с операционной системой Android

Понимание того, почему решения для desktop-платформ не могут полноценно функционировать на Android, требует детального анализа различий в архитектуре и доступных инструментах.

На настольных платформах задача воспроизведения RTSP-потоков с минимальной задержкой [4] решается использованием мультимедийных библиотек, таких как VLC и FFmpeg [5]. Ключевой особенностью desktop-реализаций является прямой доступ к низкоуровневым системным ресурсам и отсутствие многоуровневой изоляции процессов. Наборы библиотек и инструментов VLC и FFmpeg работают как единые процессы с минимальными расходами, имеют полный контроль над буферизацией на всех уровнях стека от сетевого слоя до видеовывода и могут напрямую взаимодействовать с аппаратными декодерами через интерфейсы VAAPI (Linux), DXVA2/D3D11VA (Windows) или VideoToolbox (macOS).

Попытки использовать те же подходы на платформе Android сталкиваются с фундаментальными архитектурными препятствиями. Библиотека LibVLC, портированная версия VLC для операционной системы Android, теоретически поддерживает те же параметры настройки видеопотока, однако на практике их установка не приводит к снижению задержки до уровней, сопоставимых с настольными платформами.

Фундаментальная причина заключается в том, что библиотека LibVLC на платформе Android вынуждена работать в рамках изолированной архитектуры, взаимодействуя с аппа-

ратными декодерами через интерфейс MediaCodec. Это означает, что даже если библиотека LibVLC минимизирует собственную буферизацию, она не может контролировать буферизацию, происходящую в изолированном процессе mediacodec.

Фреймворк GStreamer на Android, несмотря на теоретическую возможность установки параметра задержки latency равным нулю, на программном компоненте rtspsrc, также не свободен от проблем изоляции процессов. Хотя фреймворк GStreamer имеет превосходную архитектуру для низколатентной обработки на desktop-системах, на платформе Android он вынужден использовать либо программный декодер с высокой нагрузкой на процессор, либо обращаться к аппаратному через интерфейс MediaCodec, наследуя все его ограничения.

Стандартные Android-фреймворки для RTSP-воспроизведения демонстрируют аналогичные проблемы. Встроенный класс MediaPlayer показывает задержку 5-7 секунд при воспроизведении RTSP-потоков, причем латентность накапливается с момента начала воспроизведения и не может быть уменьшена. Набор библиотек ExoPlayer также демонстрирует задержки порядка 1–3 секунд даже при агрессивной настройке с минимальными значениями буферов.

Специализированные библиотеки, такие как rtsp-client-android, заявляют о достижении латентности 20–50 мс через прямое использование аппаратной технологии Android Low-Latency MediaCodec и отказ от буферизации. Однако эта библиотека требует тщательной настройки под конкретные модели устройств из-за фрагментации реализаций интерфейса MediaCodec, и её производительность сильно зависит от качества сетевого соединения.

Еще одним критическим отличием является обработка сетевых пакетов. На desktop-системах приложения могут использовать сырые сокеты и иметь прямой доступ к сетевому стеку с минимальными накладными расходами. Android-система же реализует дополнительный уровень изоляции сетевого доступа через систему разрешений и сетевую подсистему, что добавляет латентность. Более того, управление питанием в операционной системе Android может приводить к агрегации сетевых пакетов для экономии энергии, что увеличивает задержку доставки пакетов.

Протокол RTSP по умолчанию использует протокол UDP для транспорта RTP-пакетов, что обеспечивает минимальную латентность за счет отсутствия подтверждений доставки. Однако в условиях использования технологии NAT (Network Address Translation), типичных для мобильных сетей, UDP-пакеты могут блокироваться, что заставляет клиент переключаться на RTP-over-RTSP (туннелирование RTP через TCP-соединение RTSP). На платформе Android этот переключение на протокол TCP происходит с задержкой, что добавляет несколько секунд к начальной латентности подключения.

3. Возможные решения проблемы для платформы Android

Несмотря на архитектурные ограничения, существует несколько подходов к снижению задержки RTSP-стриминга на Android, каждый из которых имеет свои компромиссы.

Первым и наиболее доступным подходом является тщательная настройка параметров буферизации в используемых библиотеках. Для библиотек ExoPlayer рекомендуется создание кастомного компонента DefaultLoadControl с минимальными значениями длительности буферов. При настройке компонента RtspMediaSource следует явно указывать значение параметра setForceUseRtpTcp как true для использования RTP-over-TCP, что повышает надежность в NAT-окружениях за счет небольшого увеличения латентности. Альтернативно, если известно, что протокол UDP доступен, можно уменьшить таймаут для переключения на TCP через метод setTimeoutMs(), чтобы ускорить переключение при проблемах с использованием протокола UDP.

Второй подход заключается в обходе Java-уровня платформы Android и реализации RTSP-клиента на нативном языке C/C++ через Android NDK. Это позволяет использовать библиотеки настольных платформ, такие как FFmpeg, GStreamer напрямую, минимизируя

накладные расходы интерфейса JNI и получая больший контроль над буферизацией. Преимущество нативного подхода заключается в возможности прямого управления декодированием через FFmpeg или GStreamer декодеры, минимизируя взаимодействие с изолированным процессом mediacodec. Однако программное декодирование H.264 на мобильных процессорах требует значительных вычислительных ресурсов, что может привести к перегреву устройства, сокращению времени автономной работы и пропуску кадров при высоких разрешениях.

Третий подход предполагает переход с RTSP на более современные протоколы потокового вещания, лучше адаптированные для мобильных платформ. Технология WebRTC [6], изначально спроектированный для peer-to-peer коммуникаций с минимальной латентностью, демонстрирует значительно лучшие показатели на Android по сравнению с протоколом RTSP. Однако использование технологии WebRTC требует значительной модификации инфраструктуры, так как IP-камеры традиционно не поддерживают ее нативно. Необходим промежуточный сервер, который будет принимать RTSP-поток от камеры и транскодировать в WebRTC-данные для клиентов. Такая архитектура добавляет задержку транскодирования и требует вычислительных ресурсов на сервере, однако может быть оправдана для приложений с большим количеством клиентов.

Альтернативой является использование протокола SRT (Secure Reliable Transport) [7], который обеспечивает низкую задержку поверх ненадежных сетей и имеет лучшую производительность по сравнению с протоколом RTSP в условиях потерь пакетов. Библиотека RootEncoder для платформы Android поддерживает протокол SRT, предоставляя унифицированный API для потокового вещания. Протокол SRT демонстрирует латентность сопоставимую с протоколом RTSP при стабильной сети, но значительно превосходит его при наличии потерь пакетов благодаря встроенным механизмам ARQ (Automatic Repeat Request).

Для сценариев, где требуется минимальная абсолютная латентность и можно пожертвовать надежностью доставки, возможен переход на чистый протокол RTP/UDP без RTSP-слоя управления. Такой подход устраняет накладные расходы на установление связи с RTSP-потоком и позволяет получать RTP-пакеты напрямую с минимальной буферизацией. Однако он требует статической конфигурации портов и параметров кодирования, отсутствует механизм адаптации к сетевым условиям, и разработчику приходится самостоятельно обрабатывать потери пакетов и восстановление синхронизации.

Четвертый подход связан с системными оптимизациями на уровне системы Android. Критически важным является использование сервиса Foreground для RTSP-клиента, что предотвращает завершение процесса системой в условиях низкой памяти. Сервис Foreground требует отображения постоянного уведомления, что информирует пользователя о работающем приложении, но обеспечивает высокий приоритет процесса. Это гарантирует, что процесс не будет завершен механизмом LMK до тех пор, пока не исчерпаются все фоновые приложения и сервисы.

Для минимизации потребления памяти следует тщательно управлять жизненным циклом объектов MediaCodec и освобождать ресурсы немедленно после использования. Интерфейс MediaCodec может удерживать значительное количество буферов, и задержка их освобождения может привести к исчерпанию лимитов памяти кодека. Для устройств с экстремально ограниченной памятью можно рассмотреть использование меньшего разрешения потока или переключение на более эффективный кодек H.265/HEVC, который обеспечивает лучшее сжатие при том же визуальном качестве. Однако поддержка кодека HEVC на Android фрагментирована, и многие бюджетные устройства не имеют аппаратного декодера HEVC.

Заключение

Проблема высокой задержки при воспроизведении RTSP-потоков с IP-камер на Android-устройствах имеет глубокие архитектурные корни, связанные с фундаментальными принци-

пами безопасности платформы, в результате чего подходы, доказавшие эффективность на настольных платформах с использованием библиотек VLC, FFmpeg и GStreamer, не могут быть прямо применены для платформы Android без учета специфики мобильной архитектуры.

В данной статье был проведен анализ ряда практических путей снижения латентности, каждый из которых представляет компромисс между задержкой, стабильностью и универсальностью решения. Среди таких путей были рассмотрены тщательная настройка параметров интерфейсов ExoPlayer LoadControl и MediaCodec, использование нативных библиотек через инструменты NDK с программным декодированием, переход на более современные протоколы типа, а также использование сервиса Foreground и тщательного управление памятью.

С дальнейшим развитием Codec 2.0 и стандартизацией поддержки низколатентных режимов в интерфейсе MediaCodec рассмотренные пути передачи видеоданных в реальном времени могут утратить практическую ценность, однако для современных Android-устройств фундаментальное противоречие между требованиями безопасности и минимизацией задержки остается актуальным, требуя от разработчиков глубокого понимания архитектуры платформы и готовности к оптимизациям для отдельных типов устройств.

Литература

1. Security-Enhanced Linux in Android [Электронный ресурс] // Android Open Source Project : официальный сайт. – URL: https://source.android.com/docs/security/features/selinux (дата обращения: 16.10.2025).

2. Android's Stagefright Media Player Architecture [Электронный ресурс] // Quandary Peak. – 2013. – URL: https://quandarypeak.com/2013/08/androids-stagefright-media-player-architecture/ (дата обращения: 16.10.2025).

3. Android Codec 2.0: Developing Multimedia Applications for newer Android Platforms [Электронный ресурс] // Ignitarium. – 2025. – URL: https://ignitarium.com/android-codec-2-0-developing-multimedia-applications-for-newer-android-platforms/ (дата обращения: 16.10.2025).

4. *Nikoghosyan K. H.* RTSP video reader with ffmpeg c/c++ library / K. H. Nikoghosyan // Академическая публицистика. – 2022. – № 4-1. – С. 25–29.

5. *Горелов Н. Е.* Исследование методов отображения потокового видео для SCADA-приложений / Н. Е. Горелов, Л. А. Павлов // Информатика и вычислительная техника : сборник научных трудов. – Чебоксары : Чувашский государственный университет им. И. Н. Ульянова, 2024. – С. 105–110.

6. Перспективы применения технологии WEBRTC для дистанционного управления горным оборудованием / А. А. Кравцов, В. И. Анищенко, В. А. Атрушкевич, И. А. Пыталев // Устойчивое развитие горных территорий. – 2020. – Т. 12, № 4(46). – С. 592–599.

7. *Isaka K.* SRT (Secure Reliable Transport) / K. Isaka // Journal Of The Institute Of Image Information And Television Engineers. – 2020. –Vol. 74, № 1. – P. 102–104.

УДК 004.774.6

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ВИЗУАЛИЗАЦИИ ИЗОБРАЖЕНИЙ ДИЗАЙНА ИНТЕРЬЕРА

С. Н. Медведев, Е. Д. Хороших

Воронежский государственный университет

Аннотация. Трехмерное моделирование является неотъемлемой частью процесса дизайна, в частности в последнее время оно широко используется при создании дизайна интерьера. Оно позволяет визуализировать расположение заданных предметов интерьера в помещении, что значительно упрощает процесс анализа различных планировочных решений. Использование нейронных сетей сделало возможным автоматическую расстановку предметов интерьера внутри заданного помещения и получение даже неопытным пользователем результата, сравнимого с работой профессионального дизайнера. Однако, написание приложений, предоставляющих пользователю возможности использования всей мощи нейронных сетей, требует реализации специфических подходов к их созданию. В данной статье рассматривается ряд таких подходов.

Ключевые слова: веб-приложение, 3D-моделирование, фреймворк, рендеринг, язык программирования.

Введение

В работе рассматривается один из подходов к проектированию приложений, позволяющих пользователям с различным уровнем подготовки применять в своей работе предварительно обученные нейронные сети, требующие для своего функционирования сложной настройки и больших вычислительных ресурсов.

Предварительно обученная нейронная сеть функционирует на сложно настроенном мощном оборудовании. Совокупность железа и программного обеспечения, позволяющего функционировать нейросети, назовем сервером рендеринга. В первом приближении он представляет собой кластер физических серверов с мощными графическими ускорителями. Он принимает на вход заранее подготовленные 3D-модели предметов интерьера (рис. 1), расставляет их определенным образом в указанном пространстве, и генерирует на выходе ряд изображений, представляющих 2D-изображения заданных предметов в помещении (рис. 2).

В данной работе 3D-моделями являются предметы интерьера, такие как кровати, диваны и различные декоративные объекты. Сервер рендеринга расставляет их в обозначенной пространстве, называемом сценой, используя определенные внутренние алгоритмы. Он возвращает 2D-изображения помещений, оформленных с использованием заданных предметов интерьера. Процесс преобразования 3D-моделей в 2D-изображения называется рендерингом. Сервер рендеринга изображений генерирует изображения жилого помещения и видеофайл, принимая на вход 3D-модели предметов интерьера.

Отметим, что сервер рендеринга изображений требует достаточных вычислительных ресурсов генерации 2D-изображений. Для быстрого и качественного создания 2D-изображений жилых помещений требуются мощные GPU/TPU и программная экосистема, позволяющая эксплуатировать вычислительную мощь множества параллельных вычислителей.

В силу данных особенностей сервера рендеринга было принято решение разработать веб-приложение, которое предоставляет механизм, обеспечивающий возможность отправить сет товаров, предоставляющий собой несколько изображения предметов интерьера, серверу рендеринга. Также создаваемое программное обеспечение должно предлагать интерфейс для создания сетов из выбранных изображений предметов интерьера и функционал для просмотра сгенерированных 2D-объектов дизайна помещения.

Рис. 1. Примеры изображений 3D-моделей предметов интерьера

Рис. 2. Примеры 2D-изображений расстановки предметов интерьера, созданных сервером рендеринга

1. Обзор инструментария для создания веб-приложений

На данный момент написание веб-приложений «с нуля» представляет собой очень сложный и трудозатратный процесс, поэтому не является целесообразным с точки зрения финансовых и временных затрат. Гораздо эффективнее использовать готовые инструменты разработки для существенного удешевления и ускорения процесса разработки. Такие инструменты разработки называются фреймворками, т.к. в большинстве случаев предоставляют разработчику весь набор необходимых методов для повышения эффективности и ускорения создания готового приложения.

Рассмотрим несколько фреймворков и сравним их между собой по ряду критериев и выберем фреймворк, который позволит наименее трудозатратно реализовать веб-приложение для взаимодействия с сервером рендеринга изображений. Будем рассматривать фреймворки Django [1] (Python [2]) и Laravel [3] (PHP [4]).

Выбор фреймворка для разработки веб-приложения зависит от множества факторов. Рассмотрим основные критерии и особенности каждого из упомянутых фреймворков и выделим несколько критериев выбора:

1. **Опыт и знания разработчика или команды:** насколько разработчики знакомы с синтаксисом языка программирования и особенностями работы с каждым из фреймворков.

2. **Требования к проекту:** функциональность, масштабируемость, необходимость интеграции с другими системами.

3. **Сообщество и документация:** наличие обучающих материалов, активность сообщества, поддержка и обновления.

4. **Подход к разработке:** предпочтения в стиле кодирования, архитектуре приложения, использовании определённых паттернов.

Разберем особенности каждого из рассматриваемых инструментов разработки с точки зрения их преимуществ и недостатков в контексте реализуемого веб-приложения.

2. Laravel (PHP)

Фреймворк Laravel является экосистемой, предназначенной для разработки и тестирования сетевых приложений. Он обладает рядом преимуществ:

- выразительный и элегантный синтаксис;
- обширная документация и множество обучающих материалов;
- мощные функции: внедрение зависимостей, абстракция работы с базой данных, очереди и запланированные задачи, тестирование;
- масштабируемость: поддержка распределённых систем, возможность горизонтального масштабирования;
- активное сообщество и множество сторонних пакетов.

Однако данный фреймворк обладает рядом недостатков. Данный инструмент можно адаптировать практически под любую задачу, в некоторых случаях это нерационально, так как требует слишком много времени и ресурсов. Несмотря на обилие документации, новичкам может быть трудно освоить все возможности системы. А с ростом числа используемых компонентов и пакетов растет объем кода, что делает процессы отладки и обслуживания сложнее.

3. Django (Python)

Фреймворк Django высоко ценят за целый ряд объективных преимуществ, которые сделали его лидирующей технологией в веб-разработке.

Преимущества:

- множество функций «из коробки», включая административный интерфейс;
- мощная ORM [5], удобная работа с базами данных;
- большое и активное сообщество, обширная документация;
- высокая безопасность и надёжность;
- поддержка REST API [7] и GraphQL [8], что удобно для создания API [9];
- простота создания административного интерфейса.

У Django есть все инструменты для создания высоконагруженных приложений. Однако создания небольших сайтов их, зачастую, слишком много. За счет этого, не всегда есть смысл применять фреймворк для таких проектов. Помимо этого, Django может быть сложным для понимания кода начинающими разработчиками из-за обилия метапрограммирования и, как следствие, «магической» природы некоторых функций. Также иногда требуется больше усилий для кастомизации стандартных решений. Здесь надо отметить, что в большинстве случает

такая кастомизация не требуется — гораздо лучшим решением является адаптация методов разработки под предлагаемое фреймворком стандартное решение.

4. Выводы по особенностям рассматриваемых фреймворков

Для того чтобы обеспечить взаимодействие с сервером рендеринга с минимальными трудозатратами, среди рассмотренных фреймворков подходит фреймфорк Django. Он является полнофункциональным инструментом с готовой административной частью, и если разработчик или команда предпочитают язык Python, то Django является оптимальным выбором. Он обладает множеством готовых инструментов, что обеспечивает быструю разработку. Используя возможности данного фреймворка, можно организовать механизм взаимодействия с сервером рендеринга изображений, учитывая все требования по качеству и скорости разработки, а также безопасности клиент-серверного взаимодействия.

Заключение

Таким образом, в данной работе были проанализированы особенности некоторых фреймворков для веб-разработки. Были выбраны некоторые фреймворки для веб-разработки и проанализированы их достоинства и недостатки. Исследование показало, что для разработки веб-приложения, которое должно взаимодействовать с сервером рендеринга, подходит фреймворк Django, поскольку оно обладает обширным количеством встроенных инструментов, что позволит обеспечить наиболее быстрое и удобное получение результатов рендеринга с минимальными трудозатратами.

Литература

1. Django. – Режим доступа: https://ru.wikipedia.org/wiki/Django
2. Python. – Режим доступа: https://ru.wikipedia.org/wiki/Python
3. Laravel. – Режим доступа: https://ru.wikipedia.org/wiki/Laravel
4. PHP. – Режим доступа: https://ru.wikipedia.org/wiki/PHP
5. ORM. – Режим доступа: https://ru.wikipedia.org/wiki/ORM
6. REST API. – Режим доступа: https://ru.wikipedia.org/wiki/REST
7. GraphQL. – Режим доступа: https://graphql.org/
8. API. – Режим доступа: https://ru.wikipedia.org/wiki/API

УДК 004.02

МЕТОД АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ ЦЕПОЧЕК REST API НА ОСНОВЕ СОВМЕСТНОГО АНАЛИЗА BPMN И OPENAPI

А. М. Мельников, И. Е. Воронина

Воронежский государственный университет

Аннотация. В работе рассмотрен алгоритм автоматизации тестирования цепочек REST-взаимодействий на основе BPMN-моделей и спецификаций OpenAPI. Предлагается формализованная процедура сопоставления элементов процессной схемы с операциями веб-сервисов для построения сквозных сценариев и учёта передачи данных между шагами. Цель исследования — разработка алгоритма, обеспечивающего согласованную проверку процессного и интерфейсного уровней описания системы. Актуальность работы определяется ростом сложности цифровых сервисов и увеличением объёма интеграционных взаимодействий, что повышает требования к полноте и надёжности тестирования. Полученные результаты создают основу для дальнейшей автоматизации проверки многошаговых API-процессов.

Ключевые слова: REST API, BPMN, OpenAPI, автоматизация тестирования, сквозные сценарии, процессные модели, спецификации интерфейсов, контрактное тестирование, генерация тестовых данных, сопоставление элементов процесса, валидация данных, цепочки вызовов, интеграционное тестирование, структурный анализ моделей, интеллектуальная генерация данных.

Введение

Развитие современных цифровых платформ сопровождается ростом числа взаимодействующих компонентов и усложнением бизнес-процессов, реализуемых через последовательность обращений к операциям веб-сервисов [1]. Подобные цепочки характерны для финансовых онлайн-сервисов, телекоммуникационных систем, электронного документооборота и других распределённых решений. В таких условиях обеспечение корректности интеграционных взаимодействий становится критически важным, поскольку сбои в логике цепочек API напрямую отражаются на стабильности работы сервисов, достоверности обработки данных и общем пользовательском опыте [2]. Поэтому возникает необходимость в разработке методов, позволяющих повысить надёжность исполнения таких процессов и обеспечить их системное, воспроизводимое и минимально зависимое от ручных операций тестирование. Актуальность исследования определяется необходимостью повышения устойчивости многошаговых цифровых процессов и снижения уровня ручных операций при их тестировании.

Несмотря на широкое распространение инструментов проверки REST API, большинство существующих подходов ориентировано на анализ отдельных вызовов и практически не учитывает контекст бизнес-процесса [3]. Ряд утилит позволяет формировать наборы запросов, однако они не обеспечивают формального связывания шагов процесса с описаниями операций сервиса и не выявляют логические зависимости между обращениями. В то же время процессные модели, используемые аналитиками и архитекторами, содержат критически важную информацию о последовательности действий, условиях выполнения и передаче данных, но традиционно остаются вне рамок автоматизированного тестирования [4]. В итоге формируется методологический разрыв между абстрактным процессным описанием системы и её реальным поведением на уровне API.

Дополнительная сложность заключается в том, что спецификации операций сервисов предоставляют формализованное описание контрактов взаимодействия [5], однако они не имеют

автоматического сопоставления с элементами процессных диаграмм. В результате отсутствует возможность напрямую связывать шаги бизнес-процесса с соответствующими интерфейсными операциями, что затрудняет построение корректных сквозных сценариев и усложняет их сопровождение в условиях постоянной эволюции цифровых сервисов. В этих условиях возрастает потребность в использовании интеллектуальных методов анализа, включая локальные модели искусственного интеллекта, способные выявлять семантические соответствия между элементами BPMN и операциями API. Всё это подчёркивает необходимость разработки решения, объединяющего структурный анализ процессных моделей, спецификаций API и интеллектуальных механизмов обработки данных, что позволит формировать устойчивые, воспроизводимые и автоматизированные тестовые цепочки.

1. Цель, задачи и общая характеристика исследования

Целью представленного исследования является разработка методического подхода к автоматизации тестирования последовательных цепочек REST-взаимодействий на основе интегрированного анализа процессных моделей в нотации BPMN и формализованных спецификаций интерфейсов, описанных средствами OpenAPI.

Для достижения сформулированной цели последовательно решаются следующие задачи:

1. Проанализировать существующие подходы к тестированию API и методы описания бизнес-процессов.
2. Определить формальные правила сопоставления элементов BPMN и структурной модели входных и выходных параметров.
3. Разработать метод генерации сквозных сценариев тестирования на основе анализа модели процесса и описаний служебных операций.
4. Предложить механизм передачи данных между шагами цепочки и валидации контрактов.
5. Создать прототип программной системы, реализующий предложенный метод.
6. Оценить эффективность подхода на типовых примерах API-процессов.

Объектом исследования являются бизнес-процессы, функционирующие через последовательные вызовы REST-сервисов. Предмет исследования включает методы автоматизации тестирования таких процессов, основанные на анализе процессных моделей и формализованных спецификаций интерфейсных контрактов.

Научная новизна исследования отличается разработкой формальной процедуры сопоставления структурной модели процесса с описаниями операций веб-сервисов, позволяющей автоматически выявлять соответствия между элементами BPMN и спецификациями API. Кроме того, новизна проявляется в создании метода автоматизированного формирования тестовых цепочек, обеспечивающего передачу данных между шагами и учёт ограничений, определяемых контрактами API. Предложенный подход отличается тем, что объединяет процессный и технический уровни описания системы и позволяет существенно повысить полноту и согласованность тестового покрытия.

Практическая значимость работы определяется возможностью применения предложенного метода и проектируемой системы в задачах тестирования сложных корпоративных решений. Использование подхода сокращает объём ручных операций, снижает вероятность ошибок, ускоряет анализ возникающих отказов и способствует улучшению качества цифровых сервисов.

2. Анализ существующих подходов и исследований

Современные инструменты тестирования API предлагают широкий набор решений, ориентированных главным образом на проверку отдельных операций REST-сервисов. Наиболее

распространёнными являются программные библиотеки для выполнения HTTP-запросов и утилиты, такие как Postman, Swagger UI, SoapUI [3], обеспечивающие базовую проверку статусов ответов и структуры данных. Однако их функциональность ограничивается изолированным тестированием операций и не охватывает логику многошаговых процессов.

В рамках интеграционного тестирования предпринимаются попытки проверять взаимодействие нескольких сервисов, но существующие подходы остаются фрагментарными: отсутствуют механизмы формального описания последовательности шагов, связей между ними и правил передачи данных. Контрактное тестирование, основанное на JSON Schema и структурных моделях API, позволяет выявлять несоответствия спецификациям [5], однако не учитывает процессную логику и не поддерживает построение сквозных сценариев.

Параллельно развивается направление моделирования бизнес-процессов. Процессные диаграммы в нотации BPMN 2.0 [4] широко применяются для описания маршрута выполнения работ, условий переходов и логики разветвлений. Тем не менее, большинство исследований фокусируется на автоматизации исполнения процессов или анализе бизнес-логики [6]. Применение процессных моделей в задачах тестирования API остаётся ограниченным, а методы формального сопоставления элементов диаграмм с операциями веб-сервисов практически не представлены.

Отдельные работы рассматривают автоматическую генерацию тестовых данных и построение сценариев на основе спецификаций API [7]. Однако эти решения преимущественно анализируют сервисы на уровне отдельных операций и редко учитывают зависимости между шагами, определяемые реальным бизнес-процессом. Это подтверждает существование методологического разрыва между процессным и техническим описанием систем.

3. Предлагаемый подход к автоматизации тестирования цепочек REST API

Разрабатываемый метод автоматизированного формирования сквозных сценариев тестирования предназначен для анализа и проверки последовательных API-взаимодействий. В основе метода лежит интеграция двух типов артефактов: моделей бизнес-процессов, представленных в формате описания логики процесса, и спецификаций REST API, описанных файлами спецификаций API. Совместный анализ этих структур позволяет формализовать процесс построения тестовых цепочек и устранить разрыв между процессным и техническим уровнями описания системы.

3.1. Общая схема алгоритма

Алгоритм включает пять последовательных этапов:

1. Анализ структуры BPMN-модели и выделение порядка выполнения действий.
2. Извлечение операций API из спецификаций OpenAPI, включая параметры, форматы данных и ограничения.
3. Сопоставление элементов процессной схемы и операций веб-сервисов на основе структурных, семантических и контекстных признаков.
4. Построение сквозного тестового сценария с учётом ветвлений и зависимостей данных.
5. Генерация тестовых данных и выполнение цепочки с автоматической проверкой контрактов.

Совокупность этих шагов обеспечивает формализованное и воспроизводимое построение сквозных сценариев тестирования.

3.2. Анализ BPMN-модели

Метод начинается с парсинга BPMN-файла в формате XML. Из модели извлекаются:

- типы задач,
- направления потока управления,
- условия переходов,
- данные, используемые в отдельных шагах.

Использование модели потоков управления позволяет получить формализованное описание последовательности действий и зависимостей между шагами, что является основой для дальнейшего построения цепочки вызовов.

3.3. Извлечение операций API из спецификации OpenAPI

На следующем этапе производится анализ спецификаций API. Из OpenAPI-документа выделяются:

- пути и HTTP-методы,
- схемы входных и выходных данных,
- обязательные параметры,
- ожидаемые коды ответов,
- правила валидации JSON Schema.

Извлечённые сведения формируют структурированное описание поведения сервиса, которое в дальнейшем сопоставляется с процессной моделью.

3.4. Сопоставление BPMN-элементов и операций API

Ключевая часть метода — формальное определение связей между элементами процесса и операциями интерфейса.

Используются три группы признаков:

1. Структурные (ключевые слова в названиях элементов).
2. Семантические (сравнение текстовых описаний с помощью NLP-моделей).
3. Контекстные (анализ входных и выходных параметров).

Результат сопоставления определяется по интегральной метрике. Если она превышает пороговое значение, операция считается соответствующей процессному элементу.

3.5. Построение сквозной тестовой цепочки

После сопоставления формируется единая цепочка вызовов:

- шаги выстраиваются в порядке, определённом потоком управления;
- учитываются альтернативные переходы;
- выявляются зависимости данных между действиями (например, передача идентификаторов).

Полученная цепочка отражает фактическую последовательность взаимодействий в процессе.

3.6. Генерация тестовых данных

Генерация данных осуществляется с использованием комбинированного подхода, включающего:

1) JSON Schema из OpenAPI — для формирования базовой структуры данных;

2) параметры и ограничения, извлечённые из BPMN — для учёта контекстных условий выполнения процесса;

3) механизмы обработки естественного языка (NLP) и локальные модели искусственного интеллекта, применяемые для синтеза содержательных полей (имён, статусов, числовых значений), соответствующих смыслу конкретного шага.

Этот этап устраняет необходимость ручного наполнения сценариев данными.

3.7. Механизм передачи данных между шагами

Для обеспечения целостности цепочки создаётся единый контекст выполнения, куда заносятся:

- идентификаторы созданных сущностей,
- промежуточные параметры,
- входные и выходные данные операций.

Этот контекст используется для автоматического формирования запросов последующих шагов.

3.8. Проверка контрактов и обработка ошибок

Каждый запрос проверяется на соответствие спецификации:

- корректность структуры данных,
- соответствие типов,
- наличие обязательных полей,
- соответствие кодов ответов.

Ошибки фиксируются в отчёте с указанием шага, входных параметров, состояния контекста и полученного результата.

4. Архитектура программного прототипа

Для практической реализации разработанного метода предполагается программный прототип веб-системы, который должен обеспечивать загрузку BPMN-моделей и спецификаций источника структурных требований, автоматическое построение тестовых цепочек и выполнение сформированных сценариев. Архитектура системы описывается как модульная структура, включающая взаимосвязанные компоненты, каждый из которых отвечает за отдельный этап предложенного метода.

4.1. Общая структура программного прототипа

Прототип состоит из следующих функциональных модулей:

1. Модуль анализа BPMN — отвечает за загрузку BPMN-файла, парсинг XML-структуры, извлечение задач, переходов и условий выполнения процесса.

2. Модуль обработки OpenAPI — выполняет разбор спецификаций, извлечение операций API, схем данных и контрактных ограничений.

3. Модуль сопоставления элементов процесса и операций API — реализует алгоритм сравнения объектов формальной модели процесса и источника контрактных данных на основе структурных, контекстных и семантических признаков.

4. Генератор тестовых цепочек — формирует последовательность вызовов, учитывающую логику процесса, ветвления и зависимости между шагами.

5. Модуль генерации тестовых данных — создает входные данные сценария, опираясь на JSON Schema, контекст процесса и NLP-модель.

6. Исполнитель сценариев (Test Runner) — выполняет HTTP-вызовы, управляет контекстом выполнения и проверяет соответствие данных спецификации API.

7. Веб-интерфейс пользователя — обеспечивает визуализацию BPMN-модели, отображение тестовой цепочки, запуск сценариев и просмотр результатов.

Такое разделение обеспечивает гибкость системы и облегчает модификацию отдельных компонентов.

4.2. Технологический стек и используемые инструменты

Прототип реализован с использованием следующих технологий:

- Java 17 — основной язык разработки;
- Spring Boot — построение веб-приложения и REST-слоя;
- Camunda Model API — разбор BPMN-структур;
- Swagger Parser 2.x — извлечение операций из OpenAPI-документов;
- JSON Schema Validator — проверка корректности входных и выходных данных;
- DeepLearning4j — запуск локальных NLP-моделей;
- Jackson — сериализация/десериализация JSON;
- PostgreSQL — хранение результатов тестов и логов;
- React — пользовательский интерфейс.

Выбор технологий обусловлен необходимостью поддержки Java-стека, отсутствием зависимости от внешних облачных сервисов и требованием локального выполнения всех операций.

4.3. Логическая схема взаимодействия компонентов

Работа системы организована в виде следующей последовательности (рис. 1):

1. Пользователь загружает BPMN-файл и документ модели контрактов через веб-интерфейс.

2. Модуль анализа диаграммы формирует внутреннее представление процесса и передаёт его в модуль сопоставления.

3. Модуль обработки OpenAPI извлекает операции API и формирует базу контрактов.

4. Модуль сопоставления определяет связи между задачами процесса и операциями API и передаёт структуру в генератор цепочек.

5. Генератор формирует тестовый сценарий, определяя порядок вызовов и зависимости данных.

6. Модуль генерации данных создаёт входные значения для каждого шага.

7. Исполнитель сценариев выполняет цепочку вызовов, проверяет ответы и записывает результаты.

8. Веб-интерфейс отображает журнал выполнения и структурированный отчёт.

Такой поток данных обеспечивает воспроизводимость сценариев и прозрачность результатов тестирования.

Заключение

В результате проведённого исследования предложен подход к автоматизации тестирования многошаговых REST-взаимодействий на основе объединённого анализа процессных моделей и спецификаций API. Разработанная методика формализует сопоставление элементов бизнес-процесса с операциями веб-сервисов, обеспечивает автоматическое построение сквозных сценариев и позволяет контролировать корректность передачи данных между последовательными шагами. Представленные решения подтверждают, что интеграция описаний логики

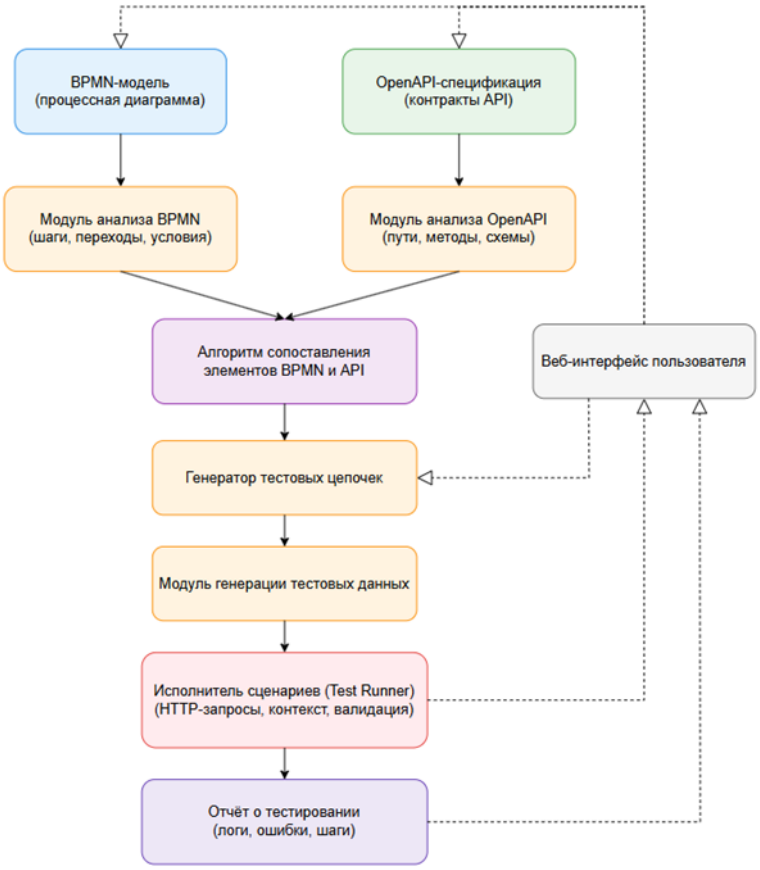

Рис. 1. Архитектурная схема реализации алгоритма автоматизации тестирования API-цепочек

процесса и контрактов API формирует целостную основу для автоматизации тестирования сложных цифровых сервисов.

Проведённый анализ и апробация метода показывают, что использование комбинированного подхода снижает объём ручной подготовки тестов, повышает полноту охвата сценариев и способствует раннему выявлению несоответствий контрактам и логике процесса. Согласование процессного представления и структурной модели данных создаёт более достоверное отражение фактического поведения системы, что особенно важно для распределённых сервисов с высокой плотностью интеграций [1, 3, 6].

Полученные результаты подтверждают практическую ценность разработанного подхода и демонстрируют его применимость для задач обеспечения качества программных систем. Дальнейшее развитие работы может быть связано с поддержкой параллельных ветвей процесса, расширением набора поддерживаемых моделей данных, а также внедрением современных методов искусственного интеллекта для повышения точности сопоставления элементов и генерации тестовых сценариев.

Литература

1. *Ньюман С.* Создание микросервисов. Практическое руководство по разработке распределённых систем. – Москва : O'Reilly Media, 2021. – 352 с.

2. *Рак М.* Тестирование RESTful API: обзор современных подходов : Журнал системной интеграции / М. Рак, А. Куомо, С. Вентичинкве. – 3-е изд. – 2020. – С. 23–35.

3. *Вахтангашвили М.* Практическое тестирование API / М. Вахтангашвили, Дж. Кимбер. – М. : Packt Publishing, 2022. – 298 с.

4. Object Management Group. Нотация и моделирование бизнес-процессов BPMN 2.0. – Стандарт OMG, 2011.

5. Инициатива OpenAPI. Спецификация OpenAPI версии 3.1. – 2021. – URL: https://spec.openapis.org/oas/latest.html (дата обращения: 10.11.2025).

6. Fundamentals of Business Process Management / M. Dumas, M. La Rosa, J. Mendling, H. Reijers. – Springer, 2018. – 527 p.

7. *Ed-dalata M.* Automatic Test Generation for REST APIs Using OpenAPI Specification : International Journal of Software Engineering / M. Ed-dalata, F. Cuppens, N. Cuppens. – 2021. – С. 112–124.

УДК 004.42

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ УЧЕТА И КОНТРОЛЯ СТРЕЛЬБ «SHOTSCOPE»

А. В. Никитцев

Воронежский государственный университет

Аннотация. В статье представлена концепция мобильного приложения «ShotSCOPE», разрабатываемого для автоматизации тренировочного процесса и контроля в пулевой стрельбе. Проведен анализ существующих решений, выявивший их ключевые недостатки. Описаны разработанные алгоритмы, включая метод нормирования поправок для оружия и оригинальный метод оценки кучности стрельбы. Особое внимание уделено реализации безопасной системы авторизации через мессенджер Telegram. В результате спроектирована система, которая комплексно решает задачи сбора статистики, хранения данных и формирования отчетов для спортсменов и тренерского состава.
Ключевые слова: пулевая стрельба, мобильное приложение, тренировочный процесс, кучность стрельбы, поправки прицеливания, Flutter, автономный клиент, Telegram-аутентификация, ShotSCOPE, синхронизация данных.

Введение

В тренировочном процессе спортсменов-стрелков всегда есть контроль как со стороны тренера, так и самостоятельная оценка. Для самоконтроля каждый стрелок имеет собственный дневник, в который он записывает свои результаты и ощущения, полученные во время тренировочного процесса и соревнований. Также результаты самостоятельной работы интересуют и руководство спортшколы, для контроля за тренерской работой.

Тренерский штаб сборной Воронежской области по пулевой стрельбе обратился за разработкой мобильного приложения, автоматизирующего тренировочный процесс и контроль за работой спортсменов и тренеров.

За реализацию данного проекта будет отвечать команда разработчиков, состоящая из трех человек.

Цель проекта — создание системы, которая будет обеспечивать работу следующих функций:

– сбор и обработка тренировочной и соревновательной информации, необходимой для анализа тренерским штабом;

– сбор подробной статистики тренировочной и соревновательной работы спортсменов;

– хранение информации об оружии стрелка (название, серийный номер, какие упражнения стреляются);

– хранение и контроль информации о последнем пройденном медосмотре;

– хранение УИН-ГТО спортсмена с возможностью быстрой выгрузки;

– формирование подробных отчетов о деятельности тренера и спортсмена.

1. Анализ существующих решений

Приложения, схожие по набору функций уже существуют на рынке. Примерами могут служить:

– SureHand [1];

– Shooting Analyzer Pro [2];

– Shoot report [3];

– ShotLog [4];

– МойСпорт [5];
– SCATT [6];
– BlackHole [7].

Все эти приложения не могут в полной мере реализовать все, что необходимо тренеру и спортсмену. Основные недостатки этих приложений:

1. SureHand — работает исключительно на IPhone, используется только английский язык, не имеет взаимодействия спортсмена и тренера, больше не поддерживается.

2. Shooting Analyzer Pro — не поддерживается современными устройствами и не поддерживается с 2019 года, нет русского языка.

3. Shoot report — нет валидации полей, только английский язык, нет подробной статистики для спорстмена.

4. ShotLog — не поддерживается русский язык, неэргономичный интерфейс, большинство функций недоступно в бесплатной версии.

5. МойСпорт — спортсмен не имеет права создавать тренировочные мероприятия, нет раздела подробной статистики спортсмена.

6. SCATT — нет взаимодействия тренера и спортсмена, работает только со специальным датчиком.

7. BlackHole — нет валидации полей, отсутствует русская локализация, часть функций не работоспособны.

В основном приложения не имеют русской локализации, работают на одной платформе (IOS или Android), набор аналитических функций в подобного рода приложениях недостаточен. В некоторых приложениях нет прямой интеграции с таким видом спорта как «Пулевая стрельба». Также весомым недостатком является отсутствие справочной информации по виду спорта.

С учетом всех минусов приложений-конкурентов было принято решение о разработке приложения «ShotSCOPE». Оно будет учитывать потребности заказчика, нюансы конкурентов и иметь свои уникальные функции.

Вариант с веб-приложением был отвергнут еще на начальных этапах, так как это накладывает слишком большие ограничения по пользованию встроенными функциями мобильных устройств, которые могут потребоваться.

2. Архитектура системы

В общем концепте система строится на принципах «толстого» клиента. Требуется высокая автономность клиентского приложения без доступа к удаленному серверу, так как большинство тиров находится в подвальных помещениях без возможности доступа к мобильной сети или интернету.

Таким образом данные, касающиеся конкретного пользователя, будут храниться на устройстве и синхронизироваться с сервером при подключении к интернету. Это позволяет дать пользователю полную автономность, при этом, как только появится возможность система перенесет данные на сервер и пользователь сможет эти данные найти с любого, подключенного к своему аккаунту устройства.

Из-за особенностей алгоритма синхронизации и работы над аналитическими функциями будущего приложения требуется реляционная система управления базами данных (далее — СУБД). Выбор конкретной СУБД будет описан позднее.

При этом важно заметить, что клиентское приложение будет единым для спортсмена и тренера, они смогут входить под разными учетными записями и иметь различный набор функций, поскольку учитывается, что один и тот же человек будет одновременно спортсменом и тренером, поэтому выбор такого подхода обоснован.

3. Работа с поправками

Введем определение поправки, для точного понимания зачем они необходимы спортсмену-стрелку.

Поправка — это осознанное изменение точки прицеливания или настройки прицельных приспособлений для компенсации отклонений пробоин от желаемой точки попадания (как правило, центра мишени).

В сегодняшних реалиях спортсмены могут стрелять в пределах одних соревнований с разных орудий, на каждое из которых выводится своя поправка в зависимости от помещения, освещения и других факторов. Поэтому контроль за изменением точки прицеливания очень важен.

Современные прицельные приспособления на спортивном оружии имеют два барабана регулировки поправок. Один отвечает за изменения по вертикали, другой за изменения по горизонтали. «Количество» поправок определяют щелчками, которые производит барабан при прокручивании в ту или иную сторону. В свою очередь прицельное приспособление не имеет на себе никаких шкал, чтобы четко определять в каком положении оно сейчас находится, поэтому все изменения записываются в виде «количества» поправок по соответствующей оси и в соответствующую сторону.

Был разработан новый метод контроля за прицельным приспособлением каждого орудия спортсмена в отдельности. Он заключается в определении начальной точки пристрелки и дальнейшего пересчета количества обратных поправок для каждого иного места стрельбы.

Принцип работы метода следующий.

1. Изначально считается, что орудие пристрелено к какому-то месту. То есть обозначаем эту точку как (0;0) на координатной плоскости и даем ей название места, например, «Место 1» как на рис. 1.

Рис. 1. Пример «базовой» поправки

2. При добавлении новой поправки будет использоваться «метод нормирования», то есть пересчет координат относительно новой точки отсчета. Теперь добавленная точка становится «базовой», то есть (0;0), а для точки с названием «Место 1» будет произведен пересчет ее координат (или поправок). Так, например, была добавлена поправка «Место 2» со значениями (3;3), результат работы алгоритма показан на рис. 2.

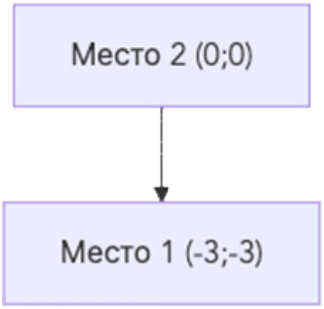

Рис. 2. Пример добавленной поправки

Для метода нормирования используется формула (1)

$$(x', y') = (x - a, y - b), \tag{1}$$

где x', y' — искомые координаты поправки, x, y — координаты поправки на данный момент, при этом a, b — координаты «базовой» точки (по осям x и y соответственно).

Стоит заметить еще несколько нюансов:

– если добавлять поправки дальше, то все предыдущие будут пересчитываться относительно нововведенной;

– если изменять параметры «базовой» поправки, по которой пристрелян пистолет, то все остальные будут пересчитываться относительно новых изменений;

– если изменять параметры поправки, которая не является «базовой» (на данный момент пистолет по ней не пристрелян), то изменения будут вноситься только в нее.

4. Алгоритм нахождения кучности с учетом ошибок стрелка

Кучность является очень важным параметром оценки работы стрелка, поэтому стоит ввести соответствующий термин.

Кучность — это показатель, отражающий степень близости точек попадания пуль друг к другу на мишени после серии выстрелов.

Возникает вопрос о том, как измеряется кучность и как ее искать ведь этот критерий довольно субъективен и будет зависеть напрямую от уровня стрелка. Тут стоит обратиться к таблице нормативов по виду спорта «Пулевая стрельба», утвержденных Минспорта России 20 декабря 2021 года [8]. По ним и будет определяться кучность. Также потребуется таблица международной федерации ISSF [9] определения достоинства пробоин.

Для примера будет взято «олимпийское» упражнение «Пистолет пневматический 60 выстрелов дистанция 10 метров». Теперь необходимо определить метки нахождения кучности. Первая будет отвечать за наличие акцентированной кучности, назовем ее условно *good_normal*, а вторая будет отвечать за выстрелы, которые не будут попадать в кучность, назовем *normal_bad*. Метки будут высчитываться расстоянию до «среднего» выстрела из «средней» серии, которую нужно попасть для выполнения нужного норматива. В итоге должна получиться мишень, выстрелы на которой раскрашены в зеленый, желтый или красный цвета, где зеленые — кучные выстрелы.

Имея соответствующие значения *good_normal* и *normal_bad* и n выстрелов на мишени, будет построен граф по следующим правилам:

1. Каждый с каждым.

2. Ребро строится только тогда, когда его длина меньше заданной метки good_normal.

После этого ищется точка с наибольшим количеством связей и берется за «базовую». Все точки, привязанные к этому выстрелу, будут кучными (зеленые), остальные будут «отрываться» от кучности. Также будет определяться дальность отрыва за счет метки normal_bad, в случае если выстрел расположен дальше, чем ограничивает нас метка, то он отмечается красным цветом, иначе — желтым.

Если такая точка не найдется, то берутся все точки, с максимальным количеством связей и отмечаются как «несколько кучностей» (желтые).

Что такое хорошая кучность? Для любого стрелка отличной кучностью будет считаться попадание всеми выстрелами в «центровую десятку», которая отмечена на рис. 3.

Для того чтобы в процентном соотношении понять на сколько стрельба кучная легко взять отношение двух площадей:

1. Площадь окружности, которую занимает «центровая десятка»

2. Площадь окружности, которая формируется окружностью, центром которой является «базовый» выстрел, а радиусом — расстояние от базового выстрела до самого далекого «кучного» выстрела.

Таким образом получим формулу (2)

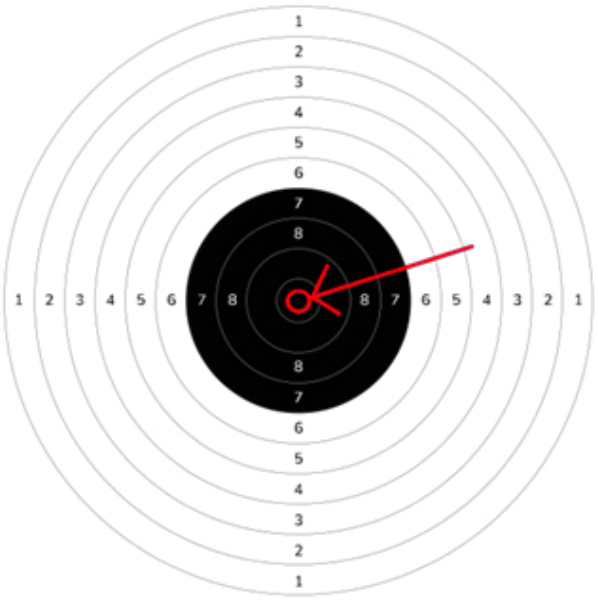

Рис 3. «Центровая десятка» на мишени для пневматического пистолета

$$\frac{\pi r^2}{\pi f^2} = \frac{r^2}{f^2} = \frac{r}{f}, \tag{2}$$

где r — радиус «центровой десятки», а f — расстояние между базовым выстрелом и самым удаленным в кучности.

5. Средства реализации

В условиях ограниченной команды разработчиков был выбран кроссплатформенный подход для оптимизации ресурсов. Среди рассмотренных решений React Native был отклонен в связи с необходимостью освоения JavaScript, а Xamarin — из-за отсутствия полноценной кроссплатформенности для интерфейсов и недостаточной представленности на рынке.

В результате выбран Flutter, который, несмотря на использование языка Dart, продемонстрировал высокую производительность и имеет растущую экосистему.

Для управления данными серверной части выбрана PostgreSQL как знакомая команде СУБД, а для локального хранения — модуль SQLite для обеспечения совместимости с мобильной платформой.

6. Требования к аппаратному и программному обеспечению

Для работы мобильного приложения необходимо мобильное устройство с операционной системой Android (версии 8.0 или новее) или IOS (версии 13 и новее).

Также для синхронизации с общей базой данных требуется подключение к интернету.

7. Реализация

7.1. Аутентификация

Для безопасного входа в мобильное приложение требуется полноценная аутентификация. Для нее потребуется логин и пароль, которые необходимо в зашифрованном или захешированном виде хранить на сервере. В целом за собой этот подход несет большой груз ответственности и детальное продумывание всей подсистемы аутентификации пользователя.

Было найдено решение, которое обеспечит безопасность, облегчит разработку, а не замедлит ее.

Для авторизации пользователей будет использоваться мессенджер «Telegram». Он является популярным и защищенным приложением с двухфакторной аутентификацией пользователей. С его помощью будет осуществляться вход в приложение. Telegram-бот будет осуществлять роль приложения-аутентификатора, который будет отдавать пользователю ключи для входа с ограниченным временем действия.

Когда новый пользователь создает свой профиль на сервер отправляется запрос на его создание в базе данных (далее — БД). Как только сервер осуществит создание пользователя, то он отдаст на клиентское приложение специальный код аутентификации. Пользователь должен использовать этот код вставив в Telegram-бота, который завершит процесс регистрации пользователя, привязав к его профилю аккаунт Telegram, через который в дальнейшем будет осуществляться процесс авторизации.

В случае авторизации пользователя алгоритм будет выстроен немного иначе. Когда пользователь хочет войти свой аккаунт он пишет специальную команду (*/entry*) в Telegram-бота, который присылает ему одноразовый ключ для доступа в приложение. Бот обращается на сервер для поиска соответствующего пользователя и непосредственной генерации ключа, чтобы прислать его пользователю.

7.2. Клиент-серверное взаимодействие

В рамках работы системы сервер будет выполнять роль общего хранилища, который будет получать запросы с пользовательских устройств и перенаправлять их в базу данных.

Все данные, которые будут пользователем создаваться, требуют перемещения на постоянное хранение на сервере, для дальнейшего их использования на других устройствах. Так как на устройстве и на сервере планируются реляционные базы данных, то для этого будет использоваться механизм синхронизации двух БД. Также на удаленном сервере будет запущен Telegram-бот, осуществляющий авторизацию.

Таким образом сервер будет выполнять следующие функции:

1. Хранить данные о пользователях.
2. Осуществлять авторизацию через Telegram-бота.
3. Осуществлять прием и обработку запросов на синхронизацию.
4. Осуществлять синхронизацию.

7.3. Бизнес-логика

Для реализации бизнес-логики используется слой «сервисов», который реализует функции бизнес-логики, независимые от слоя интерфейса. Таким образом на этом участке реализуются принципы проектирования программного обеспечения SOLID [10].

Так, например реализована логика работы с оружием при следующей иерархии модулей, показанной на рис. 4.

Видно явное разбиение слоев, при этом сам класс WeaponsService реализует статичные методы, для работы соответствующих элементов интерфейса по заданной логике.

Заключение

В данной работе рассмотрена реализация системы учета и контроля стрельб «ShotSCOPE», которое автоматизирует тренировочный процесс спортсменов по пулевой стрельбе.

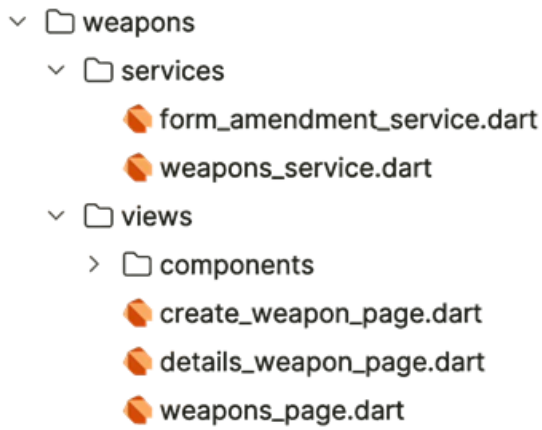

Рис. 4. Иерархия модулей для работы с оружием

Проведенный анализ предметной области и существующих решений позволил выделить уникальные особенности приложения, такие как поддержка русского языка, кроссплатформенность и интеграция с Telegram для авторизации. Выбор технологий (Flutter, PostgreSQL, SQLite) обеспечил высокую производительность и масштабируемость системы.

Разработанные алгоритмы, включая метод нормирования поправок и анализ кучности, демонстрируют инновационный подход к решению задач спортсменов и тренеров. Архитектура приложения, основанная на принципах автономности и синхронизации, гарантирует удобство использования даже в условиях отсутствия интернет-соединения.

На текущем этапе разработки мобильное приложение «ShotSCOPE» обладает базовой функциональностью и готово к внедрению в тренировочный процесс для сбора первичной обратной связи. В качестве ближайших задач развития проекта планируется реализация расширенной аналитики, а также публикация стабильной версии в официальных магазинах приложений. Конечной целью проекта является формирование готового программного продукта, который будет включать не только мобильное приложение для спортсменов, но и раздел для тренеров с функциями углубленной статистики и управления группами спортсменов.

Работа выполнена под научным руководством старшего преподавателя кафедры программного обеспечения и администрирования информационных систем Воронежского государственного университета, Матвеевой Марии Валерьевной.

Литература

1. Приложение SureHand. – URL: https://surehand.digital/ (дата обращения: 30.04.2025)

2. Приложение Shooting Analyzer Pro. – URL: http://www.sts.in/ (дата обращения: 30.04.2025)

3. Приложение Shoot report. – URL: https://www.burkhardt-sport.solutions/ (дата обращения: 30.04.2025)

4. Приложение ShotLog. – URL: https://apps.apple.com/us/app/shotlog-range-journal/id1445374487 (дата обращения: 30.04.2025)

5. Приложение МойСпорт. – URL: https://moisport.ru/ (дата обращения: 30.04.2025)

6. Приложение SCATT. – URL: https://www.scatt.ru/ (дата обращения: 30.04.2025)

7. Приложение BlackHole. – URL: https://www.blackhole-app.com/en/ (дата обращения: 30.04.2025)

8. Приказ Минспорта России от 20.12.2021. – URL: https://base.garant.ru/403336703/1d48ab41ceb4b406e6d59ae55621977c/ (дата обращения 30.04.2025)

9. Сайт ISSF. – URL: https://www.issf-sports.org (дата обращения: 30.04.2025)

10. *Контьери М.* Рецепты чистого кода / М. Контьери ; пер. с англ. Н. Григорьевой ; науч. ред. М. Мамиева. – Астана : Спринт Бук, 2024. – 411 с.

УДК 004.77

МОДЕРНИЗАЦИЯ КООРДИНАТНОГО ДОМОФОННОГО ПЕРЕГОВОРНОГО УСТРОЙСТВА

Е. В. Проценко, И. Е. Воронина

Воронежский государственный университет

Аннотация. Рассматривается архитектурное решение задачи модернизации абонентских переговорных устройств многоабонентных домофонных систем до уровня устройств умного дома. Предполагается создание универсального решения, не требующего вмешательства в общедомовое оборудование и сохраняющее обратную совместимость с существующими многоабонентными домофонными системами. В работе описывается подход к интеграции аудиосвязи и управляющих функций в локальную экосистему умного дома, включая использование локального сервера.
Ключевые слова: координатный домофон, модернизация, интернет вещей, умный дом, адаптация legacy-оборудования, internet of things, esp32, WebRTC, matter, rtc.

Введение

Координатные, многоабонентные домофонные системы широко распространены на территории России и стран СНГ. Эти системы отличаются простотой, надёжностью и низкой стоимостью обслуживания. Однако функциональные возможности абонентских переговорных устройств (АПУ) остаются ограниченными: отсутствует удалённое управление и интеграция с системами умного дома.

Полная замена общедомового домофонного оборудования сопряжена со значительными финансовыми затратами и организационными сложностями. Это требует значительных финансовых вложений и согласований с собственниками помещений, управляющими компаниями и обслуживающими организациями. На практике такие проекты оказываются сложными для внедрения и редко реализуются.

Таким образом актуальной задачей является модернизация АПУ — конечного элемента, находящегося в распоряжении пользователя. В рамках этого подхода АПУ рассматривается как «чёрный ящик», подключённый к существующей аналоговой линии домофона, а все интеллектуальные функции выполняются локальными вычислительными устройствами и сервисами [1].

1. Устройство координатных домофонных систем

Координатные домофоны используют матричную схему адресации: каждое АПУ подключено к своему сочетанию линий десятков и единиц, формируя уникальный адрес (рис. 1). Коммутатор определяет абонента по замыканию соответствующих координат, подаёт двухтоновый вызов, а при снятии трубки приводит к прекращению подачи сигнала вызова и установлению режима связи между посетителем и абонентом. Таким образом, используется двухпроводная линия, обеспечивающая:

– передачу сигнала вызова;

– определение состояния трубки;

– голосовую связь в обе стороны;

– управление электромеханическим замком посредством кратковременного изменения сопротивления цепи.

Система работает по принципу, аналогичному двухпроводным телефонным линиям: одни и те же контакты используются как для питания, так и для передачи аудиосигнала, что достигается благодаря гибридной схеме разделения направлений сигнала.

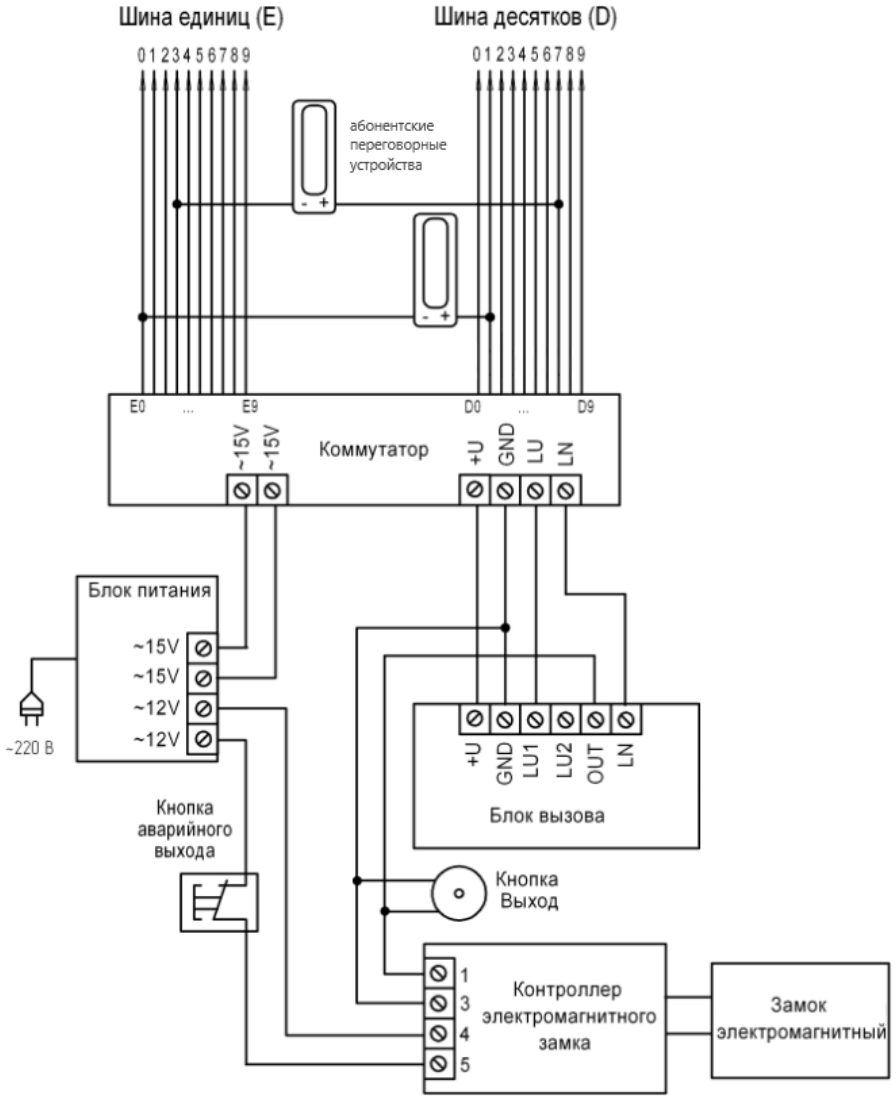

Рис. 1. Схема подключения многоабонентных домофонных систем

2. Цель модернизации

Модернизация устройства подразумевает расширение функциональности трубки с сохранением полной совместимости с существующей многоабонентной домофонной системой, включая:

– удалённое открытие двери;
– удалённую полную дуплексную связь;
– возможность записи разговоров;
– мобильные уведомления о вызове;
– режимы звонка (громко/тихо/выключено);
– пользовательские рингтоны;
– интеграция с умным домом.

3. Архитектурное решение

Архитектура аппаратно-программного комплекса (АПК) разработана по принципу «клиент-сервер». На рис. 2 приведена схема архитектуры, которая разделена на три ключевых функциональных блока:

– модернизированная АПУ;

– локальный сервер;

– интерфейс пользователя.

Несмотря на то, что *WebRTC* описывается как технология прямого соединения (*peer-to-peer*), для её работы всегда требуется сигнальный локальный сервер, обеспечивающий обмен: *SDP*-описаниями медиапотоков и *ICE*-кандидатами.

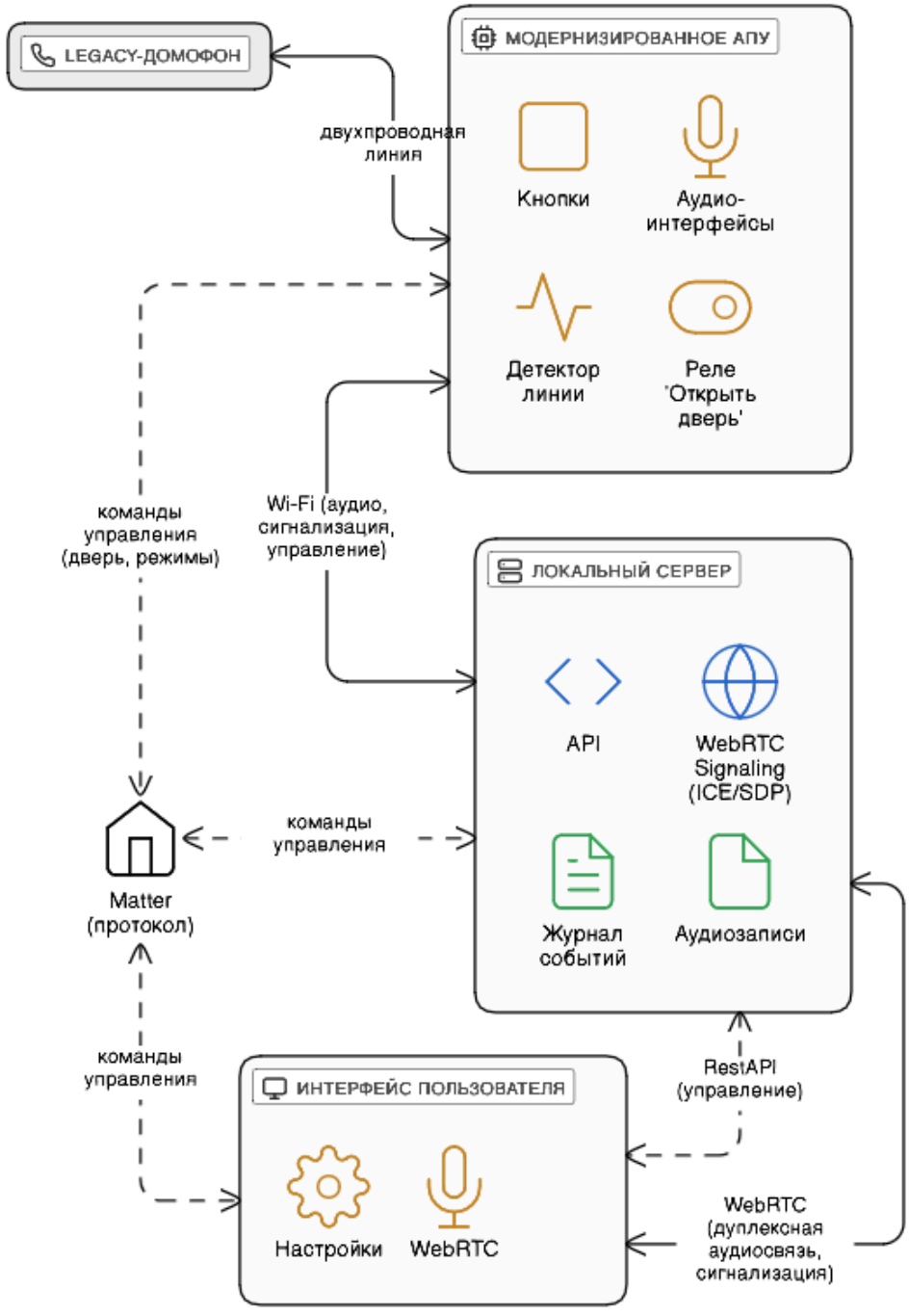

Рис. 2. Архитектура АПК

3.1. Модернизированное АПУ

Модернизированная АПУ представляет собой устройство с микроконтроллером, осуществляющее преобразование аналогового сигнала координатного домофона в цифровой поток данных (и наоборот), и поддержку интеграции *Matter* [2].

В базовой конфигурации *ESP32* использует 12-битный АЦП. Преобразованный аудиосигнал проходит стадию компандирования — нелинейного сжатия динамического диапазона. Компандирование *μ-law* (Америка) или *A-law* (Европа) реализуется табличными преобразованиями, что снижает нагрузку на микроконтроллер (рис. 3).

Рис. 3. Схема преобразования аудиосигнала

Предусмотрены следующие органы управления:
– кнопка «открыть дверь»;
– кнопка записи разговора;
– переключатель режимов звонка (тихо/громко/выкл.).

3.2. Локальный сервер

Локальный сервер выполняет ресурсоёмкие задачи для микроконтроллера и является ключевым компонентом архитектуры. Его роль — обеспечить медиакоммутацию, сигналинг *WebRTC*. В экосистемах умного дома хаб выполняет следующие функции:
– единой точки интеграции устройств;
– маршрутизации команд;
– хранения истории;
– выполнения автоматизаций;
– обеспечения совместной работы разных протоколов (*Zigbee*, *Wi-Fi*, *Matter* и др.).

В табл. 1 приведено сравнение популярных экосистем умного дома согласно целям модернизации. Полноценную поддержку с дуплексным аудио обеспечивает только *Home Assistant* [3, 4]. В версии *Matter* 1.5 добавлена поддержка *WebRTC* прямо в спецификации, но в исходном коде поддержка дуплексного аудио — отсутствует [5, 6].

3.3. Интерфейс пользователя

При использовании платформ умного дома, таких, как *Home Assistant*, *HomeKit*, *Google Home*, *Alexa Smart Home*, *SmartThings* и др. — разработчик не создаёт frontend-интерфейс вручную, т. к. он автоматически генерируется на основе файлов конфигурации (например, *YAML*, с корректной декларацией функций устройства).

Однако встроенные *frontend*-интерфейсы не предназначены для приёма и обработки входящих *WebRTC*-вызовов. В связи с этим необходим дополнительный специализированный веб-интерфейс, работающий на локальном сервере и обеспечивающий:
– отображение состояния вызова;
– приём и передачу аудиопотоков в режиме *WebRTC*;
– взаимодействие с аппаратурой АПУ (микрофоном, динамиком, кнопками).

Фрагмент интерфейса, реализующего эту функциональность, представлен на рис. 4.

Таблица 1

Сравнительная таблица архитектур

Экосистема	Поддержка WebRTC	Поддержка дуплексного аудио	Заявлена поддержка протокола Matter	Возможность работы локально без интернета
Home Assistant	Да	Да (через сторонний плагин)	Да (плагин)	Да (полностью автономно)
Amazon Alexa Smart	Да (нет публичного API, для камер)	Нет	Да	Нет (требуется облако Amazon)
Google Smart Home	Да (для камер)	Нет	Да	Нет (полностью облачно)
Apple HomeKit	Нет (нет публичного API)	Нет	Да	Да (локально с iOS Hub)
Samsung SmartThings	Да (для камер)	Нет	Да	Нет (облачное ядро)
OpenHAB	Нет	Нет	Да (плагин)	Да (полностью локально)
Hubitat	Нет	Нет	Да	Да (локальная работа)

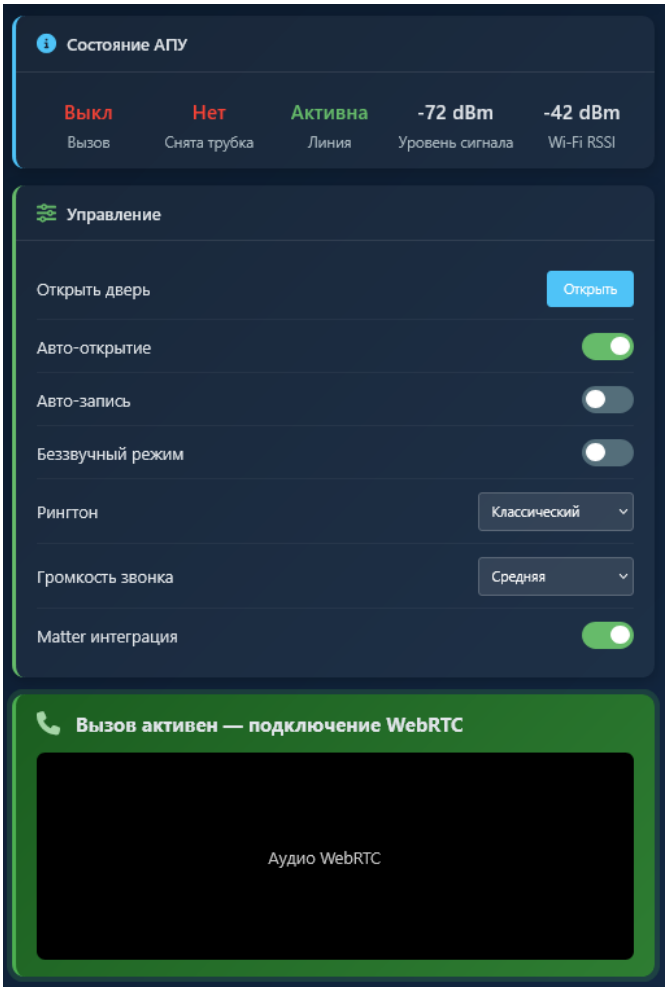

Рис. 4. Дополнительный веб-интерфейс

Заключение

Предложенная архитектурная концепция модернизации абонентского переговорного устройства для координатных домофонов отличается тем, что необходимо минимальное вмешательство в общедомовое оборудование и поддерживается обратная совместимость с существующими многоабонентными домофонными системами. АПУ выполняет роль «тонкого клиента» и обеспечивает перенос вычислений на локальный сервер и расширение набора функций до уровня полноценного устройства умного дома.

Устройство способно повысить качество жизни для маломобильных групп населения, включая людей с ограниченными возможностями передвижения и лежачих больных, поскольку предоставляется возможность удалённого управления домофоном. Тем самым устраняется барьер, связанный с необходимостью физического перемещения к трубке.

Предложенное решение имеет социальную значимость и возможность широкого внедрения.

Литература

1. *Перегудов Ф. И.* Введение в системный анализ : учебное пособие для вузов / Ф. И. Перегудов, Ф. П. Тарасенко. – Москва : Высшая школа, 1989. – 367 с.

2. ESP-Matter Solution [Электронный ресурс]. – Espressif Systems. – URL: https://www.espressif.com/en/solutions/device-connectivity/esp-matter-solution#sdk-for-matter (дата обращения: 20.11.2025).

3. Home Assistant Documentation [Электронный ресурс]. – Home Assistant. – URL: https://www.home-assistant.io/docs/ (дата обращения: 20.11.2025).

4. home-assistant / core [Электронный ресурс] : репозиторий. – GitHub. – URL: https://github.com/home-assistant/core/tree/ac465689965ca43e141ce215b215f1264368da96/homeassistant/components/go2rtc (дата обращения: 20.11.2025).

5. Matter 1.5 introduces cameras, closures and enhanced energy management capabilities [Электронный ресурс]. – CSA Connection Standards Alliance, 2023. – URL: https://csa-iot.org/newsroom/matter-1-5-introduces-cameras-closures-and-enhanced-energy-management-capabilities/ (дата обращения: 20.11.2025).

6. project-chip / connectedhomeip [Электронный ресурс] : репозиторий. – GitHub. – URL: https:// https://github.com/project-chip/connectedhomeip/blob/master/src/controller/webrtc/WebRTCClient.cpp (дата обращения: 20.11.2025).

УДК 004

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ «K-TALK» ДЛЯ ИЗУЧЕНИЯ КОРЕЙСКОГО ЯЗЫКА

А. Н. Родионова, Т. В. Курченкова

Воронежский государственный университет

Аннотация. Статья посвящена разработке мобильного приложения для изучения корейского языка «K-Talk». Цель — создать комплексный инструмент, адаптирующийся под начальный уровень пользователя. В работе описаны этапы проектирования: постановка задачи, разработка логической модели данных, интерфейса, реализация функционала с помощью языка программирования Java и среды разработки Android Studio. Основными элементами являются входное тестирование, на основе результатов которого выстраивается дальнейшее обучение, и интеграция мультимедийного контента.
Ключевые слова: мобильное приложение, изучение языков, корейский язык, Android-разработка, Android Studio, Java, модели данных, тестирование начального уровня, персонализация обучения, адаптивный контент.

Введение

С каждым годом всё больше людей открывают для себя мир корейской культуры, из-за чего растёт интерес к изучению корейского — родного языка любимых артистов и актёров. А некоторые уже владеют основами и нуждаются в практической прокачке имеющихся знаний.

При этом большинство существующих приложений либо рассчитано только на начинающих, либо содержит мало интересных практических заданий.

В статье рассматривается разработка мобильного приложения, позволяющего учиться в соответствии с запросами и стартовым уровнем пользователя.

1. Постановка задачи

Разработать мобильное приложение для изучения корейского языка. Программа должна включать следующую функциональность:

- авторизация и аутентификация пользователей;
- ведение личного кабинета: заполнение данными профиля пользователя, их изменение;
- тестирование пользователя для определения начального уровня языка;
- прохождение урока;
- просмотр дополнительных материалов (фильмы, видео, книги, новости);
- добавление карточек для запоминания слов;
- добавление заметок.

2. Логическая модель данных

В процессе разработки приложения был проведён анализ предметной области с учётом пожеланий и потребностей пользователей. Исходя из результатов анализа, была спроектирована модель данных, отражающая структуру системы и взаимосвязи между её компонентами.

Эта модель данных используется для реализации функциональности приложения и обеспечения качественного взаимодействия между его модулями.

Логическая модель данных в нотации Баркера представлена на рис. 1.

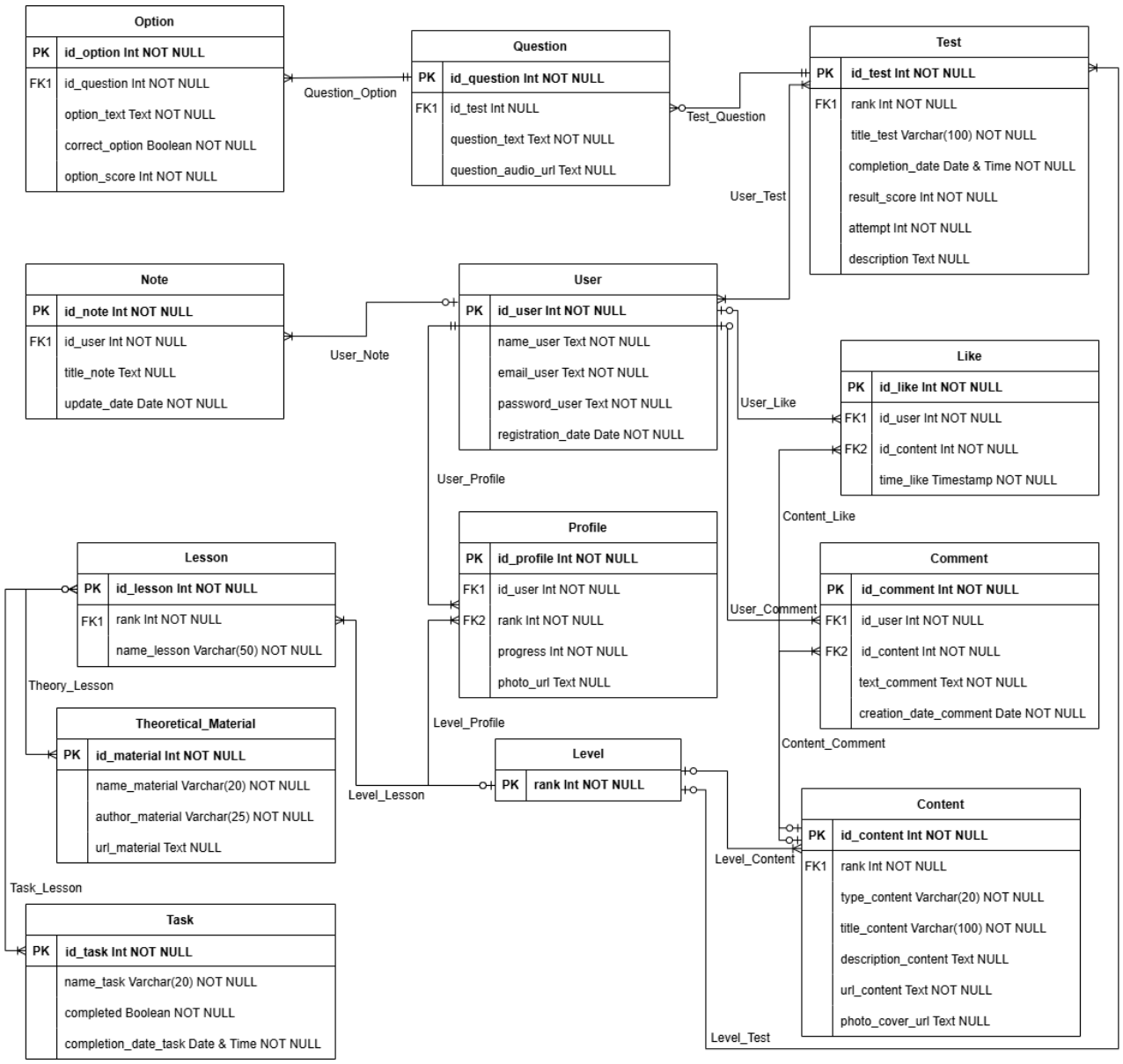

Рис. 1. Логическая модель данных

Сущность *Пользователь* (User) хранит в себе данные о пользователе: идентификатор, ник, почта, пароль, дата регистрации.

Сущность *Профиль* (Profile) содержит информацию, которая отражается в профиле пользователя: идентификатор профиля, идентификатор пользователя, уровень знания языка, показатель прогресса, ссылка на фотографию профиля.

Сущность *Уровень* (Level) хранит данные об уровне знания языка: ранг.

Сущность *Урок* (Lesson) содержит информацию об уроке: идентификатор урока, уровень знания языка, тема урока.

Сущность *Теоретические материалы* (Theoretical_Material) хранит в себе данные о теоретической части урока: идентификатор материала, название материала/тема, автор, ссылка на материалы.

Сущность *Задание* (Task) содержит информацию о практической части урока: идентификатор задания, формулировка задания/тема, статус задания (выполнено/не выполнено), дата выполнения.

Сущность *Тест* «Test» хранит данные о тестах, которые пользователь проходит после изучения темы и для определения уровня языка: идентификатор теста, уровень знания языка, тема теста, дата прохождения, результат, номер попытки, описание.

Сущность *Вопрос* (Question) содержит информацию о вопросе теста: идентификатор вопроса, идентификатор теста, формулировка вопроса, ссылка на аудио для вопроса на аудирование.

Сущность *Вариант ответа* «Option» хранит в себе данные о вариантах ответа на задания теста: идентификатор варианта, идентификатор вопроса, содержание ответа, верный ответ, балл за ответ.

Сущность *Дополнительные материалы* (Content) содержит информацию о дополнительном контенте для пользователя (новости/книги/видео/фильмы): идентификатор контента, уровень знания языка, тип контента, наименование, описание, ссылка на источник, ссылка на обложку фильма/видео/книги.

Сущность *Заметка* (Notes) содержит информацию о заметках пользователя: идентификатор заметки, идентификатор пользователя, название заметки, дата обновления.

Сущность *Лайки* (Likes) хранит в себе информацию на реакции в виде «лайков», которые может оставлять пользователь: идентификатор лайка, идентификатор контента, идентификатор пользователя, время, когда была поставлена реакция.

Сущность *Комментарий* (Comment) — сущность с данными о реакциях в виде комментариев пользователя: идентификатор комментария, идентификатор контента, идентификатор пользователя, содержание комментария, дата, когда был оставлен комментарий.

Сущность *Пользователь_Урок* (User_Lesson) является промежуточной, она соединяет сущности *Пользователь* (User) и *Урок* (Lesson) и хранит в себе: идентификатор пользователя, идентификатор урока, статус урока (пройден/не пройден).

3. Описание приложения

Для реализации приложения использовался язык программирования Java [1, 2], среда разработки Android Studio [3], для создания пользовательского интерфейса — язык разметки XML [4].

В соответствии с функциональными требованиями, на основе спроектированной модели данных, было разработано мобильное приложение «K-Talk». Рассмотрим его более подробно.

Для начала использования приложения требуется вход в аккаунт или регистрация. После входа в приложение система предлагает пройти тест, чтобы определить пользователю его текущий уровень знания языка. Сначала можно ознакомиться с общей информацией о тесте (рис. 2), который включает в себя задания на: аудирование (рис. 3), чтение (рис. 4) и письмо (рис. 5).

На странице с описанием теста показаны его структура и длительность. При нажатии кнопки *Пропустить* пользователь переходит в профиль (рис. 7), при нажатии кнопки *Начать* — приступает к вопросам.

В каждом блоке вопросы оформлены одинаково: верхняя часть экрана содержит задание на корейском и на русском, далее приведён пример ответа. На экране отображается таймер, который показывает оставшееся время на прохождение теста.

Каждому заданию соответствует определённое количество баллов в зависимости от сложности.

Переход к следующему вопросу осуществляется посредством нажатия на кнопку *Продолжить* при наличии ответа на задание или *Пропустить вопрос*, если пользователь не знает ответа или не уверен в нём.

Для заданий на аудирование предоставляется аудиофайл, для заданий на чтение и письмо — разные тексты. По содержанию и структуре тест близок к реальному TOPIK — международному экзамену на определение уровня владения корейским языком.

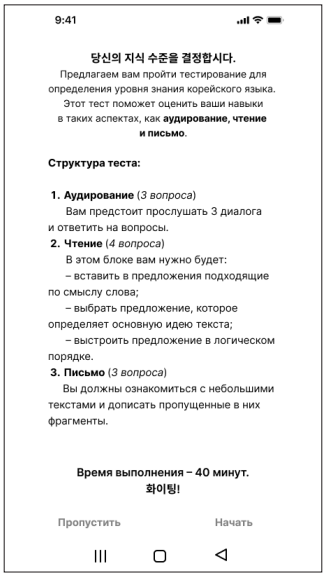

Рис. 2. Общая информация о входном тесте

Рис. 3. Пример задания на аудирование

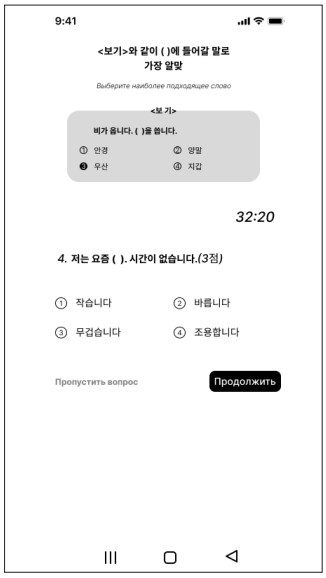

Рис. 4. Пример задания на чтение

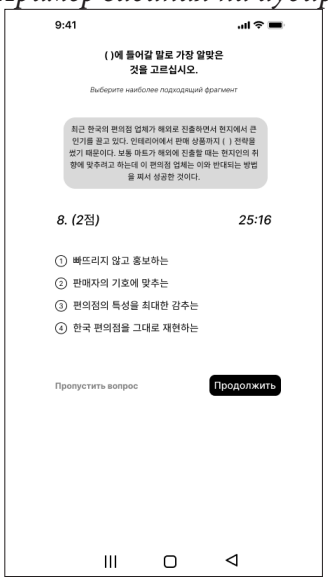

Рис. 5. Пример задания на письмо

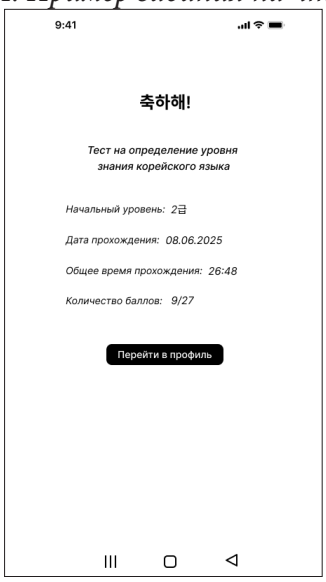

Рис. 6. Результаты теста

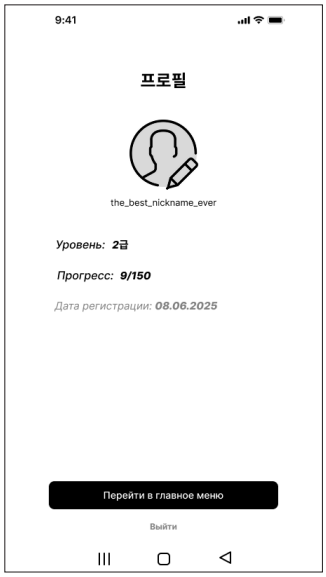

Рис. 7. Профиль пользователя

Рис. 8. Главное меню

В конце теста доступна только кнопка *Завершить* — при её активации результаты сохраняются и отображаются пользователю на экране (рис. 6).

На странице результатов также есть кнопка *Перейти в профиль*, где хранится информация, связанная с прогрессом пользователя. Из этого окна можно перейти в главное меню (рис. 8) с новостями из мира k-pop и боковым меню, откуда можно перейти в такие разделы, как *Уроки*, *Видео*, *Фильмы*, *Книги*, *Словарь* и Заметки.

Раздел *Уроки* содержит перечень тем для изучения. После выбора нужной темы открывается страница урока (рис. 9, 10): в верхней части — название темы и кнопка для определения статуса прохождения урока, далее — теоретический материал (в правом нижнем углу этого блока есть кнопка с ссылкой на допматериалы), ниже — практические задания и кнопка для перехода к тесту по этой теме.

В разделах *Видео*, *Книги* и *Фильмы* используется единый формат (рис. 11): контент сгруппирован по уровням владения языком, и для каждого из них предлагаются материалы, соответствующие этому уровню.

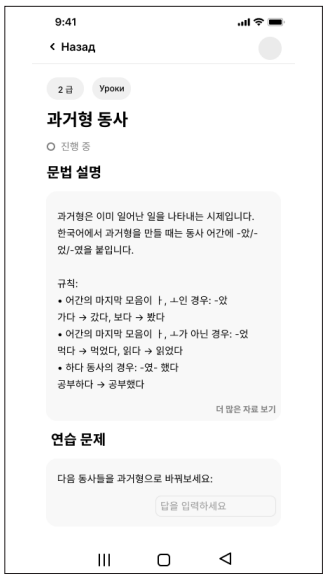

Рис. 9. Пример урока

Рис. 10. Пример урока (продолжение)

Рис. 11. Раздел Книги

Рис. 12. Словарь

В разделе *Словарь* (рис. 12) доступна функция добавления карточек для запоминания слов/фраз. Уже существующие карточки редактируются через кнопку *Изменить*, расположенную внизу самой карточки. Кнопка *Добавить карточку* в нижней части экрана позволяет создать новую карточку с переводами на русский и корейский языки.

Раздел *Заметки* оформлен аналогично разделу Словарь: на странице — перечень пользовательских заметок с кнопкой редактирования; над каждой заметкой стоит дата её изменения, а в нижней части экрана находится кнопка *Добавить заметку*.

Заключение

Было разработано мобильное приложение, позволяющее комплексно подходить к вопросу изучения корейского языка.

Приложение обеспечивает удобную среду для освоения и понимания языка, позволяя акцентироваться не только на теории и практике из учебников, но и разнообразить процесс обучения использованием интерактивных материалов. Мобильное приложение «K-Talk» будет полезно как новичкам, так и более опытным ученикам для освоения корейского языка.

Литература

1. *Шилдт Г.* Java. Полное руководство / Г. Шилдт. – 12-е изд., пер. с англ. – Санкт-Петербург : ООО «Диалектика», 2023. — 1344 с.

2. Руководство по Java Development Kit (JDK 17). – URL: https://docs.oracle.com/en/java/javase/17/ (дата обращения: 7.05.2025).

3. Документация Android Developers. – URL: https://developer.android.com/develop (дата обращения: 6.05.2025).

4. Android Layouts. Документация по верстке (XML-разметка). – URL: https://developer.android.com/guide/topics/ui/declaring-layout (дата обращения 7.05.2025).

УДК 004.424.8

РАЗРАБОТКА ПРИЛОЖЕНИЯ «СЕМЕЙНАЯ КНИГА РЕЦЕПТОВ»

П. М. Рудская, И. Е. Воронина

Воронежский государственный университет

Аннотация. Рассматривается разработка веб-приложения «Семейная книга рецептов», предназначенного для сохранения, систематизации и совместного использования кулинарных традиций внутри семьи. Проведён анализ существующих решений на рынке, определены функциональные и нефункциональные требования, а также выбраны оптимальные технологии для реализации проекта. Представлены основные этапы реализации и пользовательский интерфейс. Проведено тестирование функциональности приложения, подтверждающее корректность работы всех компонентов.
Ключевые слова: веб-приложение, семейная книга рецептов, кулинарные традиции, сохранение культурного наследия, PostgreSQL, Spring Boot, реляционная база данных, архитектура MVC, пользовательский интерфейс, совместное редактирование рецептов, мультимедийные форматы.

Введение

В современном мире, где всё меняется с невероятной скоростью, семейные традиции становятся всё более важными для сохранения стабильности и гармонии в жизни. Одной из таких традиций являются кулинарные рецепты, которые передаются из поколения в поколение. Реальность такова, что семьи часто оказываются географически разобщены, а бумажные носители постепенно утрачивают актуальность. Поэтому возникает потребность в цифровых инструментах, способных объединить родственников вокруг общего кулинарного наследия. К сожалению, существующие платформы не предоставляют возможность коллективной работы и обмена контента внутри группы. Создание специализированного веб-приложения позволяет не только оцифровать рецепты, но и превратить процесс их сохранения в форму взаимодействия между друзьями и родственниками.

Таким образом, разработка веб-платформы для создания виртуальных семейных книг рецептов представляет собой актуальной и даже социально значимой задачей, позволяющей не только сохранять уникальные гастрономические традиции, но и способствовать укреплению семейных и дружеских связей.

1. Анализ существующих решений

При разработке приложения были проанализированы три популярные кулинарные платформы — eda.ru, allrecipes.com, paprika.com.

Eda.ru ориентирована на русскоязычную аудиторию и предоставляет рецепты с фото и текстовыми инструкциями, однако практически не поддерживает социальные функции и персонализацию.

Allrecipes — международный сайт с активным сообществом, где пользователи могут публиковать рецепты, оставлять отзывы и смотреть видеоинструкции, но возможности совместного редактирования рецептов ограничены.

Приложение Paprika представляет собой офлайн-менеджер рецептов с широкими функциями импорта, экспорта и высокой степенью персонализации, но без поддержки онлайн-взаимодействия между пользователями.

На основе анализа были определены важные критерии, которые необходимо учесть при разработке собственного веб-ресурса (табл. 1).

Таблица 1

Сравнение главных критериев

Критерий	Eda.ru	Allrecipes.com	Paprika.com
социальные функции	практически отсутствуют	есть базовая социальная активность	отсутствуют
мультимедийные форматы	в основном текст и изображения, ограниченное использование видео	активно используются видео, фото, пошаговые инструкции	минимальные визуальные элементы, упор на текст и структуру
импорт и экспорт рецептов	не поддерживается	ограниченная поддержка, возможен экспорт в PDF	широкие возможности импорта, экспорта и синхронизация между устройствами
гибридная система модерации	есть автоматическая проверка, ручная модерация слабая	в основном ручная модерация, автоматизация ограничена	модерация неактуальна, так как это офлайн-приложение
персонализация и UX	интерфейс перегружен, персонализация слабая	интерфейс удобный, есть рекомендации на основе предпочтений	высокий уровень персонализации, удобный, минималистичный интерфейс

Критерии, выделенные при анализе платформ, являются ключевыми для создания востребованного и удобного веб-приложения. Социальные функции позволяют членам семьи взаимодействовать между собой — делиться мнениями, оставлять комментарии, совместно дополнять рецепты, что особенно важно для семейного контекста. Мультимедийные форматы делают рецепты более наглядными и помогают точнее передать кулинарные традиции. Импорт и экспорт рецептов обеспечивают удобство переноса данных, возможность создания резервных копий и обмена рецептами за пределами платформы. Модерация важна для сохранения качества контента и предотвращения спама, особенно в условиях открытого доступа. Персонализация и удобный пользовательский интерфейс (UX) напрямую влияют на комфорт использования приложения.

Все эти критерии формируют целостный пользовательский опыт, которого не хватает в существующих решениях, и поэтому они легли в основу проектирования приложения.

1. Реализация приложения

Реализация веб-приложения «Семейная книга рецептов» основана на архитектуре MVC, обеспечивающей разделение бизнес-логики, представления и управления данными. Основными технологиями стали Java, Spring Boot и PostgreSQL. Такой выбор обусловлен стабильностью, кроссплатформенностью и широкими возможностями интеграции данных и интерфейсов.

2.1. Архитектура и структура приложения

Приложение реализовано как клиент–серверная система. Серверная часть отвечает за обработку запросов, взаимодействие с базой данных и управление пользователями. Клиентская часть представляет собой веб-интерфейс, обеспечивающий пользователю доступ ко всем функциям системы [1]. На рис. 1 представлена структура приложения.

Рис. 1. Структура приложения

2.2. Структура данных и база данных

Основой системы хранения является реляционная база данных PostgreSQL, обеспечивающая надёжное хранение и эффективный доступ к информации. В рамках реализации создана логическая модель, включающая ключевые сущности [2]. На рис. 2 приведена логическая модель базы данных.

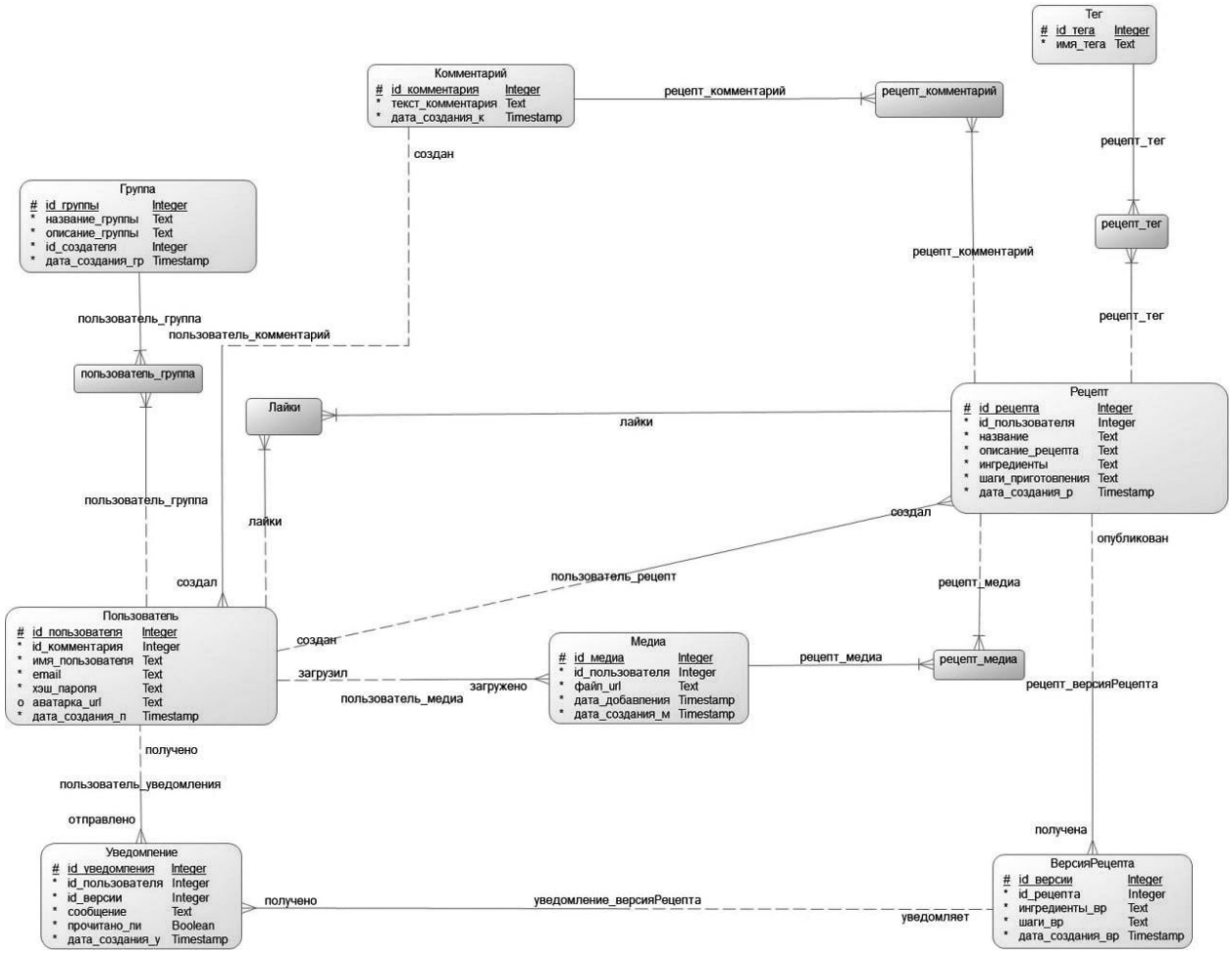

Рис. 2. Логическая модель базы данных

2.3. Безопасность и валидация данных

Для защиты пользовательских данных реализована система аутентификации и авторизации на основе Spring Security [3]. Все пароли хранятся в виде хэш-сумм. Реализованы механизмы ограничения доступа к определённым функциям.

Дополнительно внедрена система уведомлений для информирования пользователей.

2.4. Пользовательский интерфейс

Интерфейс приложения ориентирован на простоту и интуитивность. Главная страница содержит основные разделы. Пользователь может создавать и редактировать рецепты, прикреплять фото и видео, оставлять комментарии и участвовать в совместных группах.

Особое внимание уделено визуальной структуре и мультимедийным форматам. Навигация и адаптивный дизайн реализованы средствами HTML5 и CSS3.

2.5. Тестирование

Тестирование проводилось для проверки корректности выполнения основных операций. Например, при нажатии кнопки «Начать» на главной странице система корректно перенаправляет пользователя к форме регистрации, а при клике на элемент FAQ динамически раскрывается ответ без перезагрузки страницы. Все тесты показали корректную работу компонентов и стабильность приложения.

Заключение

Разработанное приложение является простым и удобным инструментом, позволяющем хранение и совместное использование семейных кулинарных рецептов. В ходе работы проведён анализ аналогичных платформ, их функциональных возможностей, преимуществ и ограничений. Удобный и интуитивно понятный пользовательский интерфейс позволяет добавлять, редактировать и структурировать рецепты, создавать семейные группы для совместного ведения кулинарной книги. Для защиты данных внедрена система аутентификации и авторизации пользователей.

В дальнейшем проект может быть дополнен новыми возможностями, которые позволят расширить функциональность приложения и сделать его более востребованным среди пользователей.

Литература

1. *Шилдт Г.* Java: полное руководство / Г. Шилдт. – 10-е изд. – Москва : Вильямс, 2018. – 1488 с.

2. *Грофф Дж. Р.* SQL: полное руководство / Дж. Р. Грофф, П. Н. Вайнберг, Э. Дж. Оппель. – 3-е изд. – Москва : Диалектика, 2020. – 960 с.

3. *Блох Д.* Эффективное программирование на Java / Д. Блох. – Москва : Диалектика-Вильямс, 2019. – 464 с.

УДК 681.3.06

СЖАТИЕ ВИДЕОДАННЫХ ДЛЯ ОПТИМИЗАЦИИ ВИДЕОСТРИМИНГА В ВЕБ-ПРИЛОЖЕНИИ «КИНОСЕРВИС» НА ПЛАТФОРМЕ ASP.NET CORE

А. Е. Сафонова

Воронежский государственный университет

Аннотация. Данная статья посвящена исследованию и практическому применению методов оптимизации хранения и передачи видеоконтента на примере веб-приложения «Киносервис», разработанного на платформе .NET 9.0. Анализируется техническая реализация конвейера обработки видео, необходимого для перехода от неоптимизированного хранения к масштабируемому видеостримингу. В работе описывается архитектура, включающая разделение хранения метаданных в СУБД MS SQL Server и видеоконтента в объектном хранилище Yandex Cloud. Особое внимание уделяется практической интеграции утилиты FFmpeg в бэкенд-логику приложения, написанную на C# 12.0, для реализации процесса транскодинга и подготовки контента к адаптивной потоковой передаче (ABR).
Ключевые слова: сжатие видеоданных, видеостриминг, видеокодеки, H.265, AV1, объектное хранилище, адаптивный стриминг, транскодинг, ABR, FFmpeg, оптимизация передачи данных, Razor Pages.

Введение

Стремительный рост популярности мультимедийного контента и, в частности, онлайн-кинотеатров предъявляет повышенные требования к инфраструктуре хранения и передачи данных. При разработке информационной системы «Киносервис» на платформе ASP.NET Core 9.0 ключевой технической проблемой стала избыточность объема исходных видеофайлов.

Подход, предполагающий непосредственное хранение видеофайлов в реляционной базе данных или их прямую загрузку с локального сервера, является неоптимизированным и передача таких массивов данных приводит к значительным финансовым издержкам, повышенной нагрузке на серверные ресурсы и ухудшению пользовательского опыта из-за низкой скорости загрузки и буферизации.

Решение данной проблемы требует комплексного подхода, базирующегося на эффективном сжатии данных и применении протоколов адаптивной потоковой передачи (стриминга). Данная статья посвящена исследованию ключевых аспектов оптимизации видеоконтента, определению роли современных видеокодеков и обоснованию архитектурного решения для реализации эффективного видеостриминга.

1. Теоретические основы и методы оптимизации видеоконтента

Основополагающим элементом в управлении объемом видеоданных является видеокодек (Coder/Decoder) — алгоритм, предназначенный для устранения избыточности в видеопотоке. Видеофайлы в исходном виде содержат колоссальный объем информации, что делает их передачу непрактичной. Кодек выполняет кодирование (сжатие), превращая исходный поток в компактный формат, и обратное декодирование (распаковку), необходимое для последующего воспроизведения на устройстве конечного пользователя. Целью кодирования является достижение максимальной степени сжатия при сохранении заданного уровня визуального качества.

Для достижения высокой эффективности современные кодеки используют две основные стратегии, направленные на устранение избыточности. Первая — пространственное сжатие (Внутрикадровое, Intra-frame) — сродни принципам сжатия неподвижных изображений

(JPEG). Алгоритм анализирует цветовое и яркостное распределение внутри кадра, применяет математические преобразования и выполняет квантование, отбрасывая наименее важную частотную информацию. Результатом является I-кадр (Intra-coded picture), который может быть раскодирован независимо от других кадров.

На уровне обработки пиксельной матрицы, из которой состоит каждый кадр, кодек идентифицирует группы пикселей с похожими характеристиками (например, цветом или яркостью). Вместо того чтобы записывать полный набор данных для каждого пикселя, применяется принцип кодирования длины серии (run-length encoding) или его аналоги, где сохраняется информация о ключевом пикселе и счетчике его повторений до момента изменения. Порог, при котором незначительные различия между пикселями считаются идентичными, прямо определяет степень сжатия и, соответственно, результирующее качество изображения. Современные и более сложные кодеки развивают этот принцип, учитывая сегментацию изображения, разноуровневое сжатие отдельных областей, а также психофизические особенности восприятия видео человеком, что позволяет значительно сократить общий объем данных без заметной для зрителя потери качества изображения.

Вторая стратегия — временное сжатие (Межкадровое, Inter-frame) — использует тот факт, что в последовательности кадров изменения минимальны. Кодек не хранит каждый кадр полностью, а записывает только разницу и векторы движения относительно опорных кадров. P-кадры (Predictive picture) кодируются на основе предыдущего опорного кадра, фиксируя только изменения, а B-кадры (Bi-predictive picture) используют предсказания как от предыдущих, так и от последующих кадров, обеспечивая максимальную степень сжатия. Таким образом, вместо тысяч полных кадров в секунду, кодек сохраняет и передает один опорный I-кадр и множество небольших наборов «изменений», что критически важно для экономии дискового пространства и пропускной способности сети.

Выбор кодека определяет не только объем хранения, но и совместимость со сторонними устройствами, что является ключевым фактором в архитектуре современного видеосервиса, использующего подход мультикодеков для охвата различных платформ. На сегодняшний день доминируют три семейства кодеков, различающихся степенью эффективности сжатия:

H.264 (AVC — Advanced Video Coding). Этот кодек долгое время являлся стандартом и остается незаменимым для обеспечения широчайшей совместимости с устаревшими устройствами и всеми веб-браузерами. Он обеспечивает приемлемое качество для HD-разрешения, но его эффективность сжатия существенно ниже по сравнению с новыми разработками.

H.265 (HEVC - High Efficiency Video Coding). Как прямое развитие H.264, данный кодек обеспечивает значительный скачок в эффективности, предлагая до 50 % лучшее сжатие при сопоставимом качестве изображения. Он является основным инструментом для хранения и передачи контента в разрешении 4K (Ultra HD) и используется для существенного снижения требований к битрейту и, соответственно, объему облачного хранилища.

AV1 (AOMedia Video 1). Представляет собой последнее поколение, разработанное консорциумом крупнейших технологических компаний. Главное преимущество AV1 — это его открытость, отсутствие лицензионных отчислений и наивысшая эффективность сжатия, которая может превышать H.265 на 20–40 %. AV1 является стратегическим выбором для минимизации операционных расходов на трафик в будущем и активно внедряется для потоковой передачи в 4K, особенно на мобильных платформах.

2. Архитектурное решение для оптимизации хранения и передачи данных

Проблема больших файлов в веб-приложениях решается не только эффективным сжатием, но и принципиальным изменением парадигмы архитектуры хранения и доставки контента.

Для создания масштабируемого веб-сервиса применяется модель разделения хранения данных, что является критически важным индустриальным стандартом для работы с медиаконтентом.

2.1. Разделение хранения и использование объектного хранилища

В архитектуре современных веб-сервисов, оперирующих мультимедийным контентом, критически важно использовать модель разделения хранения данных. Такой подход позволяет преодолеть фундаментальные ограничения реляционных СУБД, которые не предназначены для эффективного хранения больших бинарных объектов (BLOB), каковыми являются видеофайлы, а также существенно снизить нагрузку на основной сервер приложений.

В рамках проекта «Киносервис» реализовано строгое разделение данных по их типу и назначению.

Реляционная СУБД (Microsoft SQL Server 2022): используется исключительно для управления структурированными метаданными, необходимыми для функционирования каталога. В таблицах, управляемых через ORM Entity Framework Core 9.0, хранятся все ключевые атрибуты: название фильма, краткое и полное описание, год выпуска, имена актеров, режиссеров, жанры и другая служебная информация.

Облачное Объектное Хранилище (Yandex Cloud Object Storage): Данный тип хранилища выбран для размещения самого видеоконтента. В отличие от реляционных СУБД, объектное хранилище специализируется на работе с большими неструктурированными файлами. Контент хранится в логических контейнерах, называемых бакетами. Выбор Yandex Cloud и его совместимость с протоколом S3 (Simple Storage Service) обеспечивают ключевые преимущества: масштабируемость, высокую доступность и простоту интеграции с C#-кодом через стандартные SDK. Именно в этом бакете размещаются сегментированные файлы (.ts) и HLS-манифесты (.m3u8), полученные после транскодинга. Таким образом, раздача тяжелого статического контента полностью делегируется специализированной облачной инфраструктуре.

Логическая связь между метаданными и контентом обеспечивается через хранение в СУБД ссылки на мастер-манифест, генерируемый в процессе транскодинга. Например, в модели данных, описывающей фильм или другой видеофайл, может присутствовать строковое поле, содержащее публичный URL-адрес этого манифеста в облачном хранилище.

Таким образом, реляционная база данных не хранит бинарные видеофайлы, а лишь ссылки на физический контент. Данный архитектурный подход обеспечивает высокую производительность операций с метаданными (поиск, фильтрация) и в то же время делегирует задачу раздачи видеоданных масштабируемому и высокодоступному облачному хранилищу. Важно отметить, что архитектура спроектирована как провайдеро-независимая: использование стандартного S3-совместимого API позволяет при необходимости легко мигрировать между различными облачными решениями (такими как Amazon S3, Azure Blob Storage или VK Cloud Storage) путем изменения настроек подключения в ASP.NET Core.

2.2. Транскодинг

Процесс создания оптимизированных версий видео называется ***транскодингом*** и является критическим этапом, который начинается с момента, когда администратор веб-приложения загружает исходный видеофайл высокого качества на выделенный сервер или в облачный сервис обработки. Именно на этом этапе происходит основное сжатие.

Для выполнения этой многопроходной и ресурсоемкой работы используется FFmpeg — консольная утилита и одновременно мощный программный комплекс, являющийся индустриальным стандартом для транскодинга. Этот инструмент представляет собой проект с открытым исходным кодом, который включает в себя ключевые библиотеки, такие как libavcodec

для кодирования и декодирования, и libavformat для работы с контейнерными форматами (мультиплексирование и демультиплексирование). Благодаря своей модульной структуре и поддержке огромного числа кодеков и фильтров, данный комплекс является незаменимым инструментом. Он принимает исходное видео для кодирования в нужный кодек (например, H.265 или AV1).

Серверная система управления независимо от используемого фреймворка использует программные методы (например, вызов внешнего процесса) для запуска FFmpeg как внешней утилиты командной строки. Это позволяет интегрировать сложный процесс транскодинга в автоматизированный рабочий процесс. Приложение передает утилите точные команды, указывающие, какие именно кодеки, битрейты и форматы контейнеров должны быть использованы.

Процесс сжатия (кодирования) происходит в несколько критически важных этапов. Во-первых, FFmpeg прогоняет исходный файл через кодек несколько раз для создания набора файлов, известных как мультиверсии (renditions), каждая из которых имеет различное разрешение и битрейт (например, 4K, 1080p, 720p, 480p). Во-вторых, каждая из этих версий нарезается на короткие, легко загружаемые сегменты (обычно 2–10 секунд). Наконец, генерируется манифест-файл (в формате .m3u8 для HLS или .mpd для MPEG-DASH), который представляет собой список всех созданных сегментов, их качества и порядка воспроизведения. Эти сжатые, сегментированные и индексированные файлы затем загружаются в выбранное объектное хранилище. Таким образом, FFmpeg выступает как критически важный инструмент, который переводит объемный, непригодный для сети файл в набор легких, оптимизированных для потоковой передачи объектов, готовых к быстрой отдаче.

Для практической апробации в веб-приложении «Киносервис» процесс транскодинга был реализован на стороне бэкенда с использованием C# 12.0 и утилиты FFmpeg. Процесс состоит из следующих шагов:

1. Загрузка файла. Администратор через защищенную административную панель (реализованную на ASP.NET Core Razor Pages) загружает исходный мастер-файл фильма.

2. Запуск процесса. Контроллер Razor Page или специальный сервис в C#-коде перехватывает файл. Вместо простого сохранения, он инициирует асинхронный процесс транскодинга. Для этого используется класс System.Diagnostics.Process из стандартной библиотеки .NET, который позволяет запустить внешнюю утилиту FFmpeg.

3. Использование программных средств. C#-код программно формирует командную строку для FFmpeg, указывая исходный файл, целевые кодеки (например, H.264), набор битрейтов для ABR (например, 1080p, 720p, 480p) и команду нарезки в формат HLS (сегменты .ts и манифест .m3u8).

4. Обновление СУБД и загрузка в облако. После успешного завершения процесса FFmpeg, C#-код выполняет загрузку всех сгенерированных сегментов (.ts) и манифестов (.m3u8) в Yandex Cloud Object Storage с использованием соответствующего SDK (например, Amazon S3 SDK, так как Yandex использует S3-совместимый API). Затем публичный URL-адрес мастер-манифеста сохраняется в поле HlsManifestUrl в MS SQL Server с помощью Entity Framework Core.

2.3. Доставка видео и декодирование

Процесс доставки видео в «Киносервисе» полностью опирается на принцип адаптивного стриминга (ABR) и реализуется через взаимодействие трех компонентов: ASP.NET Core, Yandex Cloud и клиентского JavaScript-плеера.

Доставка видео начинается, когда пользователь инициирует просмотр фильма в браузере. В этот момент запускается механизм взаимодействия между серверной системой, облачным хранилищем и клиентским устройством, который полностью обходит проблему передачи огромных файлов.

Серверное приложение, выступающее в роли контроллера метаданных, получает запрос от пользователя. Его основная задача — не отдать сам видеофайл, а получить из реляционной СУБД MS SQL Server и передать обратно клиентскому приложению URL-адрес мастер-манифеста (например, https://<bucket_name>.storage.yandexcloud.net/movie_id/master.m3u8). Передача только ссылки на манифест, а не данных самого видео, позволяет серверной части, реализованной на ASP.NET Core Razor Pages, оставаться высокопроизводительной и разгружает ее от необходимости управления потоками большого объема.

Получив URL, клиентский видеопроигрыватель, который называется «умным» плеером (в приложении используется специализированная JavaScript-библиотека) начинает свою работу. Этот плеер поддерживает логику Adaptive Bitrate Streaming (ABR), стандартизированную протоколом HLS (HTTP Live Streaming). Плеер, считывая манифест-файл (.m3u8) от Yandex Cloud, получает «карту» всего контента: он содержит полный список всех доступных версий (мультиверсий), их кодеки, битрейты, разрешения и точные URL-адреса каждого короткого видеосегмента (.ts).

ABR — это технология воспроизведения, которая автоматически регулирует качество в реальном времени на основе состояния сети пользователя и возможностей его устройства. Плеер непрерывно мониторит состояние сети и управляет буфером, оценивая фактическую скорость загрузки предыдущих сегментов. В соответствии с этим, он динамически регулирует качество: если буфер быстро истощается или скорость сети падает, плеер немедленно переключается на сегменты с более низким битрейтом и разрешением (например, с 1080p на 480p), чтобы обеспечить непрерывное воспроизведение. При улучшении пропускной способности плеер постепенно «поднимает» качество контента, всегда стремясь обеспечить наилучшее качество просмотра.

Сегменты доставляются по HTTP-протоколу непосредственно из объектного хранилища Yandex Cloud или через сеть доставки контента (CDN). И наконец, полученные сжатые данные поступают в Media Source Extensions (MSE) API браузера. Браузер, в свою очередь, использует нативное аппаратное или программное обеспечение устройства для выполнения декодирования сжатых данных (например, H.264) и их отображения. Таким образом, вся тяжесть доставки и финальной ресурсоемкой обработки ложится на оптимизированное облачное хранение и клиентское программное обеспечение, а серверная система выступает лишь в качестве посредника.

Заключение

Данное исследование и его практическая апробация в рамках веб-приложения «Киносервис» демонстрируют, что эффективное функционирование современного видеосервиса достигается не за счет наращивания аппаратных мощностей, а благодаря внедрению комплексной архитектурной и технологической оптимизации. Ключевым элементом такой оптимизации выступает симбиоз современных видеокодеков, обеспечивающих глубокое сжатие данных без существенной потери качества, и пайплайна транскодинга для подготовки контента к адаптивной потоковой передаче, гарантирующей стабильную доставку контента в условиях нестабильной сети. Теоретические принципы сжатия (кодеки) и доставки (ABR) были успешно реализованы с использованием конкретного технологического стека: ASP.NET Core 9.0 (C# 12.0) для бэкенд-логики, MS SQL Server 2022 и Entity Framework Core 9.0 для управления метаданными, утилиты FFmpeg для «ручного» транскодинга, и Yandex Cloud Object Storage для масштабируемого хранения и раздачи контента.

Предложенная и реализованная архитектура, основанная на разделении метаданных в реляционной СУБД и видеоконтента в объектном хранилище, создает масштабируемый и экономически эффективный фундамент для любого современного веб-приложения. Такой подход

не только решает первоначальную проблему хранения и передачи больших объемов данных, но и закладывает основу для дальнейшего развития сервиса, позволяя легко интегрировать перспективные кодеки и технологии доставки. Таким образом, именно грамотное проектирование пайплайна работы с видео, включающее эффективное кодирование и адаптивную доставку, становится определяющим фактором успеха в создании конкурентного и надежного стримингового решения.

Литература

1. FFmpeg Documentation: [сайт]. – URL: https://ffmpeg.org/documentation.html (дата обращения 8.11.2025).

2. Microsoft. ASP.NET Core documentation: [сайт]. – URL: https://docs.microsoft.com/en-us/aspnet/core/ (дата обращения 8.11.2025).

3. Yandex Cloud Documentation // Yandex Object Storage: [сайт]. – URL: https://cloud.yandex.ru/docs/storage/ (дата обращения 10.11.2025).

4. Adaptive Bitrate Streaming: [сайт]. – URL: https://uploadcare.com/blog/adaptive-bitrate-streaming-what-is-it/ (дата обращения 10.11.2025).

5. Разбираем процесс видеокодирования на примере кодека H.264/AVC: [сайт]. – URL: https://www.iguides.ru/blogs/Elecard/razbiraem-protsess-videokodirovaniya-na-primere-kodeka-h264avc/ (дата обращения 10.11.2025).

УДК 004.657

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ СРАВНЕНИЯ СТРУКТУР БАЗ ДАННЫХ

А. М. Сафонова

Воронежский государственный университет

Аннотация. В статье представлено приложение для сравнения структур двух баз данных PostgreSQL, которое выявляет различия в таблицах и атрибутах. Рассмотрены методы извлечения метаданных, алгоритмы сравнения и визуализации результатов для анализа и контроля версий схем.
Ключевые слова: сравнение структур баз данных, метаданные PostgreSQL, системные таблицы, алгоритм сравнения, визуализация результатов.

Введение

В современных информационных системах сравнение структур баз данных является ключевой задачей при миграции, синхронизации и контроле версий. Для анализа схем используется множество методов, основанных на извлечении и сопоставлении метаданных. В данной работе рассматривается разработка приложения, осуществляющего автоматизированное сравнение структур двух баз данных на основе системных таблиц PostgreSQL [1]. Вводится понятие статусов элементов схемы (таблиц и столбцов) и определяется алгоритм, позволяющий выявлять и классифицировать различия в именах и типах данных. Для повышения наглядности результаты сравнения визуализируются с использованием цветовой маркировки. В работе обоснован выбор подхода сравнения метаданных, а также представлены технические детали реализации и структура приложения.

1. Обзор существующих решений и методов сравнения

На современном рынке представлен широкий спектр коммерческих и открытых инструментов для сравнения структур баз данных. Большинство коммерческих решений отличаются высокой стоимостью, что ограничивает их доступность для небольших команд и индивидуальных разработчиков. Инструменты, ориентированные на миграции, менее удобны для разовых сравнений и требуют дополнительной настройки, а некоторые поддерживают только конкретные СУБД, снижая универсальность. Бесплатные версии часто лишены расширенных функций по сравнению с платными. Для разовых сравнений и поддержки множества СУБД подходят DBeaver [2] и DataGrip [3], для профессиональной работы с SQL Server — Redgate SQL Compare [4], а для длительного контроля версий схем — Liquibase [5]. Выбор инструмента зависит от конкретных требований проекта и баланса между функциональностью, стоимостью и удобством.

Основные методы сравнения баз данных:

1. Сравнение метаданных (Schema Comparison).

Принцип работы заключается в анализе системных таблиц. В качестве достоинств можно выделить высокую скорость выполнения, точное сравнение структур, возможность генерации DDL-скриптов. Однако данный метод может не учитывать семантические различия и зависит от конкретных особенностей СУБД.

2. Сравнение SQL-дампов.

Метод экспортирует схемы в SQL-скрипты с последующим текстовым сравнением. Алгоритм прост в реализации, совместим с системами контроля версий. В качестве недостатков

выделяют низкую точность анализа, необходимость ручной обработки результатов, сложность генерации скриптов синхронизации.

3. Подход на основе миграций.

Данный метод предусматривает версионирование изменений схемы через последовательные файлы изменений. Поддерживает полное версионирование схемы, откат изменений, но не подходит для разового сравнения.

Выбор конкретного метода зависит от решаемой задачи: для разовых сравнений оптимален анализ данных и сравнение дампов. Для задач длительной разработки предпочтительны миграции.

Для реализации выбран метод сравнения метаданных, так как он обеспечивает высокую точность и скорость анализа, а также позволяет автоматически и наглядно выявлять различия в структуре баз данных. В качестве целевой СУБД выбрана PostgreSQL, обладающая широким набором типов данных и высокой производительностью. Получение информации о структуре базы реализуется через системные таблицы `pg_tables`, `pg_attribute`, `pg_class` и `pg_type`, которые предоставляют прямой и полный доступ к метаданным, в отличие от представлений `information_schema`.

2. Описание алгоритма сравнения структур баз данных

Программа предназначена для сравнения схем баз данных и состоит из нескольких ключевых компонентов: модуль подключения к БД, модуль анализа схемы, модуль сравнения и модуль формирования отчёта.

Алгоритм сравнения основан на последовательном анализе составов таблиц и их столбцов с использованием операций над множествами для выявления уникальных и общих элементов. Для каждой базы данных извлекается множество имён таблиц из системной таблицы `pg_tables`, поле tablename содержит имена таблиц, а schemaname позволяет отфильтровать системные схемы, что позволяет быстро определить, какие таблицы присутствуют в одной или обеих базах. Далее для каждой таблицы извлекается информация о столбцах и их типах данных с помощью системных таблиц `pg_attribute`, `pg_class` и `pg_type`. Таблица pg_attribute содержит данные о столбцах, pg_class - о таблицах, а pg_type — о типах данных, что позволяет получить кортежи (имя столбца, тип данных) для конкретной таблицы.

Сравнение столбцов для каждой таблицы осуществляется с помощью множеств и включает определение статусов:

- only_in_db1 — столбец присутствует только в первой базе данных;
- only_in_db2 — столбец присутствует только во второй базе данных;
- diff_types — столбец есть в обеих базах, но типы данных различаются;
- match — столбец полностью совпадает по имени и типу.

Особое внимание уделяется обработке регистра имён таблиц и столбцов для исключения ложных различий, а также возможным эквивалентностям типов данных.

3. Практическая реализация

В качестве языка программирования выбран язык Python. Возможности данного языка полностью удовлетворяют поставленной задаче, так как в нем содержится весь необходимый набор инструментов, требуемый для создания программы. При реализации использовалась среда разработки PyCharm Community Edition.

Программа представляет собой инструмент для сравнения схем баз данных. Она состоит из следующих ключевых компонентов:

1. Модуль подключения в БД обеспечивает установление соединения с базами данных на основе переданных параметров (хост, порт, имя пользователя, пароль, название БД).

2. Модуль анализа схемы извлекает метаданные о таблицах и столбцах из системных таблиц PostgreSQL.

3. Модуль сравнения сравнивает структуры двух баз данных, выявляя различия системных таблиц.

4. Модуль формирования отчета генерирует структурированные данные для последующей визуализации результатов сравнения.

На рис. 1 показана структура программы.

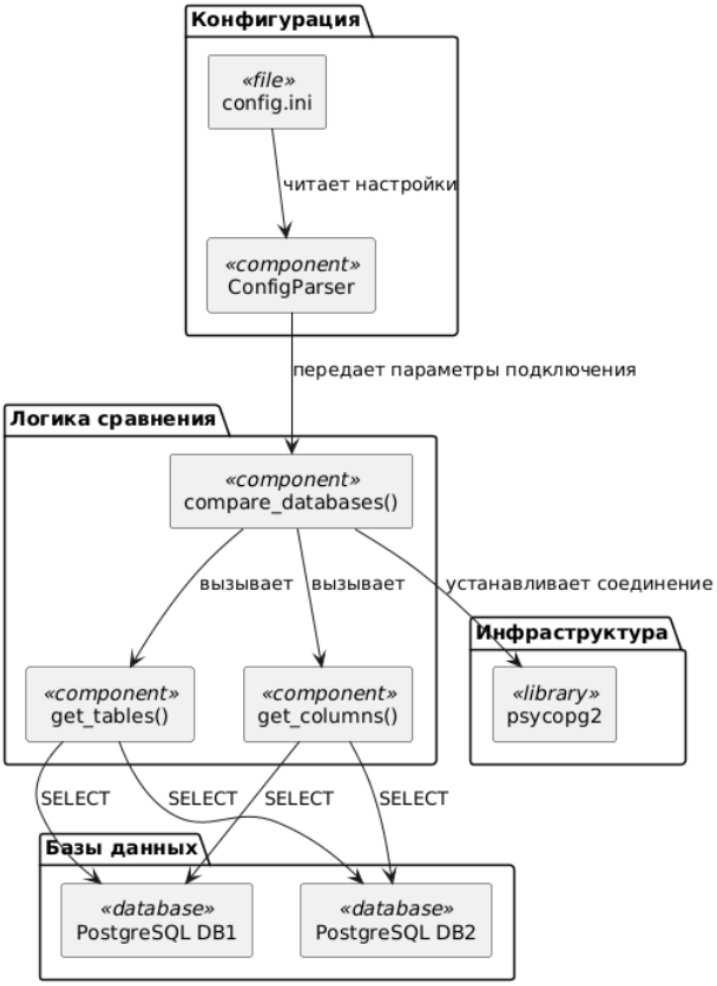

Рис. 1. Структура приложения

В реализации используется соединение с базами данных PostgreSQL через библиотеку `psycopg2`. Основные функции:

• get_tables(cursor) — получает список таблиц в базе, исключая системные схемы:

```
def get_tables(cursor):
    cursor.execute("""
        SELECT tablename
        FROM pg_tables
        WHERE schemaname NOT IN
              ('information_schema', 'pg_catalog');
        """)
    return [row[0] for row in cursor.fetchall()]
```

```
tables_db1 = set(get_tables(cur1))
tables_db2 = set(get_tables(cur2))
all_tables = tables_db1.union(tables_db2)
```

• get_columns(cursor, table_name) — извлекает столбцы и типы данных указанной таблицы:

```
def get_columns(cursor, table_name):
    cursor.execute(f"""
        SELECT a.attname, t.typname
        FROM pg_attribute a
        JOIN pg_class c ON a.attrelid = c.oid
        JOIN pg_type t ON a.atttypid = t.oid
        WHERE c.relname = %s AND a.attnum > 0 AND
                                NOT a.attisdropped;
    """, (table_name,))
    return cursor.fetchall()

columns_db1 = dict(get_columns(cur1, table)) if table in tables_db1 else {}
columns_db2 = dict(get_columns(cur2, table)) if table in tables_db2 else {}
all_columns = set(columns_db1.keys()).union(set(columns_db2.keys()))
```

• compare_databases(db1_config, db2_config) — осуществляет сравнение двух баз, формирует отчёт с информацией о таблицах и столбцах, включая сводную статистику различий:

```
def compare_databases(db1_config, db2_config):
    """Сравнивает схемы двух баз данных и возвращает данные для визуализации"""
    result = {
        'tables': {},
        'columns': {},
        'summary': {
            'tables_only_in_db1': 0,
            'tables_only_in_db2': 0,
            'columns_diff': 0,
            'tables_with_diff': 0
        }
    }

    try:
        with psycopg2.connect(**db1_config) as conn1, \
                psycopg2.connect(**db2_config) as conn2:

            with conn1.cursor() as cur1, conn2.cursor() as cur2:

                tables_db1 = set(get_tables(cur1))
                tables_db2 = set(get_tables(cur2))

                all_tables = tables_db1.union(tables_db2)
                result['summary']['total_tables'] = len(all_tables)

                for table in all_tables:
                    table_status = {
                        'exists_in_db1': table in tables_db1,
                        'exists_in_db2': table in tables_db2,
                        'columns': {}
```

```
                    }
                        columns_db1 = dict(get_columns(cur1, table)) if table in
tables_db1 else {}
                        columns_db2 = dict(get_columns(cur2, table)) if table in
tables_db2 else {}
                     all_columns = set(columns_db1.keys()).union(set(columns_db2.
keys()))
                    for column in all_columns:
                        type_db1 = columns_db1.get(column)
                        type_db2 = columns_db2.get(column)

                        if type_db1 is None:
                            status = 'only_in_db2'
                            color = '🔴'  # Красный
                            result['summary']['columns_diff'] += 1
                        elif type_db2 is None:
                            status = 'only_in_db1'
                            color = '🟢'  # Зеленый
                            result['summary']['columns_diff'] += 1
                        elif type_db1 != type_db2:
                            status = 'diff_types'
                            color = '🟡'  # Желтый
                            result['summary']['columns_diff'] += 1
                        else:
                            status = 'match'
                            color = None

                        table_status['columns'][column] = {
                            'db1_type': type_db1,
                            'db2_type': type_db2,
                            'status': status,
                            'color': color
                        }

                    if not table_status['exists_in_db1']:
                        result['summary']['tables_only_in_db2'] += 1
                    elif not table_status['exists_in_db2']:
                        result['summary']['tables_only_in_db1'] += 1

                if any(col['status'] != 'match' for col in table_status['columns'].
values()):
                        result['summary']['tables_with_diff'] += 1

                    result['tables'][table] = table_status

        return result

    except Exception as e:
        print(f"Ошибка сравнения: {str(e)}")
        return None
```

Результаты сравнения сохраняются в структуре данных, где для каждой таблицы фиксируется наличие в каждой базе, а для каждого столбца — статус и типы данных. В отчёте подсчитываются количество таблиц, присутствующих только в одной базе, количество столбцов с различиями и общее число таблиц с отличиями.

Для визуализации используется цветовая маркировка: зелёный для элементов, присутствующих только в первой базе, красный — только во второй, жёлтый — для различий в типах данных. На рис. 2 показан результат сравнения баз данных.

	Таблица	Столбец	Тип в БД1	Тип в БД2	Статус
1	dino	kindofdino	varchar	-	only_in_db1
2		dateofdeath	date	-	only_in_db1
3		id_empl_creature	-	int4	only_in_db2
4		date_of_birth	-	date	only_in_db2
5		weight	int4	int2	diff_types
6		iddino	int4	-	only_in_db1
7		id_attraction_cr	-	int4	only_in_db2
8		id_kind_of_dino	-	int4	only_in_db2
9		dateofbirthdino	date	-	only_in_db1
10		id_type_creature	-	int4	only_in_db2
11		name_dino	-	varchar	only_in_db2
12		danger_level_creature	-	int2	only_in_db2
13		id_dino	-	int4	only_in_db2
14		namedino	varchar	-	only_in_db1
15		date_of_death_dino	-	date	only_in_db2

Рис. 2. Результат сравнения типов атрибутов

Заключение

В ходе исследования было разработано приложение для сравнения структур баз данных с графическим интерфейсом.

Ключевые преимущества решения:

1. Автоматизация анализа.

Приложение устраняет необходимость ручного сравнения структур БД, предлагая точный алгоритм выявления различий на основе системных метаданных PostgreSQL.

2. Графический интерфейс.

Реализован интерфейс с темной темой оформления, включающий: формы подключения к БД, интерактивную таблицу результатов с цветовой маркировкой, поддержку быстрой настройки параметров.

Достигнутые цели:

1. Разработаны алгоритм сравнения схем БД на основе анализа системных таблиц PostgreSQL, множественных операций для выявления различий.

2. Реализован графический интерфейс с использованием PySide6, включая формы ввода параметров подключения, таблицу результатов с подсветкой расхождений.

3. Проведено тестирование функциональности, подтвердившее корректность работы: подстановки параметров по умолчанию, сравнения идентичных и различных схем.

В перспективе решение может стать частью дальнейшего расширения функциональности. Целесообразно реализовать генерацию SQL-скриптов для автоматической синхронизации схем баз данных, что позволит пользователям не только выявлять, но и оперативно устранять обнаруженные различия. Дополнительно стоит рассмотреть возможность анализа других объектов БД, таких как индексы, ограничения целостности и триггеры, что повысит комплексность проводимого сравнения.

Литература

1. PostgreSQL Documentation : System Catalogs / PostgreSQL. – URL: https://www.postgresql.org/docs/current/catalogs.html (дата обращения: 01.07.2025).

2. DBeaver : официальный сайт. – URL: https://dbeaver.com/docs/dbeaver/Schema-compare/ (дата обращения: 01.07.2025).

3. DataGrip : документация по сравнению схем и миграции / JetBrains. – URL: https://www.jetbrains.com/help/datagrip/schema-comparison-and-migration.html# (дата обращения: 01.07.2025).

4. Red Gate SQL Compare : официальный сайт продукта / Red Gate Software Ltd. – URL: https://www.red-gate.com/products/sql-compare/ (дата обращения: 01.07.2025).

5. Liquibase : управление версиями баз данных / Liquibase Inc. – URL: https://www.liquibase.com/version-control (дата обращения: 01.07.2025).

УДК 004.9

РАЗРАБОТКА ТРАНСЛЯТОРА С SQL В NOSQL

И. А. Симонов

Воронежский государственный университет

Аннотация. В статье представлено приложение, реализующее инструмент для анализа SQL запросов и последующего преобразования их в NoSQL на примере документо ориентированной СУБД MongoDB. Рассмотрены способы задания и описания синтаксиса языка, варианты построения транслятора и реализации его этапов, в частности.
Ключевые слова: трансляция с SQL в NoSQL, лексический анализ, синтаксический анализ, КС-грамматики, конечные автоматы, автоматы с магазинной памятью.

Введение

Современные системы управления базами данных представлены двумя принципиально разными подходами: реляционными и нереляционными (NoSQL) системами.

Реляционные базы данных (такие как MySQL, PostgreSQL, Oracle) используют строгую табличную модель данных с чётко определёнными связями между таблицами. Универсальным языком взаимодействия с такими системами является SQL (Structured Query Language), предоставляющий мощный и стандартизированный набор операций для работы с данными.

В свою очередь, NoSQL системы (например, MongoDB, Cassandra, Redis) предлагают принципиально иные модели хранения данных — документную, ключ значение, колоночную или графовую. Особое внимание в данной работе уделяется MongoDB — документо ориентированной СУБД, где данные хранятся в виде JSON подобных документов. Такая модель обеспечивает гибкость и масштабируемость, что делает MongoDB популярной для современных веб приложений.

Ключевой проблемой при переходе между этими типами систем является несовместимость языков запросов. SQL запросы не могут быть напрямую выполнены в NoSQL средах, что существенно осложняет процессы миграции данных и интеграции систем.

Таким образом, в работе заложены теоретические и практические основы для создания полноценного транслятора, способного связать между собой реляционные и нереляционные системы управления данными.

1. Постановка задачи

Целью данной работы является создание инструмента для анализа SQL запросов и перевода их в MongoDB формат, который можно разбить на пять взаимосвязанных частей:

1. Разработка лексического анализатора SQL, который:
   - выполняет разбор входного SQL запроса на токены;
   - проверяет корректность лексической структуры;
   - подготавливает данные для синтаксического анализа.
2. Создание синтаксического анализатора SQL, который:
   - проверяет соответствие запроса грамматике SQL;
   - строит абстрактное синтаксическое дерево (AST);
   - выявляет синтаксические ошибки.
3. Реализация промежуточного представления для MQL, которое:
   - определяет объектную модель, инкапсулирующую ключевые компоненты запроса в форме MongoDB;

- разрешает контекстные зависимости.

4. Создание генератора кода MQL, который:
   - сериализует объекты промежуточного представления в валидный JSON-код;
   - содержит шаблоны генерации для всех основных операторов.
5. Разработка веб интерфейса для:
   - ввода SQL запросов пользователем;
   - визуализации процесса лексического и синтаксического анализа;
   - отображения результатов разбора (токены, AST, ошибки);
   - вывод итогового MongoDB-запроса.

2. Анализ задачи

2.1. Общий анализ задачи

Для решения поставленной задачи необходимо реализовать программу, которая будет обеспечивать выполнение трансляции с SQL в MongoDB формат и предоставлять возможность их вывода её этапов пользователю.

Трансляция — это процесс преобразования запроса, написанного на исходном языке, в эквивалентную форму на целевом языке с сохранением семантической эквивалентности. Программы, которые реализуют логику этого процесса, называют трансляторами. В состав каждого из них входят три основных компонента:

- лексический анализатор;
- синтаксический анализатор;
- генератор кода.

Сканер выполняет первичную обработку исходного текста, преобразуя последовательность отдельных символов в структурированный поток лексем — минимальных значимых элементов языка. К ним относятся:

- ключевые слова языка;
- идентификаторы переменных;
- литералы (числовые, строковые константы);
- операторы и разделители.

На этом шаге лексемы выделяются как целостные элементы, теряя свою внутреннюю символьную структуру для последующих этапов обработки.

Парсер выполняет проверку грамматической корректности полученной на предыдущем шаге последовательности согласно формальным правилам языка. Данный модуль устанавливает структурные отношения между элементами программы.

Перед последним обязательным компонентом часто идёт генератор промежуточного представления, преобразующий проверенное синтаксическое дерево в унифицированную форму, которая:

- сохраняет семантику исходной программы;
- допускает применение оптимизаций;
- является платформенно независимой.

Генератор кода завершает процесс трансляции, преобразуя промежуточное представление в эквивалентный код целевого языка. Эта часть существенно зависит от особенностей как исходного, так и результирующего языков.

Помимо представленных этапов транслятор может быть улучшен путём добавления в него шагов семантического анализа и оптимизации запроса. Схема взаимодействия модулей представлена на рис. 1.

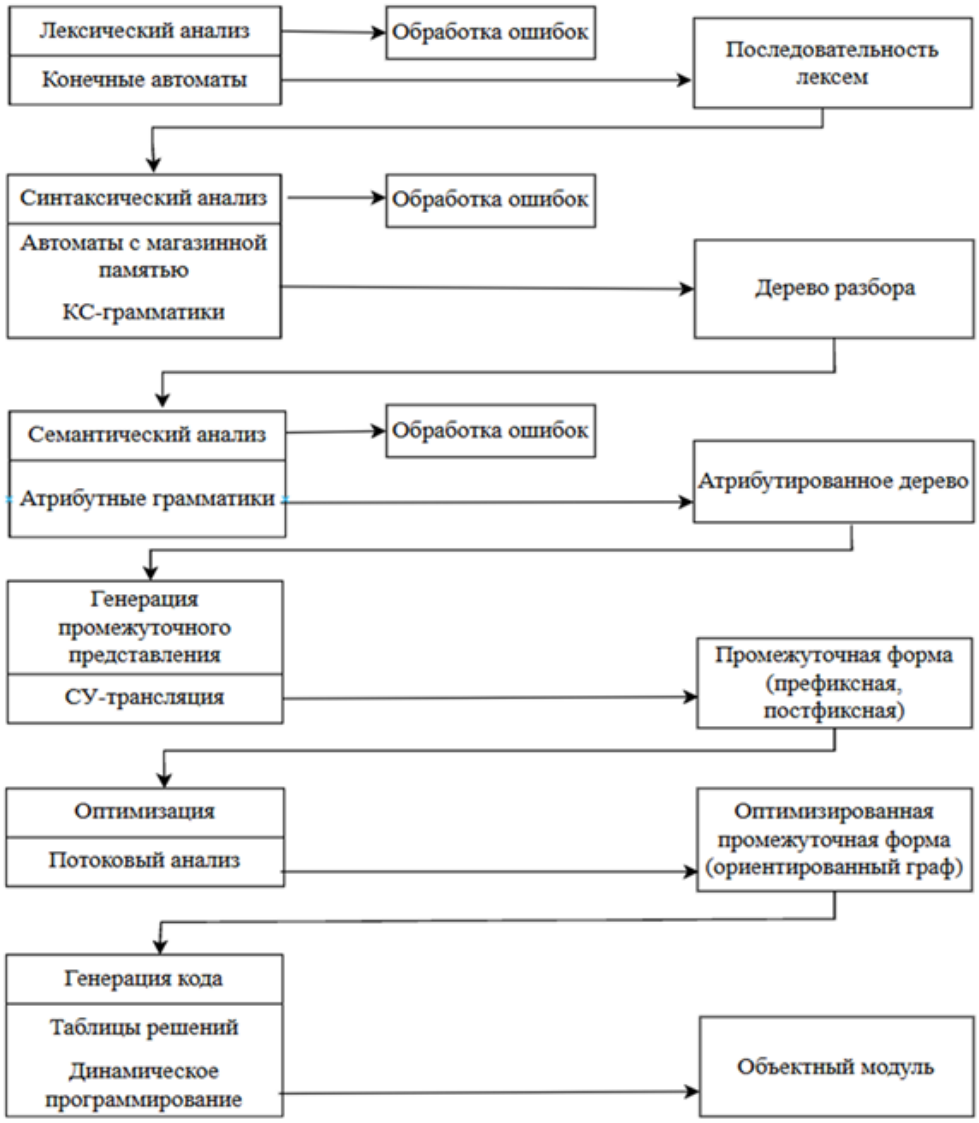

Рис. 1. Этапы трансляции

2.2. Существующие подходы к решению задачи и используемые стандарты

Язык есть множество текстов, которые состоят из отдельных знаков. Знак, называемый также буквой или символом, — минимальная единица текста, неразложимая на другие составляющие.

Синтаксис определяет структуру предложений естественных языков и программных конструкций. Регулярные грамматики обладают ограниченной выразительной силой при описании порядка слов, тогда как линейные и контекстно свободные формализмы демонстрируют значительно большие возможности. Именно поэтому последние являются стандартом для задания синтаксиса языков программирования.

Порождающая грамматика определяется четырьмя характеристиками:

- алфавитом нетерминальных символов (N);
- алфавитом терминалов (T);
- конечным множеством правил (P);
- начальным символом грамматики (S).

Контекстно свободная грамматика (КС грамматика) — это формальная порождающая грамматика, в которой все правила вывода в левой части имеют один нетерминальный сим-

вол, а в правой — произвольную цепочку терминалов и нетерминалов, включая пустую строку. КС-грамматики делятся по их пригодности для определённых методов синтаксического анализа на основные типы: LL(1), LL(k) и LR(1).

Лексический анализ — это процесс выделения во входной цепочке отдельных простейших смысловых конструкций языка — лексем.

Целью лексического анализа является обнаружение лексем входного текста, их выделение, категоризация, преобразование в удобную для последующего анализа форму, и формирование потока токенов вместо цепочки символов.

Лексемы описываются регулярными языками, имеющими самый простой тип. Регулярные языки, в свою очередь, являются регулярными множествами, для задания которых используются регулярные грамматики, регулярные выражения и конечные автоматы.

Синтаксический анализ — это процесс проверки соответствия потока токенов, полученного на этапе лексического анализа, правилам формальной грамматики. Его целью является построение дерева разбора, которое отражает иерархию и вложенность конструкций языка.

Основой для построения распознавателей КС-языков являются автоматы с магазинной памятью — МП-автоматы. Переход МП-автомата из одного состояния в другое зависит как от входного символа, так и от одного или нескольких верхних символов стека. При этом автомат называется недетерминированным, если при одной и той же его конфигурации возможен более, чем один переход. В противном случае он считается детерминированным.

3. Программный комплекс

Для описания синтаксиса языка SQL следует выбрать синтаксические диаграммы Вирта, которые отличаются наглядностью, простотой восприятия и эквивалентностью контекстно свободным грамматикам.

В качестве лексического анализатора подходит вариант, основанный на детерминированных конечных автоматах, так как их отличительными особенностями являются простота реализации, эффективность и надёжность.

Для выполнения синтаксического анализа обоснована реализация на основе комбинированного подхода, сочетающего рекурсивный спуск для простых запросов и использование стека в сложных случаях с целью просмотра и оценки впереди стоящих символов.

Для реализации серверной части приложения был выбран язык Java благодаря богатой стандартной библиотеке и наличию развитых фреймворков для работы с данными. Клиентская часть разработана с использованием React — популярной JavaScript библиотеки для создания пользовательских интерфейсов, которая обеспечивает богатую экосистему дополнительных библиотек, адаптивность и кросс браузерность.

На данном этапе реализованы синтаксический и лексический анализаторы. Их диаграмма классов показана на рис. 2.

В рамках вычислительного эксперимента была проверена корректность выполнения первых двух этапов трансляции на примере тестовых запросов, после ввода которых были получены результаты, представленные на рис. 3 и 4.

Заключение

В процессе решения поставленной задачи была создана и протестирована программа, позволяющая выполнять первые два этапа трансляции выражений с SQL в MongoDB формат: лексический и синтаксический анализы. Был реализован специальный веб-интерфейс, позволяющий вводить SQL запросы и отображать результаты их разбора.

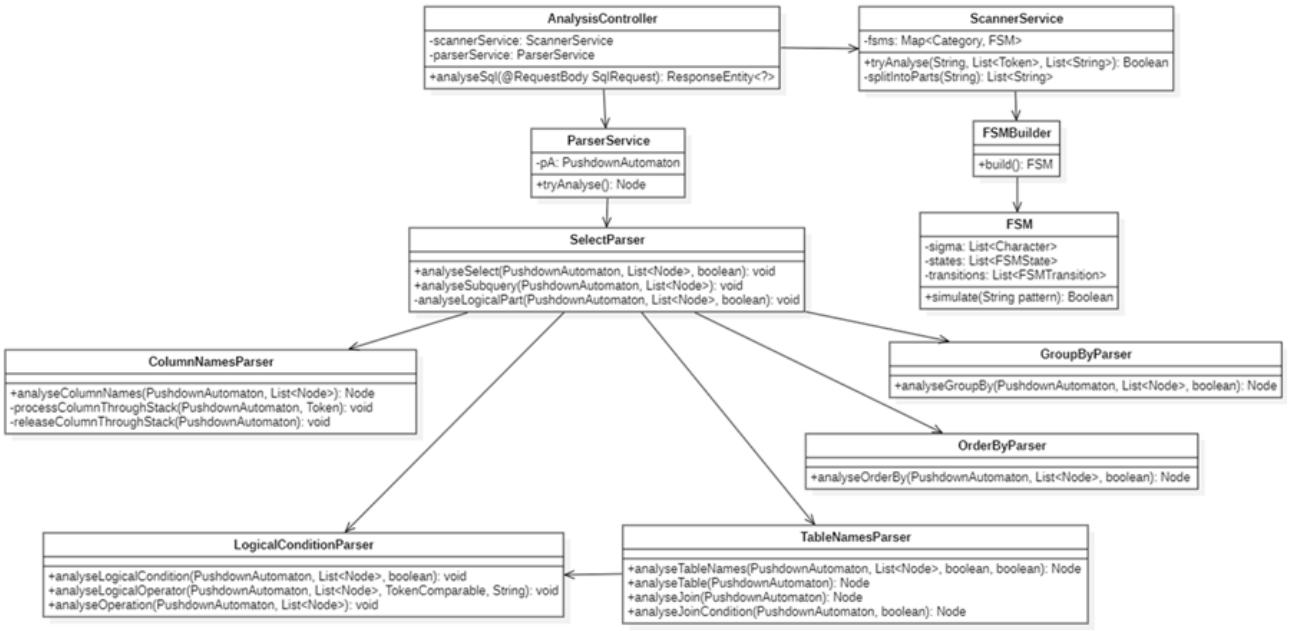

Рис. 2. Диаграмма классов синтаксического и лексического анализаторов

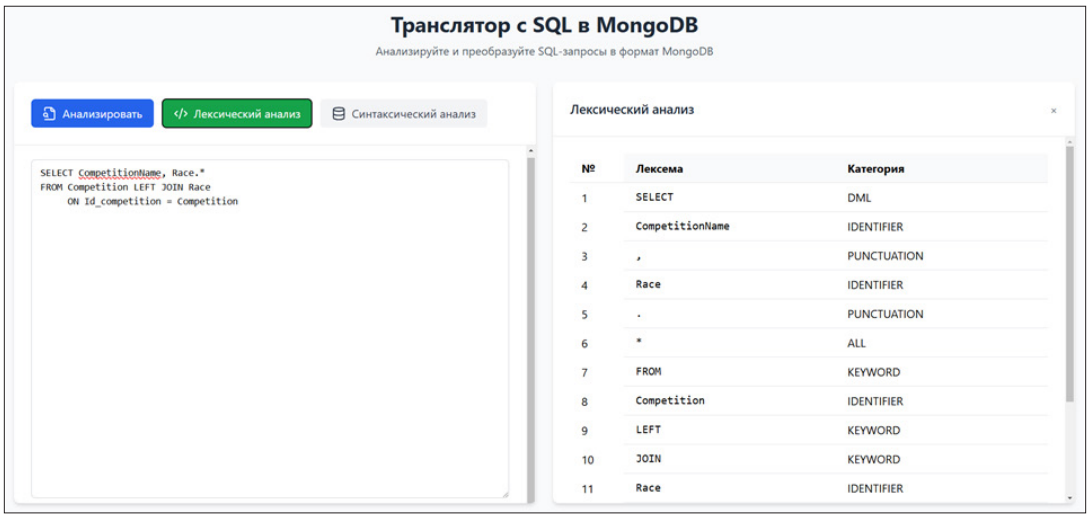

Рис. 3. Окно веб-приложения при работе с результатами лексического анализа

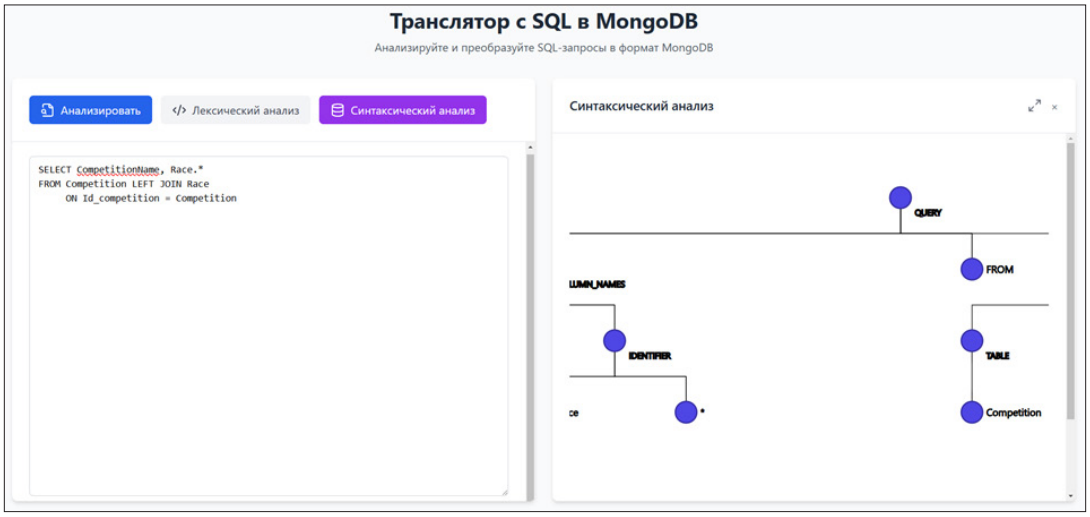

Рис. 4. Окно веб-приложения при работе с результатами синтаксического анализа в общем виде

Данная работа позволила выявить особенности построения SQL выражений на основе диалекта системы управления базами данных PostgreSQL с использованием языка программирования Java и изучить основные методы трансляции и компиляции.

Кроме того, были заложены основы для развития транслятора путём добавления этапов, связанных с генерацией кода на MQL, и поддержки большего подмножества SQL.

Литература

1. Конспективное изложение теории языков программирования и методов трансляции. Книга 1. Формальные языки и грамматики / Вл. Пономарев. – Озерск: ОТИ НИЯУ МИФИ, 2019. – С. 36–39.

2. Конспективное изложение теории языков программирования и методов трансляции. Книга 2. Лексический анализ / Вл. Пономарев. – Озерск : ОТИ НИЯУ МИФИ, 2019. – С. 7.

3. Конспективное изложение теории языков программирования и методов трансляции. Книга 3. Синтаксический анализ / Вл. Пономарев. – Озерск: ОТИ НИЯУ МИФИ, 2019. – С. 9–14.

4. Языки программирования и методы трансляции / С. З. Свердлов. – 2 е изд., испр. – СПб. : Издательство «Лань», 2019. – С. 234–235.

5. Формальные грамматики и языки. Элементы теории трансляции / И. А. Волкова, А. А. Вылиток, Т. В. Руденко. – 3 е изд – М. : Издательский отдел факультета ВМиК МГУ им. Ломоносова, 2009. – С. 48–55.

6. Математическая теория контекстно-свободных языков / С. Гинзбург. – М. : Мир, 1970. – С. 146–151.

7. Теория синтаксического анализа, перевода и компиляции, том 1 / А. Ахо, Дж. Ульман – М. : Мир, 1979. – С. 71–72.

8. Теория языков программирования и методы трансляции / П. А. Кочкарова, М. У. Эркенова. – Черкесск : БИЦ СКГА, 2024. – С. 29–30.

УДК 004.75

КРОССПЛАТФОРМЕННАЯ IoT-ПАРКОВОЧНАЯ СТАНЦИЯ С ИНТЕГРАЦИЕЙ ПЛАТЕЖНЫХ, ФИСКАЛЬНЫХ УСТРОЙСТВ И ВСПОМОГАТЕЛЬНЫХ ДАТЧИКОВ

К. С. Скоморохов, С. Ю. Болотова

Воронежский государственный университет

Аннотация. В статье рассматривается проект IoT станции оплаты парковки на базе Raspberry Pi, объединяющей устройства оплаты и датчики, такие как индукционная петля для обнаружения автомобиля, дополнительные сенсоры контроля состояния, терминалы оплаты, купюроприемник, термопринтеры для выдачи чеков, фискальные аппараты, экраны, тачпад и клавиатуры для взаимодействия с пользователем. Программная часть станции реализована кроссплатформенно на .NET с использованием Avalonia. Проводится сравнительный анализ универсальности, автономности и удобству использования существующих решений.
Ключевые слова: Raspberry Pi, GPIO, .NET, Avalonia, IoT, интернет вещей.

Введение

Рост числа автомобилей в городах порождает значительные проблемы: нехватка парковочных мест, пробки, потеря времени при поиске парковки. В связи с этим активно развивается рынок «умных парковок» (smart parking), использующий технологии интернета вещей (IoT).

Сегодня существует множество коммерческих решений автоматизации парковок, включающих как аппаратные компоненты (терминалы, шлагбаумы, датчики), так и облачные системы управления.

Однако многие из них либо крупномасштабны, либо ограничены в гибкости и не поддерживают интеграцию разных типов датчиков или фискальных устройств, либо используют проприетарное программное обеспечение.

Данная статья представляет решение по созданию IoT-парковочной станции, разработанной для обеспечения максимальной интеграции устройств, кроссплатформенной программной базы и соответствующей современным требованиям фискализации. Также приводится сравнение решения с альтернативными подходами.

1. Обзор существующих решений

В настоящее время наиболее популярные способы оплаты парковки включают мобильные приложения, веб-сайты, паркоматы и операторов. Каждый из этих вариантов обладает своими особенностями и преимуществами [3], которые представлены в табл. 1.

Мобильные приложения позволяют оплачивать парковку прямо с телефона, используя банковскую карту или электронный кошелек. Основные преимущества этого метода заключаются в возможности дистанционного продления времени стоянки, получения уведомлений и электронных квитанций, что делает его особенно удобным для постоянных пользователей. Однако для разовых посетителей или туристов использование приложения может быть менее комфортным: требуется наличие смартфона, доступ в интернет и регистрация в системе, а также стабильное подключение к сети.

Оплата парковки через веб-сайт позволяет оплатить стоянку с любого устройства с браузером, без необходимости устанавливать приложение, что удобно для тех, кто предпочитает компьютер или не хочет регистрироваться в мобильных сервисах. К преимуществам такого подхода можно отнести возможность заранее выбрать зону и время парковки, а также доступ

к деталям и квитанциям онлайн. Недостатки веб-сайта заключаются в том, что интерфейс зачастую не адаптирован для мобильных устройств. Отсутствуют также функции уведомления об окончании времени парковки и возможности продлить стоянку в один клик. Каждый раз нужно вручную вводить номер машины и зону, а также система полностью зависит от стабильного интернет-соединения — без сети оплатить парковку невозможно, что делает сервис менее удобным для ситуаций, когда требуется быстрая оплата непосредственно у автомобиля.

Таблица 1

Сравнение решений по оплате парковок

	Мобильное приложение	Веб-сайт	Оператор	Паркомат
Зависимость от сети интернет	+	+	–	–
Требование мобильного устройства	+	+	–	–
Требование установки дополнительного приложения	+	–	–	–
Дистанционная оплата	+	+	+	–
Оплата наличными	–	–	+	+
Бумажный чек	–	–	+	+
Моментальная фискализация оплаты	+	+	–	+
Необходима регистрация	+	+	–	–

Оператор или парковочный контролер позволяет оплатить парковку наличными или картой и получить помощь или консультацию на месте. Это удобно для людей, которые не пользуются цифровыми методами. Однако у этого способа есть недостатки: ограниченное время работы, возможные очереди и зависимость от оператора, что увеличивает время и стоимость обслуживания.

Паркоматы и терминалы — универсальное решение для оплаты на месте. Они принимают наличные, банковские карты и бесконтактные платежи. Пользователю не требуется смартфон или регистрация, а подтверждение оплаты предоставляется мгновенно в виде бумажного талона или электронного чека. Преимущество паркоматов в их универсальности: они подходят и для постоянного пользования, и для разовых посетителей, а процесс оплаты прост и понятен.

Паркомат выигрывает тем, что может работать полностью автономно, обеспечивая мгновенную оплату и фискализацию без участия человека, даже при отсутствии стабильного интернета или устройства для входа на сайт или приложения. Это исключает задержки и сложности при выезде автомобилей с парковки, устраняет зависимость от оператора и делает процесс прозрачным и удобным для всех водителей. Паркомат сочетает в себе универсальность, надежность, скорость обслуживания и широкий выбор способов оплаты: от наличных до банковских карт и бесконтактных платежей.

Таким образом, паркоматы представляют собой оптимальное решение для оплаты парковки, объединяя в себе автономность работы, удобство и высокую эффективность.

2. Обзор предлагаемого решения

В основе станции лежит Raspberry Pi, оснащённый специализированной платой, которая расширяет возможности GPIO и позволяет подключать множество периферийных устройств, среди которых:

1. Датчики: индукционная петля (обнаружение автомобиля), датчик открытия корпуса паркомата.

2. Платежные устройства: терминалы Pax, Kozen, Vendotek и др.; купюроприемник Nv200SmartPayout.

3. Печатающие устройства: термопринтеры Custom и другие – для печати чеков.

4. Фискализация: встроена поддержка Atol Kaznachey с фискальным накопителем, возможность подключения к фермам фискализаторов Atol.

5. Интерфейс для пользователя: экран, тачпад и опционально клавиатура. Управление шлагбаумом происходит после успешной оплаты.

Станция работает на кроссплатформенном приложении, написанном на .NET [1] с использованием Avalonia, что даёт гибкость в разработке UI и возможность запуска на разных архитектурах, а также простоту реализации взаимодействия с различными устройствами и датчиками [2]. Логика программного обеспечения отвечает за обработку событий (въезд, оплата, фискализация, выдача чека), а также за хранение информации о платежах и клиентах.

При подъезде автомобиля срабатывает датчик индукционной петли. Станция активирует терминал оплаты, клиент совершает оплату через терминал, купюроприёмник или карточку. Программа обрабатывает транзакцию, выполняет фискализацию, и терминал печатает чек. После этого шлагбаум открывается. В случаях автономной станции с камерой распознавания номеров система может автоматически идентифицировать регистрацию автомобиля и начать сессию.

Заключение

Представленная кроссплатформенная IoT-парковочная станция на базе Raspberry Pi демонстрирует возможности интеграции датчиков и устройств оплаты в единую систему. Использование индукционных петель, сенсоров, терминалов оплаты, купюроприемников, фискальных устройств и интерфейсов для пользователя обеспечивает гибкость и универсальность станции, позволяя адаптироваться под различные сценарии эксплуатации. Программная часть, разработанная на .NET с использованием Avalonia, обеспечивает совместимость с разными архитектурами и удобное управление. Серверная логика отвечает за обработку событий, фискализацию и интеграцию с внешними системами мониторинга и аналитики.

Сравнение с современными способами оплаты – мобильными приложениями, веб-сайтами, живыми кассирами и классическими паркоматами – показывает, что IoT-парковочная станция сочетает преимущества всех этих подходов: удобство, прозрачность, универсальность и надежность, а также обеспечивает мгновенную оплату наличными, картами и бесконтактными способами без необходимости использования смартфона или регистрации, что позволяет работать автономно, фискализировать операции и минимизировать зависимость от персонала.

Таким образом, представленное решение демонстрирует, как технологии IoT и кроссплатформенное программное обеспечения могут быть объединены для создания гибкой, универсальной и функциональной платформы автоматизации парковочных мест.

Литература

1. *Сазановец Ф.* C# 11 и NET 7: Создание кроссплатформенного приложения на базе .NET 7 : учеб. пособие / Ф. Сазановец ; СПб.: БХВ-Петербург; Изд-во «БХВ-Петербург», 2024. – 288 с.

2. *Клири С.* Конкурентность в C#. Асинхронное, параллельное и многопоточное программирование : учеб. пособие / С. Клири. – 2-е межд. изд. – СПб.: Питер; Изд-во «Питер», 2020. – 272 с.

3. Хабр: Парковочные системы. Сравниваем 20 производителей автоматизированных систем платной парковки. : электронный ресурс. – URL: https://habr.com/ru/companies/intems/articles/322614/ (дата обращения: 25.11.2025).

УДК 004.051

ВЛИЯНИЕ WEBVIEW НА ПРОИЗВОДИТЕЛЬНОСТЬ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ И МЕТОДЫ ЕГО ОПТИМИЗАЦИИ

А. А. Смирнов

Воронежский государственный университет

Аннотация. В работе рассматривается влияние использования компонент WebView на производительность и пользовательский опыт мобильных приложений. Для оценки этих показателей проведены измерения времени загрузки, потребления памяти и доли пользователей, покидающих экран до завершения загрузки контента. На основе полученных данных был проведен сравнительных анализ базового сценария использования и трех методов оптимизации: предварительного прогрева WebView, использования пула экземпляров и механизма post messages. По результатам измерений определяются наиболее эффективные способы повышения стабильности и отзывчивости компонента в мобильных интерфейсах.

Ключевые слова: WebView, мобильное приложение, warm up, оптимизация, user experience, производительность, время загрузки, потребление памяти, метод WebView pool, механизм post messages.

Введение

В условиях постоянной конкуренции и требования по ускорению цикла разработки мобильных приложений со стороны компаний-заказчиков все более популярными становятся технологии по безрелизной поставке новой функциональности. Одним из наиболее распространенных решений для достижения этой цели является использование специальных веб-компонентов (WebView).

WebView — это встроенный компонент операционной системы, позволяющий приложению открывать веб-страницы внутри себя, а значит, динамически менять и отображать содержимое без необходимости выпуска новой версии приложения, поскольку все изменения на веб-странице отображаются сразу.

Однако у данной технологии имеются и недостатки в сравнении с нативными экранами. Обычно такие приложения уступают по производительности и не отличаются высокой плавностью и отзывчивостью интерфейса, что обеспечивает менее оптимальный пользовательский опыт для потребителей. Как итог, WebView без должной оптимизации способно привести к медленной загрузке контента и понижению процента конверсии пользователей предоставляемых услуг.

В данной работе рассматривается влияние WebView на производительность и UX мобильного приложения на примере операционной системы iOS и описываются методы оптимизации, позволяющие смягчить его негативные стороны по сравнению с нативной разработкой. К рассмотрению предлагаются три следующих метода оптимизации:

1) Предварительная инициализация и загрузка WebView (прогрев) — запуск компонента и загрузка начальной информации до момента показа пользователю;

2) Реализация пула WebView — повторное использование уже созданных N экземпляров вместо инициализации нового для каждого экрана;

3) Взаимодействие с нативным кодом мобильного приложения через механизм Post Messages на примере отображения индикатора загрузки для информирования пользователя с целью понижения негативного впечатления при взаимодействии с экраном.

1. Постановка задачи

Цель работы — определить, как методы оптимизации влияют на производительность мобильного приложения, использующего WebView, и оценить следующие метрики:

1) Время загрузки контента — насколько быстро загружается содержимое веб-страницы;

2) Потребление оперативной памяти, особенно в ситуациях с использованием технологии на нескольких экранах;

3) Поведение пользователя — в частности долю пользователей, покидающих экран не дожидаясь загрузки контента.

Для достижения поставленной цели предполагается:

1) Разработка специальной нагруженной веб-страницы (HTML, CSS, скрипты, изображения), содержащей сложную иерархию представлений, и измерение базовых показатели загрузки этого контента в WebView;

2) Реализация и применение по отдельности трех методов оптимизации (прогрев, пул, Post Messages) для того же сценария и зафиксировать изменения во времени загрузки, использовании памяти и поведении пользователей. Для анализа последней метрики предполагается, что группа из 40 человек будет поделена пополам. Первой половине пользователей достанется экран без оптимизации, а второй — с ней;

3) Провести сравнительный анализ результатов и сделать выводы о том, какой эффект дают оптимизации и в каких аспектах.

2. Исследование базового сценария

Чтобы объективно оценить влияние WebView на производительность мобильного приложения и эффективность его оптимизации, все измерения проводились на одинаковом устройстве с одним и тем же веб-сайтом в качестве основы. В качестве контента для загрузки был разработан сайт с четырьмя страницами и с множеством элементов и большой вложенностью иерархии представлений на каждой, что позволяет сымитировать тяжелый веб-сайт со значительной нагрузкой на систему.

Тестирование проводилось на устройстве iPhone 12 под управлением iOS 18, оснащённом чипом A14 Bionic и 4 ГБ оперативной памяти. Выбор устройства обоснован его распространённостью; характеристики (процессор и память) позволяют выявить узкие места WebView даже на достаточно мощном телефоне.

Для измерения производительности использовались предоставляемые Apple инструменты профилирования: фиксировался интервал от момента инициализации WKWebView до полного завершения загрузки страницы. Потребление памяти замерялось по показателям использования RAM приложением во время загрузки (пиковое значение). Поведение пользователей оценивалось на основе A/B-теста: сравнивалась доля пользователей, покинувших экран (нажали кнопку «Закрыть») до полной загрузки контента, при разных подходах к индикации загрузки. Далее описаны результаты измерения для базового сценария без оптимизаций.

Пользователь открывает экран с WebView, который загружает веб-страницу. Первично наблюдается задержка в связи с созданием объекта WebView и запуском браузерных процессов. После отображения контента возрастает потребление памяти. Если у пользователя нестабильное и/или интернет-соединение недостаточной скорости, он наблюдает белый статичный экран на время загрузки страницы, что может вызвать ощущение зависания приложения и вынудить покинуть экран, не дожидаясь полной загрузки. Результаты измерений производительности базового сценария отражены в табл. 1.

Таблица 1

Таблица показателей производительности в базовом сценарии

Критерий	Результат
Память	150 мегабайт
Время загрузки	5,73 секунды
Процент пользователей, не дождавшихся загрузки экрана в контрольной группе	20 %

3. Метод предварительного прогрева WebView

Суть метода заключается в том, чтобы заранее (еще до момента, когда пользователь перейдет на экран с WebView) создать экземпляр WKWebView в фоновом режиме и частично подгрузить стартовые кеши ресурса. Поскольку первое открытие сайта через WebView занимает больше всего времени и ресурсов, сделав это в скрытом режиме, пользователь сможет увидеть при открытии необходимого экрана контент без длительных задержек.

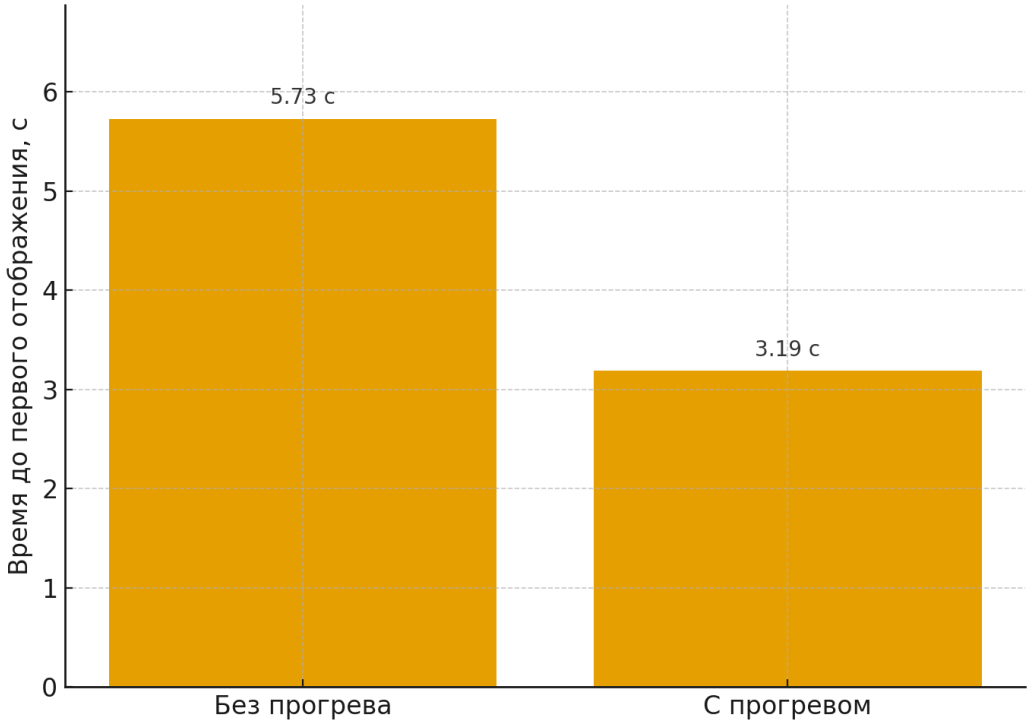

Рис. 1. Время загрузки контента на экране с WebView

Как показывают данные тестового приложения (рис. 1), предварительная инициализация и загрузка кешей сокращает время загрузки примерно на 45 %, что позволило контенту начать появляться практически мгновенно

Следует отметить, что метод прогрева требует от устройства дополнительной памяти (как и в случае с базовым сценарием). Но если пользователь так и не зайдет на экран с WebView, ресурсы будут потрачены зря. Поэтому применять данный метод следует взвешено: аргументировано выбирая точку старта инициализации компонента или при условии, что экран является часто посещаемым.

Таким образом, предварительная загрузка WebView с контентом значительно уменьшает штраф по времени при первом запуске, что приводит к улучшению пользовательского опыта и приближает производительность такого экрана к нативному решению. Основной недоста-

ток — потребление памяти даже в случае, когда пользователю так и не потребуется открытие экрана с веб-компонентом.

4. Метод организации пула и повторного использования WebView

Суть следующего метода: ограничить количество создаваемых экземпляров WebView и, по возможности, повторно использовать существующие для отображения новых страниц на экранах приложения. Как уже упоминалось ранее, создание экземпляра WKWebView это ресурсоемкая операция, потребляющая память и процессорное время. Вместо использования на каждом экране собственного экземпляра WebView предлагается ограничится конкретным количеством, повторно используя уже созданные объекты и вовремя их очищая.

Вариант реализации — выделение пула из 2 экземпляров при старте приложения. Когда нужно отобразить контент экрана, берем свободный экземпляр из общего пула. Когда он больше не нужен, мы его очищаем (загружаем пустую строку) и возвращаем в пул.

Ожидаемый эффект: снижение потребления памяти и ресурсов процессора при переходах между экранами. За счет отсутствия лишних инициализаций приложение не создает новый веб-процесс каждый раз.

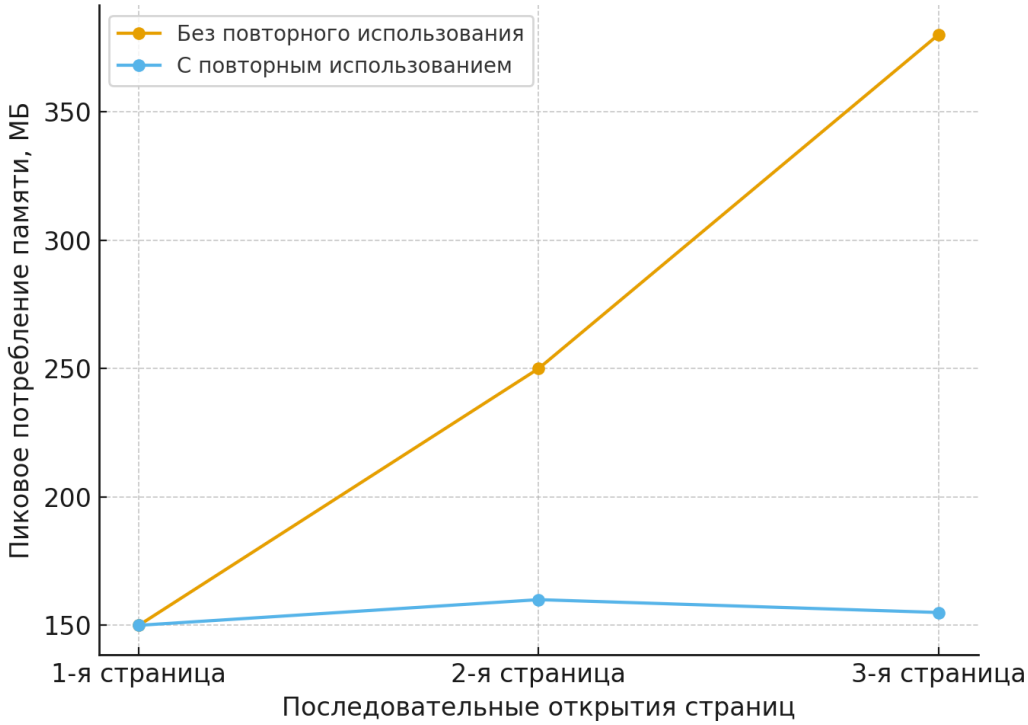

Рис. 2. Пиковое потребление памяти при последовательном открытии нескольких экранов

Если при навигации между тремя разными экранами каждый раз создавать новый экземпляр компонента (рис. 2), потребление памяти может возрасти до 380 МБ. Повторное использование не только позволяет экономить ресурсы, но и предотвращает негативные последствия в случае утечки памяти, которая может случиться по причине человеческого фактора.

Кроме памяти, значительно уменьшается и время загрузки новых страниц. После первого использования компонент уже является «разогретым» и позволяет открывать новые ссылки заметно быстрее, чем свежесозданный экземпляр. Такая организация работы с WebView нивелирует негативные впечатления при взаимодействии с экраном у пользователя.

Стоит отметить, что метод пула — это скорее организационная оптимизация: она снижает риск деградации производительности приложения при масштабировании веб-технологии

до некоторого количества экранов. За счет ограничения количества и грамотного управления компонентами достигается стабильная производительность и контролируемое потребление ресурсов.

Резюмируя, переиспользование WKWebView — отличное решение при реализации нескольких экранов внутри приложения с использованием веб-технологий. Оно значительно уменьшает нагрузку: приложение потребляет меньше памяти, является более отзывчивым при переходе между экранами и не создает лишние процессы. При этом сама организация работы с пулом не требует значительных усилий.

5. Метод взаимодействия с нативным приложением через Post Messages.

Суть метода заключается в увеличении процента удержания пользователя на экране за счет отображения информационного индикатора во время загрузки контента в WebView. Этот метод, в отличие от предыдущих двух, не ускоряет фактическую загрузку и не снижает потребление ресурсов. Вместо этого он решает UX-проблему, так как пользователю важно понимать, что загрузка продолжается и приложение не зависло.

Существует два самых распространенных вида индикаторов:

1) Спиннер — круглый виджет, стандартный индикатор активности;

2) Шиммер — блоки-заглушки на месте будущего контента, отображающиеся с анимацией мерцания. Поскольку они формируют макет будущего экрана, у пользователя создается ощущение более быстрой загрузки

Внедрение индикатора происходит посредством Post Messages — механизма связи между нативным кодом и WebView. С помощью него мы можем не только сообщить приложению, что страница готова к отображению, но и перенаправлять пользователя на другие экраны.

Без индикатора пользователь видит пустой экран и скорее всего решит, что приложение не отвечает (особенно при медленном интернете), и закроет его. С индикатором же очевидно, что идет процесс загрузки. Skeleton-экраны дают ещё больше контекста: пользователь видит макет будущего экрана, и воспринимает это так, будто бы загрузка идёт быстрее.

На рис. 3 видно, что процент ушедших с экрана пользователей, не дождавшихся загрузки контента в контрольной группе, составил 20 %, в то время как в тестовой группе – 15 %, что эквивалентно снижению отказов на 25 % относительно базового уровня. Это качественное улучшение UX: больше пользователей дождались появления реального контента. Они были готовы ждать дольше, поскольку видели индикацию загрузки, а значит, приложение вызывало больше доверия несмотря на фактическое время загрузки.

Стоит отметить, что метод индикатора — про оптимизацию восприятия пользователя, а не самой скорости. Реальное время загрузки страницы при этом не меняется. Однако пользователи в таком случае считают, что экран со скелетоном грузится быстрее, чем экран с пустотой. Этим приемом активно пользуются в индустрии, если разработчику важно удержать пользователя.

Нативная индикация загрузки существенно улучшает пользовательский опыт, хоть и не ускоряет загрузку самого контента. Наличие индикатора загрузки существенно понижает процент пользователей, не дожидающихся загрузки веб-страницы, так как в этом случае до пользователя доносится информация об активных процессах.

6. Интерпретация результатов

Измерения показали, что хоть у WebView и имеются некоторые недостатки, связанные с производительностью и UX, существуют методы, позволяющие существенно снизить негативный эффект от использования:

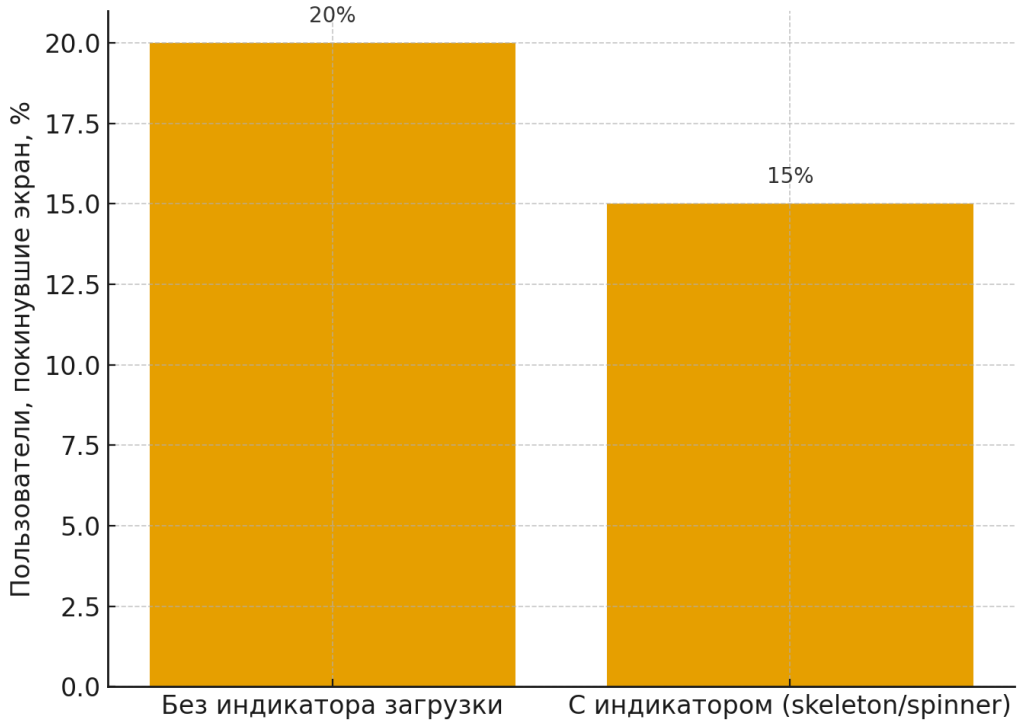

Рис. 3. Доля пользователей (%), покинувших экран до полной загрузки контента

1) Предварительный прогрев WebView, который значительно сокращает время первой отрисовки контента. Благодаря устранению задержки, связанной с инициализацией компонента и загрузки стартовых кешей, экран с WebView открывается быстрее на 45%. Однако стоит помнить, что метод всегда резервирует часть ресурсов даже в том случае, если пользователь не зайдет на экран;

2) Организация пула WebView позволяет контролировать выделение памяти. Вместо создания множества экземпляров компонента и соответствующих ему процессов, приложение переиспользует конечное (обычно небольшое) количество экземпляров WKWebView. Такой подход также ускоряет загрузку самого контента, так как при активном переключении экранов пользователь взаимодействует с уже прогретым компонентом;

3)Реализация нативной индикации загрузки с помощью механизма Post Messages. Данный метод оптимизации не влияет на фактическую производительность, но значительно улучшает восприятие скорости пользователем. Проведенный А/Б тест показал, что наличие нативных индикаторов загрузки, отображающихся мгновенно и до полного отображения веб-страницы, значительно влияет на удержание пользователей, которые готовы подождать, если видят моментальный отклик.

В ходе исследования было продемонстрировано, что компонент WebView является ресурсоемким и без должной оптимизации его проблематично назвать заменой нативной версии экрана. Поэтому, выбор в пользу WebView для реализации экрана с целью его дальнейшей безрелизной модификации следует делать осознанно.

Заключение

WebView остается полезным инструментом для интеграции динамично меняющегося контента в мобильные приложения, позволяя за минимальное время доставлять новую функциональность пользователю. При этом разработчику следует учитывать его ограничения. С помощью рассмотренных в статье подходов удалось повысить общую производительность и улучшить пользовательский опыт:

1) Предварительная инициализация снижает технические задержки;

2) Переиспользование компонента сокращает деградацию при масштабировании технологии на несколько экранов;

3) Нативная индикация позволяет удержать пользователя до полной загрузки контента.

Комплексное применение этих методов позволяет приблизить опыт работы с WebView к нативным приложениям насколько это возможно. Разумеется, компонент все еще уступает при воспроизведении сложных анимаций и общей плавности интерфейса, но в случаях, когда быстрая доставка новой функциональности является самой приоритетной задачей, с помощью оптимизаций можно достичь компромисса между скоростью и пользовательским опытом.

WebView негативно влияет на производительность и UX «из коробки», но при правильной оптимизации приложение способно обеспечивать приемлемую скорость загрузки. Разработчикам следует применять рассмотренные методы и держать баланс между веб и нативными компонентами, чтобы конечный пользователь получил лучшее от обоих технологий. WebView, при разумном подходе, остается мощным средством расширения функциональности мобильных приложений без критических потерь в качестве работы.

Литература

1. Две стороны WebView: о быстром запуске проектов и краже персональных данных // Хабр : [сайт]. – 2019. – URL: https://habr.com/ru/companies/sberbank/articles/440710/ (дата обращения: 19.11.2025).

2. Webview – что такое // Skyeng : [сайт]. – 2025. – URL: https://skyeng.ru/magazine/wiki/it-industriya/chto-takoe-webview/ (дата обращения: 19.11.2025).

3. WebView: забыть нельзя интегрировать // Хабр : [сайт]. – 2022. – URL: https://habr.com/ru/companies/cian/articles/690536/ (дата обращения: 19.11.2025).

4. Как оптимизировать производительность приложений WebView: лучшие практики // AppMaster : [сайт]. – 2023. – URL: https://appmaster.io/ru/blog/kak-optimizirovat-proizvoditelnost-prilozhenii-webview (дата обращения: 19.11.2025).

5. WebView в мобильном приложении: плюсы, минусы и экономия денег // VC.ru : [сайт]. – [б. г.]. – URL: https://vc.ru/life/851682-webview-v-mobilnom-prilozhenii-plyusy-minusy-i-ekonomiya-deneg (дата обращения: 19.11.2025).

УДК 004.4

ПРОЕКТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ ДЛЯ ОТСЛЕЖИВАНИЯ ПРОГРЕССА В ФИТНЕСЕ

А. М. Тихонова, М. В. Матвеева

Воронежский государственный университет

Аннотация. В работе рассматривается проектирование веб-приложения, которое предоставляет клиентам и тренерам фитнес-программ инструмент для отслеживания прогресса. В статье описан анализ существующих на рынке решений, на основе которого выявлены ключевые требования к функциональности приложения. Приводится разработанная модель базы данных, архитектура серверной части веб-приложения, интуитивно понятный интерфейс, а также обоснован выбор технологических решений, обеспечивающий хранение данных и обработку запросов.
Ключевые слова: веб-приложение, проектирование веб-приложения, программирование, веб-разработка, java, фитнес, отслеживание прогресса, серверная часть, пользовательский интерфейс.

Введение

В современном мире здоровье и красота стали неотъемлемыми атрибутами успешного человека. Высокий темп популяризации фитнес-индустрии превращается в настоящий тренд, охватывающий людей всех возрастов. Однако, несмотря на растущий интерес, многие сталкиваются с трудностями в достижении своих целей. Без удобных инструментов даже самые упорные усилия могут не принести желаемого результата.

В статье будет описано проектирование веб-приложения, которое позволит отслеживать фитнес-прогресс, имея все данные о тренировках, питании и физических показателях в едином месте. Это позволит анализировать динамику изменений и своевременно корректировать стратегию на пути к достижению целей.

1. Анализ существующих решений

Среди систем, предлагающих функциональность для отслеживания фитнес-прогресса, можно выделить три ключевых варианта: существующую систему на базе Excel, представляющую собой набор электронных таблиц, разработанных для ручного ввода и анализа данных, фитнес-приложение Physical Life [3] и приложение BodyLine [2]. Выбор именно этих вариантов позволяет провести сравнительный анализ по разработанным критериям, выявив возможности для проектирования нового веб-приложения. Результат сравнения представлен в табл. 1.

Таблица 1

Сравнительный анализ существующих решений

Критерий	Excel-система	Physical Life	BodyLine
Гибкость настройки под индивидуальные нужны	+	–	+/–
Автоматизация ввода данных	–	+	+
Взаимодействие клиента и тренера	+/–	–	–
Уведомления и напоминания	–	+	+
Удобство интерфейса	–	+/–	+/–
Персонализированные рекомендации	+/–	–	+

Проведенный анализ выявил существенные ограничения в каждом из рассмотренных решений. Excel-система, предлагая гибкость, обременена ручным вводом и отсутствием автоматизации. Специализированные приложения автоматизируют сбор данных, но страдают от недостаточной гибкости и слабого взаимодействия между клиентом и тренером. Комплексные платформы, такие как BodyLine, имеют необходимую функциональность, но испытывают сложности с удобством интерфейса. Таким образом, существует потребность в веб-приложении, которое сочетало бы гибкость ручных систем, автоматизацию и полноценное взаимодействие между тренером и клиентом.

Для удовлетворения этой потребности разрабатываемое веб-приложение будет включать расширенный набор показателей, позволяющий более детально оценивать прогресс. Будут внедрены настраиваемые напоминания о заполнении данных, что повысит регулярность и точность собираемой информации. Тренерам будет предоставлен полный доступ к профилю клиента с возможностью добавления комментариев и корректировки программы тренировок. Автоматическая обработка данных позволит минимизировать ошибки и освободить пользователей от рутинных операций. Наконец, интуитивно понятная структура приложения обеспечит пользователям быстрый доступ к необходимым инструментам и информации. В совокупности эти улучшения создадут инструмент для эффективного отслеживания фитнес-прогресса.

2. Требования к веб-приложению

Веб-приложение должно предоставить клиенту следующие возможности:

– регистрация (заполнение регистрационной анкеты с указанием личных данных и начальных физических параметров);

– аутентификация;

– добавление данных в отчет (вес, время сна, самочувствие, цикл, вид тренировки, оценка нагрузки, время тренировки, питание (КБЖУ + количество растительности), внетренировочная активность (количество шагов), уровень голода, замеры);

– просмотр задач в планере, который предназначен для напоминаний о необходимости добавления данных;

– просмотр справочной информации (материалы по питанию и тренировкам);

– просмотр комментариев к отчету от тренера;

– просмотр списка тренеров, с которыми работает клиент.

Должно предоставить тренерам следующие возможности:

– регистрация (заполнение регистрационной анкеты с указанием личных данных и начальных физических параметров);

– аутентификация;

– просмотр данных клиента (вес, время сна, самочувствие, цикл, вид тренировки, оценка нагрузки, время тренировки, питание (КБЖУ + количество растительности), внетренировочная активность (количество шагов), уровень голода, замеры);

– добавление и просмотр комментариев к отчету клиента;

– просмотр списка клиентов;

– загрузка и просмотр справочной информации (материалы по питанию и тренировкам);

– добавление и просмотр задач для клиента в планере;

– добавление и удаление клиента.

3. Модель данных

Для хранения пользовательской информации, задач, назначенных клиентам их тренерами, и данных из отчетов была разработана база данных. На рис. 1 представлена логическая модель.

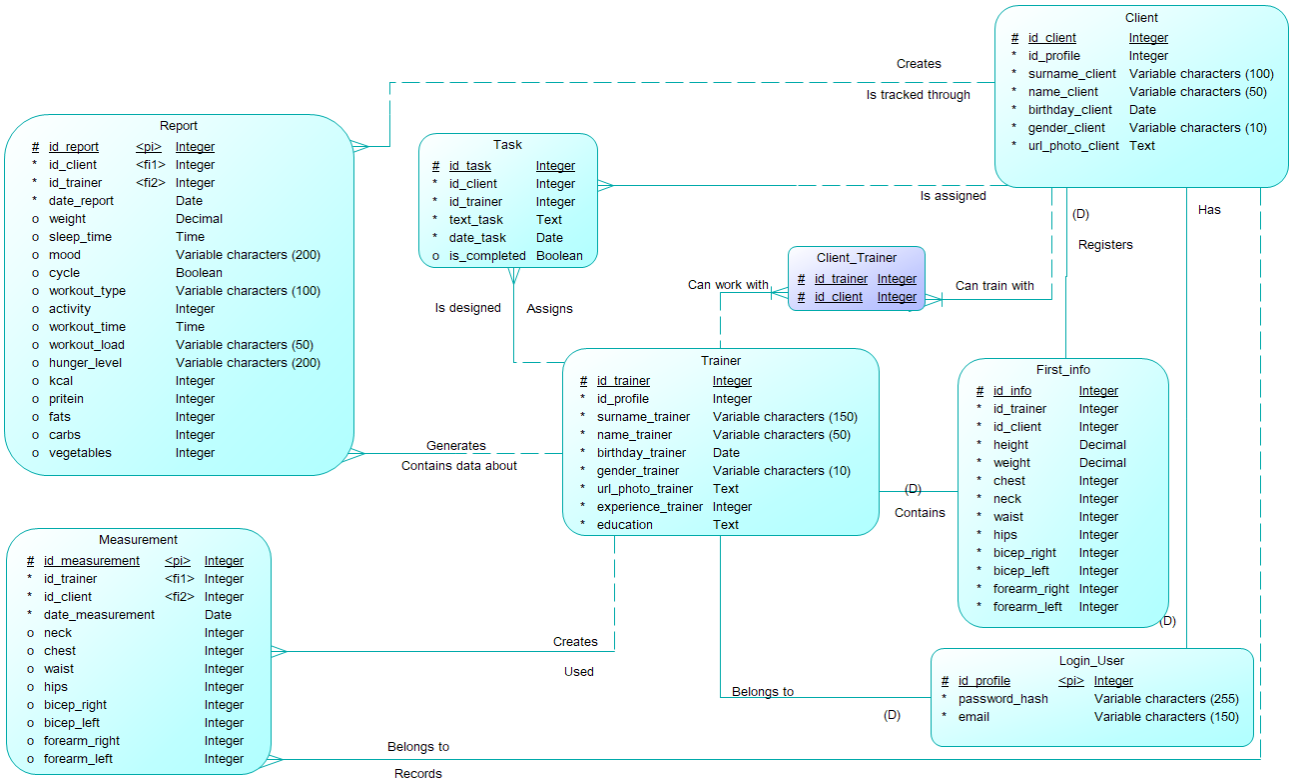

Рис. 1. Логическая модель данных

В табл. 2 представлено описание основных таблиц базы данных.

Таблица 2

Описание основных таблиц базы данных

Таблица	Описание
Trainer	Хранит информацию о каждом тренере
Client	Хранит информацию о каждом клиенте
Login_User	Хранит учетные данные для входа в систему
First_info	Хранит замеры тела клиента, сделанные в начале его тренировочной программы
Task	Хранит задачи, назначенные клиентам их тренерами
Report	Хранит данные ежедневного отчета
Measurement	Хранит данные о замерах тела клиентов

4. Структура серверной части

Серверная часть состоит из следующих частей:

– директория config содержит классы-конфигурации приложения — настройки для автодокументирования API, а также настройки безопасности и авторизации;

– директория controller включает REST-контроллеры (AuthController.java, ClientController.java, TrainerController.java), которые принимают HTTP-запросы от клиентов и тренеров, вызывают сервисы бизнес-логики и возвращают ответы в формате JSON;

– директория dto содержит объекты передачи данных (Data Transfer Objects), разделённые на запросы и ответы. MapStruct используется для автоматического преобразования между сущностями и DTO, что упрощает и ускоряет разработку;

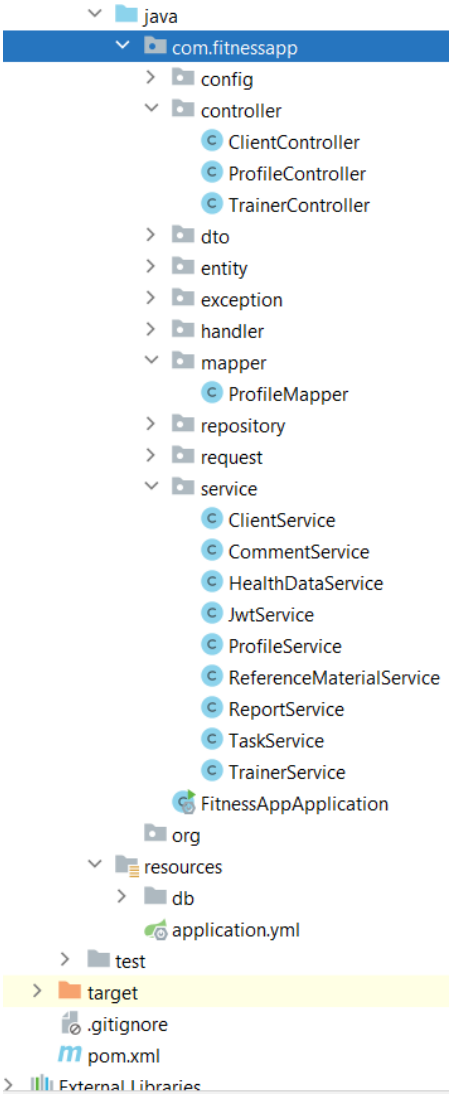

Рис. 2. Структура серверной части проекта

– директория entity содержит классы, объекты которых могут быть отображены на реляционную базу данных PostgreSQL посредством миграции при помощи Hibernate ORM;

– в директории repository реализованы операции CRUD и запросы к базе данных. Репозитории обеспечивают эффективное взаимодействие с PostgreSQL [4], используя преимущества Hibernate и JPA [5];

– директория service содержит классы бизнес-логики приложения;

– файл FitnessAppApplication.java является точкой входа в приложение;

– директория resources содержит ресурсные файлы приложения. В файле application.yml хранятся настройки конфигурации (параметры подключения к базе данных). Поддиректория db/migration содержит миграции для управления схемой базы данных;

– файл pom.xml — это Maven-файл управления зависимостями и сборкой проекта, где указаны версии библиотек, плагинов и конфигурация сборки.

Структура проекта представлена на рис. 2.

5. Интерфейс пользователя

Интерфейс приложения [1] состоит из нескольких отдельных экранов:

– страница входа;

– страница регистрации;

– отчет;

– вкладка динамика;

– прогресс;

– профиль клиента/тренера;

– настройки;

– вкладка со списком клиентов/тренеров;

– вкладка с материалами по питанию и тренировкам;

– планер с задачами.

На странице входа расположены поля ввода для электронной почты и пароля, кнопки «Войти» и «Зарегистрироваться». Если пользователь нажимает на регистрацию, он переходит на соответствующий экран, где вводит необходимую для регистрации информацию: роль, имя, фамилия, пол, дата рождения, электронная почта, пароль. Также пользователь заполняет анкету своими данными (первичная информация) — рост, вес, замеры тела.

После авторизации происходит переход на страницу с отчетом, который пользователь ежедневно заполняет. В нем находятся: текущий вес, время сна, цикл, количество шагов, самочувствие, данные для диагностики проделанных тренировок и питания. Данная страница представлена на рис. 3. Здесь же есть кнопка «Отправить замеры» для контроля прогресса, и раздел для перехода в другую часть приложения под названием «Динамика», которая представлена на рис. 4.

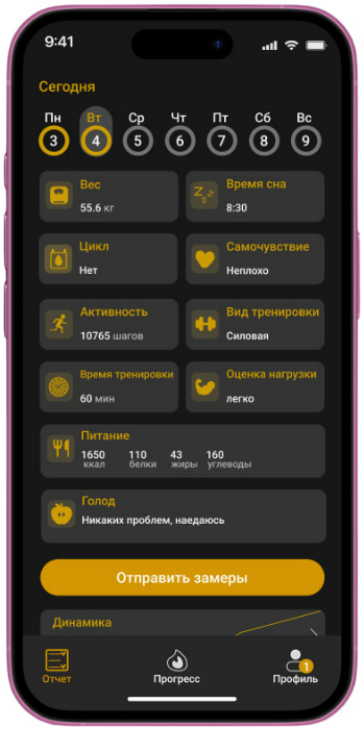

Рис. 3. Страница с отчетом

Рис. 4. Динамика

В нижней части экрана через меню можно перейти в прогресс, где пользователь отслеживает свои результаты, а также в профиль, где расположены контакты, список клиентов/тренеров в зависимости от роли пользователя, чек-листы с материалами по питанию и тренировкам, планер с задачами, а также настройки аккаунта.

Заключение

Спроектировано веб-приложение, предназначенное для отслеживания фитнес-прогресса. Были проанализированы существующие решения, выявлены их преимущества и недостатки. Разработаны требования к приложению, спроектирована модель базы данных, архитектура серверной части веб-приложения и интерфейс.

На данный момент подходит к концу реализация серверной части приложения в соответствии с описанной архитектурой. В дальнейшем планируется: разработка клиентского веб-интерфейса с использованием современных фреймворков, интеграция механизмов аутентификации и авторизации, реализация базовой функциональности для ведения дневника тренировок и питания.

Литература

1. Разработка интерфейсов. Паттерны проектирования / Д. Тидвелл, Ч. Брюэр, Э. Валенсия. – 3-е изд. – Санкт-Петербург, 2022. – 560 с.

2. Bodyline. – URL: https://bodylineapp.com/ (дата обращения: 26.11.2025).

3. Physical Life. – URL: https://pht.space/app/ (дата обращения: 26.11.2024).

4. PostgreSQL Documentation. – URL: https://www.postgresql.org/docs/ (дата обращения: 10.04.2025).

5. Spring Data JPA. – URL: https://spring.io/projects/spring-data-jpa (дата обращения: 10.05.2025).

УДК 004.93

ОПРЕДЕЛЕНИЕ МЕТОДОВ ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ И МАСШТАБИРУЕМОСТИ СОВРЕМЕННЫХ ВЕБ-ФРЕЙМВОРКОВ

Д. О. Ткаченко

Воронежский государственный университет

Аннотация. В данной статье рассматриваются подходы к оценке производительности и масштабируемости современных клиентских веб-фреймворков. Анализируются архитектурные особенности React, Angular и Vue, определяются методы оценки их эффективности при росте структуры интерфейса, увеличении количества событий и усложнении модели данных. Выделяются основные критерии, позволяющие сравнивать современные фреймворки с практической точки зрения и прогнозировать их поведение при увеличении нагрузки. Полученные результаты могут быть использованы при выборе технологий разработки высоконагруженных веб-приложений.
Ключевые слова: веб-фреймворки, производительность, масштабируемость, React, Angular, Vue, оценка эффективности, нагрузочное тестирование, время отклика.

Введение

Современные веб-приложения стремительно усложняются и становятся всё более интерактивными. Это приводит к возрастанию требований к производительности клиентской части и устойчивости приложения при интенсивных пользовательских взаимодействиях. Выбор фреймворка становится стратегическим решением, определяющим не только удобство разработки, но и возможности масштабирования в будущем.

Однако сравнительный анализ фреймворков чаще всего носит описательный характер и не опирается на формальные методы оценки. На практике разработчикам необходимо понимать, как поведение фреймворка зависит от структуры приложения, количества компонентов, частоты событий и особенностей реактивности.

Целью работы является определение методов оценки производительности и масштабируемости современных веб-фреймворков и установить критерии, которые позволяют объективно анализировать их поведение в условиях увеличения нагрузки. Для достижения данной цели решаются следующие задачи:

- Определить архитектурные особенности React, Angular и Vue, влияющие на производительность.
- Выделить ключевые факторы, определяющие масштабируемость клиентских приложений.
- Сформировать систему критериев для сравнения фреймворков.
- Определить подходы, позволяющие оценивать производительность на практике.

1. Архитектурные особенности современных веб-фреймворков

1.1. React

React основан на концепции виртуального DOM, позволяющего минимизировать количество операций обновления реальной структуры документа. Благодаря этому большинство изменений изолировано внутри отдельных компонент или их поддеревьев.

К характерным особенностям React относятся:

- использование виртуального дерева для оптимизации рендеринга;

• однонаправленный поток данных, позволяющий контролировать распространение изменений;

• высокая стабильность времени обновления при локальных изменениях;

• зависимость производительности от количества точек обновления, но не от всего дерева.

Благодаря этим свойствам React демонстрирует хорошую масштабируемость в приложениях с активным пользовательским взаимодействием.

1.2 Angular

Angular использует модель детектирования изменений, предполагающую анализ состояния компонентов при каждом событии. Это обеспечивает согласованность данных, однако увеличивает нагрузку при большом количестве связанных элементов интерфейса.

Особенности Angular:

• полный либо полуавтоматический обход дерева при обновлениях

• мощная система связывания данных

• высокая предсказуемость поведения компонентов

• рациональность в крупных корпоративных системах со сложной бизнес-логикой

Главным ограничением Angular является зависимость производительности от размеров дерева компонентов.

1.3. Vue

Vue реализует реактивную модель, основанную на отслеживании зависимостей между данными и компонентами. Такой подход позволяет обновлять только те элементы интерфейса, которые непосредственно связаны с изменением данных.

Преимущества Vue:

• высокая эффективность локальных обновлений

• минимизация числа операций рендеринга

• возможность гибкой декомпозиции интерфейса

• сравнительно низкие накладные расходы на управление состоянием

Vue обеспечивает хорошую производительность в приложениях с умеренной сложностью структуры.

2. Факторы, влияющие на производительность веб-фреймворков

Скорость работы веб-приложений зависит от нескольких важных вещей.

Во-первых, это структура интерфейса. Если страница состоит из большого числа вложенных элементов, браузеру приходится выполнять больше действий при каждом обновлении, что замедляет работу.

Во-вторых, важна частота взаимодействия пользователя с приложением. Частые клики, прокрутка или ввод данных создают нагрузку на систему, особенно если фреймворк проверяет сразу большое количество элементов на странице.

Также влияет то, как связаны данные между элементами. Если многие части страницы зависят от одного состояния, любое изменение будет затрагивать сразу несколько элементов. Фреймворки, которые обновляют только те части страницы, где произошло изменение, работают быстрее и экономнее.

Наконец, важна организация обновлений. Фреймворки, которые могут «умно» выбирать, что обновлять, и не трогать остальное, показывают лучшую производительность. Понимание этих факторов помогает прогнозировать, как фреймворк будет вести себя при усложнении приложения.

3. Методы оценки производительности и масштабируемости

Чтобы понять, насколько быстро работает веб-приложение, используют разные инструменты и показатели.

Инструменты для измерения:

• Chrome DevTools — встроенный в браузер набор инструментов для проверки скорости работы страницы, нагрузки на процессор и памяти.

• Lighthouse — программа от Google, которая автоматически проверяет скорость загрузки, отклик и готовность страницы к взаимодействию.

• WebPageTest — позволяет тестировать страницу в разных условиях сети и смотреть, сколько времени занимает её загрузка.

• React Profiler, Angular DevTools, Vue DevTools — специальные инструменты для каждого фреймворка, показывающие, сколько времени уходит на обновление отдельных элементов и какие части страницы обновляются чаще всего.

Что измеряют:

• Время отклика — сколько проходит времени между действием пользователя (например, нажатие кнопки) и реакцией страницы.

• Частота обновления интерфейса — насколько плавно и быстро обновляется страница.

• Использование памяти и процессора — сколько ресурсов тратится на работу страницы.

• Количество обновляемых элементов — показывает, как много частей страницы изменяется при каждом событии.

Такой подход позволяет реально увидеть узкие места и понять, где необходима оптимизация.

4. Сравнение масштабируемости фреймворков

Реальные тесты и исследования показывают заметные различия в поведении React, Angular и Vue под нагрузкой. React остаётся быстрым даже при глубоком дереве компонентов и частых обновлениях благодаря технологии виртуального DOM, которая позволяет обновлять только изменившиеся части страницы. Vue хорошо справляется с локальными изменениями, когда структура интерфейса относительно простая и данные связаны умеренно, что делает его удобным для проектов средней сложности. Angular отличается высокой надёжностью и предсказуемостью, но при увеличении числа элементов и частоте обновлений может демонстрировать замедление, так как проверяет состояние многих компонентов одновременно.

Практические измерения подтверждают эти наблюдения. Например, тесты с использованием Chrome DevTools и Lighthouse показали, что при обновлении 100 компонентов с частотой 50 изменений в секунду React и Vue обеспечивают более низкое время отклика и меньшую нагрузку на процессор по сравнению с Angular. Анализ потребления памяти и частоты обновлений компонентов позволяет разработчикам предсказывать, как фреймворк будет вести себя при росте нагрузки и масштабировании приложения.

Таким образом, выбор фреймворка должен учитывать не только функциональные возможности и удобство разработки, но и количественные показатели производительности и масштабируемости, полученные с помощью измерений и инструментов тестирования.

Заключение

В работе были определены методы оценки производительности и масштабируемости современных веб-фреймворков. Выделены ключевые факторы, влияющие на скорость обновления интерфейса и устойчивость при увеличении нагрузки. Проведённый анализ показал,

что фреймворки имеют значимые различия в организации реактивности, что влияет на их масштабируемость в различных сценариях.

Выявленные методы могут использоваться для практического выбора технологий при разработке высоконагруженных клиентских приложений, а также для прогнозирования поведения системы на этапах проектирования.

Литература

1. React. Официальная документация / Meta. – Режим доступа: https://reactjs.org/docs/getting-started.html.

2. Angular. Официальная документация / Google. – Режим доступа: https://angular.io/docs.

3. Vue.js. Официальная документация / Evan You. – Режим доступа: https://vuejs.org/guide/introduction.html.

4. Krause S. JS Framework Benchmark / GitHub. – Режим доступа: https://github.com/krausest/js-framework-benchmark.

5. *Воронин А. П.* Использование серверного рендеринга (SSR) для повышения производительности веб-приложений / Современные веб-технологии. – 2020. – № 3. – С. 32–38.

6. Google Chrome Labs. Анализ производительности популярных фреймворков. – Режим доступа: https://developers.google.com/web/tools/lighthouse.

УДК 004.94

РАСШИРЕНИЯ АЛГОРИТМА UTILITY AI ДЛЯ ОБЕСПЕЧЕНИЯ ПОВЕДЕНИЯ ПЕРСОНАЖЕЙ В ИГРАХ

Т. В. Тонких, Е. В. Трофименко

Воронежский государственный университет

Аннотация. В статье рассматривается алгоритм UtilityAI и расширения для данного алгоритма, которые обеспечивают поведение неигровых персонажей. Рассмотрены такие расширения как стоимость действия, личность неигрового персонажа и вероятностный выбор действий. Алгоритмы обеспечения поведения используются для симуляции выбора действий в зависимости от игровых ситуаций персонажами, которыми не управляет пользователь. В статье рассматривается базовая реализация алгоритма Utility AI и влияние его расширений на поведение игровых персонажей и на производительность на мобильных устройствах.

Ключевые слова: UtilityAI, искусственный интеллект на основе полезности, действие, выгодность действия, обстоятельство, важность обстоятельства, Unity, стоимость действия, личность неигрового персонажа, вероятностный выбор действий.

Введение

Алгоритмы обеспечения поведения используются для создания искусственного интеллекта для персонажей, которыми не управляет пользователь. Данные алгоритмы позволяют создать впечатления реалистичных действий игрового окружения. За счет алгоритмов обеспечения поведения персонажи действуют логично, адаптивно и естественно. В современных играх требования к реализму, разнообразию и интерактивности постоянно растут, что делает использование эффективных алгоритмов обеспечения поведения обязательным. Такие алгоритмы позволяют неигровым персонажам адаптироваться к стилю игры пользователя, создавая уникальные и сложные игровые ситуации. Кроме того, оптимизация поведения персонажей помогает сбалансировать производительность программного обеспечения, что важно для современных нагруженных игр с большим количеством динамических объектов. Таким образом, развитие алгоритмов обеспечения поведения является важным направлением для создания более увлекательных и живых игровых миров.

1. Обзор алгоритма Utility AI

Искусственный интеллект на основе полезности (Utility AI) — алгоритм подсчета очков на основе выгодности действий [1]. Данная система на основе заранее присвоенных доступных действий при каждом новом выборе действия начисляет каждому из них очки в зависимости от складывающихся заранее заданных доступных обстоятельств, за счет чего выбирает самое подходящее поведение для текущих условий.

Для реализации алгоритма Utility AI необходимо также реализовать два его главных аспекта: действие и обстоятельство [2]. Обстоятельство характеризуется функцией полезности (рис. 1). Функция задается разработчиком исходя из того, какой характер изменения важности обстоятельства от входного значения будет более логичен. Каждое обстоятельство связано с одной или множеством характеристик персонажа (например, уровень здоровья, количество денег), или характеристик окружающей среды (например, погода, расстояние). На вход функции полезности подается текущий уровень характеристики в процентах (отношение текущего значения характеристики к максимальному значению характеристики), выходным значением будет являться полезность в диапазоне от 0 до 1.

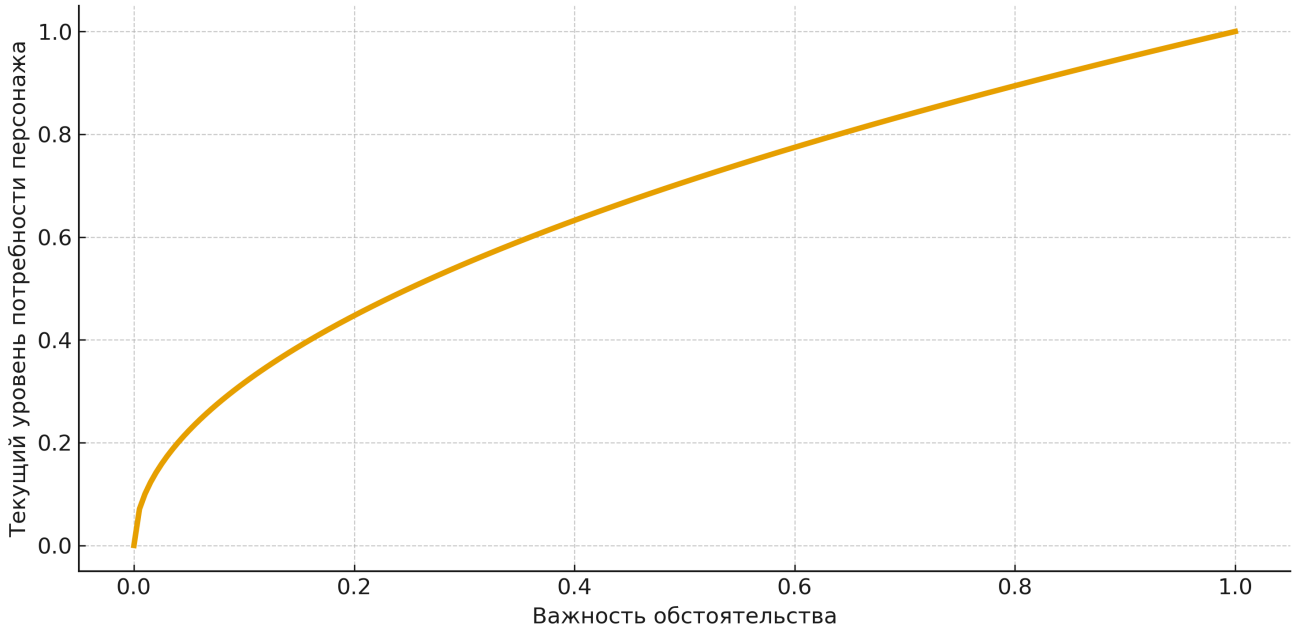

Рис. 1. Пример кривой, характеризующей обстоятельство

Действие характеризуется обстоятельством или набором обстоятельств, на основе важностей которых вычисляется выгодность действия. Для вычисления выгодности действия необходимо перемножить значения важности каждого из обстоятельств, влияющих на данное действие и применить к данному значению следующую формулу:

$$s = M + M \cdot (1 - M) \cdot \left(1 - \frac{1}{n}\right),$$

где s — выгодность действия, M — перемноженные значения важности обстоятельств, n — количество обстоятельств.

Таким образом, для каждого действия вычисляется выгодность действия, и действие с наибольшей выгодностью выбирается как самое подходящее.

2. Расширения алгоритма Utility AI

2.1. Обзор расширений алгоритма Utility AI

Стоимость действия

Расширение стоимость действия добавляет параметр или функцию, отражающую затраты на выполнение данного действия. Таким образом, действие будет обладать не только его выгодностью, но и стоимостью, которая также будет влиять на конечный выбор действия. В самом простом варианте реализации данного расширения можно задавать константные значения сложности каждому действию, или задавать динамическую стоимость действия, которая может отражать расстояние до объекта, связанного с данным действием, время, которое будет затрачено на выполнение данного действия, ресурсы, которые необходимо потратить для данного действия и другие стоимости, которые будут вписываться в контекст игрового окружения и игровых персонажей [3].

Личность неигрового персонажа

Расширение личность неигрового персонажа добавляет набор параметров, соответствующих каждому из доступных действий, выражающих личное отношение данного персонажа к действиям. Данное расширение позволяет добавить несколько заранее заданных характеров

персонажа, которые будут влиять на то, какие действия персонаж будет выбирать чаще. Таким образом, неигровой персонаж может иметь предрасположенность к работе, отдыху, сну, еде, общению или комбинации нескольких доступных действий. Данные параметры также можно реализовать динамическими и изменять их в зависимости от игрового окружения или ситуации. Влияние параметров личности можно учитывать как в важности обстоятельства, так и в выгодности действия, что дает гибкую настройку поведения персонажа. Риском расширения личность неигрового персонажа является «застревание» персонажа на одном и том же действии, если параметр данного действия достаточно высок. В таком случае используется затухающее влияние на действие, если у него и так высокая выгодность [4].

Вероятностный выбор действия

Расширение вероятностный выбор действия добавляет неигровым персонажам непредсказуемое поведение. Если в базовой реализации Utility AI выбирается то действие, которое обладает максимальной выгодностью, то с расширением вероятностного выбора действия всем действиям присваивается вероятность пропорционально их выгодности, затем случайным образом генерируется порог вероятности, после чего действие выбирается случайно из тех, что перешли данный порог [5].

2.2. Сравнение работы алгоритма Utility AI с расширениями

Сравнительный анализ алгоритма Utility AI с расширениями проводился с помощью движка Unity. Были реализованы следующие обстоятельства: количество энергии, уменьшающееся со временем, уровень голода, уменьшающийся со временем, количество денег, увеличивающееся за продажу ресурсов, заполненность инвентаря, увеличивающееся во время работы и следующие действия: сон, повышающее количество энергии, прием пищи, повышающая уровень голода, работа, повышающая количество ресурсов в инвентаре и складирование ресурсов, после которого повышается количество денег. Также было реализовано логирование действий, совершаемых персонажем в течение 2,5 минут работы алгоритма, в файл с расширением csv, на основе которого проводился сравнительный анализ.

Стоимость действия

Каждому действию была присвоена его сложность, не изменяющаяся со временем, а также учитывалось расстояние от персонажа до объекта действия. Итоговая стоимость действия учитывала базовую сложность действия и текущее расстояние от персонажа до соответствующего объекта.

Личность неигрового персонажа

Были реализованы три типа личности персонажа: трудоголик, лентяй, обжора. Трудоголику была присвоена высокая мотивация к работе, меньшая чувствительность к усталости и меньшая чувствительность к голоду, лентяю была присвоена низкая мотивация к работе, высокая чувствительность к усталости и средняя чувствительность к голоду, обжоре была присвоенная пониженная мотивация к работе, повышенная чувствительность к усталости и высокая чувствительность к голоду. Если сравнивать выбор действий персонажа с разными личностями, то трудоголик предпочитал действия работы и складирования ресурсов даже при низких значениях энергии и высоком значении голода, а лентяй и обжора предпочитали действия сна и приема пищи даже когда значения энергии немного отличалось от максимального или значение голода немного отличалось от минимального.

Вероятностный выбор действий

Для расширения вероятностного выбора действий был реализован подсчет вероятностей действий и случайного выбора действия на основе данных вероятностей. Данное расширение

добавляет непредсказуемость поведения персонажа, так как действия с низкой выгодностью могут быть выбраны с низкой вероятностью в отличие от базовой реализации, когда мы выбираем действие строго с максимальной выгодностью. Таким образом, персонаж неожиданно мог выбирать действия сна или приема пищи, даже если его потребности были на хорошем уровне.

Сравнение расширений

На основе данных из файлов, в которых записывалось время, название выбранного действия, полезность выбранного действия и значения показателей персонажа (энергия, голод, количество денег, заполненность инвентаря) были вычислены следующие метрики: среднее значение выгодности действий, дисперсия выгодности действий и отношение количества действий «Работа» к общему количеству действий, представленные в табл. 1.

Таблица 1

	Среднее значение выгодности действий	Дисперсия выгодности действий	Отношение количества действий «Работа» к общему количеству действий
Стоимость действия	0,55	0,08	74 %
Личность неигрового персонажа	0,94	0,08	75 %
Вероятностный выбор действий	0,88	0,25	60 %

Среднее значение выгодности действий считалось как среднее арифметическое выгодности у действий, выбранных в качестве итоговых действий. Дисперсия выгодности действий считалась как сумма разниц квадратов среднего арифметического выгодности действий и выгодности каждого действия, деленное на количество действий, выбранных в качестве итоговых. Отношение количества действий «Работа» к общему количеству действий — это результат деления действий с названием «Работа», выбранных в качестве итоговых, на общее количество действий, выбранных в качестве итоговых.

На основе этих данных можно сделать несколько выводов: расширение стоимость действия сильно уменьшает значения выгодности действий, а расширение вероятностный выбор действий позволяет выбирать более разные действия, что можем сказать на основе повышенной дисперсии. Больше всего на реалистичность выбора действий влияет расширение вероятностный выбор действий, так как он увеличивает дисперсию и снижает количество действий работы по сравнению с общим количеством действий.

Сравнение влияния расширений на производительность на мобильных устройствах

На основе данных из файлов, в которых записывалось время, время, затраченное на принятие решения, количество кадров в секунду, название выбранного действия и количество выделенной памяти были вычислены следующие метрики: среднее время принятия решения в секундах, среднее количество кадров в секунду и среднее количество выделенной памяти в мегабайтах, представленные в табл. 2. Для проверки производительности были выбраны несколько режимов работы программы: алгоритм UtilityAI без расширений, с расширением стоимость действия, с расширением личность неигрового персонажа, с расширением вероятностный выбор действий и со всеми расширениями, работающими одновременно. Устройство, на котором производился сбор показателей производительности, обладало следующими характеристиками: операционная система Android 15 версии, 8 Гб оперативной памяти, процессор Dimensity 8100-Ultra и графический процессор Mali-G610 MC6.

Таблица 2

	Среднее время принятия решения, сек.	Среднее количество кадров в секунду	Среднее количество выделенной памяти, Мб
Базовый UtilityAI	1,916	51,8	2,86
Стоимость действия	1,996	51,8	2,88
Личность неигрового персонажа	2,278	51,4	2,93
Вероятностный выбор действий	2,156	51,1	2,91
Все расширения	2,368	50,4	2,97

Среднее время принятия решения считалось как среднее арифметическое показателей времени, затраченного на принятие решения, среднее количество кадров в секунду считалось как среднее арифметическое показателей количества кадров в секунду, среднее количество выделенной памяти считалось как среднее арифметическое показателей количества выделенной памяти.

На основе полученных данных можно сделать вывод о том, что среди расширений самыми нагружающими оказались личность игрового персонажа и вероятностный выбор действий, так как во время их работы в среднем требовалось больше времени для принятия решения, было меньшее количество кадров в секунду и выделялось больше памяти. При всех расширениях, работающий одновременно, хоть показатели и увеличились, но они не сильно отличаются от показателей при работе самых нагружающих расширений по отдельности.

Заключение

В данной статье был приведен обзор алгоритма UtilityAI и обзор и сравнительный анализ расширений данного алгоритма. Расширения алгоритма UtilityAI помогают обеспечивать более реалистичное поведение неигровых персонажей. Также было сделано несколько выводов о работе расширений:

1. Расширение вероятностный выбор действий больше влияет на разнообразный выбор действий.

2. Самые нагружающие устройство расширения — это личность неигрового персонажа и вероятностный выбор действий.

3. Максимальное увеличение количество кадров в секунду составило 2,7 %, а максимальное увеличение выделенной памяти составило 3,8 %, что говорит о низком влиянии расширений на производительность, которая является критически важным параметром для мобильных устройств.

Литература

1. *Миллингтон И.* Artificial Intelligence for Games / И. Миллингтон, Д. Фанж. – Бока-Ратон : CRC Press, 2016. — 895 с.

2. Utility AI: Поведенческие деревья уходят в прошлое? // ProGamer. – URL: https://www.progamer.ru/dev/utility-ai.htm

3. *Марк Д.* Behavioral Mathematics for Game AI / Д. Марк. – Бостон : Course Technology, 2009. – 480 с.

4. *Рабин С.* AI Game Programming Wisdom / С. Рэбин. – Нидем : Charles River Media, 2002. – 704 с.

5. *Дилл К.* Design Patterns for the Configuration of Utility-Based AI / К. Дилл, Ю. Пюрсель, П. Гэррити, Г. Фрагомени. – Орландо : Interservice/Industry Training, Simulation ans Education Conference, 2012. – 12 с.

УДК 004.057.4

СЕМЕЙСТВО ПРОМЫШЛЕННЫХ ПРОТОКОЛОВ MODBUS

К. С. Ухина

Воронежский государственный университет

Аннотация. Статья посвящена промышленным протоколам Modbus RTU и Modbus TCP, являющимся одними из наиболее распространенных стандартов для обмена данными в системах автоматизации технологических процессов. Modbus представляет собой открытый, простой и надежный протокол, обеспечивающий взаимодействие между различными устройствами, такими как программируемые логические контроллеры (ПЛК), датчики, приводы и системы SCADA. В статье рассмотрены основные принципы работы протоколов, их ключевые особенности и детально проанализированы отличия между последовательной (RTU) и сетевой (TCP) реализациями.
Ключевые слова: Modbus RTU, Modbus TCP, АСУ ТП, SCADA, Master, Slave, промышленная сеть, ведущее устройство, ведомое устройство, регистр, дискретный вход, катушка, последовательный порт, TCP/IP.

Введение

В автоматизированных системах управления технологическими процессами (АСУ ТП) критически важна надежная и простая передача данных между устройствами нижнего уровня (датчики, исполнительные механизмы, ПЛК) и системами верхнего уровня (SCADA, MES). Среди множества промышленных протоколов семейство Modbus, разработанное компанией Modicon (ныне Schneider Electric) в 1979 году, остается одним из самых популярных благодаря своей простоте, открытости и широкой поддержке производителями оборудования.

Modbus существует в нескольких версиях, наиболее распространенными из которых являются Modbus RTU (Remote Terminal Unit), работающий по последовательным интерфейсам (RS-485, RS-232), и Modbus TCP/IP, адаптированный для сетей Ethernet. Несмотря на различия в физической и канальной реализации, оба протокола используют единую семантику обмена данными на уровне приложения, что упрощает интеграцию систем.

1. Термины протоколов Modbus

Ведущее устройство (Master) — активное устройство, которое инициирует запросы к ведомым устройствам.

Ведомое устройство (Slave) — пассивное устройство, которое обрабатывает запросы от ведущего устройства и возвращает ответы. Каждое ведомое устройство в сети имеет уникальный адрес (1-247).

Код функции (Function Code) — однобайтовое поле в запросе, которое указывает ведомому устройству, какое действие необходимо выполнить (например, прочитать регистры, записать катушку).

PDU (Protocol Data Unit) — полезная нагрузка пакета Modbus, которая включает код функции и данные. Для Modbus RTU и Modbus TCP PDU идентична.

ADU (Application Data Unit) — полный пакет Modbus. Для Modbus RTU это [Адрес Slave] + [PDU] + [CRC]. Для Modbus TCP это [Заголовок MBAP] + [PDU].

Заголовок MBAP (Modbus Application Protocol Header) — 7-байтовый заголовок в Modbus TCP. [1]

Дискретный вход (Discrete Input) — битовая переменная, доступная только для чтения.

Катушка (Coil) — битовая переменная, доступная для чтения и записи.
Регистр входа (Input Register) — 16-битная переменная, доступная только для чтения.
Регистр хранения (Holding Register) — 16-битная переменная, доступная для чтения и записи.

2. Описание работы Modbus RTU и Modbus TCP

Modbus — это протокол типа «запрос-ответ» (Master-Slave), основанный на следующих принципах:

Архитектура «Ведущий-Ведомый» (Master-Slave) (рис. 1). В сети Modbus всегда есть одно ведущее устройство (Master), которое инициирует все транзакции и запросы обмена данными, иначе клиент, и одно или несколько ведомых устройств (Slave), которые отвечают на запросы, иначе говоря сервер. Ведущее устройство формирует запрос, содержащий адрес ведомого устройства, код функции и данные. Ведомое устройство с соответствующим адресом выполняет запрос и возвращает ответ. [2]

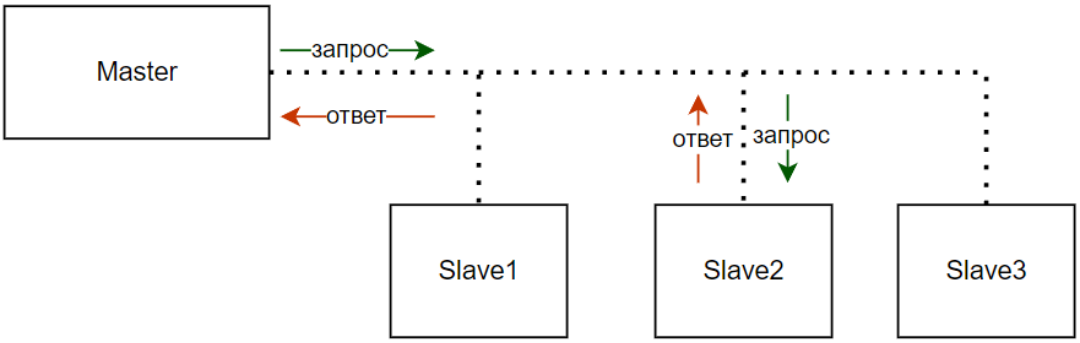

Рис. 1. Архитектура «Ведущий-Ведомый» (Master-Slave)

Простота и открытость. Спецификация Modbus является открытой и не требует лицензионных отчислений, что способствовало его широкому распространению.

Единая модель данных. Несмотря на различные реализации Modbus использует общую модуль данных. Modbus оперирует четырьмя типами данных, которые организованы в таблицы на ведомом устройстве:

– дискретные входы (discrete inputs) — только для чтения, 1 бит. Состояние дискретного датчика (вкл/выкл);

– катушки (coils) — чтение и запись, 1 бит. Управление дискретным выходом (вкл/выкл);

– регистры входов (input registers) — только для чтения, 16 бит. Значение аналогового датчика или счетчика.

– регистры хранения (holding registers) — чтение и запись, 16 бит. Хранение данных конфигурации, уставок и т. д. [2]

2.1. Реализация Modbus RTU

Modbus RTU работает исключительно на последовательных интерфейсах:
– RS-485 — многоточечное соединение (до 32 устройств на сегмент)
– RS-232 — точка-точка (для соединения двух устройств)
Структура сообщения (рис. 2)
– адрес ведомого (1 байт);
– код функции (1 байт);
– данные (N байт);
– CRC-16 (2 байта) [3].

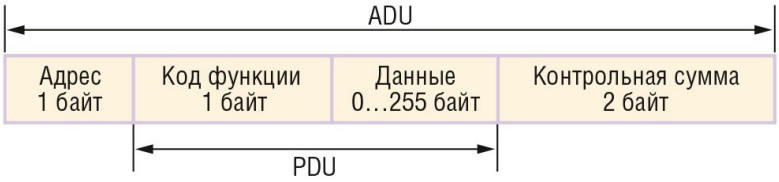

Рис. 2. Структура сообщения протокола Modbus RTU [5]

Фреймирование на основе временных пауз — границы сообщения определяются интервалами тишины:

– минимальная пауза 3.5 символа между сообщениями;

– пауза 1.5 символа между символами внутри сообщения;

Адресация устройств:

– только один уровень адресов (номера устройств 1-247);

– отсутствие понятий IP-адреса, порта, MAC-адреса;

– широковещательный адрес 0 для команд всем устройствам.

Особенности реализации:

Отсутствие встроенной маршрутизации — сообщения не могут покидать локальную шину без шлюза.

Нет фрагментации — размер сообщения ограничен возможностями последовательного порта.

Требует прямого физического подключения — нельзя использовать через стандартные сетевые коммутаторы

Режим передачи данных

– данные передаются только в одном направлении в каждый момент времени;

– требуется механизм управления направлением (RTS/CTS для RS-485);

– все устройства слушают шину, но передает только одно.

Контроль целостности через CRC-16:

– двухбайтная контрольная сумма вычисляется по всему сообщению;

– алгоритм CRC отличен от TCP-checksum;

– при несовпадении CRC сообщение молча отбрасывается. [4]

2.2. Реализация Modbus TCP

Modbus TCP работает поверх TCP/IP стека:

– использует стандартный порт 502;

– работает поверх Ethernet (IEEE 802.3);

– поддерживает маршрутизацию через IP-сети;

– может работать через Wi-Fi, оптоволокно, сотовые сети.

Структура сообщения и заголовки (рис. 3)

MBAP-заголовок (Modbus Application Protocol) заменяет простую адресацию RTU:

– идентификатор транзакции (2 байта) – для сопоставления запросов и ответов;

– идентификатор протокола (2 байта) — всегда 0 для Modbus;

– длина (2 байта) — количество последующих байтов;

– идентификатор устройства (1 байт) — аналог адреса slave в RTU;

– отсутствие CRC — контроль целостности обеспечивается средствами TCP/IP.

Рис. 3. Структура сообщения протокола Modbus TCP [5]

Адресация и сетевая идентификация

– IP-адрес и порт для идентификации устройства в сети;

– Unit Identifier (в MBAP) для выбора конкретного устройства за шлюзом;

– поддержка широковещания через UDP (Modbus UDP).

Особенности реализации

Автоматическое фреймирование — границы сообщения определяются TCP-пакетом.

Встроенная фрагментация — TCP самостоятельно разбивает большие сообщения.

Потоковая передача — данные передаются как непрерывный поток байтов.

3. Применение протоколов Modbus

Modbus RTU и Modbus TCP находят широкое применение в следующих сферах:

Энергетика: сбор данных с электронных счетчиков, мониторинг параметров электрических сетей (напряжение, ток, мощность), управление автоматическими выключателями.

Системы вентиляции и кондиционирования (HVAC): управление климатическими установками, контроль температуры и влажности.

Водоподготовка и водоотведение: мониторинг уровня в резервуарах, управление насосами и задвижками, контроль расхода и давления.

Нефтегазовая промышленность: сбор данных с датчиков давления и температуры на скважинах и трубопроводах.

Производственная автоматизация: связывание разнородного оборудования (ПЛК, частотные преобразователи, панели оператора) в единую систему SCADA. [4]

Заключение

Семейство протоколов Modbus RTU/TCP остается популярным в мире промышленной автоматизации процессов. Такие протоколы имеют преимущество за счет своей простоты, открытости и проверенной временем надежности, что делает их идеальным выбором для множества задач промышленной автоматизации.

Выбор между RTU и TCP определяется конкретными задачами: Modbus RTU идеален для локальных, недорогих и устойчивых к помехам последовательных сетей, в то время как Modbus TCP является стандартом де-факто для интеграции промышленного оборудования в современные информационные системы на базе Ethernet. Также эти протоколы могут использоваться вместе через шлюзы, что позволяет создавать гибридные системы, сочетающие преимущества обоих стандартов.

Литература

1. Modbus Organization. Modbus Application Protocol Specification v1.1b3 : [сайт]. – 2007. – URL: https://modbus.org/specs.php (дата обращения: 03.11.2025).

2. Modbus over serial line specification and implementation guide V1.02 // Modbus.org : [сайт]. – 2006. –URL: https://ftp.owen.ru/ModbusCourse/00_ModbusSpecs/Modbus_over_serial_line_V1_02.pdf (дата обращения: 07.11.2025).

3. The Modbus Protocol in Depth // National Instruments : [сайт]. – 2025. – URL: https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/the-modbus-protocol-in-depth.html (дата обращения: 07.11.2025).

4. Modbus Protocol Overview // Fernhill Software : [сайт]. – 2010. – URL: https://www.fernhillsoftware.com/help/drivers/modbus/modbus-protocol.html (дата обращения: 10.11.2025).

5. Протоколы и сети Modbus и Modbus TCP : [сайт]. – 2010. – URL: https://www.cta.ru/articles/cta/spravochnik/v-zapisnuyu-knizhku-inzhenera/124866/ (дата обращения: 10.11.2025).

УДК 004.891.2

АРХИТЕКТУРА TTS-СИСТЕМЫ ДЛЯ ИИ-РЕКРУТИНГОВОЙ ПЛАТФОРМЫ AIVIANA

И. М. Чжан, М. В. Матвеева

Воронежский государственный университет

Аннотация. В статье рассматривается архитектура и техническая реализация системы синтеза речи (TTS) для рекрутинговой платформы AIVIANA. Представлено микросервисное решение, обеспечивающее естественное голосовое взаимодействие в ходе интервью. Описаны ключевые модули системы: API-шлюз с механизмами безопасности, оркестратор задач с приоритетными очередями, нейросетевое ядро на базе модели XTTS и модуль потоковой передачи данных.
Ключевые слова: синтез речи, TTS-система, рекрутинговая платформа, микросервисная архитектура, голосовое взаимодействие, нейросетевая модель, потоковый синтез, система очередей, задержка генерации, параллельная обработка, оркестрация задач.

Введение

Современный рынок рекрутинга характеризуется возрастающей сложностью процессов подбора персонала, обусловленной большими объемами обработки данных, субъективностью человеческой оценки и значительными временными затратами на проведение первичных собеседований. В условиях цифровой трансформации кадровых процессов особую актуальность приобретают интеллектуальные платформы автоматизации рекрутинга, такие как AIVIANA. Данная платформа представляет собой комплексное решение на основе искусственного интеллекта, предназначенное для оптимизации ключевых этапов подбора персонала.

Одной из фундаментальных задач при создании подобной системы является организация естественного и эффективного голосового взаимодействия между искусственным интеллектом и кандидатом. Система синтеза речи (Text-to-Speech, TTS) выступает критически важным компонентом, обеспечивающим не только преобразование текстовых данных в аудиопоток, но и создание иллюзии естественной коммуникации, сопоставимой по качеству с взаимодействием с человеческим рекрутером.

В настоящей статье рассматриваются архитектурные решения и техническая реализация TTS-системы, разработанной для платформы AIVIANA. Особое внимание уделяется анализу требований, структурным компонентам системы и их интеграции в общую экосистему платформы.

Архитектура и принципы построения TTS-системы платформы AIVIANA

Технические требования и функциональные характеристики

Разработка TTS-системы для платформы AIVIANA требует тщательного анализа функциональных требований, вытекающих из специфики рекрутинговых процессов. Были идентифицированы критические аспекты, определяющие архитектурные решения:

• **коммуникативные требования:** *непрерывность диалога* (непрерывность диалога (система должна поддерживать длительные сессии интервью в 30–60 минут без деградации качества), *контекстуальная адаптивность* (способность изменять интонационную окраску в зависимости от типа вопроса и заданного стиля поведения) и *естественность пауз* (соответствие ритма речи человеческим паттернам для поддержания вовлеченности кандидата);

• **технические ограничения:** *задержка генерации* (максимально допустимая задержка между получением текста и началом аудиовывода не должна превышать 200 мс), *параллельная обработка* (поддержка минимум 5 одновременных интервью-сессий).

Архитектурные компоненты и их взаимодействия

TTS-система платформы AIVIANA реализована на основе принципов микросервисной архитектуры [6], что обеспечивает распределение функциональности между специализированными компонентами. Каждый модуль системы инкапсулирует определенную бизнес-логику и отвечает за конкретный этап обработки запроса - от аутентификации до генерации и доставки аудиоконтента.

Архитектура системы построена вокруг четырех ключевых модулей, взаимодействующих через REST API и gRPC-интерфейсы. Для обеспечения надежности и отказоустойчивости реализованы механизмы асинхронной обработки запросов с использованием очередей сообщений. Система использует реляционную базу данных PostgreSQL [1] для хранения состояния задач и метаданных аудиофайлов, что гарантирует целостность данных и возможность восстановления после сбоев.

Модуль API-шлюза и управления запросами

Модуль API-шлюза и управления запросами служит единой точкой входа в TTS-систему, обеспечивая прием и первичную обработку запросов от основной платформы AIVIANA. Данный компонент реализует два специализированных программных интерфейса: /media/tts/ для генерации стандартного аудио и /media/tts-realtime/ для потокового синтеза речи в режиме реального времени. Последний особенно важен для сценариев интерактивного собеседования — сеансов голосового взаимодействия, в рамках которых речевые реплики искусственного интеллекта формируются динамически непосредственно во время диалога с кандидатом.

Ключевой функцией модуля является обеспечение безопасности системы через механизм валидации API-ключей, который проверяет авторизацию каждого входящего запроса и блокирует несанкционированные попытки доступа к сервису синтеза речи. Механизм реализован с использованием декоратора @api_key_required, который автоматически сверяет переданный клиентом ключ с эталонным значением из настроек системы (settings.API_KEY_TTS).

Архитектурно модуль построен как FastAPI-роутер [3] с раздельной обработкой двух сценариев использования:

• **статический синтез:** для генерации заранее подготовленных вопросов и текстовых материалов;

• **потоковый синтез:** для оперативного формирования речевых реплик в процессе живого диалога.

Оба эндпоинта используют единую модель данных, содержащую текст для синтеза и аутентификационный ключ. После успешной проверки авторизации запрос направляется в соответствующий менеджер очередей tts_filename_manager или tts_filename_manager_realtime, что обеспечивает эффективное разделение потоков обработки и соблюдение строгих требований к временным задержкам для различных типов запросов.

Модуль оркестрации и управления очередями

Данный компонент выполняет функции диспетчера, осуществляющего интеллектуальное распределение задач синтеза между доступными вычислительными ресурсами. Основная очередь предназначена для обработки стандартных задач синтеза с максимальным размером оче-

реди, ограниченным параметром MAXSIZE_QUEUE_TTS. Приоритетная очередь реального времени обслуживает операции интерактивного диалога с ограничением MAXSIZE_QUEUE_TTS_REALTIME.

Модуль реализует алгоритмы планирования, которые учитывают множественные факторы. Система приоритетов предусматривает три уровня важности задач (от 1 — низкий до 3 — высокий), что позволяет обеспечить обслуживание в первую очередь реплик активного диалога с кандидатом. Алгоритм fetch_one_high_prio_over обеспечивает вытесняющую обработку высокоприоритетных задач. Механизмы атомарного захвата задач используют технологию SKIP LOCKED, что гарантирует отсутствие дублирования обработки запросов в распределенной среде. Это особенно важно при работе воркеров на разных серверах.

Управление состоянием задач включает отслеживание полного жизненного цикла. Модуль автоматически отслеживает «зависшие» задачи через механизм requeue_stuck_running и возвращает их в очередь при необходимости. Балансировка нагрузки осуществляется через систему fetch_tasks_for_worker, позволяющую забирать пачки задач с сортировкой по приоритету и времени создания. Это обеспечивает оптимальное использование вычислительных ресурсов и минимизацию времени отклика. Такая архитектура гарантирует соблюдение строгих требований к задержкам генерации речи в интерактивных сценариях, обеспечивая при этом стабильность работы системы в условиях пиковых нагрузок.

Модуль нейросетевого синтеза речи

Основу системы синтеза составляет нейросетевая модель XTTS [4], взятая из открытых источников. Это современная модель преобразования текста в речь, обеспечивающая высокое качество генерации, что позволило достичь показателей задержки генерации менее 200 мс при поддержке множественных параллельных сессий, что подтверждает эффективность выбранного архитектурного подхода [6] и используемых технологических решений [1, 2, 3, 5].

Модель была доработана для оптимальной работы в сценариях рекрутинговых собеседований. Она анализирует пунктуацию текста и автоматически расставляет паузы соответствующей длительности - короткие для запятых и более продолжительные для точек и других знаков завершения предложений.

Модуль хранения и передачи аудио

Модуль реализует распределенную систему управления аудиоданными через gRPC-сервис, обеспечивающий потоковую передачу, хранение и мониторинг файлов. Архитектура поддерживает конкурентную обработку запросов и предоставляет статистику операций в реальном времени.

Для гарантии надежности реализованы механизмы проверки целостности данных, контроля статуса файлов и автоматического восстановления при сбоях. Система обеспечивает минимальную задержку передачи и отказоустойчивость в условиях сетевой нестабильности.

Заключение

Проведенное исследование показало эффективность применения микросервисной архитектуры для построения систем синтеза речи, работающих в условиях строгих требований к задержкам и параллельной обработке.

Разработанная система демонстрирует возможность обеспечения естественного голосового взаимодействия в ходе продолжительных интервью. Ключевыми особенностями решения

являются: раздельная обработка потоковых и статических запросов, система приоритетного планирования задач, механизмы интеллектуального управления паузами и интонацией.

Особенностью реализации стало использование модифицированной модели XTTS в сочетании с оптимизированной системой очередей и механизмами безопасности. Это позволило достичь показателей задержки генерации менее 200 мс при поддержке множественных параллельных сессий.

Литература

1. PostgreSQL 15.2 Documentation [Электронный ресурс]. – URL: https://www.postgresql.org/docs/current/index.html (дата обращения: 10.08.2024).

2. Pydantic Documentation: Data validation using Python type hints [Электронный ресурс]. – URL: https://docs.pydantic.dev/latest/ (дата обращения: 15.03.2024).

3. PyTorch 2.0: Tensors and Dynamic neural networks in Python with strong GPU acceleration [Электронный ресурс]. – URL: https://pytorch.org/docs/stable/index.html (дата обращения: 15.03.2024).

4. SQLAlchemy 2.0 Documentation: The Python SQL Toolkit and Object Relational Mapper [Электронный ресурс]. – URL: https://docs.sqlalchemy.org/en/20/ (дата обращения: 15.08.2025).

5. Microservices Pattern: Language-agnostic API contracts [Электронный ресурс] / C. Richardson. – URL: https://microservices.io/patterns/index.html (дата обращения: 15.03.2024).

6. XTTS: A Voice Cloning and Cross-lingual Voice Conversion Model [Электронный ресурс] / Coqui AI. – URL: https://github.com/coqui-ai/TTS/ (дата обращения: 15.03.2024).

УДК 004.056.53

ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ КОНТЕЙНЕРИЗОВАННЫХ ПРИЛОЖЕНИЙ: АНАЛИЗ И ВНЕДРЕНИЕ ПОЛИТИК БЕЗОПАСНОСТИ В KUBERNETES

К. А. Шанина

Воронежский государственный университет

Аннотация. Kubernetes стал отраслевым стандартом для оркестровки контейнеров на фоне развития облачных вычислений и микросервисной архитектуры. Несмотря на преимущества автоматизации, использование платформы сопряжено с серьёзными рисками безопасности. В данной статье анализируются современные угрозы, включая проблемы конфигурации, уязвимости образов и сетевую изоляцию. Обсуждаются практические аспекты построения системы безопасности, включая контроль доступа, управление секретами и возможность наблюдения. Предлагается набор решений для создания безопасных приложений на Java и Spring Boot в рамках современных DevOps-процессов, объединяющих контроль доступа, мониторинг и автоматизированное тестирование.
Ключевые слова: Kubernetes, безопасность контейнеров, оркестрация, контейнеризация, политики безопасности, микросервисная архитектура, Spring Boot, DevOps, CI/CD, сетевые политики, управление секретами, анализ уязвимостей, мониторинг, наблюдаемость.

Введение

Современная индустрия разработки программного обеспечения характеризуется широким внедрением облачных технологий и микросервисной архитектуры, что привело к тому, что Kubernetes стал доминирующей платформой для оркестровки контейнерных приложений. Гибкость управления, мощные возможности автоматизации и развитая экосистема делают его незаменимым инструментом для построения масштабируемых распределенных систем. Однако стремительное развитие технологий опережает развитие методов обеспечения безопасности, создавая значительные операционные риски для организаций.

Многоуровневая архитектура Kubernetes, которая сочетает в себе сложные механизмы управления, динамическую сетевую инфраструктуру и разнородные рабочие нагрузки, увеличивает возможности для атаки. Типичные сценарии компрометации включают несанкционированный доступ к управлению кластером, горизонтальное перемещение между рабочими нагрузками и потерю конфиденциальных данных, что может привести к значительному финансовому и репутационному ущербу. Актуальность этой проблемы подчеркивается повторяющимися инцидентами безопасности в производственных средах, зафиксированными в отчетах ведущих компаний, занимающихся ИТ-безопасностью. Целью данного исследования является систематический анализ современных угроз безопасности в средах Kubernetes и разработка практических рекомендаций по построению эффективной защиты. Особое внимание уделяется изучению всего жизненного цикла безопасности — от первоначальной настройки кластера до оперативного мониторинга запущенных приложений. Интеграция инструментов безопасности в процессы CI/CD имеет первостепенное значение в соответствии с современными принципами DevSecOps (Development Security Operations).

1. Современные вызовы безопасности в средах Kubernetes

Безопасность кластера Kubernetes — сложная проблема, требующая комплексного подхода к защите нескольких уровней инфраструктуры. Анализ инцидентов показывает, что большинство успешных атак обусловлено сочетанием ошибок конфигурации и недостаточного контроля доступа.

Одной из наиболее критических областей является безопасность конфигурации основных компонентов кластера. API-сервер, как центральный элемент управления, часто становится основной целью атак, особенно если он уязвим к таким недостаткам конфигурации, как разрешение анонимных запросов или отсутствие шифрования передаваемых данных. Не менее значимая угроза исходит от неправильной настройки механизма управления доступом на основе ролей (RBAC) [8], когда избыточные привилегии пользователей или учетных записей служб создают возможности для эскалации привилегий внутри кластера. Распределенное хранилище etcd, содержащее критически важную информацию, такую как секретные данные приложений и данные аутентификации, требует особых мер безопасности, поскольку его компрометация равносильна полному контролю над всей инфраструктурой. Не менее значимые риски существуют для безопасности образов контейнеров и зависимостей приложений. Статистические исследования, включая отчёт Snyk за 2023 год [9], показывают, что значительная доля образов из публичных репозиториев содержит известные уязвимости. Эта проблема усугубляется использованием устаревших базовых образов и недостаточным контролем транзитивных зависимостей. Это создаёт условия для эксплуатации таких уязвимостей, как Log4Shell, в производственных средах.

Сетевой уровень Kubernetes представляет собой ещё один критически важный вектор атак. Отсутствие сегментации сети позволяет злоумышленникам, получившим доступ к поду, перемещаться горизонтально по всему кластеру. Сложность конфигураций сервисных сеток и механизмов балансировки нагрузки часто приводит к уязвимостям, позволяющим перехватывать трафик или обходить аутентификацию [7].

2. Стратегии построения эффективной системы защиты

Обеспечение безопасности сред Kubernetes требует многоуровневой защиты, охватывающей все компоненты инфраструктуры и этапы жизненного цикла приложений. Практический опыт промышленной эксплуатации позволяет разрабатывать эффективные стратегии и методы.

Основополагающим принципом безопасности является строгое соблюдение принципа наименьших привилегий при настройке управления доступом [4, 8]. Это требует не только тщательной настройки политик RBAC, но и внедрения механизмов автоматической проверки соответствия ролей реальным потребностям пользователей и сервисов. Интеграция с корпоративными системами идентификации через протокол OpenID Connect обеспечивает унифицированное управление доступом и надежную аутентификацию пользователей.

Обеспечение сетевой безопасности требует внедрения детальных политик, ограничивающих взаимодействие между компонентами системы. Сетевые политики обеспечивают эффективную сегментацию кластера и изолируют различные среды и приложения друг от друга [8]. Сервисные сети обеспечивают дополнительный уровень защиты, не только шифруя трафик между сервисами, но и предоставляя механизмы детального управления доступом на уровне отдельных операций. Проблемы безопасности образов контейнеров решаются путем внедрения методов безопасной разработки и автоматизированного тестирования. Использование минимального количества базовых образов значительно сокращает поверхность атаки, а регулярное сканирование на уязвимости позволяет своевременно выявлять проблемы. Механизмы политик доступа обеспечивают соблюдение стандартов безопасности организации, автоматически блокируя развертывание неподписанных или уязвимых образов [8].

Управление конфиденциальными данными требует особого подхода, учитывающего ограничения основных механизмов Kubernetes. Шифрование данных в etcd с использованием внешних систем управления ключами, интеграция со специализированными хранилищами секретных данных и строгий контроль доступа к конфиденциальной информации составляют комплексную систему защиты данных [8].

3. Инструментальная поддержка безопасной разработки Java-приложений

Разработка безопасных приложений Java и Spring Boot для использования в средах Kubernetes требует тщательного выбора подходящих инструментов для обеспечения безопасности на каждом этапе жизненного цикла разработки.

Для реализации механизмов аутентификации и авторизации наиболее эффективным решением является использование Spring Security Framework в сочетании с OAuth 2.0 и OpenID Connect [9]. Такой подход не только обеспечивает надежную защиту конечных точек приложения, но и легко интегрируется с корпоративными системами идентификации. Обработка токенов JWT с проверкой подписи и срока действия обеспечивает дополнительный уровень защиты от несанкционированного доступа.

Безопасное хранение конфиденциальных данных достигается благодаря интеграции с внешними системами управления секретами, такими как HashiCorp Vault [8]. Используя spring-cloud-starter-vault-config, секреты можно динамически извлекать во время выполнения, устраняя необходимость их хранения в файлах конфигурации или образах контейнеров. Для дополнительной защиты рекомендуется реализовать автоматическую ротацию секретов и подробное протоколирование их использования. Обеспечение безопасности зависимостей требует внедрения автоматизированных процессов мониторинга. Инструменты статического анализа, такие как OWASP Dependency Check и Snyk, выявляют известные уязвимости в сторонних библиотеках во время сборки приложения [9]. Интеграция этих инструментов в процесс CI/CD обеспечивает автоматическую блокировку сборок с критическими уязвимостями, что соответствует принципам DevSecOps.

Процесс создания образа контейнера должен включать несколько этапов контроля качества. Многоэтапные сборки позволяют минимизировать размер конечного образа за счет удаления инструментов разработки и исходного кода. Статический анализ Dockerfiles с помощью Hadolint выявляет потенциальные проблемы конфигурации, а сканирование собранных образов с помощью Trivy или Grype гарантирует отсутствие известных уязвимостей в базовых слоях.

Эффективный мониторинг и наблюдение за приложениями в Kubernetes требуют комплексной системы сбора и анализа данных. Spring Boot Actuator предоставляет подробные метрики производительности приложений, которые можно экспортировать в Prometheus для последующего анализа и визуализации в Grafana [9]. Структурированное ведение журнала с использованием SLF4J и корреляционных идентификаторов позволяет отслеживать распределенные транзакции и быстро выявлять аномалии в работе системы.

Заключение

В настоящем исследовании представлен комплексный подход к обеспечению безопасности контейнерных приложений в средах Kubernetes. Анализ актуальных угроз показал, что наибольшие риски представляют ошибки конфигурации, уязвимости сетевой изоляции и пробелы в безопасности цепочки поставок программного обеспечения. Разработанные практические рекомендации охватывают все ключевые аспекты безопасности: от настройки базовых компонентов кластера до организации безопасного процесса разработки и развертывания приложений [1–3].

Предлагаемые технологические решения для Java-приложений на Spring Boot обеспечивают эффективную защиту на всех уровнях: аутентификацию и авторизацию через Spring Security, безопасное хранилище секретных данных с помощью HashiCorp Vault, контроль зависимостей с помощью OWASP Dependency Check и Snyk, а также создание безопасных образов контейнеров с интеграцией в процесс CI/CD. Особое значение имеет реализация комплексного мониторинга, который позволяет не только оперативно выявлять инциденты безопасности, но и

проводить проактивный анализ потенциальных уязвимостей [4, 7]. Реализация предлагаемых мер значительно повышает безопасность сред Kubernetes и минимизирует риск компрометации критически важных систем. Дальнейшие исследования в этой области могут быть сосредоточены на разработке специализированных фреймворков для автоматизированного аудита соответствия и создании интеллектуальных систем для обнаружения аномальной активности в режиме реального времени.

Литература

1. *Фаулер М.* Архитектура корпоративных программных приложений. 2-е изд. – М. : Вильямс, 2020. – 576 с.

2. *Эванс Э.* Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. – СПб. : Питер, 2020. – 560 с.

3. *Браун С.* Модульная архитектура: как организовать разработку микросервисов. – М. : Питер, 2019. – 320 с.

4. *Ким Дж., Хамбл Дж., Уэст Дж.* DevOps: как создавать продуктивные и безопасные IT-системы. – М. : Манн, Иванов и Фербер, 2021. – 416 с.

5. *Блюм Б.* Безопасность информации: принципы и практика. – М. : Вильямс, 2017. – 672 с.

6. *Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J.* Borg, Omega, and Kubernetes // ACM Queue. – 2016. – Vol. 14, No 1. – P. 70–93.

7. *Hightower K., Burns B., Beda J.* Kubernetes: Up and Running. 3rd ed. – Sebastopol, CA : O'Reilly Media, 2022. – 352 p.

8. Документация Kubernetes. URL: https://kubernetes.io/ru/docs/home/ (дата обращения: 25.05.2025).

9. Документация Spring Security. URL: https://spring.io/projects/spring-security (дата обращения: 25.05.2025).

УДК 004.627

АЛГОРИТМ КОМПРЕССИИ И ДЕКОМПРЕССИИ ФАЙЛОВ ЖУРНАЛА

В. Ю. Шипилова

Воронежский государственный университет

Аннотация. Статья посвящена разработке алгоритма компрессии и декомпрессии для файлов журнала, характеризующихся высокой степенью избыточности из-за повторяющихся структурных элементов. Предлагаемый метод основан на выделении и замене повторяющихся фрагментов шаблонными конструкциями. Особенностью алгоритма является гибридный подход к поиску шаблонов, сочетающий принципы частотного анализа, позиционного индексирования и кластеризации текстов. Описаны основные этапы алгоритма. Особое внимание уделено оптимизации алгоритма для работы с большими файлами за счёт фрагментации и параллельной обработки. Приводятся рассуждения о влиянии порогового значения на эффективность компрессии, а также результаты экспериментальной проверки на файлах различного объёма.

Ключевые слова: компрессия данных, декомпрессия, файлы журналов, логи, шаблонизация, гибридный алгоритм, алгоритм сжатия, шаблон, журнал событий, частотный анализ, индексирование, кластеризация, пороговое значение, фрагментная обработка, параллельные вычисления.

Введение

Компрессия файлов логов позволяет значительно уменьшить объем занимаемого пространства, что особенно важно при обработке большого количества данных. Разработанный алгоритм эффективно сжимает данные и может быть использован в качестве предобработки перед применением стандартных архиваторов, таких как ZIP, позволяя достичь более высокого уровня сжатия. В отличие от универсальных алгоритмов сжатия, предлагаемое решение учитывает структурные особенности логов — высокую повторяемость определенных шаблонов сообщений при изменении отдельных параметров (IP-адресов, идентификаторов, кодов ошибок).

Целью работы является разработка алгоритма компрессии на основе гибридного метода шаблонизации, сочетающего преимущества нескольких подходов к анализу текста. Для достижения этой цели необходимо решить следующие задачи: реализовать комбинированный метод поиска шаблонов, разработать механизм оптимизации для работы с большими файлами и исследовать влияние пороговых значений на эффективность компрессии.

1. Общий анализ

Для решения поставленной задачи необходимо разработать программу, которая реализует алгоритмы компрессии и декомпрессии файлов журнала. Программа должна обеспечивать возможность преобразования исходного файла журнала в три различных файла: текстовый файл-словарь, текстовый файл-шаблонов и бинарный файл с временными метками и кодами.

Для выполнения задачи требуется разработка следующих ключевых функций:

– компрессия файла журнала с формированием необходимых выходных файлов;

– декомпрессия данных для восстановления исходного файла журнала;

– проверка правильности восстановления исходного файла журнала.

Также необходимо обеспечивать корректную работу функции заполнения файла пользовательскими данными.

В фазу компрессии входят:

– анализ исходных данных и поиск шаблонов;

– оптимизацию найденных шаблонов для повышения точности компрессии;

– кодирование и запись в бинарный файл временных меток и сообщений с использованием словаря и шаблонов.

В фазу декомпрессии входят:

– считывание данных из бинарного файла;

– восстановление записей файла журнала.

Оценка эффективности алгоритма компрессии будет произведена аналитически на основе полученных результатов.

Применённые оптимизации и подходы к разработке алгоритмов позволяют достичь значительного повышения эффективности обработки данных, что делает данный метод компрессии пригодным для работы с большими объёмами файловых журналов.

2. Предполагаемые структуры данных

Для реализации алгоритма сжатия понадобятся следующие структуры:

Структура запись в журнале будет иметь следующие поля:

1. Временная метка — значение даты и времени.

2. Сообщение — строка, являющаяся оставшейся частью после временной метки в записи журнала.

Для поиска шаблонов понадобятся две структуры.

Структура информация о шаблоне имеет следующие поля:

1. Количество — целое число, обозначающее число совпадений шаблона.

2. Переменные слова — набор строк, представляющий изменяемые слова в шаблоне.

Структура хэш ключа будет иметь перегруженный оператор определяющий хэш для ключа.

3. Предполагаемый формат файла журнала

Файл журнала представляет собой последовательность строк, каждая из которых содержит несколько слов, разделённых пробелами или другими разделителями. В начале строки обычно присутствует метка времени, фиксирующая момент записи и позволяющая отслеживать последовательность событий.

Структура записей может варьироваться в зависимости от типа журнала, но обычно включает идентификаторы процессов или пользователей, команды или операции, статусы выполнения и дополнительные параметры, такие как порты, IP-адреса и пути к ресурсам.

Высокая повторяемость отдельных элементов делает такой формат удобным для алгоритмического сжатия.

4. Содержимое и форматы выходных файлов

После обработки журнал преобразуется в несколько выходных файлов, различающихся по структуре и назначению:

1. Файл шаблонов содержит структурированные записи, в которых выделены повторяющиеся элементы. Эти записи представляют собой обработанные строки журнала, где одинаковые части сведены в единые шаблонные конструкции.

2. Файл словаря включает фрагменты данных, которые заменяют изменяемые элементы в шаблонах. В отличие от файла шаблонов, словарь хранит конкретные уникальные значения, которые встречаются в разных местах журнала, например, имена пользователей, порты, IP-адреса.

3. Бинарный файл представляет собой закодированные ссылки на шаблоны и словарные значения. Если алгоритм рассматривается как алгоритм сжатия, этот файл является финальной версией сжатого журнала. Если алгоритм используется для предобработки, бинарный файл служит промежуточным этапом перед дальнейшей компрессией.

4. Архивированный файл (ZIP) является финальной версией после дополнительного архивирования для максимального уменьшения размера.

5. Описание алгоритма

5.1. Этап компрессии

Алгоритм на этапе компрессии анализирует журнал и выполняет выделение повторяющихся фрагментов, заменяя их шаблонами для уменьшения объёма данных и оптимизации хранения. Это позволяет значительно уменьшить размер хранимых данных, сохраняя всю необходимую информацию и структуру записей.

Основные этапы компрессии:

1. Чтение журнала и предобработка данных

– Загружается файл журнала, разбираются строки, извлекаются временные метки и сообщения.

– Дата и время преобразуются в единообразный формат временной метки для дальнейшего упрощения обработки.

2. Выделение шаблонов

– Каждое сообщение анализируется на предмет повторяющихся элементов.

– Используются регулярные выражения для проверки соответствия шаблонам.

– Составляются шаблоны, где изменяемые части, такие как имена пользователей, IP-адреса, номера портов, заменяются переменными.

3. Оптимизация шаблонов

– Шаблоны сортируются по количеству фиксированных слов и алфавитному порядку.

– В шаблонах объединяются схожие фрагменты, минимизируя количество уникальных конструкций и сокращая объём хранимых данных.

4. Создание файлов компрессии

– Файл шаблонов формируется на основе выделенных повторяющихся конструкций сообщений, позволяя сократить их общий объём.

– Файл словаря сохраняет значения переменных, заменённых в шаблонах, обеспечивая возможность их восстановления при декомпрессии.

– Бинарный файл записывает закодированные ссылки на шаблоны и словарные данные.

5.2. Этап декомпрессии

Этап декомпрессии предназначен для полного восстановления исходного журнала из его сжатого представления. Он использует файлы шаблонов, словаря и бинарного журнала, чтобы реконструировать сообщения в их первоначальном виде.

Основные этапы декомпрессии:

1. Чтение данных

– Загружается файл словаря, содержащий соответствия между закодированными значениями и оригинальными словами в журнале.

– Загружается файл шаблонов, где каждой записи сопоставлены шаблонные конструкции.

– Открывается бинарный журнал, содержащий ссылки на шаблоны и словарные значения.

2. Восстановление временных меток

– Из бинарного файла извлекаются временные метки и их изменения.

– Метки корректируются, учитывая изменения, чтобы восстановить точные моменты записи.

3. Реконструкция сообщений

– Если сообщение связано с шаблоном, восстанавливаются его изменяемые части с помощью словаря.

– Если сообщение не использует шаблон, оно восстанавливается исключительно на основе данных словаря.

– Итоговые строки формируются с полным соблюдением исходной структуры.

– Все восстановленные сообщения объединяются в единый текстовый файл, который полностью соответствует исходному журналу.

6. Особенности реализации алгоритма

При разработке алгоритма должны быть применены методы, позволяющие эффективно выявлять повторяющиеся структуры в текстовых данных и адаптировать процесс обработки для работы с большими файлами.

6.1. Методы поиска шаблонов в текстовых данных

При поиске шаблонов в текстовых данных будут использоваться методы, сочетающие в себе разные подходы: частотный анализа, построения индексов и кластеризация текстов.

1. Частотный анализ (подход Apriori). Основная идея Apriori-алгоритма — выявление часто встречающихся элементов и использование этой информации для формирования более сложных структур [2]. В контексте поиска шаблонов в логах применяется аналогичный принцип: сообщения анализируются на предмет повторяющихся последовательностей, а слова или фразы, которые встречаются в разных сообщениях, формируют основу будущих шаблонов.

2. Построение индексов (методы информационного поиска). Информационный поиск базируется на индексировании данных, чтобы быстро находить релевантные совпадения [3]. В алгоритме поиска шаблонов применяется построение позиционного индекса, где каждое слово связывается с его позицией в сообщении. Это позволяет эффективно группировать сообщения с схожими структурами и выявлять закономерности в их построении.

3. Шаблонизация на основе совпадений (кластеризация текстов). Кластеризация текстовых данных позволяет группировать схожие элементы в единую структуру [1]. В алгоритме поиска шаблонов этот метод применяется при анализе пар сообщений, где фиксируются совпадающие слова. На основе этих совпадений строятся шаблоны, заменяя переменные части на универсальные маркеры (_?). Это аналогично принципу кластеризации, где объекты объединяются в группы по сходству.

Таким образом, алгоритм объединит три ключевых подхода: выявление повторяющихся паттернов, характерное для метода Apriori [2], эффективное индексирование позиций, основанное на принципах информационного поиска [3], и кластеризацию на основе совпадений, применяемую в анализе текстов [1].

6.2. Оптимизация алгоритма для работы с большими файлами

Для работы с большим файлом потребуется оптимизация алгоритма, чтобы снизить нагрузку на оперативную память и обеспечить корректное выполнение процесса.

Один из возможных подходов — применение временных файлов для хранения промежуточных данных. В этом случае обработка выполняется по частям, а результаты записываются на диск. Однако у этого метода есть существенный недостаток — высокая нагрузка на систему ввода-вывода. Запись и чтение данных из файлов существенно замедляют выполнение алгоритма, особенно при работе с жёсткими дисками. Такой способ эффективен только при наличии быстрых SSD-дисков, способных минимизировать задержки при доступе к данным.

Другой вариант — обработка файла фрагментами, чтобы избежать превышения допустимого объёма используемой памяти. Однако этот метод требует адаптации порогового значения, которое определяет, какие шаблоны будут выделены. Если пороговое значение превышает размер обрабатываемого фрагмента, шаблоны не будут найдены. В других случаях будет выделено меньшее количество шаблонов, чем при анализе всего файла целиком. Чтобы компенсировать этот эффект, пороговое значение уменьшается пропорционально количеству частей, на которые делится файл. Такой подход гарантирует, что шаблон, который мог бы быть найден при анализе всего массива данных с исходным пороговым значением, будет обнаружен хотя бы в одной из частей при скорректированном пороге. Если же порог изначально небольшой, то он останется прежним.

При этом подходе также решается проблема длительного времени поиска шаблонов в большом файле. Части можно обрабатывать параллельно. В данной программе части обрабатываются группами по восемь частей, внутри каждой группы обработка выполняется параллельно. Количество частей, обрабатываемых параллельно, зависит от числа ядер процессора.

Таким образом, алгоритм будет адаптирован так, чтобы учитывать особенности работы с большими объёмами данных и обеспечивать корректное выделение шаблонов при ограниченных ресурсах памяти.

7. Анализ влияния порогового значения на степень сжатия журнала

Пороговое значение — один из важнейших параметров компрессии журнала, определяющий, какие шаблоны будут выделены. Оно задаёт минимальную частоту появления последовательностей в файле, при которой они становятся шаблонами. Изменение порога влияет на степень сжатия файла, поскольку от него зависит количество найденных повторяющихся структур. При слишком низком пороге алгоритм фиксирует множество малозначимых последовательностей, что увеличивает размер словаря и снижает эффективность сжатия. Напротив, высокий порог исключает полезные паттерны, уменьшая степень компрессии.

Порог не может быть равен 1, так как при таком значении шаблон будет включён в словарь даже при единственном вхождении в файле. В результате алгоритм будет фиксировать случайные и неинформативные последовательности, что значительно увеличит размер словаря и снизит эффективность сжатия. Кроме того, единичные повторения не позволяют выявить устойчивые закономерности, необходимые для эффективной компрессии.

Также порог не должен быть равен 100% от общего числа строк в файле или близким к этому значениям, поскольку в этом случае алгоритм сможет выделить только шаблоны, встречающиеся во всех строках, полностью игнорируя частично повторяющиеся конструкции. Это приведёт к утрате важной информации и сделает алгоритм непригодным для анализа файлов со сложной структурой данных.

8. Эксперименты по сжатию

Эксперименты были проведены на двух файлах: первый размером в 2500 строк, а второй 5000 строк. Размер файлов оценивается в байтах. Значения порога обозначаются как в проценн-

тах от общего числа строк, так и численно. Рассматриваются две цели: максимальное сжатие после компрессии алгоритма и максимальное сжатие архиватором ZIP компрессионных файлов.

Для первого файла был запущен алгоритм начиная с порога 2 (0.08%) до порога 2250 (90%). Полные результаты приведены в табл. 1.

Таблица 1

Результаты для файла с 2500 строк

Пороговые значения	2	3	4	5	7	10	12	25
Пороговые значения (в процентах)	0.08 %	0.12 %	0.16 %	0.2 %	0.3 %	0.4 %	0.5 %	1 %
Размер изначального файла	253314	253314	253314	253314	253314	253314	253314	253314
Размер словаря	10618	10642	10737	10584	10611	10791	10791	10996
Размер бинарного файла	59145	58797	57141	57009	57145	57455	54555	59144
Размер файла шаблонов	11670	10557	5863	5674	5391	5171	5171	3977
Общий размер компрессионных файлов	81433	79996	73741	73267	73147	73417	70517	74117
Размер исходного файла после ZIP	24670	24670	24670	24670	24670	24670	24670	24670
Размер компрессионных файлов после ZIP	23876	23659	22692	22479	22506	22593	22593	22895

Продолжение таблицы 1

Пороговые значения	50	125	250	500	750	1000	1250	1500	1750
Пороговые значения (в процентах)	2 %	5 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %
Размер изначального файла	253314	253314	253314	253314	253314	253314	253314	253314	253314
Размер словаря	11087	11153	11175	11186	11186	11186	11186	11250	11250
Размер бинарного файла	59744	60708	60834	62981	63301	64400	64624	92491	92608
Размер файла шаблонов	3479	2030	1881	1048	978	885	822	755	710
Общий размер компрессионных файлов	74310	73891	73890	75215	75465	76471	76632	104496	104568
Размер исходного файла после ZIP	24670	24670	24670	24670	24670	24670	24670	24670	24670
Размер компрес-сионных файлов после ZIP	22893	22792	22815	22915	23070	23236	23411	23962	23927

Минимальное значение сжатия алгоритмом было достигнуто при пороговом значении 0.5 %. Слишком маленький порог влияет отрицательно на степень сжатия, но в данном аспекте разница небольшая. Сильное увеличение размера файлов следует только после порога в 50 %.

При оценивании алгоритма в качестве предобработки перед применением ZIP, слишком маленький порог становится почти так же неэффективен, как и слишком большой порог. Слишком большим порогом по-прежнему считается порог больше 50 %, а слишком маленьким, который меньше 0.1 %. В этом случае наиболее эффективный порог составил 0.2%.

Для второго файла был запущен алгоритм начиная с порога 5 (0.1 %) и до порога 4500 (90 %). Полные результаты приведены в табл. 2.

Таблица 2

Результаты для файла с 5000 строк

Пороговые значения	5	10	15	20	25	50	100
Пороговые значения (в процентах)	0.1 %	0.2 %	0.3 %	0.4 %	0.5 %	1 %	2 %
Размер изначального файла	515830	515830	515830	515830	515830	515830	515830
Размер словаря	21411	21411	21411	21411	21591	21653	21734
Размер бинарного файла	114423	114423	114423	114423	114815	114551	122739
Размер файла шаблонов	7020	6349	6274	6217	5953	5425	4599
Общий размер компрессионных файлов	142854	142183	142108	142051	142359	141629	149072
Размер исходного файла после ZIP	49208	49208	49208	49208	49208	49208	49208
Размер компрессионных файлов после ZIP	44045	43951	43920	43918	43949	43769	45870

Продолжение таблицы 2

Пороговые значения	250	500	1000	1500	2000	2500	3000	3500
Пороговые значения (в процентах)	5 %	10 %	20 %	30 %	40 %	50 %	60 %	70 %
Размер изначального файла	515830	515830	515830	515830	515830	515830	515830	515830
Размер словаря	21894	21937	21942	21959	21959	21959	22023	22023
Размер бинарного файла	125210	126824	126880	127832	127836	127836	180525	180525
Размер файла шаблонов	3649	2334	1919	1654	1584	1584	1517	1477
Общий размер компрессионных файлов	150753	151095	150741	151445	151379	151379	204065	204025
Размер исходного файла после ZIP	49208	49208	49208	49208	49208	49208	49208	49208
Размер компрессионных файлов после ZIP	45951	45768	45964	45817	45809	45809	46911	46909

Как в случае исследования предыдущего файла, при оценке программы и как самостоятельного компрессионного алгоритма, и как предобработки перед архивированием ZIP порог выше 50 % демонстрирует наихудшую эффективность. Но в этот раз, по сравнению с прошлым, фиксируются два резких скачка размеров компрессионных файлов: после 1 % и после 50 %. Также порог в 1 % обеспечивает здесь наилучшие результаты и в качестве алгоритма компрессии, и в качестве алгоритма предобработки.

Для разных файлов будут разные оптимальные пороги, они будут зависеть не только от количества строк и от структуры самих записей, поэтому невозможно подобрать единый лучший порог для всех файлов. Тем не менее, есть определённые взаимосвязи, которые помогут определить оптимальный порог для случайного файла.

1. Порог не должен быть больше 50 %. Такой порог приведёт к снижению эффективности для всех файлов.

2. Порог не должен быть сильно маленьким, то есть меньше 0.1 % и не может быть равен 1.

3. Оптимальный порог для случайного файла находится в диапазоне от 0.2 % до 1 %.

В файле из 2500 строк с оптимальным значением порога было достигнуто сжатие примерно в 3.5 раза, а экономия при использовании алгоритма в качестве предобработки составила 2191 байт, что составляет 10 % от размера файла при архивации без алгоритма. В файле на 5000 строк было достигнуто сжатие в 3.6 раза, а предобработка экономит 5439 байт, что превышает 11 % от размера заархивированного файла без использования алгоритма.

Заключение

В работе представлен алгоритм компрессии и декомпрессии файлов журнала, основанный на гибридном методе поиска шаблонов. Основные преимущества предложенного решения включают:

1. Высокую эффективность сжатия за счет учета структурных особенностей логов.

2. Адаптивность к большим объемам данных благодаря механизму фрагментации и параллельной обработки.

3. Универсальность гибридного подхода к шаблонизации, позволяющего выявлять разнотипные повторяющиеся структуры.

Литература

1. *Айвазян С. А.* Классификация многомерных наблюдений / С. А. Айвазян, Э. Н. Бежаева, О. В. Староверов. – Москва : Статистика, 1974. – 238 с.

2. Анализ данных и процессов. 3-е изд. / А. Барсегян, М. Куприянов, И. Холод [и др]. – Санкт-Петербург : БХВ-Петербург, 2009. - 512 с. – ISBN 978-5-9775-0368-6.

3. Маннинг К. Введение в информационный поиск / К. Маннинг, П. Рагхаван, Г. Шютце ; пер. с англ. и ред. Д. А. Клюшин, П. И. Браславский. – Москва : Вильямс, 2011. – 528 с.

4. Страуструп Б. Язык программирования C++. Специальное издание / Б. Страуструп ; пер. с англ. – Москва : Бином, 2011. – 1136 с.

УДК 004.657

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ СРАВНЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ SQL-ЗАПРОСОВ

В. В. Юрасова, М. В. Матвеева

Воронежский государственный университет

Аннотация. В статье описывается процесс проектирования веб-приложения, предназначенного для сравнения производительности SQL-запросов в СУБД PostgreSQL версий 14 и 15. Для изоляции сред и работы с разными версиями PostgreSQL применяются Docker-контейнеры. Основные функции системы включают загрузку дампа базы данных, ввод SQL-запроса и сравнение его плана выполнения в разных версиях СУБД. Архитектура приложения разделена на три слоя: представление, домен и источник данных. Серверная часть реализует взаимодействие с Docker и PostgreSQL через JDBC, а клиентская часть построена на HTML, CSS и Thymeleaf. Результатом работы является отчет с планами выполнения запроса и рекомендацией по выбору версии СУБД с лучшей производительностью.
Ключевые слова: веб-приложение, СУБД, SQL-запрос, PostgreSQL, производительность, EXPLAIN ANALYZE.

Введение

Выполнение SQL-команд в любой системе управления базами данных (СУБД) включает несколько этапов. Одной из ключевых фаз является определение плана выполнения запроса, на которой специальный компонент системы — оптимизатор — рассматривает различные варианты планов выполнения запроса. Для оценки каждого варианта используется математическая модель, которая учитывает статистику по хранимым данным. Оптимизатор рассчитывает стоимость выполнения каждого возможного плана и выбирает наиболее эффективный вариант с наименьшими затратами ресурсов.

Важно отметить, что каждая СУБД обладает собственной реализацией оптимизатора и механизмов выполнения отдельных шагов плана запроса. Это означает, что даже одинаковые запросы, выполняемые над одинаковыми структурами данных и одними и теми же наборами данных, могут демонстрировать различную производительность в зависимости от используемой СУБД или её версии. Даже внутри одной и той же СУБД различия реализаций между версиями могут влиять на работу планировщика и итоговую стоимость выполнения запросов.

Сравнение производительности SQL-запросов в разных версиях одной СУБД позволяет оценить влияние обновлений, выбрать подходящую версию СУБД и подготовиться к миграции, минимизируя риски.

Таким образом, в рамках задачи исследования производительности SQL-команд возникла необходимость разработки программного инструмента, способного сравнить производительность SQL-запросов в различных системах управления базами данных и разных версиях одной и той же СУБД.

1. Постановка задачи

Веб-приложение для сравнения производительности SQL-запросов в СУБД PostgreSQL в 14 и 15 версиях, должно предоставлять следующие возможности:

– загрузка дампа базы данных;

– ввод и выполнение пользовательских SQL-запросов;

– отчет по выполнению запроса, содержащий план выполнения запроса в соответствующей версии PostgreSQL, а также рекомендацию СУБД, в которой производительность запроса выше.

2. Анализ существующих решений

На данный момент нет универсального приложения, которое бы позволяло загружать SQL-скрипты и получать отчет со сравнением производительности запроса в разных версиях СУБД. Но существуют инструменты, комбинация которых может помочь осуществить сравнительный анализ производительности SQL-скриптов на разных версиях PostgreSQL.

Поставленную задачу можно решить, используя Docker. Данное приложения позволяет создать несколько контейнеров с разными версиями PostgreSQL и запускать SQL-скрипты в каждом из них. Docker стоит использовать для решения данной задачи, так как он обеспечивает изоляцию приложений, создавая отдельные пространства для их работы. Каждый контейнер использует образы, которые запускаются в изолированном окружении, но при этом работают на общем ядре операционной системы [1]. Для решения задачи важна изоляция версий СУБД, таким образом осуществляется предотвращение возможных конфликтов.

Для получения плана запроса можно использовать команду EXPLAIN ANALYZE, который фактически выполняет запрос и предоставляет реальные метрики времени выполнения, количества возвращаемых строк и информацию о времени инициализации каждого узла плана [2].

Для большинства запросов важна общая стоимость, но в некоторых контекстах, например, когда есть подзапрос в EXISTS, планировщик будет минимизировать стоимость запуска, а не общую стоимость, так как исполнение запроса завершится сразу после получения одной строки [3].

После получения планов запроса необходимо сравнить показатели и выбрать версию СУБД, в которой производительность запроса выше. Сравнение планов приходится делать вручную, что занимает некоторое время.

3. Архитектура приложения

Веб-приложение будет построено по архитектуре с тремя основными слоями: представление, домен, источник данных.

Слой представления выполняет функции отображения информации и интерпретации вводимых пользователем команд с преобразованием их в соответствующие операции в контексте домена (бизнес-логики). В разрабатываемом приложении слой представления делится на клиентскую и серверную части. Клиентская часть реализована при помощи HTML и CSS. Она включает в себя форму для ввода SQL-запросов, загрузки dump-файла базы данных и выбора версий СУБД, отправляя данные на сервер через стандартный механизм HTML-форм. Серверная часть ответственна за прием и маршрутизацию введённых пользователем данных, а также за отображение результата сравнения запросов.

Источник данных обеспечивает взаимодействие с PostgreSQL и Docker. Основными компонентами сервиса являются Docker-контейнеры с разными версиями PostgreSQL и JDBC-драйвер PostgreSQL.

Логика домена (бизнес-логика) ответственен за бизнес-логику, включая сравнение производительности SQL-запросов, а непосредственное выполнение запросов делегируется слою источника данных. Данный слой не зависит от того, как именно выполняются запросы, т. е. отсутствует инфраструктурный код, работа происходит только с интерфейсом слоя данных [4].

4. Интерфейс веб-приложения

При реализации клиентской части приложения использовались:

- язык гипертекстовой разметки — HTML
- язык описания стилей — CSS
- среда разработки Microsoft Visual Studio 2022.

Интерфейс пользователя должен быть простым и отражать основную идею приложения. Он состоит из одной страницы и использует Thymeleaf для динамического отображения данных с сервера. Основная функциональность сосредоточена в двух частях: форма ввода SQL-запроса и блок результатов. Блок результатов отображается, когда сервер возвращает данные, которые динамически отображаются в виде таблицы, каждая строка которой отображает версию PostgreSQL и план выполнения. Ниже таблицы располагается текстовый блок с указанием оптимальной версии. Система обработки ошибок интегрирована непосредственно в форму, сообщения об ошибках выводятся в специальном блоке, который становится видимым при наличии соответствующего атрибута в модели.

При открытии сайта загружается главная страница с полем для ввода SQL-запроса, кнопкой для загрузки файла для восстановления базы данных и кнопкой для сравнения производительности запроса в 14 и 15 версиях СУБД PostgreSQL. Главная страница представлена на рис. 1.

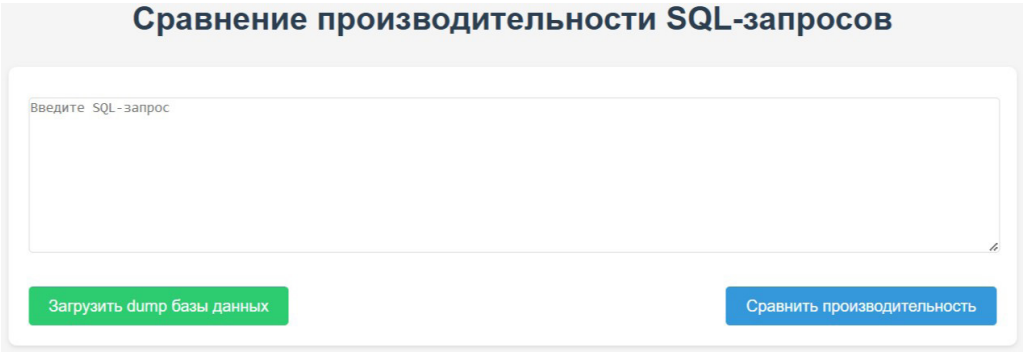

Рис. 1. Главная страница

После ввода SQL-запроса, загрузки dump-файла и нажатия кнопки «Сравнить производительность» появится таблица с версией PostgreSQL и планом выполнения запроса, а под ней — поле с сообщением о версии СУБД, в которой производительность введенного запроса выше. Страница с результатом работы приложения представлена на рис. 2.

5. Проектирование приложения

5.1. Диаграмма классов

Для проектирования был использован язык графического описания UML. На рис. 3. представлена диаграмма классов.

На диаграмме представлены основные классы: QueryController, DatabaseInterfaceImpl, DockerOperationsImpl, ExplainPlanResultDto, QueryResultDto, ComparisonResultDto, ConnectionParamsDto, QueryComparator, PostgresExplainPlanParser и интерфейсы DatabaseInterface и DockerOperationsInterface.

Класс QueryComparator осуществляет сравнение производительности SQL-запросов в разных версиях PostgreSQL, анализируя планы выполнения. Основными критериями сравнения являются: стоимость выполнения запроса (cost), полученная из плана EXPLAIN ANALYZE и время выполнения (Execution Time). В первую очередь выбирается версия PostgreSQL с меньшей стоимостью выполнения. Если значения стоимости равны, сравнение производится по времени выполнения, и оптимальной считается версия, у которой оно меньше. Класс осуществляет валидацию входных данных, т. е. проверку SQL-запроса и размера dump-файла, создает временный файл и передает в слой данных путь к этому файлу. Возвращает результат в виде DTO (ComparisonResultDto), содержащего планы выполнения для 14 и 15 версий PostgreSQL и рекомендацию по выбору оптимальной.

Сравнение производительности SQL-запросов

Введите SQL-запрос:

```
EXPLAIN ANALYZE SELECT b.id_box,
                       COUNT(DISTINCT o.id_client) AS unique_clients,
                       SUM(o.cost_of_the_order) AS total_revenue,
                       AVG(o.cost_of_the_order) AS avg_revenue
FROM ordering o JOIN box b ON o.id_box = b.id_box
WHERE o.cost_of_the_order > 5000
GROUP BY b.id_box
HAVING COUNT(o.id_ordering) > 10
ORDER BY total_revenue DESC
LIMIT 10;
```

Загрузить dump базы данных | Сравнить производительность

Результаты сравнения:

Версия СУБД	План выполнения запроса
PostgreSQL 14	Limit (cost=1926.47..1926.49 rows=10 width=52) (actual time=14.359..14.363 rows=10 loops=1) -> Sort (cost=1926.47..1926.64 rows=67 width=52) (actual time=14.357..14.359 rows=10 loops=1) Sort Key: (sum(o.cost_of_the_order)) DESC Sort Method: top-N heapsort Memory: 26kB -> GroupAggregate (cost=1666.48..1925.02 rows=67 width=52) (actual time=9.557..14.300 rows=200 loops=1) Group Key: b.id_box Filter: (count(o.id_ordering) > 10) -> Sort (cost=1666.48..1709.07 rows=17036 width=16) (actual time=9.482..10.589 rows=17023 loops=1) Sort Key: b.id_box Sort Method: quicksort Memory: 1566kB -> Hash Join (cost=6.50..469.17 rows=17036 width=16) (actual time=0.087..5.178 rows=17023 loops=1) Hash Cond: (o.id_box = b.id_box) -> Seq Scan on ordering o (cost=0.00..417.00 rows=17036 width=16) (actual time=0.027..2.546 rows=17023 loops=1) Filter: (cost_of_the_order > 5000) Rows Removed by Filter: 2977 -> Hash (cost=4.00..4.00 rows=200 width=4) (actual time=0.047..0.047 rows=200 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 16kB -> Seq Scan on box b (cost=0.00..4.00 rows=200 width=4) (actual time=0.005..0.018 rows=200 loops=1) Planning Time: 4.968 ms Execution Time: 14.570 ms
PostgreSQL 15	Limit (cost=1440.48..1440.50 rows=10 width=52) (actual time=12.389..12.391 rows=10 loops=1) -> Sort (cost=1440.48..1440.64 rows=67 width=52) (actual time=12.387..12.389 rows=10 loops=1) Sort Key: (sum(o.cost_of_the_order)) DESC Sort Method: top-N heapsort Memory: 26kB -> GroupAggregate (cost=0.43..1439.03 rows=67 width=52) (actual time=0.199..12.343 rows=200 loops=1) Group Key: b.id_box Filter: (count(o.id_ordering) > 10) -> Merge Join (cost=0.43..1223.98 rows=16964 width=16) (actual time=0.029..10.011 rows=16971 loops=1) Merge Cond: (o.id_box = b.id_box) -> Index Scan using u_ordering on ordering o (cost=0.29..1002.78 rows=16964 width=16) (actual time=0.013..8.055 rows=16971 loops=1) Filter: (cost_of_the_order > 5000) Rows Removed by Filter: 2977 -> Index Only Scan using box_pkey on box b (cost=0.14..8.64 rows=200 width=4) (actual time=0.013..0.063 rows=200 loops=1) Heap Fetches: 200 Planning Time: 0.359 ms Execution Time: 12.456 ms

Оптимальный выбор версии:

Производительность запроса выше в 15 версии

Рис. 2. Результат работы приложения

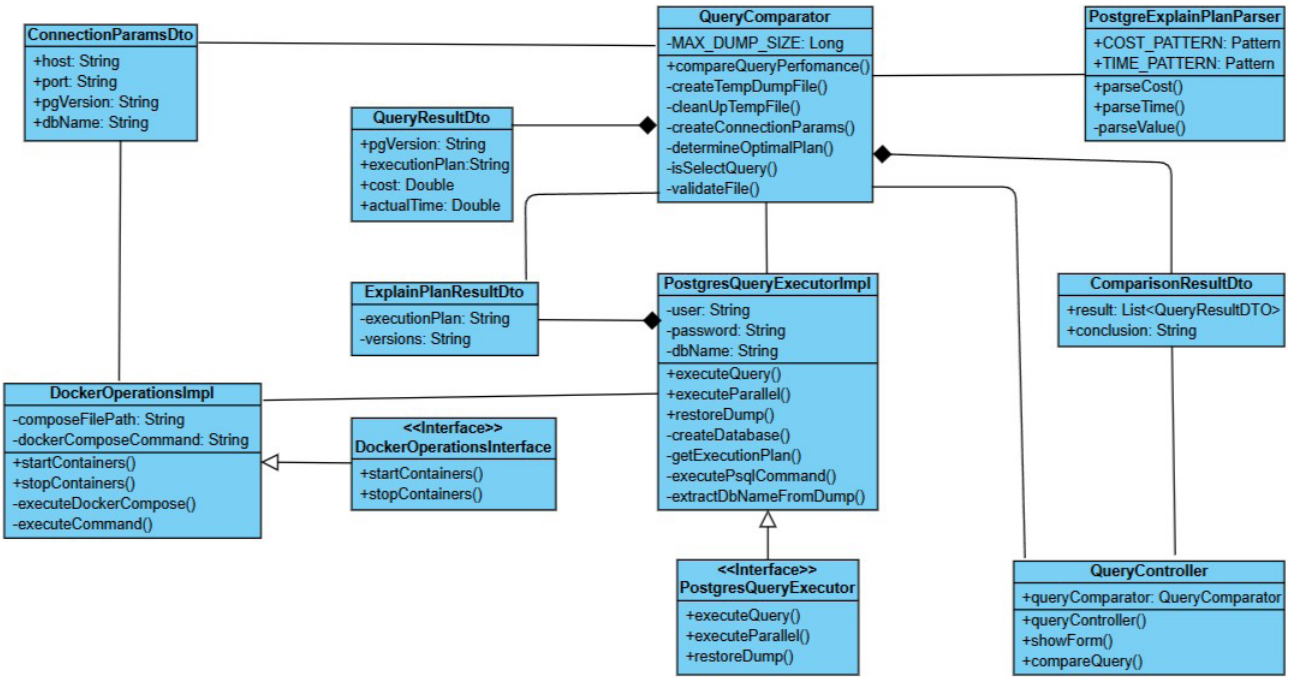

Рис. 3. Диаграмма классов

Класс ConnectionParamsDto содержит поля для подключения к СУБД: имя пользователя и пароль, которые берутся из .env файла, порт, хост, и версию СУБД.

Класс DockerOperationsImpl управляет жизненным циклом Docker-контейнеров с PostgreSQL. Он имеет метод startContainers для запуска контейнеров, а также метод stopContainers для остановки контейнеров. Класс использует ProcessBuilder для выполнения команд Docker CLI, обрабатывает ошибки, оборачивая их в DockerException. Также содержит вспомогательные методы для копирования dump-файлов БД в контейнеры.

Класс PostgresQueryExecutorImpl отвечает за параллельное выполнение SQL-запросов в контейнерах с PostgreSQL. Содержит методы для создания и восстановления базы данных, а также вспомогательные методы для создания и запуска соответствующих SQL-команд. Класс конфигурируется через application.properties. Для работы с базой данных использует JDBC.

QueryController имеет метод showForm() для отображения HTML-страницы start.html для ввода SQL-запроса и загрузки dump-файла. Метод compareQuery() принимает POST-запросы с параметрами: sql-query — SQL-запрос для анализа и database-dump — файл для восстановления базы данных, которые затем передаются в сервис QueryComparator для выполнения сравнения. Класс возвращает результаты работы приложения или ошибку на эту же страницу.

ComparisonResultDto содержит список из QueryResultDto и рекомендацию о выборе версии. Создается в классе QueryComparator и передается в QueryController для отображения пользователю результата работы.

QueryResultDto содержит поля, хранящие план выполнения запроса, версию СУБД, стоимость выполнения запроса и время.

DatabaseInterface определяет метод executeQuery для получения плана выполнения запроса, метод executeParallel обеспечивает параллельное выполнение запроса и собирает результаты в список из ExplainPlanResultDto, метод restoreDump ответственен за восстановление базы данных из dump-файла с помощью ConnectionParamsDto для подключения к СУБД.

DockerOperationsInterface содержит метод startContainers для запуска контейнера с выбранной версией СУБД, а также метод stopContainers для остановки, и последующего удаления контейнера.

5.2. Диаграмма последовательности

На рис. 4. представлена диаграмма последовательности.

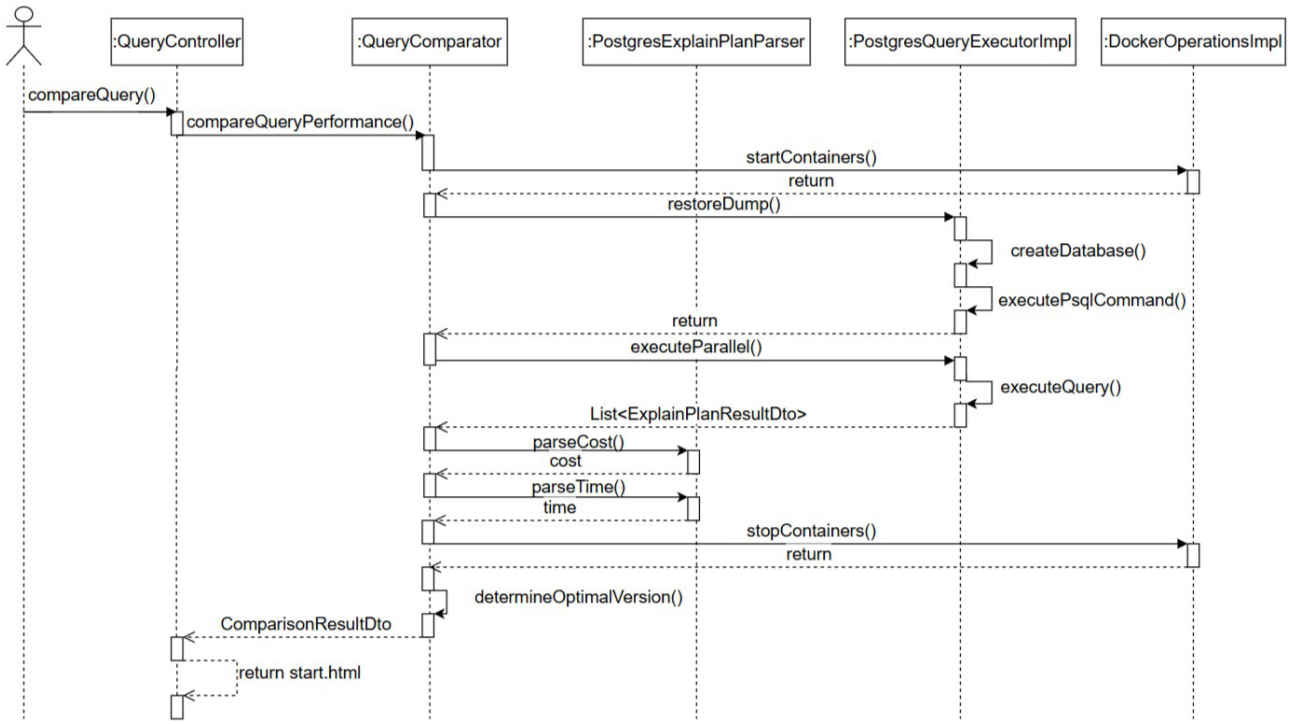

Рис. 4. Диаграмма последовательности

На диаграмме последовательности отражены вызовы основных методов для реализации сравнения производительности SQL-запроса в двух версиях СУБД.

Класс QueryController получает загруженный dump-файл и SQL-запрос, после чего передает эти данные в виде параметров при вызове метода compareQueryPerfomance принадлежащего классу QueryComparator.

Класс QueryComparator в методе compareQueryPerfomance создает список из ConnectionParamsDto и передает его в качестве параметра в метод startContainers класса DockerOperationsImpl, который создает и запускает контейнеры, используя переданные параметры подключения.

После запуска контейнеров вызывается метод restoreDump класса PostgresQueryExecutorImpl. В метод передаются параметры для подключения к СУБД и путь к dump-файлу.

Метод restoreDump вызывает в этом же классе метод для создания базы данных createDatabase, а затем вызывает метод, в котором формируется команда для восстановления базы данных из файла в базу, которую создал метод createDatabase.

Метод executeParallel вызывается после восстановления базы данных. Он используется для параллельного выполнения запроса в разных контейнерах. Параметры метода: SQL-запрос и список ConnectionParamsDto. Метод вызывает executeQuery, передавая ему запрос и ConnectionParamsDto. После выполнения запроса, executeQuery возвращает ExplainPlanResultDto, который содержит версию СУБД и план выполнения запроса.

Класс QueryComparator, получив список ExplainPlanResultDto извлекает из каждого плана стоимость и время его выполнения, осуществляет сравнение и формирует строку с выводом об оптимальной версии СУБД, результатом работы является ComparisonResultDto, содержащий список из QueryResultDto и строку с выводом о рекомендуемой версии.

QueryController получает ComparisonResultDto и обновляет HTML-страницу, для того чтобы на ней отобразилась таблица с результатами сравнения.

Заключение

Результатом работы является проект веб-приложения для сравнения производительности SQL-запросов в двух версиях СУБД PostgreSQL. Были разработаны требования к приложению, спроектирована его архитектура и основные классы. На данный момент приложение находится на стадии реализации. Планируется реализовать клиентскую часть, разработать улучшенную функциональность для серверной части, добавив ещё одну СУБД, а также перенести ресурс на хостинг и ввести его в эксплуатацию.

Литература

1. Документация Docker. – URL: https://docs.docker.com/get-started/docker-overview/#docker-architecture (Дата обращения: 28.02.2025).

2. *Домбровская Г., Новиков Б., Бейликова А.* Оптимизация запросов в PostgreSQL / пер. с англ. Д. А. Беликова. – М. : ДМК Пресс, 2022. – 278 с.

3. Документация PostgreSQL. – URL: https://postgrespro.ru/docs/postgresql/ 17/sql-explain (Дата обращения: 28.02.2025).

4. *Фаулер М.* Архитектура корпоративных программных приложений. : Пер. с англ. – М. : Издательский дом «Вильямс», 2007. – 544 с.