МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ, ПРОГРАММИРОВАНИЕ

УДК 004.65

ИСПОЛЬЗОВАНИЕ ГРАФОВЫХ БАЗ ДАННЫХ ДЛЯ АНАЛИЗА ИНФОРМАЦИОННОЙ СИСТЕМЫ

А. Е. Акулова

Воронежский государственный университет

Аннотация. В рамках этой статьи рассмотрены распространенные способы взаимодействия с пользователем в информационной системе для выявления наиболее эффективного инструмента, приведены средства работы с данными для решения задач, основанных на выявлении пользовательских действий, проанализированы и проиллюстрированы на конкретных примерах алгоритмы (Лувена, Лейдена, Walktrap) поиска сообществ в графах.
Ключевые слова: БД, СУБД, РСУБД, граф, графовая база данных, Walktrap, кластеризация.

Введение

Большинство информационных систем в современном мире стремится к привлечению большего количества пользователей, увеличению экранного времени, затрачиваемого именно их системой. Все стараются внедрить наиболее эффективное средство взаимодействия с пользователем. Но перед тем, как ответить на вопрос «Какое?», стоит узнать «Как?»:

С помощью чего можно добиться заинтересованности пользователей и ее удержания? Какие существуют наиболее действенные средства по достижению этого? Как определить основную черту приложения и основываться именно на ней для привлечения клиентов? Как реализовать наиболее эффективное средство? Ответив на эти вопросы, станет возможным добиться значительных результатов.

1. Обзор способов взаимодействия с пользователем

Помимо внедрения новых функций, современные информационные системы используют такие наиболее эффективные средства взаимодействия с пользователем, как персонализированные уведомления, программа поощрений, основанные на анализе действий пользователя; рекомендательные системы, основанные на интересах, позволяют расширять количество получаемых данных, что увеличивает время использования.

Однако, мало того, чтобы пользователь просто зашел в ваше приложение из-за пришедшего сообщения, важно, чтобы ему это было интересно и хотелось проводить там все больше и больше времени. Исходя из этого появляется понимание важности адаптации приложения под увлечения пользователя.

Рекомендательные системы, помимо удержания заинтересованности пользователя, помогают ему определиться с выбором в огромном количестве данных, поэтому знания о пользователе и контенте, с которым он взаимодействует, способствует получению более точного каталога объектов, основанных на его предпочтениях. При создании приложения, ориентированного на действия пользователя, возможность основываться на связях между ним и остальными элементами приложения, является значительным преимуществом.

2. Обзор графовых баз данных

Использование графовых баз данных оптимально подходит для решения задач, основанных на выявлении пользовательских действий, из-за их преимущества в скорости выполнения

запросов и удобстве работы с самой базой данных, относительно реляционных БД, где данные структурированы в виде таблиц. Внедрение графовых систем также возможно и в реляционной БД, однако, это требует более сложных алгоритмов анализа данных [1] .

Граф — структура, состоящая из узлов и ребер, которые их соединяют. Узлами являются объекты в графе, к примеру, пользователи, новости. Ребра — линии, соединяющие пару вершин, могут быть направленными и ненаправленными и определяют связи между узлами (подписки, сообщения, комментарии).

К основным графовым программным средствам относят нереляционную СУБД Neo4j. Отсутствие необходимости строго фиксировать структуру данных на начальном этапе, гибкость, способность масштабироваться [2] делает данную графовую систему популярным средством с открытым кодом.

Внедрение и использование специальных библиотек в реляционный БД также является одним из средств работы с графами. При работе с РСУБД PostgreSQL возможно использование таких расширений, как pgRouting или AGE (A Graph Extension) предоставляющих возможности для работы с графами, включая маршрутизацию и анализ, а также создание и управление графами с помощью реляционных структур.

Для работы с графами существуют различные алгоритмы выявления сообществ, на основе которых пользователям будет выдаваться больше рекомендаций.

3. Алгоритмы для анализа графов

Алгоритм Лувена для обнаружения сообщества в больших сетях является методом жадной оптимизации и работает по принципу группировки узлов на основе максимизации общей модульности. Вычислительная сложность алгоритма Лувена равна O(n*logn), где n — количество узлов.

1. Каждому узлу присваивается собственное сообщество.

2. Для каждого узла в графе рассматриваются его соседи и определяется насколько изменится модулярность. Под модулярностью понимается метрика для разбиения графа на сообщества, отображающая отношение плотности связей внутри сообщества к плотности связей между группами.

Если перемещение узла в соседний кластер повышает модулярность графа, то узел перемещается в это сообщество.

Процесс повторяется для всех узлов до тех пор, пока не будет достигнуто максимальное улучшение. Это улучшение выполняется для каждого узла и его соседей.

3. Создается новая сеть, узлами которой являются сообщества, найденные на первом этапе.

4. Шаги 2 и 3 повторяются до тех пор, пока не перестанут вноситься изменения и не будет достигнута максимальная модулярность.

Алгоритм Лейдена для кластеризации графа аналогичен методу Лейдена, но дает более качественные решения. Он был разработан как модификация метода Лувена для решения проблем с отключёнными сообществами. Данный алгоритм состоит из трех шагов:

1. Каждый узел является отдельным кластером и имеет свою уникальную метку.

2. На каждом шаге алгоритм рассматривает соседей каждого узла и переносит его в тот кластер, который имеет наибольшее количество соседей.

3. Алгоритм продолжает перераспределение до тех пор, пока каждое сообщество не будет состоять только из одного узла.

4. Алгоритм завершает свою работу, когда перестают происходить значительные изменения в кластерах. В конечном итоге мы получаем множество кластеров, каждый из которых представляет собой группу с похожими интересами.

Алгоритм обнаружения сообществ Walktrap. Его работа основывается на идее того, что более близкие вершины в графе будут иметь более короткие случайные пути между собой.

1. Дан граф $G = (x,u)$. Для каждого узла v в графе генерируется случайное блуждание длины L, которое начинается с узла v и переходит к случайным соседям на каждом шаге. На k-м шаге случайного блуждания для узла u осуществляется переход к одному из соседей этого узла, с учётом вероятности перехода $p_{vu}^{(l)}$.

$$p_{vu}^{(l)} = P(\text{узел } v \xrightarrow{l} u), \tag{1}$$

где $p_{vu}^{(l)}$ зависит от структуры графа и может быть определено как:

$$p_{vu}^{(l)} = \frac{A_{vu}}{d_v}. \tag{2}$$

Здесь

A_{vu} — элемент матрицы смежности графа, равный 1, если между узлами v и u существует ребро, и 0 в противном случае,

d_v — степень вершины v, то есть количество рёбер, инцидентных вершине v.

2. Для каждого узла v и u вычисляется метрика сходства, основанная на их случайных блужданиях. Если $p_v^{(l)}$ — это распределение вероятностей посещения узлов графа при случайном блуждании длины l, то сходство между узлами v и u можно выразить через скалярное произведение их распределений:

$$S(v,u) = \sum_{l=1}^{L} \left(p_v^{(l)} * p_u^{(l)} \right). \tag{3}$$

Здесь $p_v^{(l)}$ и $p_u^{(l)}$ — это векторы вероятностей для узлов v и u соответственно на l-м шаге случайного блуждания. Чем выше значение сходства $S(v,u)$, тем более вероятно, что узлы v и u принадлежат одному сообществу.

3. Для построения матрицы расстояний D, которая отражает степень сходства между всеми парами узлов, можно использовать евклидово расстояние между этими распределениями:

$$D(v,u) = \sqrt{1 - S(v,u)}. \tag{4}$$

Это расстояние характеризует, насколько сильно различаются вероятности случайных блужданий для узлов v и u. Если $D(v,u)$ маленькое, это означает, что узлы v и u принадлежат к одному сообществу.

4. Далее используется агломеративная кластеризация для построения иерархии сообществ. На каждом шаге выбираются два сообщества с минимальным расстоянием (или максимальным сходством), и они объединяются в одно. Пусть C_v и C_u — два сообщества, которые объединяются на шаге k. Тогда на каждом шаге кластеризации для объединения сообществ используется следующая формула для обновления расстояния между объединёнными сообществами:

$$D(C_v, C_u) = \frac{1}{|C_v| * |C_u|} \sum_{v \in C_v} \sum_{u \in C_u} D(v,u). \tag{5}$$

Здесь

$|C_v|$, $|C_u|$ — размеры сообществ C_v и C_u,

$D(v,u)$ — расстояние между узлами v и u.

Это позволяет обновлять расстояния между объединёнными сообществами на каждом шаге кластеризации.

Критерии останова:

– Желаемое количество сообществ k достигнуто.

– Минимальное расстояние между оставшимися сообществами больше порогового значения.

4. Применение алгоритмов

Рассмотрим информационную систему, в которой создание рекомендаций основывается на общих интересах пользователей. Пусть узлами графа будут являться:

Пользователи $(P1, P2, ..., Pn)$; Мероприятия $(E1, E2, ..., En)$; Категории мероприятий $(C1, C2, ..., Cn)$.

Тогда ребра графа будут:

– Мероприятие → Категория (Мероприятия принадлежат к определенным категориям)

– Пользователь → Категория (Пользователь выбирает интересующие его категории)

Алгоритм Лувена.

1. Каждому узлу назначается новая метка (рис. 1).

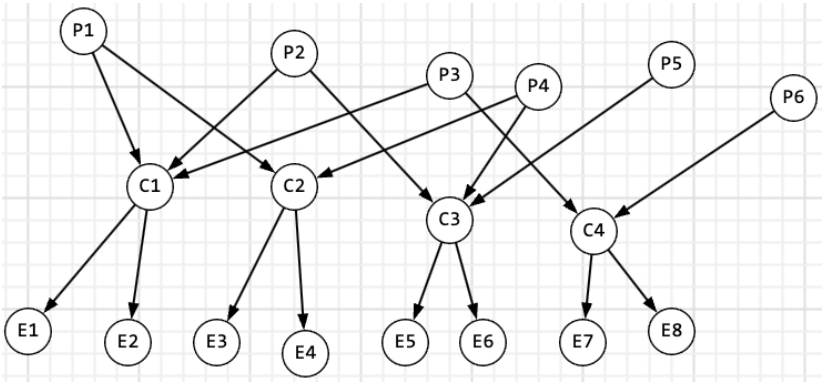


Рис. 1. Граф информационной системы

2. Для каждого узла в графе рассматриваются его соседи и определяется насколько изменится модульность. (рис. 2).

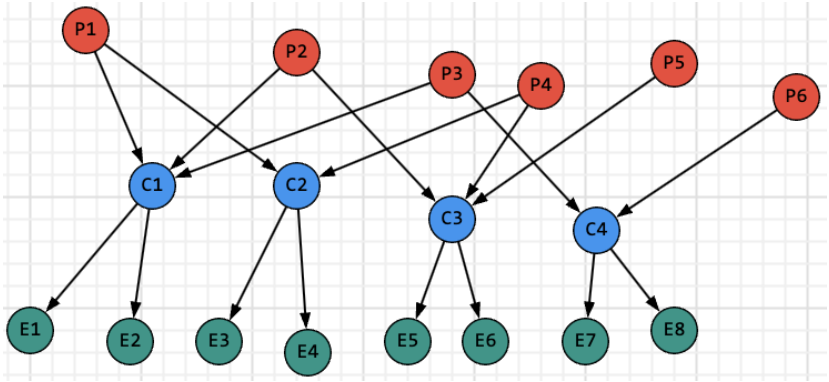


Рис. 2. Рассмотрение узлов графа

3. Перераспределение узлов между сообществами для повышения модулярности графа. Каждый пользователь будет стремиться попасть в то сообщество, где его связи с другими пользователями максимально сильны. Если пользователи выбирают одинаковые категории и схожие мероприятия, то они должны быть в одном сообществе. (рис. 3).

4. Перераспределение продолжается, пока не перестанут вноситься изменения и не будет достигнута максимальная модулярность (рис. 4).

Когда модулярность стабилизируется, мы получаем окончательные сообщества, которые оптимально отражают схожесть интересов пользователей. Таким образом становится возможно создание системы рекомендаций:

Пользователи P1, P2, P3 будут рекомендоваться друг другу так как находятся в одном кластере, интересуясь общей категорией C1.

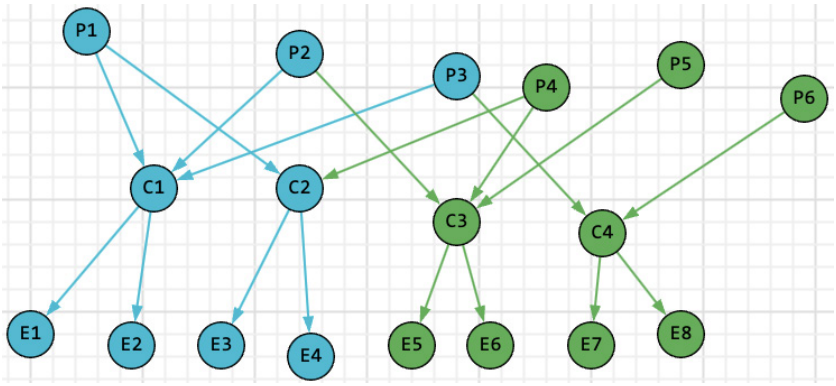


Рис. 3. Перераспределение узлов в графе

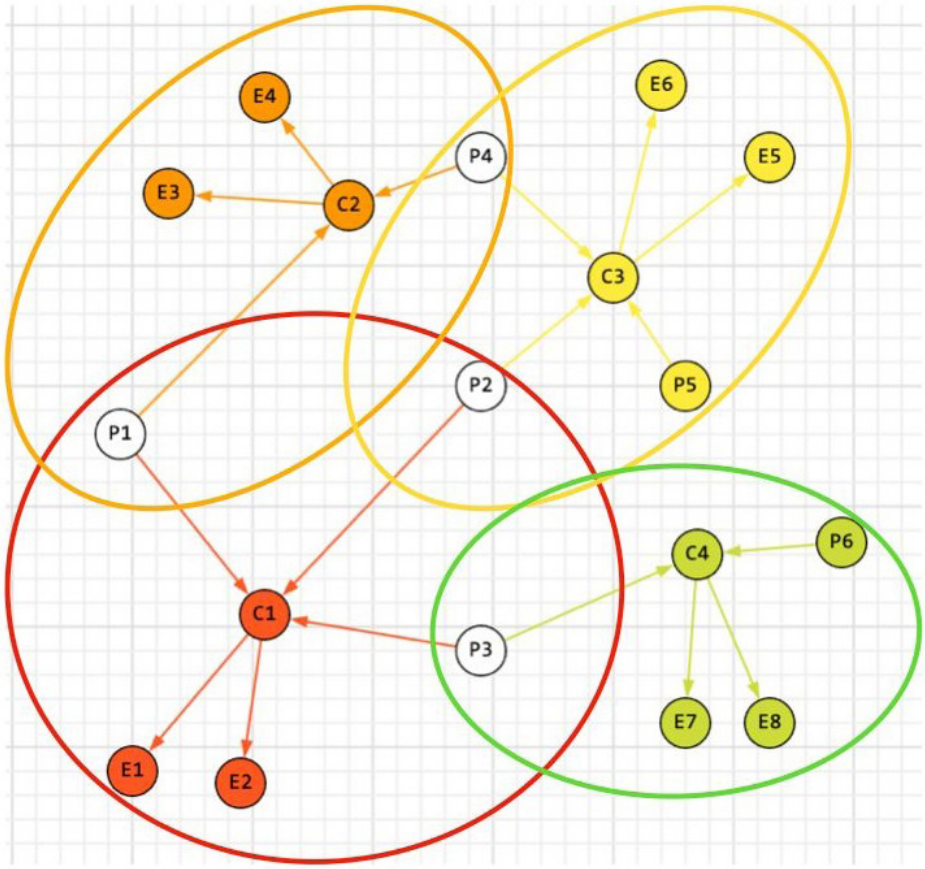


Рис. 4. Вид перераспределенного графа

Пользователи P1, P4 будут рекомендоваться друг другу так как находятся в одном кластере, интересуясь общей категорией C2.

Пользователи P4, P2, P5 будут рекомендоваться друг другу так как находятся в одном кластере, интересуясь общей категорией C3.

Пользователи P6, P3 будут рекомендоваться друг другу так как находятся в одном кластере, интересуясь общей категорией C4.

По итогу, взяв для рассмотрения пользователя P1, увидим, что ему будут рекомендоваться P2, P3, P4.

Рассматривая рекомендационную систему для мероприятий на примере пользователя P1, можно получить такое разбиение: P1 будет получать рекомендации мероприятий из категорий C2, C3, C4, поскольку P1 находится в одном кластере с P2, P3, другой кластер содержит вместе P1 и P4.

Производительность данного алгоритма является преимуществом по сравнению с остальными, рассматриваемыми в этой статье, однако в точности кластеризация при работе с сложными системами он уступает.

Шаги алгоритма Лейдена схожи с алгоритмом Лувена, его преимуществом является то, что сообщества будут гарантированно связаны, что позволит находить более качественные кластеры и сделает это с большей скоростью. Несмотря на то, что реализация усложняется за счет разбиения на более мелкие шаги, оптимизация кластеров на каждом шаге позволяет избежать имеющихся в предыдущем алгоритме проблем. При создании рекомендательной системы, улучшение качества сообществ за счет дополнительной оптимизации способствует получению более точных данных, хоть это и влияет на замедление скорости работы.

Алгоритм Walktrap использует матрицу расстояний между узлами и пытается найти сообщества, минимизируя расстояние между узлами внутри каждого сообщества.

1. Строится граф $G = (x, u)$ с восемнадцатью узлами $X = \{x_1, x_2, ..., x_{18}\}$ и ребрами U, где узлы представляют сущности, а рёбра отражают связи между ними (рис. 5).

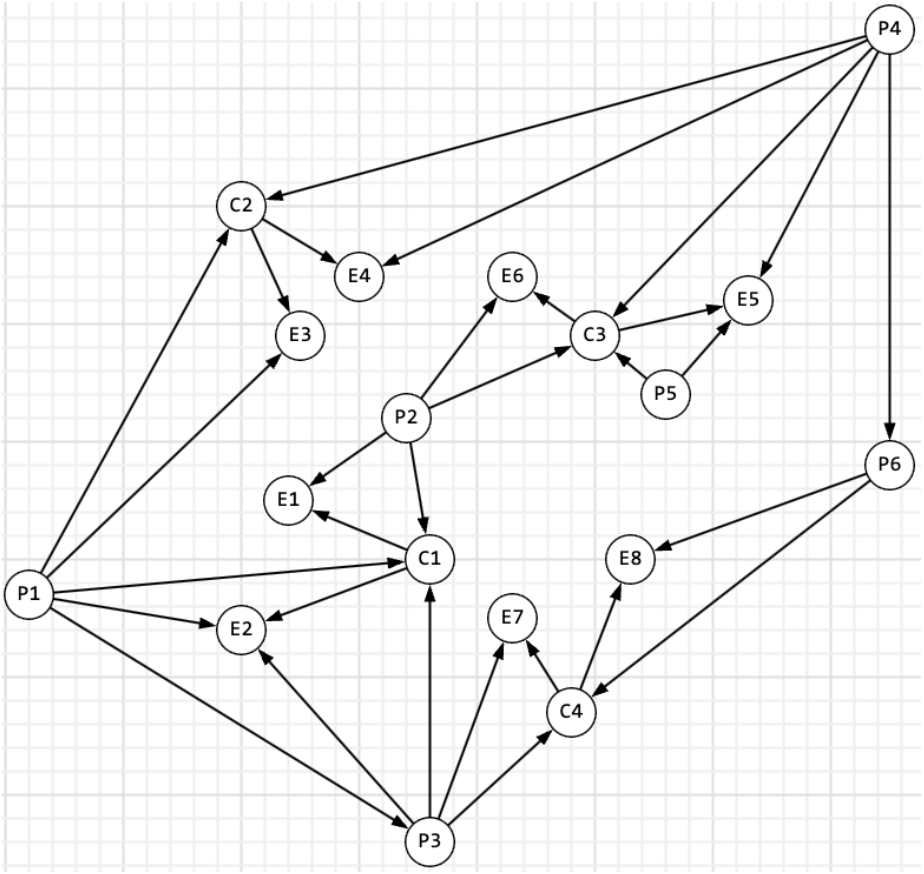


Рис. 5. Изначальный вид графа

Добавим еще связи между пользователями, где ребра будут выражать подписку на пользователя.

2. Граф можно представить в виде матрицы смежности A (6), где элемент $A_{ij} = 1,$ если между узлами есть ребро и 0 в противном случае. Таким образом, узлы будут включать пользователей, мероприятия и категории мероприятий, а ребрами будут являться их взаимодействия.

Если пользователи подписаны друг на друга, то элемент матрицы будет равен 1.

Если пользователь собирается посетить мероприятие, то элемент $A_{ij} = 1.$

Если мероприятие принадлежит к конкретной категории, то элемент будет равен 1.

Если пользователь заинтересован в конкретной категории мероприятий, то взаимосвязь между ними будет определять ребро с индексом 1.

$$A = \begin{pmatrix} 0010000011000001100 \\ 0000001000010010 10 \\ 1000000100001010 01 \\ 0000010001100001 10 \\ 0000000000100000 10 \\ 0000000000000100 01 \\ 0100000000000010 00 \\ 1010000000000010 00 \\ 1000000000000001 00 \\ 0001000000000001 00 \\ 0001100000000001 0 \\ 0100000000000001 0 \\ 0010000000000000 1 \\ 0000010000000000 1 \\ 1110001100000000 0 \\ 1001000011000000 0 \\ 0101100000110000 0 \\ 0010010000001100 0 \end{pmatrix} \quad (6)$$

Рассчитаем степени узлов:

$$d_1 = 5, d_2 = 4, d_3 = 5, d_4 = 4, d_5 = 2, d_6 = 2,$$
$$d_7 = 2, d_8 = 3, d_9 = 2, d_{10} = 2, d_{11} = 3, d_{12} = 2, \quad (7)$$
$$d_{13} = 2, d_{14} = 2, d_{15} = 5, d_{16} = 4, d_{17} = 5, d_{18} = 4$$

Выберем длину случайного блуждания $L = 3$.

3. Для каждого узла графа генерируются случайные блуждания. Пусть v — узел, с которого начинается блуждание. Алгоритм выбирает случайного соседа для каждого шага блуждания.

Рассмотрим в качестве примера узел P1:

$$p_{P1,P3} = \frac{A_{1,3}}{d_1} = \frac{1}{5}, p_{P1,E2} = \frac{A_{1,8}}{d_1} = \frac{1}{5}, p_{P1,E3} = \frac{A_{1,15}}{d_1} = \frac{1}{5}, p_{P1,C1} = \frac{A_{1,15}}{d_1} = \frac{1}{5}, p_{P1,C2} = \frac{A_{1,16}}{d_1} = \frac{1}{5} \quad (8)$$

Если мы пришли в C1, то:

$$p_{C1,P1} = \frac{A_{15,1}}{d_{15}} = \frac{1}{5}, p_{C1,P2} = \frac{A_{15,2}}{d_{15}} = \frac{1}{5}, p_{C1,P3} = \frac{A_{15,3}}{d_{15}} = \frac{1}{5}, p_{C1,E1} = \frac{A_{15,7}}{d_{15}} = \frac{1}{5}, p_{C1,E2} = \frac{A_{15,8}}{d_{15}} = \frac{1}{5} \quad (9)$$

На третьем шаге, если мы пришли в P3, то вероятность перехода в P1, E1, E7, C1, C4 будет:

$$p_{P3,P1} = \frac{A_{3,1}}{d_3} = \frac{1}{5}, p_{P3,E1} = \frac{A_{3,7}}{d_3} = \frac{1}{5}, p_{P3,E7} = \frac{A_{3,13}}{d_3} = \frac{1}{5}, p_{P3,C1} = \frac{A_{3,15}}{d_3} = \frac{1}{5}, p_{P3,C4} = \frac{A_{3,18}}{d_3} = \frac{1}{5} \quad (10)$$

Теперь для P1 случайное блуждание может быть представлено в виде:

На первом шаге $p_{P1,C1} = \frac{1}{5},$ на втором шаге $p_{C1,P3} = \frac{1}{5},$ на третьем — $p_{P3,P1} = \frac{1}{5}.$ Итак, вероятность перехода $\frac{1}{5} * \frac{1}{5} * \frac{1}{5} = \frac{1}{125}$:

Для узла P3 аналогично, но в обратном порядке.

Теперь для P3 случайное блуждание может быть представлено в виде:

На первом шаге $p_{P3,C1} = \frac{1}{5}$, на втором шаге $p_{C1,P1} = \frac{1}{5}$, на третьем — $p_{P1,P3} = \frac{1}{5}$. Итак, вероятность перехода: $\frac{1}{5} * \frac{1}{5} * \frac{1}{5} = \frac{1}{125}$.

4. Вычислим сходство между узлами, используя скалярное произведение их вероятностей на каждом шаге блуждания:

$$S(P_1, P_3) = \left(\frac{1}{5} * \frac{1}{5}\right) + \left(\frac{1}{5} * \frac{1}{5}\right) + \left(\frac{1}{5} * \frac{1}{5}\right) = 0.12. \quad (11)$$

5. Теперь вычислим расстояние между узлами с помощью евклидова расстояния:

$$D(P_1, P_3) = \sqrt{1 - S(P_1, P_3)} = \sqrt{1 - 0.12} = 0.9381. \quad (12)$$

6. Алгомеративная кластеризация. Сначала каждое сообщество будет состоять из одного узла. После чего, объединяем самые близкие сообщества:

Если $D(P_1, P_3)$ будет наименьшим, то P1 и P3 объединяются. И так до тех пор, пока не будет выполнен критерий останова.

Таким образом, применение алгоритма Walktrap к информационной системе позволит выявить сообщества пользователей, у которых совпадают интересы (предпочтения в выборе одинаковых категорий). Также, за счет «случайных блужданий» можно получить неочевидные зависимости, что будет преимуществом при расширении системы. Относительно двух предыдущих, производительность данного алгоритма ниже из-за того, что помимо расчета модулярности, на каждом шаге выбирается случайный путь для определения вероятности того, что два узла окажутся в одном сообществе.

Заключение

Графовые базы данных эффективно работают в системах, основанных на действиях пользователей. Их реализация возможна не только с помощью специализированных платформ и языков, но и систем управления реляционными базами данных. Каждый алгоритм имеет свои особенности и при выборе стоит отталкиваться от требований конкретной системы для настройки рекомендаций. Оптимальным выбором будет являться алгоритм Лейдена, он имеет среднюю производительности, хорошую точность получаемых данных и справляется с сложными сетями. Выбор в пользу алгоритма Лувена стоит делать в случае, если в системе нет сложных зависимостей и получаемым качеством можно пожертвовать в пользу скорости и простоты разработки. Для информационной системы, основными критериями которой являются качество, желание опережать мысли клиентов, касательно предлагаемых рекомендаций, подойдет использование алгоритма Walktrap, чья производительность может снижаться на больших графах, однако он эффективен с разнообразными зависимостями.

Литература

1. *Абрамский М. М.* Сравнительный анализ использования реляционных и графовых баз данных в разработке цифровых образовательных систем. // Вестник НГУ. Серия «Информационные технологии». – 2018. – Т. 16, № 4. – С. 5–12.

2. *Еремеев А. П., Панявин Н. А.* Унификация модели представления данных и преобразование форматов на основе нереляционной СУБД Neo4j. // Программные продукты и системы. – 2022. – Т. 35, № 4. – С. 549–556.

3. *Засядко Г. Е., Карпов А. В.* Проблемы разработки графовых баз данных. // Инженерный вестник Дона, Nel. – 2017. – ivdon.ru/ru/magazine/archive/nly2017/3994

4. *Кочкаров А. А., Калашников Н. В., Кочкаров Р. А.* Сравнительный анализ алгоритмов выявления сообществ в сложных сетевых системах на примере социальных сетей //Программные продукты и системы // Sofware & Systems. – 2020. – Т. 33, № 2. – С. 349–356.

УДК 004.415.2

ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ПОДГОТОВКИ К ЕГЭ ПО ИНФОРМАТИКЕ

Е. Д. Александрова, И. И. Каширская

Воронежский государственный университет

Аннотация. В работе рассматривается проектирование веб-приложения, которое предоставляет выпускникам возможность самостоятельной подготовки к ЕГЭ по информатике. В статье приводится общая архитектура приложения, модель базы данных и структура пользовательского интерфейса.
Ключевые слова: программирование, веб-приложение, веб-разработка, проектирование веб-приложения, ЕГЭ по информатике.

Введение

За последние несколько лет появилась потребность в использовании дистанционного формата обучения. Большая часть граждан отдает предпочтение обучению в онлайн формате [1].

На данный момент можно найти большое количество ресурсов для подготовки к ЕГЭ по информатике. Однако найти среди этого обилия структурированные и понятные источники — трудная задача, особенно для человека, который начинает подготовку с самого начала.

Из вышесказанного вытекает, что создание веб-приложения для самостоятельной подготовки к ЕГЭ по информатике представляет собой перспективное и актуальное направление.

1. Анализ существующих решений

Наиболее популярными ресурсами для самостоятельной подготовки к ЕГЭ по информатике являются: kompege [2], Сайт Константина Полякова [3], Решу ЕГЭ [4], Яндекс Образование [5], ФИПИ [6]. Необходимо проанализировать указанные ресурсы по разработанным критериям. Результат сравнения представлен в табл. 1.

Таблица 1

Сравнительный анализ существующих решений

Критерий сравнения	КЕГЭ	Сайт Константина Полякова	Решу ЕГЭ	Яндекс Образование	ФИПИ
Прикрепленные решения задач	–	–	–	–	–
Возможность решения готовых вариантов	+	+	+	+	–
Автоматическая проверка заданий	+	–	+	+	–
Доступ к теории по номерам	–	+	–	–	–
Возможность просмотра результатов теста через время	–	–	–	–	–

Все вышеперечисленные ресурсы не предусматривают возможности просматривать свои результаты в любой момент использования сайта, а также не реализуют возможность самостоятельного разбора ошибок. Почти все ресурсы не предоставляют материалы для самостоятельного изучения теоретического материала по заданиям, представленным в ЕГЭ.

Отталкиваясь от вышеизложенного, появляется необходимость создания такого веб-приложения, которое бы учитывало все недочеты рассмотренных ресурсов и поддерживало их достоинства.

2. Требования к веб-приложению

Веб-приложение должно предоставлять пользователю следующие возможности:
- регистрация в системе;
- аутентификация в системе;
- просмотр информации о доступных тестах;
- выбор теста;
- прохождение теста с отображением результата, указанием ошибок и правильных ответов;
- просмотр решений заданий, в которых совершена ошибка;
- просмотр теоретического материала по каждому заданию;
- просмотр результатов теста, через некоторое время после его завершения;
- одинаковое отображение графического интерфейса в наиболее распространенных браузерах.

3. Модель данных

Для хранения пользовательской информации, теории по заданиям и практических задач была разработана модель данных, представленная на рис. 1. По данной модели данных была создана база данных [7]. Краткое описание основных таблиц приведено в табл. 2.

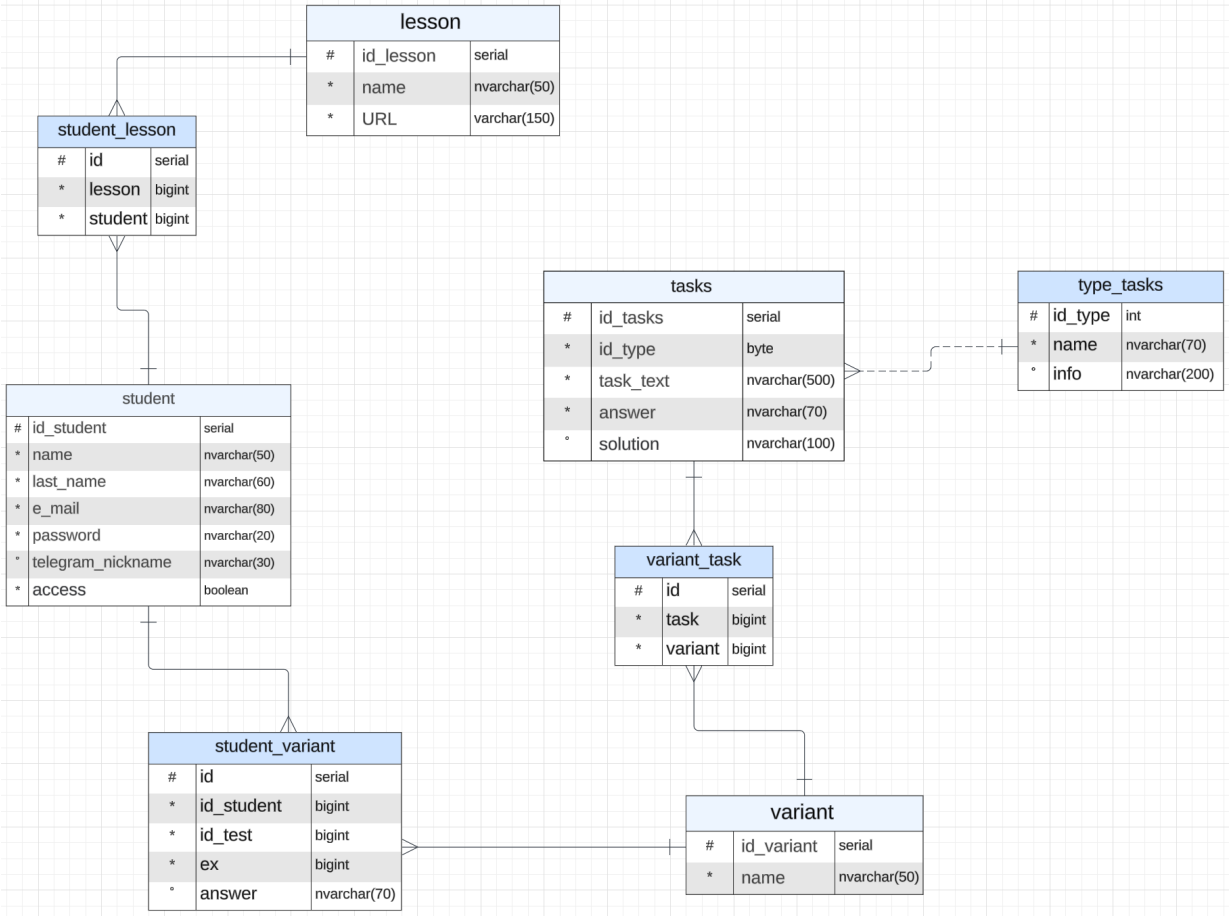


Рис. 1. Модель данных

Таблица 2

Описание основных таблиц базы данных

Таблица	Описание
lesson	Содержит информацию о теоретическом материале
tasks	Содержит информацию о задачах
variant	Содержит информацию о варианте
student	Содержит информацию о пользователях

4. Архитектура веб-приложения

Веб-приложение состоит из 4 основных компонентов:

• клиентская часть — реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него;

• серверная часть — получает и обрабатывает запросы от клиента, формирует ответы, отвечает за взаимодействие с базой данных;

• база данных — хранит информацию о пользователях и метаданные;

• файловый сервер — хранит изображения, используемые в задачах.

Диаграмма компонентов веб-приложения представлена на рис. 2.

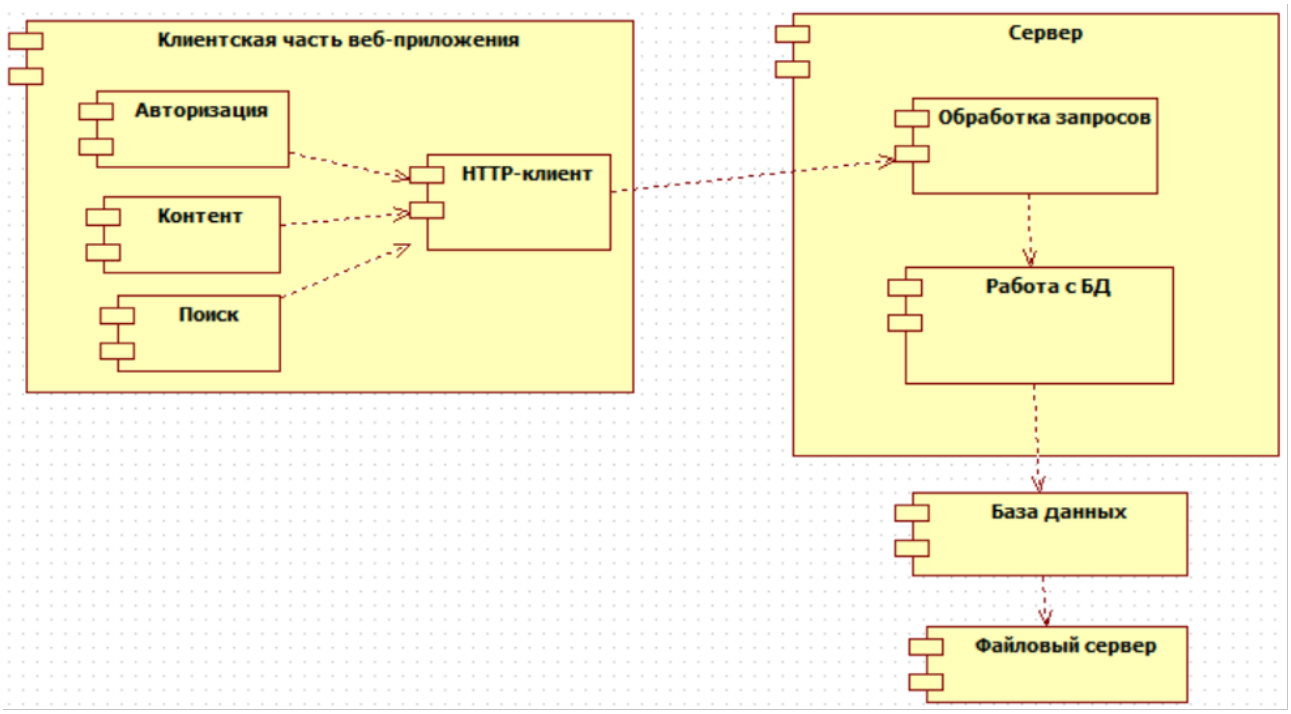


Рис. 2. Диаграмма компонентов веб-приложения

5. Интерфейс пользователя

Интерфейс веб-приложения должен соответствовать принципам юзабилити [8] и обеспечивать необходимую функциональность. Он будет состоять из 12 страниц, включая главную страницу. На рис. 3 представлена схема переходов между страницами проектируемого приложения.

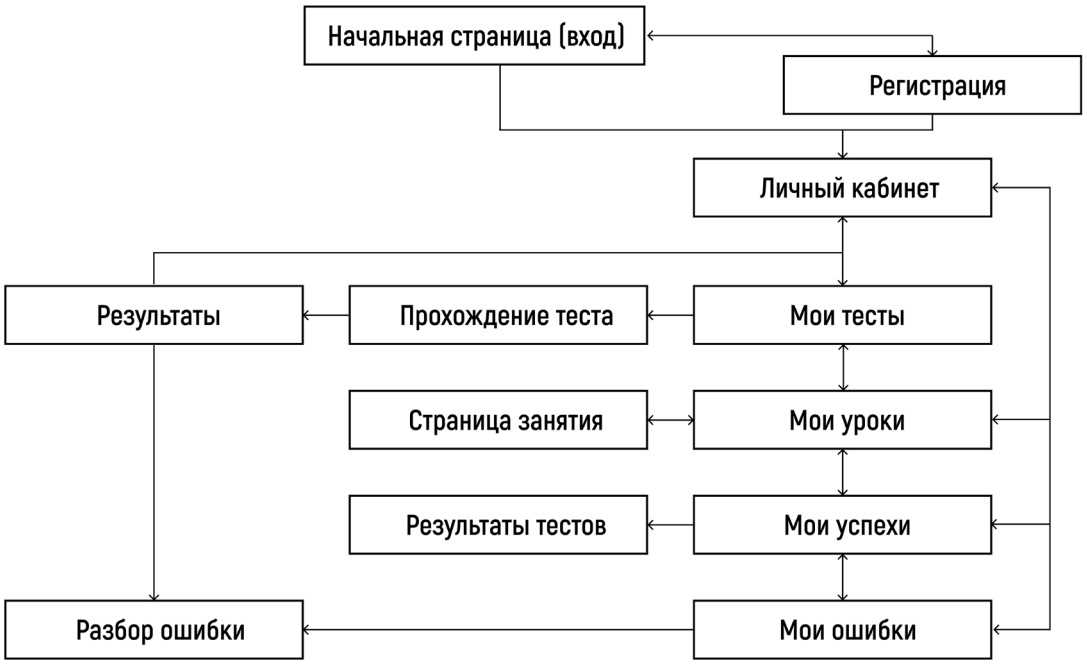


Рис. 3. Схема переходов между страницами

Заключение

Спроектировано веб-приложение, предназначенное для самостоятельной подготовки к ЕГЭ по информатике. Были проанализированы аналогичные решения, их преимущества и недостатки. Разработаны требования, спроектирована архитектура и база данных. Планируется реализовать клиентскую и серверную часть, а также перенести ресурс на хостинг и ввести его в эксплуатацию. Для реализации выбраны язык программирования Python и фреймворк для веб-приложений Django.

Литература

1. Сколько россиян предпочитают дистанционную форму обучения // РИА НОВОСТИ. – URL: https://ria.ru/20240327/opros-1936229091.html (дата обращения 7.10.2024)

2. КЕГЭ – URL: https://kompege.ru (дата обращения: 7.11.2024).

3. Сайт Константина Полякова – URL: https://kpolyakov.spb.ru/school/ege.htm (дата обращения: 7.11.2024).

4. Решу ЕГЭ – URL: https://inf-ege.sdamgia.ru/ (дата обращения: 7.11.2024).

5. Яндекс Образование – URL: https://education.yandex.ru/ege (дата обращения 7.11.24)

6. ФИПИ – URL: https://fipi.ru/navigator-podgotovki/navigator-ege#inf (дата обращения 7.11.24)

7. 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.

8. Нильсен Я. Дизайн Web-страниц. Анализ удобства и простоты использования 50 узлов / Я. Нильсен – Москва : Вильямс, 2002. – 326 с.

УДК 004.75

МЕТОДЫ ПОИСКА ПОДВИЖНЫХ ОБЪЕКТОВ: АНАЛИЗ И РЕАЛИЗАЦИЯ

Д. Д. Ануфриев, В. М. Мельников

Воронежский государственный университет

Аннотация. В статье рассмотрены методы хранения и поиска пространственно-временных данных, востребованные в современных приложениях для работы с подвижными объектами, такими как навигационные и социальные сервисы. На примере индексации пространственных данных для сотен тысяч объектов, передающих координаты в режиме реального времени, исследованы R-деревья и Grid-индексы как основные подходы к решению задачи. Описаны принципы работы, преимущества и ограничения структур R-дерева и его модификаций (R*-дерева и R+-дерева), а также Grid-индексов. Проведено их сравнение по ключевым параметрам, определяющим производительность, точность и масштабируемость методов в задачах, связанных с отслеживанием и быстрым обновлением координат подвижных объектов.

Ключевые слова: подвижные объекты, индексация пространственных данных, R-дерево, R*-дерево, R+-дерево, Grid-индекс, поиск объектов в реальном времени, навигационные системы, геоинформационные системы.

Введение

Современные технологии, такие как смартфоны и GPS, способствуют росту объема пространственно-временных данных. Различные приложения, от служб навигации до социальных сетей, собирают и используют эти данные, что требует эффективных методов хранения, обновления и поиска информации об объектах, движущихся в пространстве. В этом контексте задача состоит в том, чтобы обрабатывать данные о 100 000 подвижных объектах, которые отправляют свои координаты (широта и долгота) в систему в реальном времени.

Для решения задачи необходимо проанализировать методы хранения и поиска пространственных данных, которые могут обеспечивать высокую производительность при частых обновлениях данных о местоположении объектов. Два перспективных подхода, которые часто используются в таких системах, — это R-деревья и Grid-индексы. В статье будут рассмотрены R-дерево и его альтернативы, а также предложена реализация сервиса на основе REST-интерфейса для хранения и поиска данных о подвижных объектах.

1. R-дерево

R-дерево является одной из наиболее широко используемых структур данных для индексации двумерных и многомерных пространственных объектов [1]. В контексте задач, связанных с поиском подвижных объектов, таких как автомобили, находящиеся в движении, R-дерево предоставляет эффективные средства для поиска и обработки запросов на пересечение или нахождение объектов в конкретных областях пространства.

1.1. Принцип работы R-дерева

R-дерево строится на основе иерархии прямоугольных областей, называемых ограничивающими прямоугольниками, которые заключают в себе объекты. Каждый узел дерева хранит несколько таких прямоугольников, которые, в свою очередь, могут содержать либо другие прямоугольники (внутренние узлы), либо непосредственно объекты (листья).

R-дерево представляет собой иерархическую структуру, организованную в виде узлов, каждый из которых имеет свою роль в упорядочении и хранении пространственных данных. На вершине этой структуры располагается корневой узел, содержащий ограничивающие прямоугольники, каждый из которых охватывает определённые группы объектов, задавая общие границы для них. Далее следуют внутренние узлы, которые отвечают за организацию прямоугольников меньшего размера, создавая уровни вложенности внутри общей структуры. На самом нижнем уровне находятся листовые узлы, содержащие конечные объекты, например, автомобили с указанием их координат. Эти листовые узлы формируют базу данных R-дерева, обеспечивая возможность быстрого доступа к объектам в пространственных запросах.

1.2. Алгоритм работы R-дерева

При добавлении объекта в дерево, дерево ищет подходящий листовой узел, который может вместить новый объект. Если такой узел найден, объект добавляется. Если узел переполняется, происходит разбиение узла на два новых, что может вызвать разделение на верхних уровнях дерева.

Для поиска объектов, находящихся в определённой области (например, для определения всех машин, находящихся рядом с пользователем), R-дерево начинает с корневого узла, исключая области, не пересекающиеся с запросом. Процесс продолжается рекурсивно до тех пор, пока не будут найдены все объекты в пределах запрашиваемой области.

1.3. Преимущества и недостатки R-дерева

R-дерево обладает рядом преимуществ, которые делают его особенно полезным для работы с пространственными данными. Одним из ключевых достоинств этой структуры является высокая эффективность при выполнении запросов на пересечение. Это позволяет легко находить все объекты, расположенные в пределах заданной области, что удобно для задач вроде отображения автомобилей, находящихся рядом с пользователем в реальном времени.

Кроме того, R-дерево отличается встроенной балансировкой и гибкостью своей структуры, благодаря чему оно остаётся компактным даже при хранении больших объемов данных [2]. Такой подход не только экономит место, но и поддерживает высокую скорость обработки данных.

Ещё одним важным преимуществом R-дерева является поддержка динамических данных. Оно позволяет легко обновлять информацию, добавляя и удаляя объекты, что особенно ценно для приложений, работающих с подвижными объектами, координаты которых постоянно изменяются. Эти особенности делают R-дерево идеальной структурой для приложений, где требуется оперативное обновление пространственных данных.

R-дерево, несмотря на свои преимущества, имеет и некоторые недостатки, которые могут ограничить его эффективность в ряде задач. Одной из основных проблем являются сложности с частыми обновлениями. Когда объекты перемещаются слишком часто, процесс вставки и удаления данных может существенно замедлить работу дерева, поскольку для каждого обновления необходимо находить подходящий листовой узел и, в некоторых случаях, перестраивать структуру. Это особенно заметно в приложениях с высокой динамикой объектов.

Ещё один недостаток R-дерева связан с его ограниченными возможностями по работе с временными данными [3]. Изначально R-дерево не поддерживает временные запросы, поэтому для обработки пространственно-временных данных, например, поиска автомобилей, которые находились в определённой точке 10 минут назад, его необходимо адаптировать. Это требует усложнения структуры и добавления дополнительных механизмов, что повышает сложность и ресурсоёмкость обработки запросов.

1.4. Примеры использования R-дерева

В навигационных системах R-дерево может использоваться для быстрого поиска ближайших к пользователю объектов (автомобилей или магазинов) [4]. Когда пользователь передвигается, его координаты обновляются, и необходимо быстро находить ближайшие к нему объекты.

В приложениях, где пользователи могут видеть местоположение своих друзей на карте, R-дерево помогает эффективно обрабатывать запросы на поиск друзей вблизи.

2. Варианты оптимизации R-дерева

R*-дерево является улучшенной версией классического R-дерева, специально разработанной для уменьшения перекрытий между ограничивающими прямоугольниками в узлах дерева. Цель такой модификации — сократить количество ложных срабатываний и повысить точность и производительность при выполнении запросов, особенно для поиска ближайших соседей и поиска по диапазону.

Основные отличия R*-дерева от классического R-дерева заключаются в его более продвинутых методах управления и оптимизации данных. Во-первых, R*-дерево ориентировано на уменьшение площади перекрытия между ограничивающими прямоугольниками. Это снижает вероятность обработки лишних узлов, не содержащих искомого объекта, что ускоряет поиск. Такой эффект достигается за счёт улучшенных алгоритмов вставки и удаления элементов.

Во-вторых, при добавлении новых объектов R*-дерево автоматически перестраивает узлы, стремясь минимизировать не только площадь каждого прямоугольника, но и его периметр и центр тяжести [5]. Это позволяет формировать более компактные группы объектов и снижать риск ложных срабатываний, обеспечивая большую точность поиска.

Третьей особенностью является механизм реинсерции объектов. Когда при добавлении элементов узел перегружается, R*-дерево не просто делит узел, но и переносит часть объектов в другие узлы. Такой подход позволяет эффективно перераспределить объекты, дополнительно оптимизируя размещение и уменьшая площади пересечений между прямоугольниками.

R+-дерево представляет собой модификацию классического R-дерева, которая отличается тем, что запрещает перекрытие прямоугольников на одном уровне дерева. Такая структура упрощает процесс поиска, так как каждый объект может находиться только в одном узле на каждом уровне. В результате при поиске уменьшается число узлов, которые нужно проверить, что значительно повышает производительность при выполнении запросов.

R+-дерево обладает рядом особенностей, которые отличают его от R-дерева и R*-дерева [6]. Одной из главных характеристик является отсутствие перекрытий между узлами на каждом уровне. В R+-дерево узлы располагаются так, чтобы минимизировать пересечения, и если объект по своим координатам попадает сразу в несколько узлов, то его разделяют и добавляют в каждый подходящий узел.

Эта особенность связана с механизмом разбиения объектов. Когда объект занимает пространство, пересекающее несколько областей, его копируют в каждый соответствующий узел. Несмотря на увеличение числа дубликатов, такое разбиение упрощает поиск: для нахождения объекта не требуется проверять все пересекающиеся узлы.

Благодаря отсутствию перекрывающихся областей, R+-дерево обеспечивает повышенную производительность поиска. Оно обходится меньшим количеством узлов, что делает его особенно эффективным для задач, требующих высокой скорости поиска, таких как диапазонные запросы.

Ниже приведено сравнение R-дерева, R*-дерева и R+-дерева в табл. 1.

Таблица 1

Сравнительные характеристики деревьев

Метод	Преимущества	Недостатки	Пример применения
R-дерево	Простая реализация, сбалансированная структура	Возможно значительное перекрытие узлов	Общие базы данных, навигация
R*-дерево	Минимизация перекрытий, высокая точность поиска	Сложность реализации, высокие требования к ресурсам	Геоинформационные системы, 3D-моделирование
R+-дерево	Запрещены перекрытия, высокая производительность поиска	Дублирование данных, сложность вставки и обновления	ГИС, системы управления базами данных, навигация

3. Grid-индексы

Grid-индексы представляют собой структуру данных, использующую разделение пространства на ячейки фиксированного размера. Такой подход обеспечивает высокую скорость работы и простоту в реализации, что делает его особенно востребованным

в системах, где необходимо обрабатывать данные в реальном времени с частыми обновлениями, как в логистике, навигации или системах отслеживания подвижных объектов. В отличие от деревьев, Grid-индексы работают с простой сеточной структурой, где пространство делится на равные ячейки, и каждый объект привязывается к одной или нескольким из них.

3.1. Основные принципы работы Grid-индексов

Grid-индексы организуют пространство в виде сетки, разделённой на регулярные ячейки, что обеспечивает высокую скорость поиска и обновления данных [7]. Работа Grid-индексов строится на нескольких этапах.

Первым этапом является деление пространства: вся область, в которой отслеживаются объекты, разбивается на равные ячейки, формируя сетку. После этого каждый объект в зависимости от своих координат привязывается к одной или нескольким ячейкам этой сетки. Такая привязка позволяет легко обновлять положение объектов и эффективно обрабатывать запросы на их поиск.

Для поиска объектов в определённой области система проверяет только те ячейки, которые пересекаются с заданной областью. Это уменьшает объем обрабатываемых данных и ускоряет поиск, так как нет необходимости просматривать всю сетку. При изменении положения объектов их координаты обновляются: объекты перемещаются между ячейками, к которым они привязаны, что позволяет гибко отслеживать изменения их размещения.

3.2. Преимущества и недостатки Grid-индексов

Grid-индексы обладают рядом преимуществ, которые делают их отличным выбором для задач, требующих высокой скорости обработки данных в реальном времени. Прежде всего, их структура отличается простотой: Grid-индексы легко реализовать и использовать, так как они не требуют сложных операций, таких как балансировка дерева, что снижает вычислительную нагрузку по сравнению с более сложными структурами вроде R-деревьев.

Одним из ключевых достоинств Grid-индексов является их высокая скорость работы с данными в реальном времени. Координаты объектов можно обновлять очень быстро: при переме-

щении объекта достаточно просто переназначить его привязку к ячейкам сетки. Grid-индексы также масштабируемы: их можно разделить на множество сегментов и обрабатывать каждый отдельно, что позволяет распределять нагрузку между серверами и работать с большими объемами данных.

Кроме того, Grid-индексы экономны с точки зрения вычислительных затрат на поиск. Чтобы найти объекты в пределах определённой области, достаточно проверить лишь ячейки, которые пересекаются с этой областью. Это позволяет сократить объём обрабатываемых данных и ускорить работу системы.

Однако у Grid-индексов есть и свои недостатки. Например, при неравномерном распределении объектов по пространству плотность некоторых ячеек может значительно превышать другие, что приводит к неравномерному расходу памяти: в областях с высокой плотностью объекты скапливаются, тогда как в других – ячейки могут оставаться пустыми.

Ещё одним минусом является ограниченная точность при поиске ближайших соседей. В отличие от R-деревьев, Grid-индексы менее эффективны в таких запросах, поскольку приходится проверять дополнительные ячейки для получения точного результата, что может замедлить работу в условиях высокой плотности объектов.

Grid-индексы чувствительны к размеру ячеек. Слишком крупные ячейки снижают точность поиска, затрагивая больше областей, чем необходимо. С другой стороны, слишком мелкие ячейки могут повысить затраты памяти и сделать поиск менее эффективным из-за увеличения числа операций.

3.3. Применение Grid-индексов в системах отслеживания подвижных объектов

Grid-индексы могут быть очень полезны для отслеживания транспортных средств в условиях большого города. Рассмотрим, как их использование поможет в задаче мониторинга нескольких тысяч автомобилей с целью быстрой обработки запросов на поиск ближайших объектов и своевременного обновления информации о местоположении.

Во-первых, город можно разбить на сетку, где каждая ячейка представляет собой участок определённого размера, например, 100 на 100 метров. Это позволяет легко разделить городскую территорию на удобные зоны для отслеживания объектов.

Далее, в зависимости от своих координат, каждый автомобиль привязывается к конкретной ячейке сетки. Когда автомобиль пересекает границу между ячейками, его привязка обновляется, и он перемещается в соответствующую ячейку. Такая структура позволяет эффективно отслеживать местоположение объектов.

Если возникает запрос на поиск ближайших автомобилей к определённой точке, например, для подачи такси, система проверяет только объекты в текущей ячейке и соседних, что сокращает объём данных для обработки и ускоряет поиск.

Кроме того, Grid-индексы обеспечивают быстрое обновление информации: при перемещении автомобиля в пределах одной и той же ячейки обновление данных происходит локально, без необходимости перестраивать сетку. Это делает систему эффективной и оптимизированной для обработки частых перемещений, что особенно важно для приложений реального времени.

4. Сравнение методов индексации R-дерева и Grid-индекса

Методы R-дерева и Grid-индекса отличаются по своей структуре и применимости, особенно в задачах пространственной индексации и поиска [8].

R-дерево представляет собой сложную иерархическую структуру данных, которая предназначена для эффективного поиска и индексации объектов в двумерных и многомерных пространствах. Оно организует объекты с помощью иерархии прямоугольных областей, где

каждый узел дерева охватывает множество объектов, создавая вложенную структуру прямоугольников. Это позволяет R-дереву быстро выполнять точные пространственные запросы, включая задачи поиска ближайших объектов и пересечения. Однако, частое обновление данных может увеличивать сложность дерева, что постепенно снижает его производительность.

Grid-индекс, напротив, основан на простом делении пространства на ячейки фиксированного размера, образующих сетку. Объекты привязываются к одной или нескольким ячейкам в зависимости от их положения. Такая структура делает Grid-индекс особенно удобным для задач, требующих частых обновлений в реальном времени, поскольку его просто реализовать и масштабировать. Однако Grid-индекс имеет ограничения: при высокой плотности данных он менее точен для поиска ближайших объектов, и его эффективность снижается при неравномерном распределении объектов по пространству.

Основные преимущества и недостатки обоих подходов приведены в табл. 2.

Таблица 2

Сравнение подходов R-Дерева и Grid-индекса

Метод	Основное преимущество	Основной недостаток	Применение
R-дерево	Эффективный поиск в двумерных и многомерных данных	Может снижать производительность при частых обновлениях перекрытие узлов	Поддержка динамических объектов
Grid-индекс	Простота реализации, высокая скорость обновлений	Неэффективен при неравномерной плотности объектов	Масштабируемая система для объектов в реальном времени

Заключение

R-дерево является мощным инструментом для индексации пространственных данных и их поиска. В задачах, связанных с отслеживанием подвижных объектов, оно обеспечивает быструю обработку запросов на пересечение областей и поиск ближайших объектов. Однако, для систем, где необходимы частые обновления данных, могут потребоваться дополнительные оптимизации или использование альтернативных структур, таких как Grid-индексы.

Grid-индексы являются удобным и эффективным инструментом для задач, где требуется быстрое обновление и поиск объектов в реальном времени. Преимущества, такие как высокая скорость, простота реализации и возможность масштабирования делают их оптимальным выбором для систем отслеживания подвижных объектов.

Литература

1. *Гутмен Д.* Алгоритмы поиска и сортировки в многомерных данных / Д. Гутмен. – Москва : Наука, 2005. – 412 с.

2. *Хан Д.* Пространственно-временные базы данных : теория и практика / Д. Хан, М. Аль-Хабиб. – Москва : Мир, 2015. – 293 с.

3. *Mahmood A., Punni S., Aref W. G.* Spatio-Temporal Access Methods: A Survey (2010–2017) // GeoInformatica. – 2018. – P. 1–36.

4. *Браверман М.* Пространственные индексы и их применение в геоинформационных системах / М. Браверман // Программирование. – 2016. – Т. 40, № 4. – С. 55–67.

5. *Zhao W., Chen L.* An Optimized R*-Tree Structure for High-Density Spatial Data // IEEE Transactions on Knowledge and Data Engineering. – 2017. – Vol. 29, No 3. – P. 670–682.

6. *Sidiropoulos D., Manolopoulos Y.* R+-Tree Performance Analysis in Spatial Database Management Systems / D. Sidiropoulos, Y. Manolopoulos // ACM Transactions on Database Systems. – 2018. – Vol. 43, No 2. – P. 112–128.

7. *Розенфельд А.* Методы пространственно-временной обработки данных / А. Розенфельд, Дж. Хаттон. – СПб. : БХВ-Петербург, 2008. – 374 с.

8. *Лионс Дж., Хикс П.* Структуры данных и алгоритмы для пространственных баз данных / Дж. Лионс, П. Хикс. – Санкт-Петербург : Питер, 2011. – 586 с.

УДК 004.428.4

ПРИЛОЖЕНИЕ ДЛЯ МОДЕЛИРОВАНИЯ БИОСИСТЕМ КАК ИНСТРУМЕНТ ДЛЯ ИССЛЕДОВАНИЯ ПОВЕДЕНИЯ ЭКОСИСТЕМ

В. Ю. Богушов, В. Д. Воронов, Д. В. Каспер, Р. В. Скворцов

Воронежский государственный университет

Аннотация. В работе представлено приложение для моделирования биосистем, разработанное с использованием языка программирования Rust и современных технологий визуализации. Основное внимание уделено моделированию роевых насекомых с использованием муравьиного алгоритма, позволяющего воспроизводить коллективное поведение колонии. Приложение также обладает инструментами для задания параметров, визуализации данных и проведения анализа взаимодействий в экосистемах. Особенности реализации обеспечивают высокую производительность и кроссплатформенность, что делает его полезным инструментом для исследования и прогнозирования экологических процессов.

Ключевые слова: моделирование экосистем, биосистемы, муравьиный алгоритм, Rust, феромоны, визуализация данных, роевые насекомые, многозадачность, Canvas API, симуляция, поведение экосистем, экология, биоинформатика.

Введение

Биосистемы представляют собой сложные и многокомпонентные системы, состоящие из живых организмов и их взаимодействий с окружающей средой. Они охватывают разные уровни организации: от клеточного и органного до экосистемного и биосферного. Ключевыми особенностями таких систем являются их сложная иерархическая структура, высокая степень взаимозависимости компонентов и способность реагировать на внешние изменения, сохраняя функциональную целостность, что делает их важным объектом исследования.

Изучение биосистем имеет фундаментальное значение для понимания основных причин эволюции живых существ. Также изучение таких систем позволяет предсказывать динамику экосистем, разрабатывать меры по их сохранению и создавать искусственные среды, обеспечивающие выживание исчезающих видов.

1. Постановка задачи

Одной из актуальных задач современной экологии и биоинформатики является создание программных средств для моделирования и анализа экосистем.

Необходимо разработать программное решение, способное моделировать поведение экосистем в заданных пользователем рамках и позволяющее в удобной визуальной форме отображать данные. К основным требованиям к такой программе относятся:

- Возможность задавать параметры экосистемы, такие как начальные условия и внешние воздействия на компоненты этой системы.
- Возможность предоставлять пользователю графических средства визуализации данных, таких как анимации жизнедеятельности экосистемы и статистические графики, для наблюдения за развитием экосистемы во времени.

Такая система должна обладать функционалом, благодаря которому можно будет воспроизводить сложные экологические процессы, прогнозировать развитие экосистем, а также тестировать выдвинутые гипотезы, о поведении биосистем.

2. Реализация модели с помощью языка программирования Rust

В данном разделе представлена реализация модели для симуляции биосистемы, с фокусом на роевых насекомых. В качестве примера была выбрана модель муравьев. Для моделирования этой системы, использовался язык программирования Rust.

Этот язык был выбран потому, что Rust предоставляет высокий уровень контроля над памятью при сохранении безопасности, что критично при работе с моделями, где требуется высокая производительность и минимальные накладные расходы. Также, Rust сочетает в себе удобство работы с высокоуровневыми абстракциями и низкоуровневое управление ресурсами, что позволяет использовать многозадачность и многопоточность для повышения производительности, при этом гарантируя безопасность данных и предотвращая такие ошибки, как гонки данных или утечки памяти.

Для точного и быстрого расчета столкновений между объектами биосистемы был использован алгоритм Гилберта — Джонсона — Кирти (GJK), который применяет метод симплекса для проверки пересечения двух объектов.

Сам алгоритм начинается с построения пустого симплекса, который затем итеративно ищет новые направления, добавляя точки в симплекс, пока не будет найдено пересечение. Но для повышения скорости вычислений было принято решение на первом этапе проводить проверки пересечения описанных окружностей объектов. И в случае успешного пересечения окружностей применять полный алгоритм GJK. Такой подход позволяет значительно сократить вычислительные ресурсы.

Для повышения производительности программы также была использована многозадачность. Основной поток приложения выполняет симуляцию биосистемы, в то время как дополнительный поток занимается подготовкой и передачей данных о системе через веб-сокеты для дальнейшей их обработки. Этот подход позволяет избежать блокировки основного потока, улучшая общую производительность программы.

Также важным элементом оптимизации является механизм передачи данных только при активном соединении, что позволяет уменьшить накладные расходы.

2.1. Муравьиный алгоритм

Для моделирования поведения объектов в биосистеме был использован Муравьиный алгоритм. Этот алгоритм является одним из методов поиска оптимальных решений, который основан на поведении реальных муравьёв в природе. Муравьиный алгоритм базируется на способности муравьев находить кратчайшие пути к источникам пищи, используя феромоны, которые они оставляют на пути. Система феромонов в алгоритме помогает оптимизировать поиск путей и решений путем усиления или ослабления следов муравьёв.

В контексте симуляции муравьёв, алгоритм используется для моделирования коллективного поведения, взаимодействий между муравьями и принятия решений на основе локальной информации. В данной модели муравьи «общаются» между собой, оставляя феромоны на виртуальной среде, что позволяет им адаптироваться к изменениям в экосистеме и оптимизировать свои действия в зависимости от текущих условий. Такой подход особенно эффективен для моделирования роевых насекомых, где индивидуальное поведение определяется взаимосвязью с окружающими членами группы.

При каждом шаге муравей начинает с того, что выбирает путь на основе вероятности, который зависит от следующих двух факторов: длины пути и интенсивности феромонов (чем больше феромонов на пути, тем более вероятен выбор этого пути). То есть, муравьем с большей вероятностью будет выбран тот путь, который короче и который имеет больше феромонов.

В смоделированной в этой программе симуляции количество феромонов, оставляемых муравьями, изменяется во времени и зависит от следующих факторов:

– Каждый раз, когда муравей проходит по пути, он оставляет базовый уровень феромонов.

– Феромоны имеют тенденцию исчезать со временем. Это моделирует естественное испарение феромонов в реальном мире.

– Когда муравей достигает ресурса, например, пищи, он оставляет больше феромонов на пути, который привёл его к этому ресурсу.

Также для приближения нашей модели к реальной колонии, был добавлен параметр стадности, который регулирует насколько сильно феромоны влияют на поведение других муравьёв. И чем выше будет значение стадности, тем сильнее будет влияние феромонов, оставленных другими муравьями.

Таким образом, феромоны в симуляции функционируют как средство передачи информации между муравьями: чем больше феромонов на пути, тем выше вероятность, что другие муравьи выберут тот же маршрут. Этот процесс способствует поиску кратчайших путей и эффективному распределению ресурсов в экосистеме. Моделирование феромонов в симуляции позволяет не только учитывать их природные свойства, но и отражать адаптивное поведение муравьёв, которое изменяется в зависимости от текущих условий в экосистеме.

3. Технологии для удобной визуализации и использования приложения

Приложение для моделирования биологических систем состоит из трёх взаимосвязанных компонентов: симуляции, основного сервера и клиентского приложения, что позволяет моделировать различные виды биосистем.

Приложение разработано для работы в браузерной среде, что обеспечивает его кроссплатформенность. Благодаря этому приложение можно запускать на любой операционной системе и в любом современном браузере. Основным языком разработки был выбран JavaScript.

Для успешного запуска моделирования пользователю необходимо подготовить карту. Для этого был разработан редактор (рис. 1), обеспечивающий удобное создание ландшафта. После

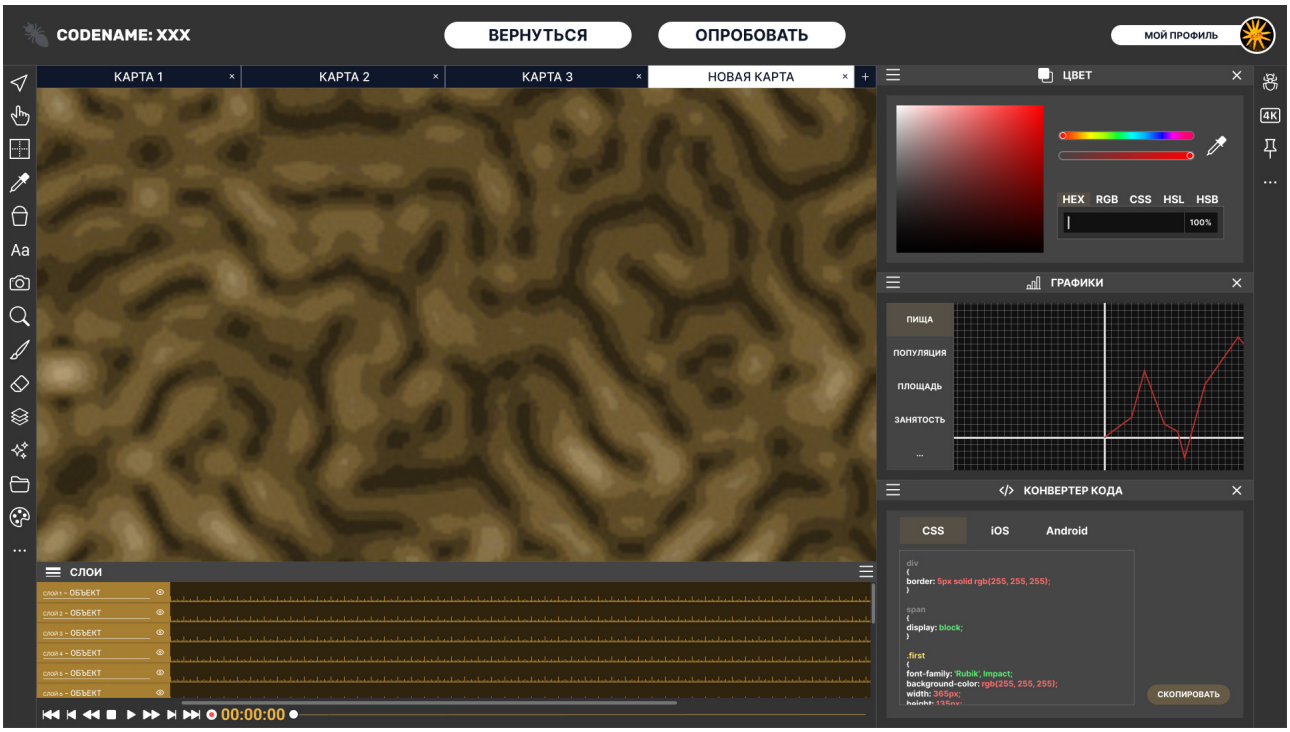


Рис. 1. Редактор карты

создания карты её данные передаются на сервер, в котором происходит моделирование биосистем, где выполняются основные вычисления. Этот сервер постоянно отправляет данные о положении муравьёв на основной сервер, который отвечает за взаимодействие с пользователем и использует TCP-соединение для надежной передачи данных.

Для снижения задержек и стабильной визуализации моделируемой колонии используется UDP протокол, обеспечивающий устойчивость системы к потере пакетов, поскольку пропуск отображаемого кадра не влияет на моделируемую симуляцию.

Для визуального отображения колонии была выбрана растровая графика с использованием Canvas API, поскольку она обеспечивает высокую производительность и стабильность работы системы даже при обработке большого количества движущихся объектов, как, например, на рис. 2 с оптимизированным видом на симуляцию колонии.

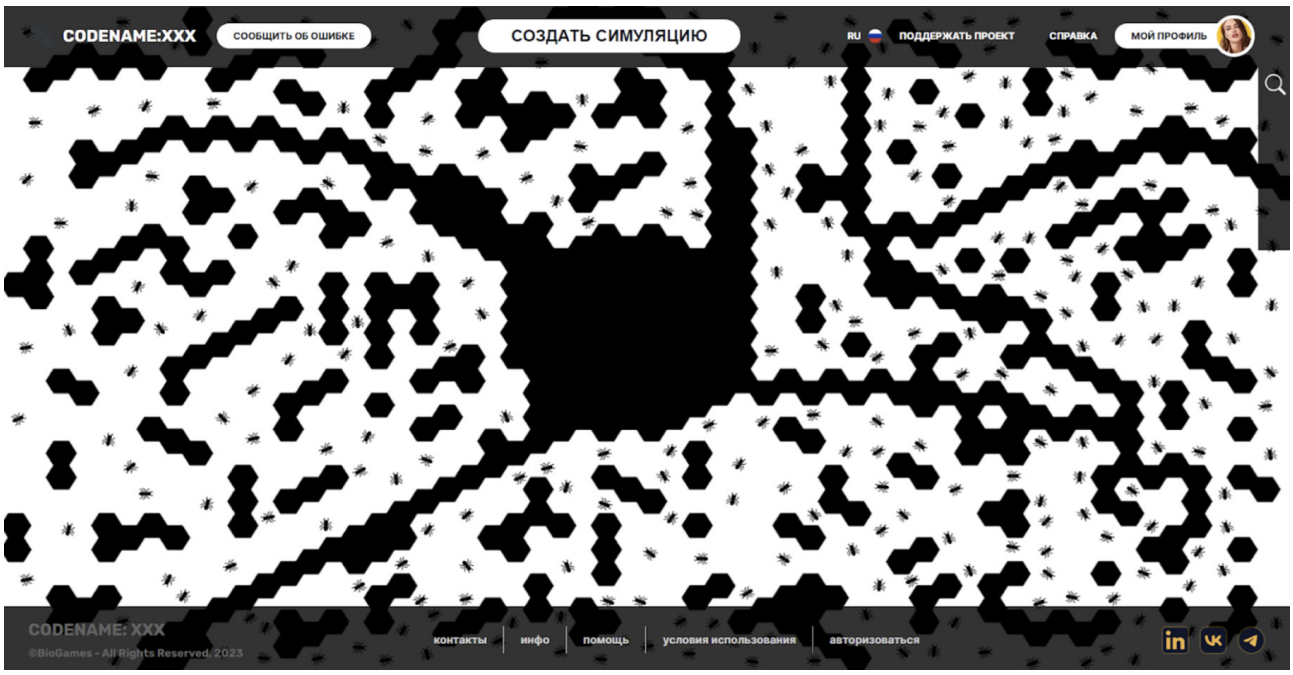


Рис. 2. Оптимизированный вид симуляции колонии муравьёв

Для повышения удобства использования приложения был разработан дизайн, который при обильной экспериментальной информации о моделируемой системе оставался бы удобным для пользователя. Для этого использовалась методология атомарного дизайна, которая подразумевает разбиение интерфейса на более мелкие компоненты (атомы).

4. Результат моделирования экосистем

Разработанное приложение для моделирования биосистем предоставляет возможности, которые трудно или даже невозможно реализовать в реальной природе. Использование компьютерных моделей позволяет ускорить исследование биосистем за счёт более высокой скорости проведения симуляций. Моделирование даёт возможность устанавливать такие начальные условия и параметры, которые сложно или даже невозможно создать в природных условиях.

Заключение

В рамках проведенной работы было создано приложение для моделирования биосистем, которое выступает мощным инструментом для исследования поведения этих систем. Разра-

ботка включает использование современных технологий и методов, что обеспечивает высокую производительность и удобство для пользователя.

Данное приложение позволяет воспроизводить сложные экологические процессы, исследовать влияние различных параметров на развитие экосистем и тестировать гипотезы, которые трудно проверить в реальных условиях. Таким образом, созданный инструмент открывает новые возможности для изучения и понимания взаимодействий в биосистемах, что имеет как научное, так и практическое значение.

Литература

1. Rust lang book – URL: https://doc.rust-lang.ru/book/ (дата обращения: 11.11.2024).

2. Муравьиные алгоритмы. – URL: https://blog.bullgare.com/wp-content/uploads/2019/05/aca.pdf (дата обращения: 03.11.2024).

3. TCP/IP. – URL: https://selectel.ru/blog/tcp-ip-for-beginners/ (дата обращения: 05.11.2024).

4. Клиент-серверное и межсервисное взаимодействие: разбираемся в REST, GraphQL, RPC и WebSocket. – URL: https://habr.com/ru/articles/729528/ (дата обращения: 10.11.2024).

5. Atomic Design by Brad Frost. – URL: https://atomicdesign.bradfrost.com (дата обращения: 08.11.2024).

6. *Дженифер Тидвелл, Чарли Брюэр, Эйнн Валенсия*. Разработка интерфейсов. Паттерны проектирования. 3-е изд. – СПб. : Питер, 2022. – 560 с.

УДК 004.023

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АРХИТЕКТУРНЫХ ПОДХОДОВ К ОБЪЕКТНО-РЕЛЯЦИОННОМУ ОТОБРАЖЕНИЮ В C++ 14

А. Д. Бузуверов

Воронежский государственный университет

Аннотация. Рассматриваются проблемы объектно-реляционного отображения (ORM) в разработке программного обеспечения, сосредотачиваясь на преимуществах и недостатках различных архитектурных паттернов: Table Gateway, Row Gateway, Active Record и Data Mapper. Рассматриваются цели и задачи ORM, а также трудности, связанные с несоответствием объектной и реляционной моделей данных. Каждый паттерн оценивается с точки зрения сложности реализации, производительности, масштабируемости и тестируемости в контексте C++14.
Ключевые слова: объектно-реляционное отображение (ORM), семантический разрыв, бизнес-логика, доступ к данным, Table Gateway, Row Gateway, Active Record, Data Mapper, CRUD операции.

Введение

Использование объектно-ориентированного подхода в разработке программного обеспечения влечет за собой необходимость решения задачи долговременного хранения объектов. Использование реляционной БД для хранения объектно-ориентированных данных приводит к семантическому разрыву между способами обработки объектов и их хранения [1]. Разработка бизнес-логики приложения часто требует использования разных парадигм: объектно-ориентированного подхода для кода приложения и реляционной модели для базы данных. Это приводит к постоянному переключению между ними при каждом обращении к БД, что влечет за собой либо избыточный код, либо необходимость создания промежуточного слоя. Более эффективным решением является разработка специальной библиотеки, которая выполняет преобразования между объектами приложения и реляционной моделью БД. Такие библиотеки называются ORM (Object-Relational Mapping).

Альтернативой использования объектно-реляционного отображения является хранение объектов в объектно-ориентированных базах данных [2]. Предлагаемый подход устраняет несоответствие между представлениями данных. Однако, поскольку реляционные базы данных по-прежнему широко распространены, эффективное отображение объектно-ориентированных моделей в реляционные остаётся важной задачей.

1. ORM Цели и задачи объектно-реляционного отображения

1.1. Цели и за объектно-реляционного отображения

Цели использования ORM библиотек:
1. Минимизировать ошибки при взаимодействии объектной и реляционной моделей данных.
2. Разделить бизнес-логику от логики доступа к данным.
3. Обеспечить независимость приложения от конкретной базы данных.
4. Уменьшить вероятность нарушения целостности данных.
5. Повысить безопасность работы с данными.
6. Сократить объем кода приложения.

1.2. Задачи объектно-реляционного отображения

Разработка универсального подхода к автоматическому прямому и обратному отображению объектно-ориентированных и реляционных данных связана с трудностями (в англ. литературе — «object-relational impedance mismatch» [1, 3]):

1. Объектная модель, в отличие от реляционной, обладает более высокой степенью детализации и может описывать больше сущностей (классов), чем представлено в соответствующей базе данных таблицами.

2. Наследование — ключевая концепция объектно-ориентированного программирования, отсутствующая в современных реляционных системах управления базами данных.

3. В отличие от реляционных СУБД, использующих первичные ключи для идентификации записей, в объектно-ориентированном программировании идентификация объектов определяется программистом.

4. В объектно-ориентированных системах ассоциации реализуются однонаправленными ссылками, в отличие от реляционных баз данных, где используются внешние ключи. Для создания двунаправленной связи в ООП требуется явное определение ссылки в обеих сторонах.

5. В отличие от эффективного доступа к данным в реляционных базах данных с помощью SQL-запросов и JOIN-операций, навигация по графу объектов в объектно-ориентированных системах может быть неэффективной при извлечении данных из реляционной БД, требуя множества отдельных обращений.

2. Сравнительный анализ возможных подходов

Существует несколько подходов (паттернов) к созданию библиотеки объектно-реляционного отображения (англ. Data Source Architectural Patterns): Table Gateway, Row Gateway, Active Record, Data Mapper [4].

2.1. Table Gateway

В архитектуре программного обеспечения Table Gateway представляет собой шаблон проектирования, инкапсулирующий все операции доступа к данным (CRUD — создание, чтение, обновление, удаление) для конкретной таблицы базы данных. Взаимодействие с базой данных осуществляется посредством вызова методов объекта Table Gateway, что абстрагирует непосредственный контакт с SQL-запросами от остальной части приложения.

Этот подход обладает рядом преимуществ: централизация логики доступа к данным упрощает сопровождение и модификацию кода, повышая его читаемость и поддерживаемость. Повторное использование одного объекта Table Gateway в разных частях приложения сокращает дублирование кода. Наконец, простота создания моков Table Gateway существенно облегчает процесс юнит-тестирования.

Однако использование Table Gateway имеет и свои ограничения. Сложные запросы, не учтенные в первоначальном интерфейсе объекта, могут потребовать модификации самого Table Gateway, что снижает гибкость. Для выполнения ресурсоемких или сложных операций, требующих оптимизированных SQL-запросов, использование единого объекта Table Gateway может быть неэффективным. Наконец, сильная привязка Table Gateway к конкретной таблице базы данных делает его чувствительным к изменениям схемы базы данных; рефакторинг может потребовать значительных изменений в коде. Поэтому применение этого шаблона оптимально для таблиц с относительно простой структурой и запросами.

Table Gateway в C++14 подходит для небольших проектов с простыми моделями данных и где важен контроль над SQL-запросами и требуется высокая производительность. Однако,

для больших и сложных проектов с множеством взаимосвязанных таблиц, лучше рассмотреть более сложные ORM-фреймворки или архитектурные подходы, предлагающие более высокий уровень абстракции и функциональности. Его простота реализации делает его хорошей отправной точкой для изучения принципов ORM, прежде чем переходить к более сложным решениям.

2.2. Row Gateway

Шаблон проектирования Row Gateway представляет каждую запись из таблицы базы данных в виде отдельного объекта, структура которого отражает структуру таблицы. Взаимодействие с базой данных полностью скрыто внутри этих объектов.

К преимуществам Row Gateway относятся простота и интуитивность: разработчик работает с объектами, не используя SQL-запросы напрямую. Это способствует интеграции данных с объектно-ориентированным кодом. В ряде случаев, создание mock-объектов упрощает юнит-тестирование.

Однако, данный подход имеет и существенные недостатки. Внедрение доступа к данным непосредственно в объекты, содержащие бизнес-логику, нарушает принцип разделения ответственности и усложняет разработку и сопровождение кода. Более того, частые обращения к базе данных для получения каждого объекта приводят к снижению производительности приложения, особенно заметному при обработке больших объемов данных и во время тестирования. Поэтому Row Gateway наиболее эффективен для небольших наборов данных и простых приложений, где производительность не является критическим фактором.

Row Gateway в C++14 — это простой и эффективный подход для ORM, идеально подходящий для проектов с простыми схемами данных и небольшим количеством запросов. Однако, для сложных приложений с многочисленными отношениями между сущностями и сложными запросами, Row Gateway может оказаться непрактичным и потребует значительных усилий для поддержания кода. Его лучше рассматривать как один из вариантов в сравнительном анализе, подходящий для конкретных, ограниченных задач.

2.3. Active Record

Шаблон проектирования Active Record представляет собой подход, в котором объект одновременно содержит данные и методы для работы с ними, причем основная часть данных хранится в базе данных. Этот паттерн тесно интегрирует логику доступа к данным с бизнес-логикой объекта.

Его простота и интуитивность делают Active Record привлекательным, особенно для разработчиков с ограниченным опытом работы с базами данных. Код становится более кратким и читабельным, а разработка — быстрее, что особенно ценно для небольших проектов.

Однако, по мере роста и усложнения проекта, преимущества Active Record начинают уступать место недостаткам. Тесная связь данных и логики приводит к снижению поддерживаемости и усложнению модификации кода. Тестирование таких объектов становится сложной задачей, требующей использования моков или других изоляционных техник. Наконец, Active Record часто нарушает фундаментальные принципы объектно-ориентированного проектирования, такие как принцип единственной ответственности и принцип открытости/закрытости, что в долгосрочной перспективе негативно влияет на архитектуру и масштабируемость приложения. Переход на другую СУБД также может потребовать значительных усилий. Таким образом, Active Record подходит для небольших проектов с ограниченной сложностью, но может стать препятствием для больших и сложных систем.

Реализация паттерна Active Record в C++14 для объектно-реляционного отображения (ORM) характеризуется определёнными удобствами и недостатками. Удобство прежде всего заключается в простоте концепции: класс напрямую отражает таблицу базы данных, а его методы — операции CRUD (Create, Read, Update, Delete). Это интуитивно понятно и облегчает начальную разработку.

2.3. Data Mapper

Объектно-ориентированные и реляционные модели данных обладают фундаментальными различиями: объектные модели используют механизмы наследования и коллекций, отсутствующие в реляционных базах данных. В сложных приложениях с богатой бизнес-логикой это различие приводит к несоответствию между объектной моделью и схемой базы данных.

Шаблон проектирования Data Mapper решает эту проблему, создавая промежуточный слой между объектной моделью приложения и базой данных. Этот слой отвечает за преобразование данных между двумя моделями и обеспечивает их независимость друг от друга. Объекты в приложении, использующем Data Mapper, не знают о структуре базы данных, не содержат логики доступа к данным и не работают с SQL-запросами напрямую.

Преимущества Data Mapper очевидны: чёткое разделение логики обработки данных и логики доступа к данным повышает читаемость, поддерживаемость и тестируемость кода. Изменение базы данных или объектной модели не требует изменений в другой части приложения; миграция на другую СУБД сводится к замене Data Mapper. Этот шаблон хорошо масштабируется и подходит для больших и сложных проектов. Data Mapper легко тестируется благодаря возможности изоляции от базы данных и использованию моков. Наконец, один и тот же Data Mapper может использоваться для работы с различными объектами, обеспечивая единообразие интерфейса.

Однако, использование Data Mapper имеет и некоторые недостатки. Внедрение этого паттерна более сложно, чем использование более простых подходов, таких как Active Record; потребуется больше кода и времени на разработку. Дополнительный уровень абстракции может незначительно снизить производительность. Наконец, потребуется время на освоение паттерна для разработчиков, не знакомых с ним. Таким образом, Data Mapper — мощный инструмент для больших и сложных проектов, но требует тщательного взвешивания сложности внедрения и потенциальных преимуществ.

Data Mapper в C++14 представляет собой элегантный и гибкий подход к ORM, если приоритетом является контроль и производительность. Он подходит для проектов, где критична оптимизация запросов и требуется глубокое понимание работы с базой данных. Однако, для быстрой разработки и проектов с простой моделью данных, более автоматизированные ORM-фреймворки могут показаться более удобными, несмотря на возможную потерю производительности и гибкости.

Заключение

Таким образом, проведенный сравнительный анализ архитектурных подходов к объектно-реляционному отображению (ORM) демонстрирует, что каждый паттерн имеет свою нишу и может быть эффективным в определенных контекстах. Отличия в производительности, сложности реализации и степени абстракции указывают на отсутствие универсального решения.

В контексте C++14 Table Gateway и Row Gateway подходят для небольших проектов с простыми моделями данных, предлагая простоту реализации и контроль над SQL-запросами, но ограничены в масштабируемости и сложности обрабатываемых данных. Active Record, несмотря на свою интуитивность и быстроту разработки, страдает от недостатков в масштабируе-

мости и тестируемости по мере роста проекта. Data Mapper, являясь наиболее сложным в реализации, оптимален для крупных проектов с богатой бизнес-логикой, обеспечивая высокую гибкость, масштабируемость и тестируемость за счет четкого разделения ответственности. Выбор конкретного шаблона должен основываться на компромиссе между простотой реализации, производительностью и требованиями к масштабируемости и поддерживаемости проекта.

Литература

1. *Johnson R.* J2EE Design and Development // Wrox. – 2002. – P. 255–256.

2. *Dietrich S. W., Urban S. D.* Fundamentals of Object Databases: ObjectOriented and Object-Relational Design, – Morgan & Claypool Publishers, 2011. – P. 66–100.

3. *Bernard E.* «The Object-Relational Impedance Mismatch,» [В Интернете]. Available: http://hibernate.org/orm/what-is-an-orm/#the-object-relationalimpedance-mismatch.

4. *Fowler M.* Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002.

УДК 004.42

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ БИБЛИОТЕК AR FOUNDATION И VUFORIA

Д. О. Бутузова, Е. В. Трофименко

Воронежский государственный университет

Аннотация. Дополненная реальность (AR) представляет собой технологию, которая позволяет интегрировать виртуальные объекты в реальное пространство, обеспечивая новые формы взаимодействия с окружающим миром. В статье рассматриваются две библиотеки для разработки AR-приложений в Unity: AR Foundation и Vuforia. Оцениваются их возможности, особенности безмаркерного трекинга, а также производительность на различных устройствах. Сравнение этих библиотек помогает выбрать оптимальное решение для различных типов AR-приложений, учитывая специфику проекта и требования к совместимости с устройствами.

Ключевые слова: дополненная реальность, augmented reality(AR), Unity, AR Foundation, Vuforia, безмаркерный трекинг, кроссплатформенность, производительность, распознавание поверхностей, разработка AR-приложений.

Введение

Дополненная реальность(AR) — это технология, которая в реальном времени совмещает виртуальные объекты с элементами физического мира. В отличие от виртуальной реальности(VR), которая полностью погружает в искусственную среду, AR позволяет расширить реальное окружение. AR широко используется в играх и мобильных приложениях, улучшая пользовательский опыт и предоставляя новые формы взаимодействия с окружающей средой.

В игровой индустрии AR открывает новые возможности для создания интерактивных приложений, где игроки могут взаимодействовать с виртуальными объектами, помещенными в их физическое окружение. Unity [1, 2] — один из самых популярных движков для разработки игр и приложений, предоставляет мощные инструменты для создания приложений с дополненной реальностью.

1. Библиотеки для работы с безмаркерной технологией в AR в Unity

Безмаркерная технология дополненной реальности предоставляет возможность определять и отслеживать поверхности, объекты и их перемещения без использования маркеров [3]. Она опирается на функции пространственного распознавания и трекинга, что открывает широкий спектр возможностей для взаимодействия с окружающим миром. Благодаря этому пользователи могут размещать виртуальные объекты в реальном пространстве, взаимодействовать с ними и наблюдать, как они динамически адаптируются к изменениям окружения [4, 5].

Для отслеживания положения и ориентации устройства в реальном пространстве, и правильного интегрирования виртуальные объекты с реальным миром реализован процесс трекинга. Это ключевая часть технологии AR, которая позволяет виртуальным объектам, таким как 3D-модели, текстуры и анимации, оставаться стабильно привязанными к реальным объектам или определённым точкам в окружающем пространстве.

Трекинг без маркеров использует данные с устройства, такие как GPS, акселерометры, гироскопы и камеры, чтобы отслеживать изменения в положении и ориентации. Это позволяет отслеживать объекты и пространства без необходимости в явных маркерах.

В рамках данной статьи будет подробно рассмотрена безмаркерная технология. Для сравнения, существует и маркерная технология, где размещение виртуальных объектов зависит от специальных меток, размещённых в физическом пространстве.

1.1. Библиотека AR Foundation

AR Foundation [6, 7] — это кроссплатформенная библиотека Unity, упрощающая создание и развертывание AR-приложений на различных платформах с использованием единого API, что делает разработку удобнее и быстрее. AR Foundation поддерживает безмаркерную технологию, в том числе распознавание плоскостей, 3D-объектов и лиц. Простота настройки и широкие возможности трекинга поверхностей и объектов позволяют эффективно использовать AR Foundation для самых разных задач.

Главное преимущество AR Foundation — это кроссплатформенность. Библиотека поддерживает устройства на iOS и Android, а также совместима с актуальными версиями ARKit (для iOS) и ARCore (для Android). Она легко интегрируется в Unity, что ускоряет процесс разработки.

Однако у AR Foundation есть и недостатки. Она не полностью поддерживает устаревшие устройства, а её функциональность зависит от возможностей ARKit и ARCore, что может ограничивать разработчиков в использовании некоторых функций.

1.2. Библиотека Vuforia

Vuforia [8] — это платформа для разработки AR, предназначенная для мобильных устройств и очков дополненной реальности. Изначально она разрабатывалась как маркерная система, но со временем получила поддержку безмаркерной технологии, включая распознавание плоскостей и трекинг объектов.

Vuforia предлагает расширенные возможности трекинга, совмещая маркерные и безмаркерные подходы. Она обеспечивает высокую точность отслеживания объектов, поддерживая различные форматы изображений и моделей. Библиотека также включает функции трекинга плоскостей, пространственного позиционирования и распознавания 3D-объектов.

Среди преимуществ Vuforia — поддержка большего числа устройств, включая некоторые устаревшие модели, и возможность интеграции с платформами и библиотеками, которые недоступны в AR Foundation.

Однако у Vuforia есть и недостатки. Поддержка новых функций ARKit и ARCore ограничена, а производительность на современных устройствах может уступать AR Foundation. Кроме того, для доступа к полному набору возможностей требуется приобретение платной лицензии.

1.3. Сравнение библиотек AR Foundation и Vuforia

Сравнение двух библиотек по критериям: доступность платформ для работы приложения, поддержки безмаркерной технологии, точности трекинга, качество распознавания поверхностей, возможность кастомизации, вариантов лицензий и производительности приведено в табл. 1.

Выбор между библиотеками AR Foundation и Vuforia зависит от специфики проекта и целевых устройств. AR Foundation, интегрированная с Unity и использующая возможности ARKit и ARCore, является отличным выбором для приложений, которые требуют поддержки современных устройств с высокой точностью трекинга и безмаркерной технологии. Она идеально подходит для приложений, ориентированных на мобильные платформы iOS и Android.

Vuforia, в свою очередь, предоставляет более широкие возможности для использования маркеров и 3D-объектов, а также поддерживает различные устройства, включая очки дополненной реальности. Она подходит для более универсальных решений, в том числе для старых устройств, и предоставляет большую гибкость в кастомизации. Однако, для доступа к полной функциональности необходимо приобретение платной версии, что может быть ограничением для некоторых пользователей.

Таблица 1

Критерий	AR Foundation	Vuforia
Поддержка платформ	iOS, Android (через ARKit и ARCore)	iOS, Android, очки дополненной реальности
Безмаркерная технология	Отличная, поддерживает плоскости и лица	Хорошая, но ориентирована больше на маркеры
Точность трекинга объектов	Зависит от ARKit/ARCore, высокое качество	Широкий спектр, включая маркеры, 3D-объекты
Качество распознавания поверхностей	Очень точное для новых устройств	Универсальное, подходит для старых и новых
Гибкость и кастомизация	Ограничено платформами	Высокая, можно использовать в различных AR-решениях
Лицензирование	Бесплатно с Unity	Бесплатно с ограничениями, полная версия платная
Производительность	Высокая на новых устройствах	Зависит от устройства, может требовать больше ресурсов

Таким образом, AR Foundation лучше подходит для современных устройств и приложений с фокусом на мобильные платформы и безмаркерные технологии, в то время как Vuforia предоставляет больше возможностей для кастомизации и поддержки широкого спектра устройств, но с учетом платных лицензий для полного функционала.

2.4. Сравнение производительности библиотек AR Foundation и Vuforia

Для оценки производительности были созданы два приложения с идентичным функционалом, использующие AR Foundation и Vuforia. Оба приложения включали реализацию распознавания поверхностей с применением стандартных функций библиотек, визуализацию обнаруженных поверхностей, а также возможность размещения и перемещения 3D-объекта по распознанной поверхности. Пример работы приложения изображен на рис. 1.

Рис. 1. Пример работы приложения

С помощью Unity Profiler были измерены различные критерии производительности приложений, такие как средний FPS(частота обновления экрана), стабильность трекинга, среднее время обнаружения первой плоскости, потребление ресурсов CPU и GPU, точность позиционирования, использованная память и использованная память для хранения текстур. Также путем вывода логов в файл было рассчитано среднее время распознавания первой поверхности. Данные, полученные в процессе тестирования приложений приведены в табл. 2.

Таблица 2

Данные тестирования

Критерий	AR Foundation	Vuforia
Средний FPS	30 FPS	30 FPS
Стабильность трекинга	Высокая стабильность	Средняя стабильность
Среднее время обнаружения плоскости (сек)	15	12
CPU (мс/кадр)	33.85	56
GPU (мс/кадр)	1.72	1.92
Точность позиционирования	Высокая точность	Хорошая точность
Использованная память(MB)	260	194
Использованная память для хранения текстур(MB)	20.4	29.3

AR Foundation обеспечивает более стабильный трекинг и высокую точность позиционирования, но требует больше ресурсов памяти и нагрузки на CPU.

Vuforia быстрее распознает поверхности и использует меньше оперативной памяти, но имеет чуть менее стабильный трекинг и меньшую точность позиционирования.

Память текстур у Vuforia выше, что может указывать на большую нагрузку на графическую подсистему.

Обе библиотеки — AR Foundation и Vuforia — имеют свои сильные и слабые стороны, и выбор между ними должен зависеть от специфики проекта.

Заключение

AR Foundation идеально подходит для приложений, ориентированных на новые устройства с поддержкой ARKit и ARCore. Она обеспечивает высокую производительность, точность и стабильность трекинга на современных устройствах. Библиотека особенно эффективна для задач безмаркерного трекинга, что делает её отличным выбором для приложений, требующих высокой точности и стабильности работы.

Vuforia, в свою очередь, предоставляет большую гибкость и поддерживает широкий диапазон устройств, включая старые модели и AR-очки. Это делает её лучшим вариантом для проектов, где важно обеспечить совместимость с различными типами устройств. Также Vuforia поддерживает как маркерный, так и безмаркерный трекинг, что позволяет кастомизировать приложение под разные требования. Однако её производительность и точность на новых устройствах могут быть ниже по сравнению с AR Foundation.

Если основным приоритетом является высокая производительность и поддержка новых устройств, то лучшим выбором будет AR Foundation. В случае, когда требуется поддержка различных типов устройств и гибкость в трекинге, включая маркерный AR, предпочтительнее использовать Vuforia.

Литература

1. Unity Technologies / Unity. – URL: https://unity.com/ (дата обращения: 08.10.2024).

2. AR development in Unity/ Unity. – URL: https://docs.unity3d.com/2023.1/Documentation/Manual/AROverview.html (дата обращения: 10.10.2024).

3. Augmented Reality and Virtual Reality: New Trends in Immersive Technology / M. Claudia tom Dieck, Timothy H. Jung, Sandra M. C. Loureiro. – 1-е изд. – Берлин : Springer, 2021. – 325 с.

4. Robert Wells. Unity 2020 By Example: A project-based guide to building 2D, 3D, augmented reality, and virtual reality games from scratch / Robert Wells. – Бирмингем : Packt Publishing, 2020. – 676 с.

5. Micheal Lanham. Augmented Reality Game Development / Micheal Lanham. – Бирмингем : Packt Publishing, 2017. – 334 с.

6. Введение в ARFoundation / Habr. – URL: https://habr.com/ru/companies/otus/articles/688306/ (дата обращения: 10.10.2024).

7. AR Foundation / Unity. – URL: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.0/manual/index.html (дата обращения: 10.10.2024).

8. Vuforia Developer Portal. – URL: https://developer.vuforia.com/library/unity-extension/working-vuforia-engine-and-unity/ (дата обращения: 19.10.2024).

УДК 004.9, 628.1

АНАЛИЗ ЦИФРОВЫХ ИНСТРУМЕНТОВ ДЛЯ ПРОЕКТИРОВАНИЯ СИСТЕМ ВОДОСНАБЖЕНИЯ

В. С. Войнова

Астраханский государственный архитектурно-строительный университет

Аннотация. В статье проведен сравнительный анализ программ, используемых для проектирования и расчета систем водоснабжения. Исследование позволило выявить преимущества и недостатки каждой программы, а также определить их эффективность в решении конкретных задач. Результаты исследования будут использоваться для разработки информационной системы для проектирования водопроводных сетей.
Ключевые слова: учёт расхода воды, расчётная модель, функциональные особенности, решение задач проектирования систем водоснабжения.

На сегодняшний день проектирование систем водоснабжения достаточно трудоёмкое и ресурсозатратное занятие. Люди долгое время проектировали все эти системы вручную, используя таблицы Exel или бумажные носители, что создавало определённые неудобства: постоянно приходится ссылаться на нормативные документы, на бумаге можно было легко ошибиться с расчётами, а файлы Exel легко поддаются изменениям, что тоже сказывается на точности расчётов, и оба типа хранения данных могут легко потеряться.[1] Ввиду большой трудоёмкости расчётов были разработаны различные цифровые инструменты для расчёта и проектирования водопроводных сетей, такие как ZuluHydro, УМНАЯ ВОДА и WaterSupply. Рассмотрим их функциональные особенности, достоинства и недостатки.

Сравнительный анализ проводился по ряду критериев:

– кроссплатформенность,
– доступ пользователя к ИС,
– техническая поддержка,
– функциональность ИС,
– тип лицензии.

Эти критерии наиболее важны для определения удобства в работе пользователя в системе, поэтому в данном сравнительном анализе описанные выше информационные системы будут рассматриваться с позиции каждого из перечисленных критериев.

ZuluHydro — это набор программ для расчетов водопроводных сетей, предназначенных для решения различных отраслевых задач [2]. ZuluHydro доступна только на операционных системах Windows и устанавливается локально на каждый компьютер, из-за чего программа не доступна пользователям, работающим на Linux, Mac OS, Android, iOS.

В функционал данной программы входит создание расчетной математической модели сети, выполнение паспортизации сети. На основе созданной модели возможно решение информационных задач, задач топологического анализа и выполнение различных гидравлических расчетов. ZuluHydro позволяет рассчитывать водопроводную сеть большого объема и любой сложности. Расчету подлежат тупиковые и кольцевые сети водоснабжения, в том числе с оптимизирующими насосными станциями и дросселирующими устройствами, работающие от одного или нескольких источников [2].

Построение электронных моделей систем водоснабжения включает применении следующих этапов [3]:

1. Разработка расчетной схемы модели: выбор типа расчетной схемы (укрупненной или подробной) в зависимости от поставленных задач, формировании топологии сети в про-

граммный комплекс, создании электронных баз данных характеристик элементов для проведения гидравлических расчетов.

2. Разработка балансовой характеристики модели заключается в разработке балансовой схемы в соответствии выбранному типу расчетной схемы. На данном этапе проводится анализ балансов потребления воды в городе.

3. Проведение предварительных расчетов системы. Целью предварительных расчетов является определение потокораспределения в водопроводной сети, подачи и напора источников при известных диаметрах труб и отборах воды в узловых точках.

4. Калибровка гидравлической модели. Калибровка гидравлической модели уточняет математическую модель сети с использованием измерений, полученных на водопроводной сети. В результате калибровки подбираются новые гидравлические сопротивления и коэффициенты утечек для участков сети так, чтобы математическая модель давала минимальные отклонения от давлений в контрольных точках, распределяла суммарные утечки по участкам сети и обеспечивала баланс расходов между источниками.

Интерфейс программы для проектировщика водопроводных сетей, который не обладает высокими навыками работы с компьютером, сложен и загромождён избыточными функциями, не предназначенными напрямую для расчёта.

Программный продукт является платным. Для бесплатного пользования программой с официального сайта скачивается демо-версия программы на рабочий компьютер, но она имеет множество ограничений по количеству параметров, что уже не подходит для решения более крупных задач. Обеспечивается ежегодная платная техническая поддержка [4].

Программа для проектирования систем внутреннего водопровода и канализации зданий УМНАЯ ВОДА зарегистрирована в Реестре программ для ЭВМ с 25 ноября 2016 г. [5].

В отличии от ZuluHydro, в программе УМНАЯ ВОДА можно работать как через браузер в веб-версии программы, так и через клиентское приложение. Однако «УМНАЯ ВОДА» не является самостоятельным приложением: работа в этой программе осуществляется через тонкий клиент 1С.

Основной функционал программы — это создание «одного окна», в котором будут выполняться все необходимые расчёты в соответствии с нормативно-технической документацией. Как и ZuluHydro в УМНОЙ ВОДЕ имеются функции, которые являются избыточными для пользователя, например, санитарный расход воды в секунду, а не в час [6].

УМНАЯ ВОДА ориентирована на расчет внутренних систем водоснабжения и позволяет проектировать:

1) хозяйственно-питьевой водопровод,
2) противопожарный водопровод,
3) хозяйственно-противопожарный водопровод,
4) бытовая канализация,
5) дождевая канализация (внутренний водосток).

Информационная система «УМНАЯ ВОДА» в режиме онлайн позволяет выполнять мониторинг уровня и расхода воды по каналу и отводу. Путем нажатия на ярлык дислокации водного объекта достигается автоматический учет расхода воды. В данной программе моделируется зарастание трубопроводов, приводящее к увеличению потерь напора, производится балансировка циркуляционных колец, осуществляется подбор необходимого оборудования, материалов и их специфицирование. Гидравлические расчеты выполняются не по табличным значениям, а по аналитическим формулам, которые наиболее точно описывают физические зависимости.

Сформированные отчеты сохраняются в форматах .doc/docx, .xls/xlsx или .pdf [7]:

– баланс водопотребления и водоотведения;
– спецификация оборудования и материалов;

– паспорт системы ГВС / ХВС;
– расчетные расходы воды;
– гидравлический расчет в режиме водоразбора;
– тепловой расчет;
– гидравлический расчет в режиме циркуляции;
– настройка балансировочных клапанов.

Пользователю доступен пробный период 1,5 месяца, по окончании которого функционал по выполнению расчетов становится ограниченным.

WaterSupply относится к типу программ «Технический расчет» (ТП) и создан для расчета различных инженерных проектов. В отличии от программы УМНАЯ ВОДА, WaterSupply доступна только для пользователей ОС Windows и устанавливается локально на компьютер, также как ZuluHydro.

Основной функционал программы основан на положении СНиП 2.04.01-85 [8] и реализует его основные пункты для гидравлического проектирования водопроводных сетей внутри зданий. Для заданной конфигурации сети (известными величинами считаются длины участков, материал и диаметры труб в первом приближении, характер водопотребления, свободные напоры у водоразборной арматуры, количество потребителей, а также количество и места расположения водоразборной арматуры) определяется требуемый напор на вводе. Также программа предоставляет возможность назначения диаметров участков в соответствии с «экономичными» скоростями движения воды [9].

В WaterSupply реализована возможность определения необходимого напора воды для конкретной конфигурации водопроводной сети. Конфигурация сети определяется в программе выбором значений следующих параметров:

1) длина участка трубы;
2) материал труб;
3) диаметр труб в первом приближении;
4) характер потребления воды;
5) свободные напоры водоразборной арматуры;
6) количество потребителей;
7) количество водоразборной арматуры и ее расположение в сети.

С целью оптимизации скоростей движения воды предусмотрена возможность изменения диаметра участков труб для готовой схемы.

В отличии от рассмотренных выше программ, WaterSupply обновляется и не имеет технической поддержки. Информационная система имеет устаревший интерфейс и ограниченный функционал, например, реализован расчет общего расхода воды, при этом нельзя выполнить просмотр результатов среднесуточного расхода воды.

Для бесплатного пользования WaterSupply доступна только демо-версия, которая включает в себя возможность расчета данных лишь для «контрольной сети» одного типа водопотребителей [10].

Проведенный сравнительный анализ позволил выявить как достоинства, так и недостатки рассмотренных информационных систем. Результаты анализа представлены в табл. 1.

В результате проведенного исследования были рассмотрены три информационные системы для проектирования водопроводных сетей. Каждая система обладает своими особенностями, преимуществами и недостатками. По результатам анализа была актуализирована и поставлена задача по разработке информационной системы для автоматизации расчетов водопотребления, которая будет учитывать выявленные недостатки рассмотренных выше программ и позволит проектировщикам более качественно решать задачи проектирования систем водоснабжения.

Таблица 1

Результат сравнительного анализа ZuluHydro, УМНАЯ ВОДА, WaterSupply

Название ИС / Критерии	ZuluHydro	УМНАЯ ВОДА	WaterSupply
Компания разработчик (правообладатель)	Политерм	Элита	Istok
Кроссплатформенность	отсутствует	присутствует	отсутствует
Доступ пользователя	локально	локально, веб	локально
Техническая поддержка	реализуется	реализуется	не реализуется
Функциональность	избыточная	избыточная	ограничена
Лицензия	платно, цена уточняется у компании, демо-версия	платно: 1 400 – 36 000р., пробный период 1,5 месяца	платно, цена уточняется у компании, демо-версия

Литература

1. Разработка гидравлических электронных моделей водоснабжения // Научная электронная библиотека ELibrary.Ru : [сайт]. – 2000–2024. – URL: https://www.elibrary.ru/item.asp?id=57182400 (дата обращения: 28.10.2024).

2. Расчет систем водоснабжения – ZULUHYDRO 9.0 («ZULUHYDRO 9.0») // Научная электронная библиотека ELibrary.Ru : [сайт]. – 2000–2024. – URL: https://elibrary.ru/item.asp?id=45823250 (дата обращения: 28.10.2024).

3. 06.12-19И.292 Применение современных информационных технологий при расчете гидравлического удара в системах водоснабжения // Научная электронная библиотека ELibrary.Ru : [сайт]. – 2000–2024. – URL: https://elibrary.ru/item.asp?id=9800504 (дата обращения: 28.10.2024).

4. Сайт разработчиков «Политерм». ZuluHydro : официальный сайт. – Санкт-Петербург, 1999–2024. – URL: https://www.politerm.com/products/hydro/zuluhydro/ (дата обращения: 28.10.2024).

5. «УМНАЯ ВОДА» : сайт группы компаний «CSoft». – Санкт-Петербург, 2002–2024. – URL: https://www.csoft.ru/soft/umnaya-voda/umnaya-voda.html (дата обращения: 28.10.2024).

6. Оптимизация управления водными ресурсами с использованием системы «УМНАЯ ВОДА» в ирригационных системах: методы и результаты // Научная электронная библиотека ELibrary.Ru : [сайт]. – 2000–2024. – URL: https://www.elibrary.ru/item.asp?id=59462825 (дата обращения: 05.11.2024).

7. Сайт программы «УМНАЯ ВОДА» : официальный сайт. – Санкт-Петербург, 2024 – URL: https://smartwater.su/ (дата обращения: 05.11.2024).

8. СНиП 2.04.01-85 «Внутренний водопровод и канализация зданий» – М. : Министерство строительства РФ, 1996.

9. Программа для расчета гидравлических характеристик внутреннего водопровода «WaterSupply» : сайт компании «Агроводком». – Москва, 2003–2024 –URL: https://clck.ru/3ERYaT (дата обращения: 05.11.2024).

10. Сайт программы «WaterSupply» : официальный сайт. – Санкт-Петербург, 2024. – URL: https://www.kommproekt.ru/index.php/blog/item/157-istokcalc-watersupply (дата обращения: 05.11.2024).

УДК 004.512

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ПОДДЕРЖКИ ПРИНЯТИЯ КЛИНИЧЕСКИХ РЕШЕНИЙ ПРИ ЭНДОСКОПИЧЕСКОМ ГЕМОСТАЗЕ ЯЗВЕННЫХ ГАСТРОДУОДЕНАЛЬНЫХ КРОВОТЕЧЕНИЙ

А. И. Воротилина, И. Л. Каширина

Воронежский государственный университет

Аннотация. В работе рассматривается разработка мобильного приложения для поддержки принятия врачебных решений при эндоскопическом гемостазе язвенных дуоденальных кровотечений. Приложение включает рекомендательную систему, основанную на дереве решений, которая помогает в принятии клинических решений и диагностике типа кровотечения. Система предоставляет врачам рекомендации по лечению, а также информацию о необходимом оборудовании для проведения процедуры. В приложении реализованы модули для обучения, оказания медицинской помощи, оценки качества и справочной информации. Планируется внедрение приложения в медицинские учреждения Воронежа.

Ключевые слова: мобильное приложение, эндоскопический гемостаз, рекомендательная система, клинические решения, язвенные гастродуоденальные кровотечения.

Введение

Внутренние желудочные кровотечения представляют собой серьёзное медицинское состояние. Современные методы диагностики и лечения внутренних желудочных кровотечений разнообразны и зависят от тяжести состояния, причины кровотечения и общего состояния пациента. Однако, несмотря на развитие медицинских технологий, процесс принятия решений врачами остаётся сложным и требует глубоких знаний и опыта.

В Российской Федерации язвенной болезнью страдает 1,7–5 % населения, причем количество пациентов с кровотечениями из хронических язв желудка и 12-перстной кишки составляет 90–160 человек на 100 000 населения и имеет отчетливую тенденцию к увеличению [1]. Несмотря на значительные успехи хирургии и анестезиологии, летальность при острых желудочно-кишечных кровотечениях составляет 5–14 %, а при рецидиве возрастает до 30–40 % и не имеет тенденции к уменьшению [2].

Принятие клинических решений при эндоскопическом гемостазе язвенных гастродуоденальных кровотечений связано с рядом сложностей. Точность диагностики зависит от субъективной оценки врача и опыта в интерпретации эндоскопических изображений, а также ограниченное время на принятие решения в экстренных ситуациях создает давление на врача, что увеличивает риск ошибок.

Описанные факторы подчеркивают важность разработки вспомогательных инструментов, основанных на современных технологиях. Мобильное приложение, оснащённое рекомендательной системой предоставить врачам своевременные и точные рекомендации по лечению и помочь в диагностике типа кровотечения, улучшая тем самым качество медицинской помощи.

Мобильные технологии играют все более важную роль в медицине, предоставляя клиницистам доступ к необходимой информации и инструментам в режиме реального времени. Современные технологии позволяют улучшить точность диагностики, автоматизировать процессы сбора данных и повысить скорость принятия клинических решений. Поддержка клиницистов через мобильные приложения особенно важна в условиях необходимости быстрого нахождения оптимальной тактики лечения.

Целью исследования является разработка и внедрение мобильного приложения для поддержки принятия врачебных решений в процессе эндоскопического гемостаза.

1. Материалы и методы

Разработка дерева решения для грамотной рекомендательной системы имеет критическое значение для нескольких аспектов медицинской практики, особенно в контексте лечения внутренних желудочных кровотечений. В результате взаимодействия с группой квалифицированных врачей Воронежской области описана рекомендательная система для поддержки принятия врачебных решений, а также составлено дерево решений для мобильного приложения.

Все рекомендации имеют идентификационный номер. Из конечной рекомендации при нажатии кнопки «продолжить» совершается переход в блок оценки приложения. При помощи кнопки назад из любой странички (кроме стартового окна) совершается переход либо на предыдущую рекомендацию, либо на блок выбора, из которого был совершен переход на текущую страницу. Примеры блоков дерева решений представлены на рис. 1–2.

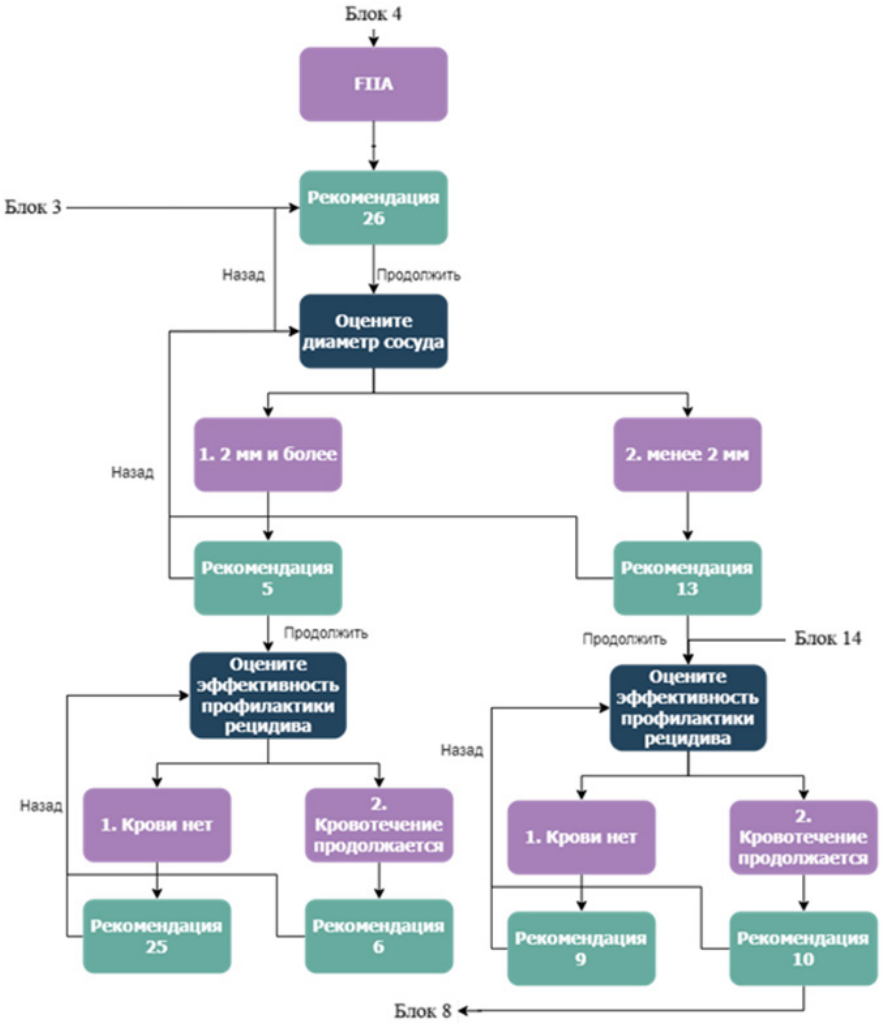


Рис. 1. Пример блока дерева решений

Для разработки мобильного приложения выбран фреймворк React Native языка программирования JavaScript. Одним из ключевых преимуществ React Native [3] является возможность разработки приложений для iOS и Android с использованием единого кода. Выбор React Native для разработки мобильного приложения предоставляет множество преимуществ, таких как высокая производительность, богатая экосистема, интеграция с нативным кодом, возможность повторного использования кода и улучшенное пользовательское взаимодействие. Для создания React Native проекта использовался Expo CLI [4].

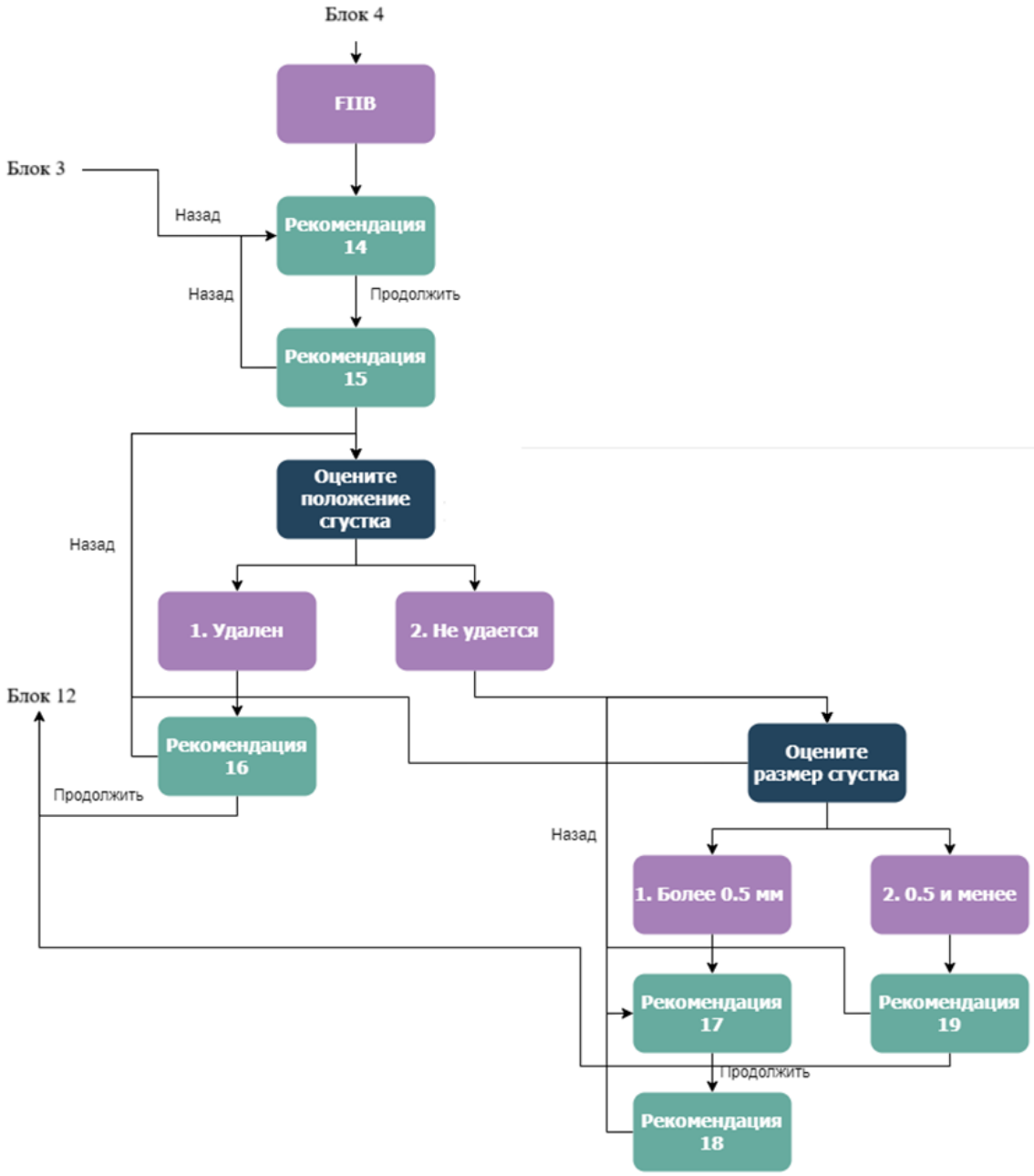


Рис. 2. Пример блока дерева решений

Основываясь на дереве решения и специфике проекта, были определены следующие ключевые особенности дизайна мобильного приложения:

1. Header с заголовком экрана или содержанием вопроса;
2. Кнопки «назад» и «продолжить», обеспечивающие удобную навигацию внутри мобильного приложения;
3. Возможность заполнять данные о медицинской организации;
4. Возможность оставлять отзыв об алгоритме рекомендательной системе;
5. Возможность изменять цветовую схему мобильного приложения.

Исходя из этого был разработан макет интерфейса мобильного приложения.

На стартовом экране приложения отображены название проекта и кнопка «продолжить», осуществляющая переход на главную страницу приложения. Также на стартовом экране расположен переключатель, который позволяет сменить цветовую тему приложения. На главной

странице приложения осуществляется выбор режима работы приложения. Каждый модуль приложения выполнен в едином стилистическом оформлении.

Приложение включает опрос, разработанный для сбора мнений врачей о функциональности и значимости алгоритма рекомендательной системы. Из ответа пользователя формируется email письмо на почту врача, который анализирует получаемые данные.

Для реализации описанных функциональных возможностей разработаны следующие компоненты: Footer, Header, Info, BackButton, LaterButton, MenuButton, NextButton, QAForm, SaveAnswers, RecommendationStyle, StartBox.

Все рекомендации хранятся в файле формата json, который содержит информацию об идентификационном номере рекомендации и текст рекомендации.

На базе компонент реализованы экраны EquipmentScreen, InformationScreen, StartScreen, HomeScreen, а также страницы типов QAScreens, QuestionsScreens, RecommendationScreens.

Мобильное приложение имеет разветвленную логику принятия решений и большое количество экранов, поэтому React Navigation выступает ключевым инструментом для обеспечения навигации внутри приложения. Параметры передаются через метод navigate, так как это позволяет динамически адаптировать содержимое экранов на основе пользовательского ввода и минимизирует дублирование кода. Такая архитектура облегчает добавление новых экранов и маршрутов.

Интеграция React Navigation поддерживает централизованное управление навигацией при помощи файла Navigation, который содержит в себе настройки и информацию о всех экранных переходах.

2. Результаты и обсуждение

Когда пользователь открывает приложение, он видит стартовое окно, на котором у него есть возможность изменить цветовую схему. Все экраны приложения отображаются корректно, навигация осуществляется согласно дереву решения. Пример экрана приложения изображен на рис. 3.

В мобильном приложении реализованы следующие основные модули:

1. Модуль «Обучение» позволяет ознакомиться с алгоритмом и изучить клинические рекомендации, сформированные на основе симптомов пациента;

2. Модуль «Оказание медицинской помощи» запрашивает данные о названии медицинской организации, номере истории болезни пациента и кратком анамнезе, а после открывает доступ к алгоритму поддержки принятия решений. Пользователь отвечает на вопросы с выбором вариантов и получает таким образом полный список рекомендаций для текущего пациента;

3. Модуль «Оборудование для эндоскопического гемостаза» полный перечень необходимого оборудования для выполнения процедуры эндоскопического гемостаза;

4. Модуль «Оценить качество приложения» предоставляет возможность пройти опрос и дать обратную связь о работе приложения и алгоритма;

5. Модуль «Справочная информация» включает полные сведения о разработчиках приложения, их квалификации и участии в проекте.

Разработанное приложение обладает высоким потенциалом для интеграции в повседневную клиническую практику. Оно обеспечивает врачей быстрым доступом к актуальным клиническим рекомендациям и алгоритму, что ускоряет принятие решений в условиях ограниченного времени. Список необходимого оборудования помогает оптимизировать подготовку к процедурам, снижая риск упущений. Кроме того, функционал обратной связи позволяет пользователям делиться опытом работы с приложением, что способствует его постоянному совершенствованию.

Рис. 3. Главный экран в светлой теме

Заключение

В ходе работы было реализовано мобильное приложение для поддержки принятия врачебных решений в процессе эндоскопического гемостаза, которое включает качественную рекомендательную систему. Планируется поэтапное внедрение приложения в медицинские учреждения Воронежа и Воронежской области. На первом этапе будет проведено пилотное тестирование в ряде больниц, с последующей адаптацией на основе обратной связи.

Литература

1. Общероссийская общественная организация Российское общество хирургов. Язвенные гастродуоденальные кровотечения. Клинические рекомендации. Приняты на Общероссийской согласительной конференции по принятию национальных клинических рекомендаций, 6 июня 2014 года, г. Воронеж. – Воронеж : 2014.

2. *Петров Ю. В.* Хирургическая тактика при язвенных гастродуоденальных в зависимости от вида кровотечения: дис. …. канд.мед.наук / Ю. В. Петров. – Уфа, 2016. – 139 с.

3. React Native Documentation // React Native URL: https://reactnative.dev/ (дата обращения: 12.10.2024).

4. Expo CLI Documentation // Expo CLI URL: https://docs.expo.dev/ (дата обращения: 12.10.2024).

УДК 004.622

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ДАННЫХ ОБ АКТИВНОСТЯХ СТУДЕНТОВ В РАЗНЫХ ЦИКЛАХ ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ НА ОСНОВЕ ИНСТРУМЕНТОВ КЛАСТЕРНОГО АНАЛИЗА

А. А. Гамбаров, Н. А. Демидов

МИРЭА – Российский технологический университет

Аннотация. В данной работе проведен сравнительный анализ активности студентов в системе дистанционного обучения программированию на Python (ЦАП) за 2023 и 2024 годы с использованием методов кластеризации. Были извлечены 33 признака из журналов событий, описывающих активность студентов по 11 задачам.
Ключевые слова: python, кластеризация, k-means, UMAP, анализ данных, журналы событий, активность студентов, метрики качества кластеризации, ЦАП.

В современном мире дистанционное обучение стало неотъемлемой частью образовательного процесса, особенно в сфере программирования. Системы дистанционного решения задач, такие как Цифровой Ассистент Преподавателя (ЦАП), играют ключевую роль в обеспечении эффективного обучения студентов. ЦАП представляет собой автоматизированную систему, которая предлагает каждому студенту 11 уникальных разнотипных задач по программированию на языке Python, проверяет их решения и фиксирует результаты в журналах событий. Эти журналы содержат важную информацию о деятельности студентов, включая время решения задач, количество попыток и способы решения [1].

Анализ данных, собранных в журналах ЦАП, позволяет выявить закономерности в активности студентов, определить успешных и неуспешных студентов, а также выявить потенциальные проблемы в процессе обучения. Одним из эффективных инструментов анализа таких данных является кластеризация, которая позволяет группировать студентов по схожим признакам, таким как время решения задач, количество попыток и другие [3].

В предлагаемом исследовании был выполнен анализ данных, собранных в журналах ЦАП за 2023 и 2024 годы, с использованием алгоритма кластеризации k-means [4]. Цель работы заключалась в выявлении различий/сходства в активности студентов в этих двух циклах обучения. Для достижения поставленной цели из журналов событий посредством предварительной обработки для каждой из 11 задач были извлечены три признака, характеризующие время успешной загрузки решения задачи, длительность решения задачи и общее число попыток при решении задачи. В результате были сформированы 2 набора данных, каждый из которых содержал 33 признака. Эти наборы соответствовали 2023 и 2024 года и содержали соответственно по 1056 и 1111 записей, описывающих активности студентов, успешно сдавших системе ЦАП все 11 задач. Эти наборы данных были объединены в один. К новому объединенному набору данных был применен алгоритм k-means. При этом оптимальное число кластеров было определено с использованием индекса кластерного силуэта. Кроме того, этот набор данных был разбит на 11 поднаборов, соответствующих 11 задачам. К этим поднаборам данных также был применен алгоритм k-means.

При выполнении исследования была разработана программа на языке Python в среде Jupyter Notebook. Программа выполняет сравнение активности студентов в решении задач по программированию на Python в течение двух циклов обучения в 2023 и 2024 годах и определяет, можно ли корректно разделить активности, относящиеся к разным циклам с помощью инструментов кластеризации. При этом активность может описывать как все 11 задач, так и каждую задачу по отдельности.

Основные этапы работы программы включают в себя следующие этапы.

1. **Загрузка и предобработка данных.** Программа загружает данные из журналов ЦАП за 2023 и 2024 годы. Каждый набор данных содержит информацию о 11 задачах, для каждой из которых извлекаются три признака: время успешной загрузки решения задачи, длительность решения задачи и общее число попыток при решении задачи. В результате для каждого цикла обучения формируется набор, содержащий 33 признаков, характеризующих активности студентов. Наборы данный объединяются в один общий с сохранением меток, определяющих разные циклы обучения.

2. **Кластеризация с использованием k-means.** Программа реализует кластеризации данных с использованием алгоритма k-means, который является одним из наиболее распространенных алгоритмов неконтролируемого обучения. Для выбора оптимального числа кластеров k программа использует индекс кластерного силуэта, который должен быть максимизирован.

3. **Оценка качества кластеризации.** Программа реализует расчет различных метрик, позволяющих оценить качество кластеризации. В частности, вычисляется индекс кластерного силуэта silhouette, который максимизируется. При его расчете считается, что метки кластеров неизвестны. Кроме того, вычисляются такие метрики, как homogeneity, completeness, V-measure, ARI (Adjusted Rand Index), AMI (Adjusted Mutual Information), позволяющие сопоставить предсказанные метки кластеров фактическим.

4. **Визуализация результатов.** Программа использует UMAP (Uniform Manifold Approximation and Projection) алгоритм [5] для визуализации результатов кластеризации. Визуализация помогает наглядно представить распределение данных и выявить различия между кластерами.

На рис. 1–12 приведены результаты визуализации исследуемых наборов данных с применением UMAP алгоритма как для всех 11 задач, так и для каждой задачи в отдельности. На рис. 1–12, слева маркерами разного цвета изображены предсказанные с использованием алгоритма k-means кластеры. На рис. 1–12, справа маркерами разного цвета изображены фактические кластеры, соответствующие 2 циклам обучения.

Результаты кластеризации показали, что разделить данные на кластеры, соответствующие годам 2023 и 2024, не удалось ни в целом по всем 11 задачам, ни по отдельности по каждой задаче. Кластеры, полученные в результате кластеризации, не соответствуют циклам обучения. При этом предсказанное число кластеров составляло от 2 до 4. В случае больших значений такой метрики качества кластеризации, как индекс кластерного силуэта для некоторых задач, можно предположить, что сформированные кластеры определялись, например, с учетом способностей студентов решать задачи быстро при малом числе ошибочных попыток и медленно

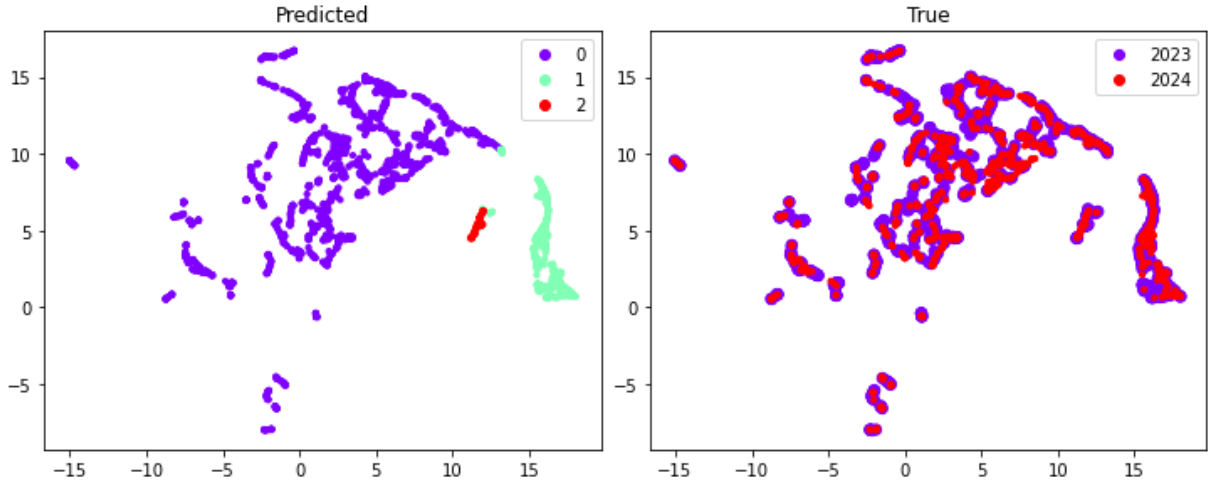


Рис. 1. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 11 задачам: homogeneity=0.002, completeness=0.002, V-measure=0.002, ARI=0.001, AMI=0.002, silhouette=0.448

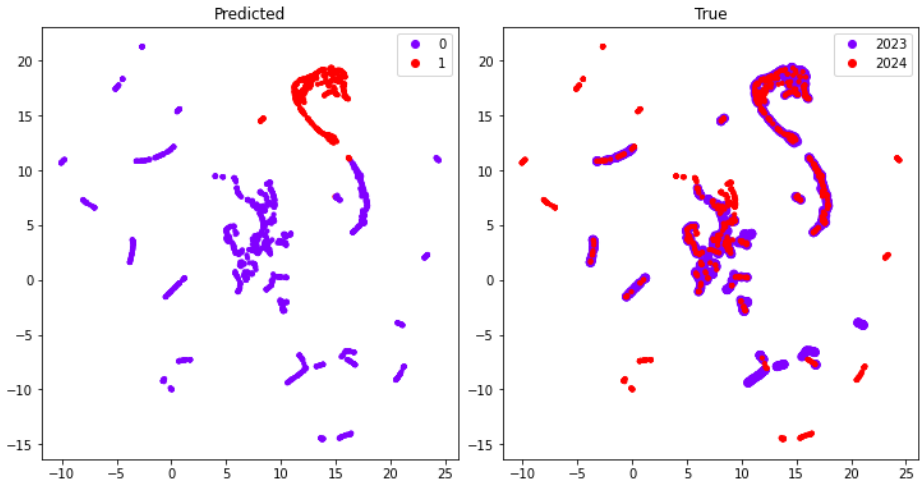


Рис. 2. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 1-й задаче: homogeneity=0.006, completeness=0.007, V-measure=0.007, ARI=0.001, AMI=0.006, silhouette=0.825

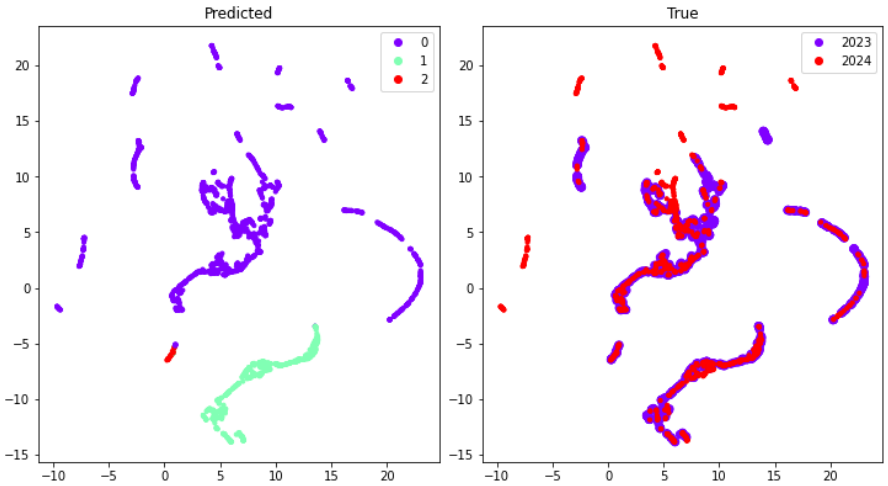


Рис. 3. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 2-й задаче: homogeneity=0.0002, completeness=0.0003, V-measure=0.0002, ARI=0.0003, AMI=–0.0001, silhouette=0.828

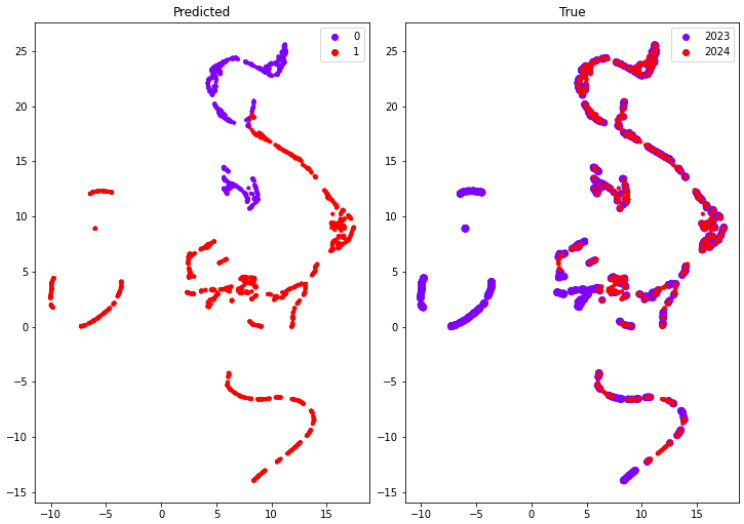


Рис. 4. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 3-й задаче: homogeneity=0.0003, completeness=0.0003, V-measure=0.0003, ARI=–0.0006, AMI=–0.0004, silhouette=0.826

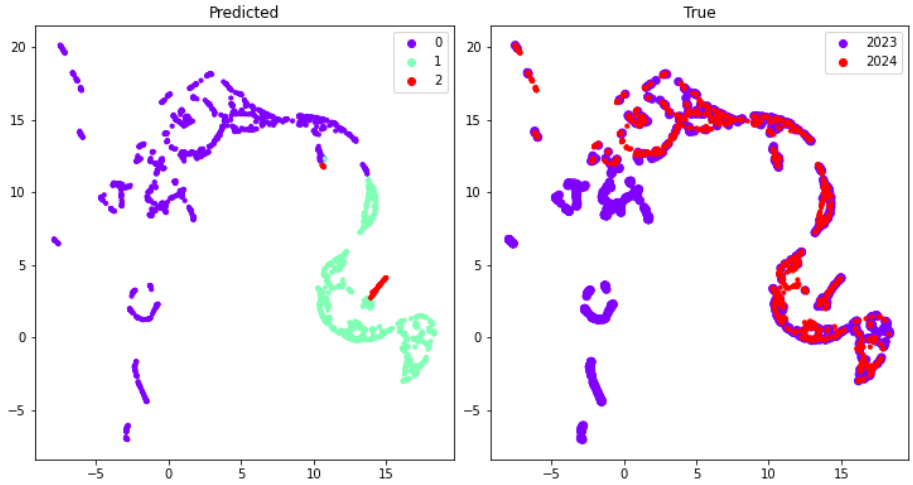


Рис. 5. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 4-й задаче: homogeneity=0.006, completeness=0.007, V-measure=0.006, ARI=0.004, AMI=0.006, silhouette=0.812

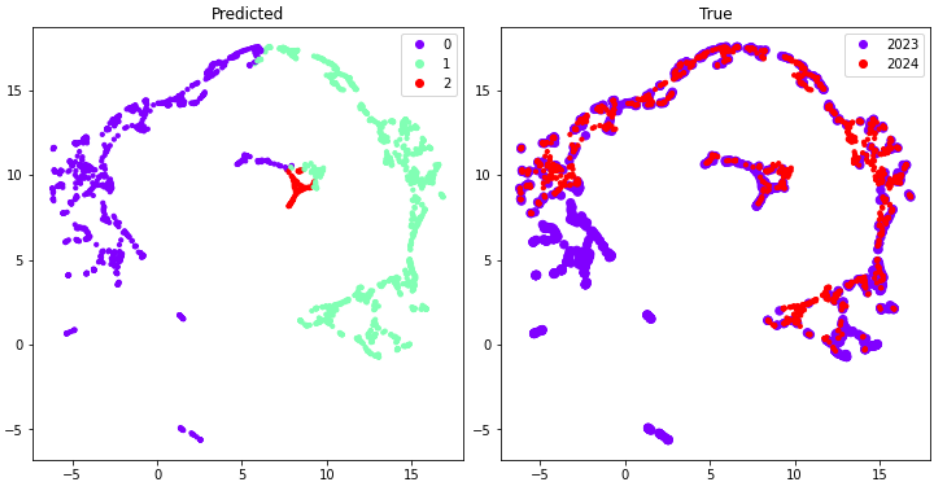


Рис. 6. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 5-й задаче: homogeneity=0.019, completeness=0.017, V-measure=0.018, ARI=0.019, AMI=0.017, silhouette=0.756

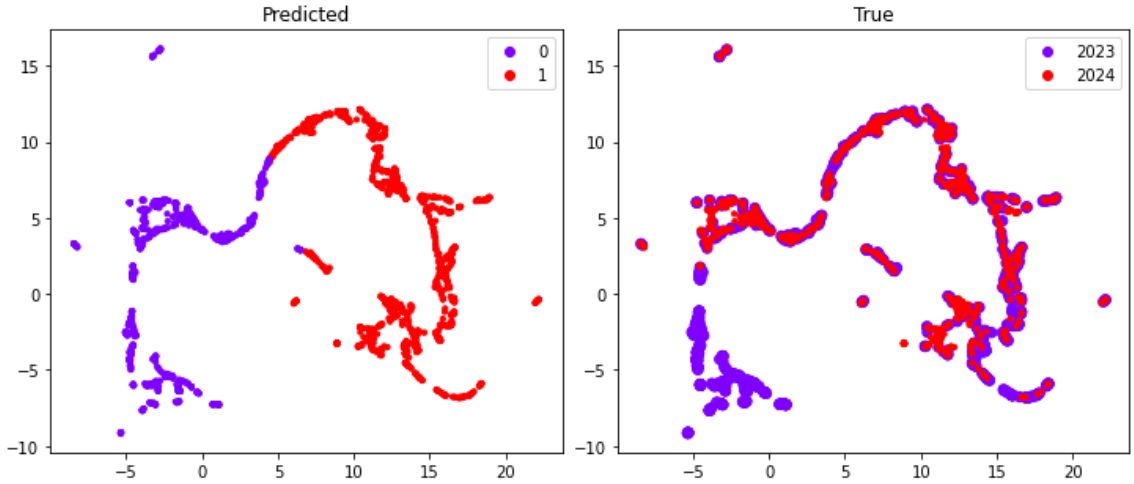


Рис. 7. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 6-й задаче: homogeneity=0.034, completeness=0.029, V-measure=0.031, ARI=0.043, AMI=0.031, silhouette=0.707

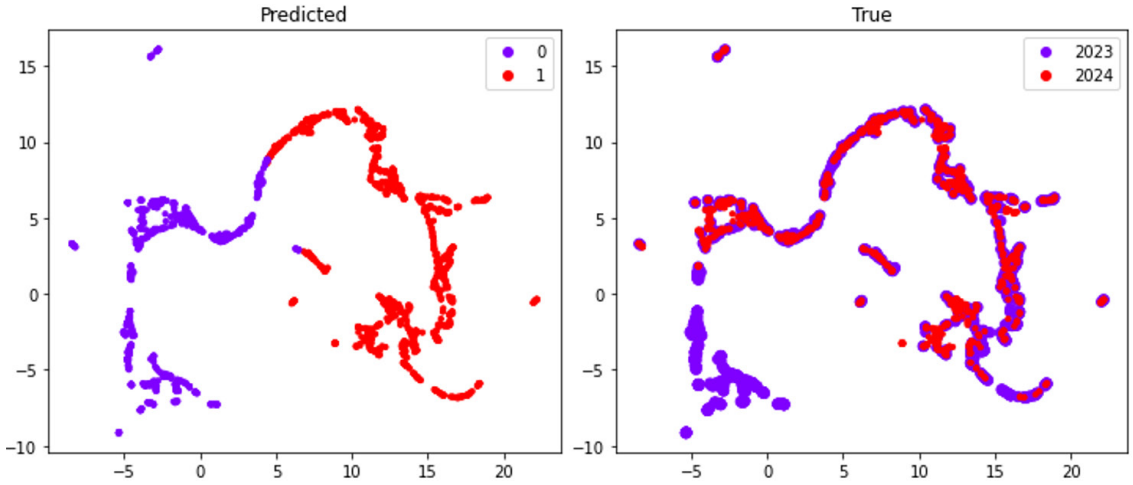


Рис. 8. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 7-й задаче: homogeneity=0.026, completeness=0.027, V-measure=0.026, ARI=0.036, AMI=0.026, silhouette=0.695

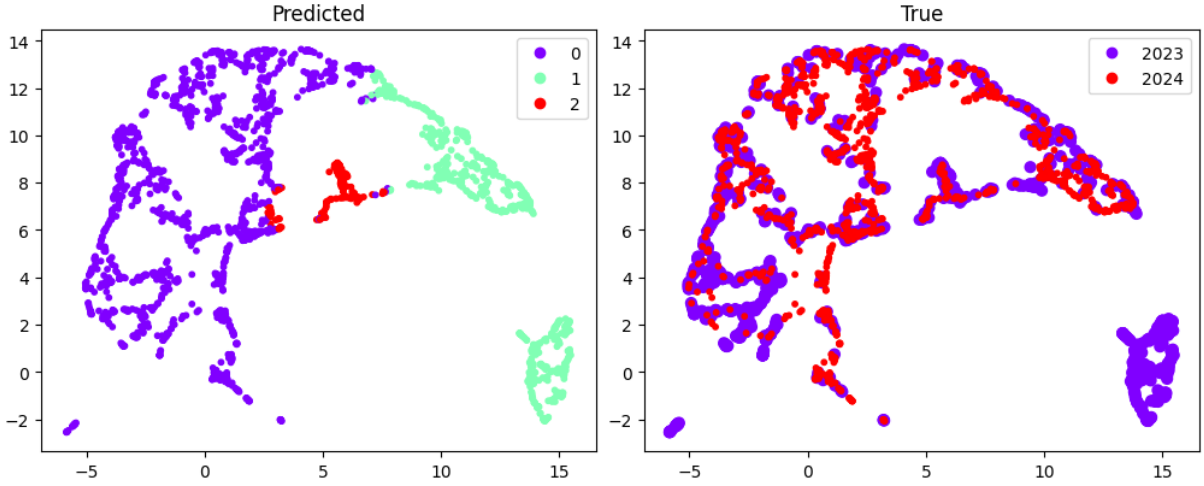


Рис. 9. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 8-й задаче: homogeneity=0.029, completeness=0.024, V-measure=0.026, ARI=0.038, AMI=0.026, silhouette=0.647

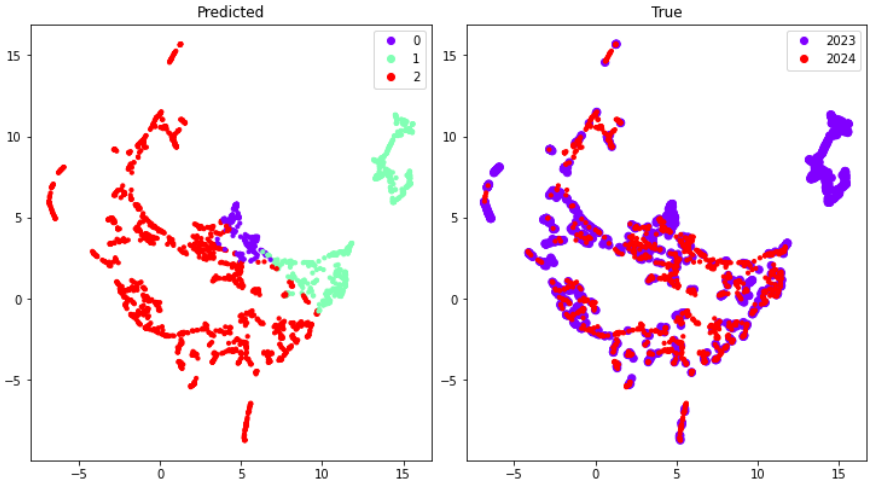


Рис. 10. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 9-й задаче: homogeneity=0.028, completeness=0.023, V-measure=0.026, ARI=0.034, AMI=0.025, silhouette=0.586

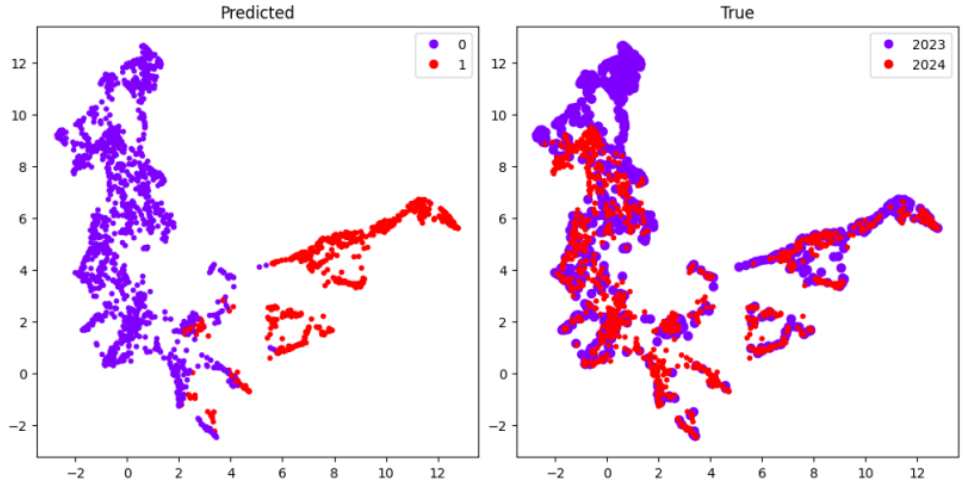


Рис. 11. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 10-й задаче: homogeneity=0.034, completeness=0.029, V-measure=0.031, ARI=0.036, AMI=0.030, silhouette=0.537

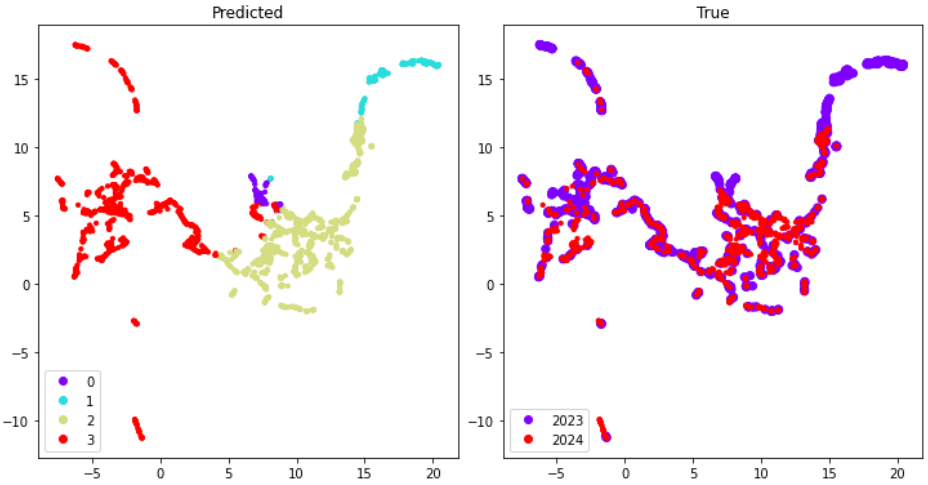


Рис. 12. Визуализация предсказанных и фактических кластеров на основе UMAP алгоритма для набора данных, соответствующего 11-й задаче: homogeneity=0.165, completeness=0.106, V-measure=0.129, ARI=0.095, AMI=0.128, silhouette=0.551

при большом числе ошибочных попыток. Малые значения других метрик качества кластеризации свидетельствуют о плохой разделимости активностей студентам по 2 циклам обучения (как в целом по всем 11 задачам, так и по отдельности по каждой задаче).

Таким образом, результаты исследования свидетельствуют о том, что активности студентов в решении задач по программированию на Python в 2023 и 2024 годах похожи между собой. Кластеризация данных не позволила разделить активности студентов на группы, соответствующие разным циклам обучения, что указывает на отсутствие значительных различий в активностях студентов в этих циклах.

Литература

1. *Горчаков А. В., Демидова Л. А., Советов П. Н.* Интеллектуальный учёт учебных достижений в системе цифровой ассистент преподавателя // International Journal of Open Information Technologies. – 2023. – Т. 11, № 4. – С. 106–115.

2. *Wang C., Dai J., Xu L.* Big data and data mining in education: a bibliometrics study from 2010 to 2022 // 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), IEEE. – 2022. – P. 507–512.

3. *Demidova L. A., Sovietov P. N., Andrianova E. G., Demidova A. A.* Anomaly Detection in Student Activity in Solving Unique Programming Exercises: Motivated Students against Suspicious Ones // Data. – 2023. – Vol. 8, No 8. – P. 129. – https://doi.org/10.3390/data8080129.

4. *Sinaga K. P., Yang M.-S.* Unsupervised K-Means Clustering Algorithm // IEEE Access. – 2020. – Vol. 8. – P. 80716–80727.

5. *McInnes L., Healy J., Melville J.* UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction // arXiv:1802.03426. – 2018.

УДК 512.622

ВЫЧИСЛЕНИЕ КОЭФФИЦИЕНТОВ БИНОМИАЛЬНОГО РАЗЛОЖЕНИЯ МНОГОЧЛЕНА В СКМ MAPLE

Я. В. Гамилов, И. Н. Попов

Северный (Арктический) федеральный университет имени М. В. Ломоносова

Аннотация. Различные разложения многочленов используются в решении многих задач, связанных как с исследованиями самих многочленов, так и их применениями. Под разложением понимается представление многочлена в виде линейной комбинации определённых многочленов со скалярными коэффициентами из коммутативного кольца с единицей. Одним из разложений является биномиальное разложение. В работе предлагается способ вычисления коэффициентов биномиального разложения по известным коэффициентам исходного многочлена, который записывается в стандартной форме. Компьютерная реализация вычислений искомых коэффициентов осуществлена в СКМ Maple. Итогом реализации является представление многочлена в биномиальном виде.

Ключевые слова: многочлен, стандартная запись многочлена, линейная комбинация многочленов со скалярными коэффициентами, разложение многочлена, биномиальный коэффициент, биномиальное разложение многочлена, рекуррентная формула, комбинаторика, алгоритм разложения, СКМ Maple.

Введение

При выяснении свойств многочленов и их применения в решении задач нередко многочлен раскладывается по многочленам определённого множества. Под разложением понимается представление многочлена в виде линейной комбинации данных многочленов со скалярными коэффициентами, которые, в общем случае, принадлежат коммутативному кольцу с единицей [1]. Разложения используются для выяснения свойств приводимости многочлена, вычисления значений многочлена и его производных, а также приближенных значений, в решении задач аппроксимации и так далее [2]. Одним из разложений является биномиальное разложение, которое, в частности, используется для решения комбинаторных задач (например, суммирование степеней натурального ряда) и в качестве способа многочленного шифрования.

Ставится задача об определении коэффициентов биномиального разложения по коэффициентам исходного многочлена, записанного в стандартном виде. Решение этой задачи приводится в рекуррентном виде с использованием СКМ Maple [3]. Для получения формул вычисления используются знания о комбинаторных элементах [4].

1. Теоретические аспекты

Представление многочлена, записанного в стандартной форме,

$$\sum_{k=0}^{n} a_k x^{n-k} = \sum_{k=0}^{n} b_k C_x^{n-k}, \text{ где } C_x^k = \begin{cases} \dfrac{x(x-1)(x-2)\ldots(x-(k-1))}{k!}, k > 0, \\ 1, k = 0, \end{cases}$$

называется биномиальным разложением. Слагаемые исходного многочлена записываются по убыванию их степеней от n до 0.

Для вычисления коэффициентов биномиального разложения составляются две вспомогательные таблицы.

А) В таблице A (табл. 1) построчно записываются коэффициенты произведения $x(x-1)(x-2)\ldots$, которые вычисляются рекуррентно по формуле $A_{i,j+1} = -(i-1)A_{i-1,j} + A_{i-1,j+1}$

(два первых слева столбца — вспомогательные, третий столбец состоит из единиц, так как при раскрытии скобок в выражении $x(x-1)(x-2)\ldots$ старший коэффициент равен 1; каждая строка заканчивается нулём для удобства вычислений).

Таблица 1

Таблица коэффициентов разложения произведения $x(x-1)(x-2)\ldots$

x	–1	1	0				
$x(x-1)$	–2	1	–1	0			
$x(x-1)(x-2)$	–3	1	–3	2	0		
$x(x-1)(x-2)(x-3)$	–4	1	–6	11	–6	0	
$x(x-1)(x-2)(x-3)(x-4)$	–5	1	–10	35	–50	24	0
$x(x-1)(x-2)(x-3)(x-4)(x-5)$	–6	1	–15	85	–225	274	–120	0
...							

Б) В таблице B в столбцы записываются скаляры, вычисляемые по рекуррентной формуле, опираясь на элементы матрицы A:

$$B_{i,j} = -\sum_{k=j}^{i-1} B_{k,j} A_{n-k+1,i-k+1},\ B_{i,i} = 1 \text{ для всех } i = 1\ldots n, j = 1\ldots n, j-1 < i..$$

Например, для случая $n = 6$ элементы таблицы B указаны в табл. 2.

Таблица 2

Таблица B (случай $n = 6$)

1					
15	1				
65	10	1			
90	25	6	1		
31	15	7	3	1	
1	1	1	1	1	1

В этом случае коэффициенты b_k биномиального разложения выражаются через коэффициенты a_k исходного многочлена следующим образом:

$b_6 = 6!\cdot a_6$

$b_5 = 5!\cdot(15a_6 + a_5)$

$b_4 = 4!\cdot(65a_6 + 10a_5 + a_4)$

$b_3 = 3!\cdot(90a_6 + 25a_5 + 6a_4 + a_3)$

$b_2 = 2!\cdot(31a_6 + 15a_5 + 7a_4 + 3a_3 + a_2)$

$b_1 = 1!\cdot(a_6 + a_5 + a_4 + a_3 + a_2 + a_1)$

$b_0 = 0!\cdot a_0$

Отметим, что для любого n последняя строка таблицы B состоит из единиц и свободные члены разложений совпадают.

2. Практическая реализация в СКМ Maple

Предлагаются программы в СКМ Maple для вычисления элементов таблиц A и B и вывода результатов.

Программа 1

Генерация матриц A и B

```
restart:
n:=6:
# Generating product coefficients x(x-1)(x-2)...(x-(n-1)), the matrix A
A[1]:=[1,0]:
for i from 2 to n do
   A[i]:=[1];
   for j from 1 to i-1 do
      A[i]:=[op(A[i]),-(i-1)*A[i-1][j]+A[i-1][j+1]];
   od:
   A[i]:=[op(A[i]),0]:
od:
# Generating matrix elements B
for j from 1 to n do B[j]:=[] od:
for j from 1 to n do
   B[j]:=[op(B[j]),1];
   for i from j+1 to n do
      S:=-sum(B[k][j]*A[n-k+1][i-(k-1)],k=j..i-1):
      B[i]:=[op(B[i]),S];
   od:
od:
seq(B[i],i=1..n);
for i from 1 to n do c[i]:=[]: od:
for i from 1 to n do
   c[n-i+1]:=sum(B[i]['j']*a['n-j+1'],'j'=1..nops(B[i]))
od;
```

Результат работы программы для случая $n = 6$ представлен на рис. 1.

$$[1], [15, 1], [65, 10, 1], [90, 25, 6, 1], [31, 15, 7, 3, 1], [1, 1, 1, 1, 1, 1]$$

$$c_6 := a_6$$

$$c_5 := 15\,a_6 + a_5$$

$$c_4 := 65\,a_6 + 10\,a_5 + a_4$$

$$c_3 := 90\,a_6 + 25\,a_5 + 6\,a_4 + a_3$$

$$c_2 := 31\,a_6 + 15\,a_5 + 7\,a_4 + 3\,a_3 + a_2$$

$$c_1 := a_6 + a_5 + a_4 + a_3 + a_2 + a_1$$

Рис. 1. Выражение составных частей коэффициентов биномиального разложения

Программа 2

Биномиальное разложение многочлена

```
# Expression of b_n,...,b_1,b_0 in terms of a_n,...,a_1,a_0
# Input data by volume n: a[...]; output data by volume n: b[...]
a:=[1,1,1,1,1,1,1]:
b:=[]:
```

```
5 for i from 1 to n do
6    b:=[op(b), sum(a[k]*B[i][k]*(n-i+1)!,k=1..i)];
7 od:
8 b:=[op(b),a[n+1]]:
9 sort(sum('a[k]*x^(n-k+1)','k'=1..n+1))=sum('b[k]*C[x]^(n-k+1)','k'=1..
  n+1);
```

Комментарии к программе 2.

1) В строке 3 задаются коэффициенты исходного многочлена, начиная с наивысшей степени одночлена и до нулевой (до константы).

2) В строках 4–8 генерируются коэффициенты при биномиальных слагаемых на основе коэффициентов исходного многочлена.

3) В строке 9 осуществляется вывод результата — запись исходного многочлена и его биномиальное разложение.

Результат работы программы представлен на рис. 2.

$$a := [1, 1, 1, 1, 1, 1, 1]$$

$$x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 = 720\, C_x{}^6 + 1920\, C_x{}^5 + 1824\, C_x{}^4 + 732\, C_x{}^3 + 114\, C_x{}^2 + 6\, C_x + 1$$

Рис. 2. Биномиальное разложение многочлена

Результатом реализации программы 2 является представление многочлена в биномиальном виде.

3. Приложение

Биномиальное разложение используется, в частности, для вычисления сумм чисел натурального ряда.

Пример. Пусть многочлен $f(x)$ записан в биноминальном виде:

$$f(x) = b_k C_x^k + b_{k-1} C_x^{k-1} + \cdots + b_0 C_x^0.$$

Используя равенство

$$C_0^r + C_1^r + C_2^r + \cdots + C_n^r = C_{n+1}^{r+1},$$

получаем

$$f(0) + f(1) + f(2) + \cdots + f(n) = b_k C_{n+1}^{k+1} + b_{k-1} C_{n+1}^k + \cdots + b_0 C_{n+1}^1.$$

Используя указанные выше предложенные программы, получаем результат для случая $f(x) = x^3$ (рис. 3).

$$a := [1, 0, 0, 0]$$

$$x^3 = 6\, C_x{}^3 + 6\, C_x{}^2 + C_x$$

Рис. 3. Биномиальное разложение многочлена $f(x) = x^3$

Так как $C_n^m = \dfrac{n!}{m!(n-m)!}$, то

$$0^3 + 1^3 + 2^3 + \cdots + n^3 = 6C_{n+1}^4 + 6C_{n+1}^3 + C_{n+1}^2 = \left(\frac{(n+1)n}{2}\right)^2.$$

Следовательно, получаем формулу суммирования кубов натурального ряда ■

Отметим, что биномиальное разложение также может рассматриваться как пример многочленного шифрования по схеме $a_n a_{n-1} \ldots a_0 \leftrightarrow b_n b_{n-1} \ldots b_0$.

Заключение

Итогом работы является компьютерное решение в СКМ Maple задачи о нахождении биномиального разложения многочлена.

Литература

1. *Гриншпон С. Я.* Многочлены над областями целостности (теория и приложения) / С. Я. Гриншпон, И. Э. Гриншпон. – Томск : Издательский Дом Томского государственного университета, 2016. – 152 с.

2. *Прасолов В. В.* Многочлены / В. В. Прасолов. – Москва : МЦНМО, 2003. – 336 с.

3. *Говорухин В. Н.* Введение в Maple. Математический пакет для всех / В. Н. Говорухин, В. Г. Цибулин. – Москва : Мир, 1997. – 208 с.

4. *Виленкин И. Я.* Комбинаторика / А. И. Виленкин, П. А. Виленкин. – Москва : ФИМА, МЦНМО, 2006. – 400 с.

УДК 004.62

ЭФФЕКТИВНАЯ МАРШРУТИЗАЦИЯ И УПРАВЛЕНИЕ МЕТАДАННЫМИ В СИСТЕМАХ ХРАНЕНИЯ НА БАЗЕ TELEGRAM

И. А. Грошев

Воронежский государственный университет

Аннотация. В статье рассматриваются подходы к маршрутизации и управлению метаданными в распределенных системах хранения данных, построенных на базе Telegram. Описаны методы сохранения метаданных: в текстовых сообщениях и внешних базах данных. Проанализированы стратегии поиска информации о фрагментах, включая стандартизированное именование, использование хэшей и организацию дерева каталогов. Полученные результаты позволяют оптимизировать процессы хранения и восстановления данных, повышая надежность и удобство использования Telegram как распределенного хранилища.

Ключевые слова: распределенное хранилище данных, Telegram как система хранения, метаданные, фрагменты данных, управление метаданными, хэширование данных, контроль целостности, внешние базы данных, поиск по метаданным, стандартизированное именование, репликация данных, масштабирование системы, архитектура хранения данных, оптимизация восстановления файлов.

Введение

В последние годы мессенджеры, такие как Telegram, стали неотъемлемой частью нашей цифровой жизни. Их удобство и функциональность делают их привлекательными платформами не только для общения, но и для хранения данных. Однако, при использовании мессенджеров для хранения больших объемов информации возникает проблема эффективной организации и поиска данных. В данной статье рассматривается задача управления метаданными в контексте использования Telegram как хранилища данных.

1. Методы хранения метаданных в Telegram

Метаданные играют ключевую роль в организации и управлении данными в распределенных системах хранения, особенно в случаях, когда данные разбиваются на множество частей, как в случае с Telegram. Эффективное хранение и поиск метаданных — это важный аспект, который позволяет обеспечивать целостность данных, их быстроту и доступность при восстановлении. Метаданные, в частности, содержат информацию о файлах (например, имя, размер), структуру распределения данных, идентификаторы сообщений или ссылок, а также контрольные суммы для проверки целостности. В случае использования Telegram в качестве распределенного хранилища важно грамотно организовать хранение этих данных, учитывая особенности самой платформы и ограниченные возможности устройства пользователя. Для этого могут быть использованы различные подходы, включая хранение метаданных в текстовых сообщениях и использование внешних баз данных.

1.1. Использование сообщений Telegram

Один из интуитивных способов хранения метаданных — использование сообщений в Telegram. Текстовые сообщения могут содержать структурированную информацию о файлах, такую как имя, размер, дата создания и хэш. Однако у этого метода есть несколько проблем.

Во-первых, существуют задержки при поиске сообщений с метаданными, так и сложность их инерпритации. Во-вторых, возникает проблема разбивания сообщений: Telegram ограничивает размер сообщения, поэтому длинные метаданные будут разделены на несколько частей. Это требует разработки фрейма метаданных, который будет учитывать эту особенность, поддерживая разбивку и обеспечение целостности данных.

Пример фрейма метаданных:

[Номер фрейма/Общий номер фрейма][Дата и время][Строка метаданных]

Каждое сообщение будет содержать часть метаданных, а система будет собирать и обрабатывать их в правильном порядке, обеспечивая восстановление исходной информации.

1.2. Внешние базы данных

Использование внешних баз данных позволяет хранить более объемные и сложные метаданные. Такие базы данных, как SQLite, PostgreSQL или MongoDB, предлагают гибкие возможности для организации и поиска данных. Информация о файлах может быть представлена в виде таблиц с полями, соответствующими различным атрибутам метаданных.

Выбор базы данных для локального устройства зависит от специфики проекта, включая платформу, требования к производительности и объему данных. SQLite является предпочтительным выбором для большинства локальных приложений, в то время как Realm Database и PouchDB могут быть более эффективными для мобильных приложений с высокими требованиями к скорости доступа и синхронизации данных.

2. Методы поиска информации о фрагментах

Поиск фрагментов метаданных может быть организован различными способами в зависимости от используемой технологии хранения данных. Рассмотрим два подхода: поиск через API Telegram и использование внешних баз данных.

2.1. Поиск через API Telegram

Telegram API предоставляет возможность взаимодействия с сообщениями через методы, такие как *getMessages*, что позволяет извлекать и искать метаданные внутри сообщений. Однако для поиска фрагментов метаданных в Telegram необходимо учитывать особенности этой платформы.

Одной из основных проблем является ограничение длины сообщений в Telegram. Из-за этого метаданные могут быть разбиты на несколько сообщений. В этом случае важно учитывать порядок сообщений, чтобы собрать все фрагменты в правильном порядке. Для этого можно использовать уникальные идентификаторы или метки, которые будут привязаны к каждому фрагменту, и обеспечат восстановление исходных данных.

Поиск фрагментов можно осуществлять с помощью фильтрации по ключевым словам или меткам. Например, можно искать сообщения, содержащие текст, связанный с метаданными, такие как «имя файла», «размер» или «хэш». Такой подход полезен, если метаданные имеют определённую структуру, позволяя находить сообщения по заранее определённым параметрам.

Тем не менее, при поиске по большому объёму сообщений могут возникнуть задержки и проблемы с производительностью. Метод *getMessages* не всегда оптимален для быстрого поиска, особенно если сообщения разбросаны по множеству чатов или каналов. Чтобы уменьшить задержки, можно использовать индексацию сообщений, сохраняя информацию о фрагментах

в базе данных и связывая её с идентификаторами сообщений в Telegram. Это позволит восстанавливать данные быстрее, не обращаясь к API при каждом запросе.

2.2. Поиск через внешние базы данных

Использование внешних баз данных для хранения метаданных предоставляет больше возможностей для эффективного поиска и управления фрагментами. Рассмотрим несколько популярных баз данных.

SQLite является оптимальным решением для локальных приложений, где важно быстро хранить и искать метаданные. В этой базе данных можно организовать структуру с таблицами, содержащими поля для номера фрейма, идентификатора файла, даты и времени получения данных, а также строк метаданных. Запросы к базе данных позволяют быстро находить нужные фрагменты и восстанавливать оригинальные данные. Это решение особенно полезно для небольших и средних объёмов данных, где требуется высокая производительность при поиске.

PostgreSQL и MongoDB подходят для более крупных проектов с высокими требованиями к производительности и масштабируемости. PostgreSQL поддерживает реляционные структуры данных, что делает её идеальной для сложных запросов, а MongoDB используется для хранения данных в виде документов. Обе эти базы данных поддерживают индексацию, что позволяет эффективно искать фрагменты, даже если объём данных значительно велик. Например, можно искать фрагменты метаданных по таким атрибутам, как «имя файла» или «хэш», а также восстанавливать данные по порядку фреймов, что существенно упрощает работу с большими объёмами информации.

Realm Database и PouchDB предназначены для мобильных приложений и предлагают быстрые способы синхронизации данных с сервером. Они обеспечивают высокую скорость доступа к локальному хранилищу и удобство работы с данными на мобильных устройствах. В таких базах данных также можно эффективно управлять фрагментами метаданных и быстро выполнять поисковые запросы, что делает их отличным выбором для приложений с высокими требованиями к синхронизации и производительности.

Использование внешних баз данных обладает рядом преимуществ, таких как гибкость, высокая производительность и поддержка сложных запросов. В отличие от Telegram, где необходимо учитывать ограничения на размер сообщений, внешние базы данных позволяют хранить более сложные структуры данных и выполнять поиск с минимальными задержками.

Заключение

В данной статье рассмотрены два подхода к хранению и поиску метаданных: использование сообщений в Telegram и внешних баз данных. Методы, основанные на Telegram, удобны для простых решений, но имеют ограничения по длине сообщений и требуют дополнительных усилий для управления фрагментами данных. Внешние базы данных, такие как SQLite, PostgreSQL и MongoDB, предоставляют более гибкие и масштабируемые решения для работы с большими объёмами данных и сложными запросами. Выбор подхода зависит от специфики проекта: Telegram оптимален для обмена данными в реальном времени, а базы данных — для сложных и масштабируемых систем.

Литература

1. A Digital Signature Based on a Conventional Encryption Function. – 1988. – URL: https://ieeexplore.ieee.org/document/5371620/сенкс (дата обращения: 20.11.2024).

2. The Google File System. – 2003. – URL: https://dl.acm.org/doi/10.1145/940072.940073/ (дата обращения: 20.11.2024).

3. The Design and Implementation of Data Caching Systems for High-Performance Applications. – 2007. – URL: https://dl.acm.org/doi/10.1145/1270001.1270011/ (дата обращения: 20.11.2024).

4. SQLite: The Definitive Guide. O'Reilly Media. – URL: https://www.sqlite.org/docs.html/ (дата обращения: 21.11.2024).

5. Efficient Search Techniques in Large Distributed Systems. – URL: https://www.sciencedirect.com/science/article/pii/S0743731504000810/ (дата обращения: 21.11.2024).

6. Optimal File Chunking and Retrieval Algorithms for Distributed File Systems. – URL: https://ieeexplore.ieee.org/document/5070422/ (дата обращения: 21.11.2024).

7. Automatic Information Retrieval. – URL: https://www.researchgate.net/publication/220539160_Automatic_Information_Retrieval/ (дата обращения: 21.11.2024).

8. Telegram Bot API Documentation. – URL: https://core.telegram.org/bots/api (дата обращения: 21.11.2024).

9. Database Systems: Design, Implementation, and Management. – URL: https://www.cengage.com/c/database-systems-design-implementation-and-management-12e-connolly/ (дата обращения: 21.11.2024).

10. Patterns of Enterprise Application Architecture. – URL: https://martinfowler.com/eaaCatalog/ (дата обращения: 21.11.2024).

УДК 004-4

BLOC И CUBIT ВО FLUTTER: УПРАВЛЕНИЕ СОСТОЯНИЕМ ПРИЛОЖЕНИЯ

Н. В. Денисов, Н. А. Каплиева

Воронежский государственный университет

Аннотация. В статье рассматриваются паттерны управления состоянием BLoC и Cubit, используемые в разработке приложений на Flutter. BLoC предоставляет мощный механизм управления через события и состояния, в то время как Cubit упрощает реализацию для менее сложных сценариев. Описываются основные отличия, преимущества и области применения каждого подхода. Приводятся примеры кода, демонстрирующие их использование. Статья помогает выбрать подходящий инструмент для разработки приложений с четкой архитектурой.
Ключевые слова: Flutter, BLoC, Cubit, управление состоянием, события, состояния, потоки, Streams, архитектура приложений, кроссплатформенная разработка, бизнес-логика, пользовательский интерфейс, масштабируемость, тестируемость, примеры кода, оптимизация разработки.

Введение

Современная разработка приложений требует четкой архитектуры, обеспечивающей масштабируемость, тестируемость и легкость сопровождения. Flutter, как один из ведущих фреймворков для кроссплатформенной разработки, предоставляет широкий выбор инструментов для управления состоянием, среди которых особое место занимают BLoC (Business Logic Component) и его упрощенная версия Cubit.

BLoC основывается на концепциях однонаправленного потока данных и четкого разделения ответственности между пользовательским интерфейсом и бизнес-логикой. Этот подход позволяет минимизировать связность компонентов, упрощая сопровождение и масштабирование приложений. Cubit, в свою очередь, представляет собой упрощенную реализацию BLoC, исключая обработку событий, что делает его подходящим для менее сложных задач.

Целью данной статьи является изучение особенностей, преимуществ и сценариев использования BLoC и Cubit. Мы рассмотрим их концепции, ключевые отличия, а также предоставим примеры кода, демонстрирующие их интеграцию в Flutter-приложениях.

1. Суть работы BLoC

BLoC — это паттерн управления состоянием, основанный на использовании потоков (Streams) для обработки событий и предоставления нового состояния. Он помогает реализовать однонаправленный поток данных, что упрощает поддержку и тестирование приложений. Этот подход позволяет четко разделить пользовательский интерфейс и бизнес-логику, минимизируя связность компонентов и улучшая читаемость кода [1].

BLoC активно используется для построения масштабируемых и гибких приложений, так как предоставляет мощный механизм для обработки сложных сценариев с большим количеством взаимодействий. Рассмотрим работу BLoC на рис. 1 [2].

Основные элементы BLoC [3]:

1. События (*Events*) — входные данные, отправляемые в BLoC. События могут быть вызваны действиями пользователя, системными изменениями или любыми другими внешними триггерами. Примеры событий: *LoginButtonPressed*, *FetchData*, *ItemSelected*.

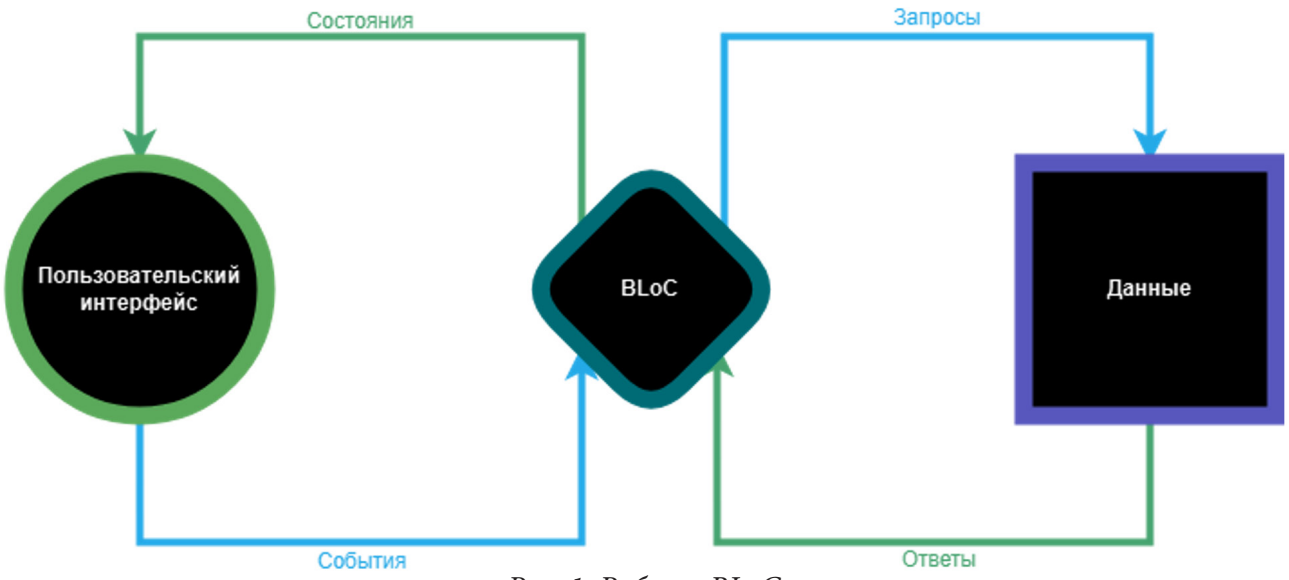


Рис. 1. Работа BLoC

2. Состояния (*States*) — выходные данные, которые представляют текущее состояние приложения. Они описывают, как интерфейс должен выглядеть в данный момент. Примеры состояний: *Loading*, *Loaded*, *Error*.

3. BLoC — центральный компонент, который принимает события, обрабатывает их и возвращает новое состояние. Он содержит всю бизнес-логику приложения.

2. Преимущества использования BLoC

1. *Однонаправленный поток данных*. Обеспечивает предсказуемость поведения приложения.

2. *Тестируемость*. Так как бизнес-логика отделена от интерфейса, ее легко покрывать модульными тестами.

3. *Масштабируемость*. Структура BLoC позволяет легко добавлять новые функции без значительных изменений в существующем коде.

4. *Поддерживаемость*. Четкое разделение ответственности между слоями делает код более читаемым и простым в сопровождении.

5. *Улучшенная производительность*. BLoC позволяет оптимизировать перерисовку UI, обновляя только те части интерфейса, которые действительно изменились.

3. Пример работы BLoC

Для начала работы с BLoC в Flutter-приложении необходимо определить структуру событий и состояний. Рассмотрим пример создания структуры BLoC для простого приложения-счетчика [4].

3.1. Создание событий

События в BLoC представляют действия или изменения, которые могут произойти в приложении. Они определяются как классы, наследующиеся от базового абстрактного класса. Это позволяет создать типобезопасную систему обработки событий. Следующий код определяет базовую структуру для событий в нашем приложении-счетчике:

```
abstract class CounterEvent {}
class IncrementCounter extends CounterEvent {}
class DecrementCounter extends CounterEvent {}
```

Этот код работает следующим образом:

1. *abstract class CounterEvent* — абстрактный класс, служащий основой для всех событий, связанных со счетчиком. Его использование позволяет нам создавать иерархию событий и обеспечивает типобезопасность при обработке событий.

2. *class IncrementCounter extends CounterEvent* — класс, представляющий событие увеличения счетчика на единицу. Он наследуется от *CounterEvent*. Это означает конкретное действие, которое может произойти с нашим счетчиком.

3. *class DecrementCounter extends CounterEvent* — аналогично с п. 2, этот класс представляет событие уменьшения счетчика на единицу.

3.2. Создание состояний

Состояния в BLoC представляют текущее состояние приложения. Они определяются как классы, наследующиеся от базового абстрактного класса. Это позволяет создать иерархию состояний и обеспечивает типобезопасность при обработке состояний. Пример структуры состояний для приложения-счетчика:

```
abstract class CounterState {}
class CounterValueState extends CounterState {
  final int value;
  CounterValueState(this.value);
}
```

Этот код работает следующим образом:

1. *abstract class CounterState* — абстрактный класс, который служит основой для всех состояний, связанных со счетчиком. Использование абстрактного класса позволяет нам создавать иерархию состояний и обеспечивает типобезопасность при обработке состояний.

2. *class CounterValueState extends CounterState* —класс, представляющий текущее состояние счетчика после какого-либо действия (увеличения или уменьшения). Он содержит значение счетчика.

В более сложных сценариях можно добавить дополнительную информацию в классы-состояния. Например, если нужно отслеживать ошибки или загрузку данных, можно добавить соответствующее состояние:

```
class LoadingState extends CounterState {}
class ErrorState extends CounterState {
  final String error;
  ErrorState(this.error);
}
```

3.3. Создание BLoC

Рассмотрим код для создания BLoC для приложения-счетчика:

```
import 'package:flutter_bloc/flutter_bloc.dart';
class CounterBloc extends Bloc<CounterEvent, CounterState> {
  CounterBloc() : super(CounterState()) {
    on<IncrementCounter>((event, emit) {
      final currentState = state as CounterValueState;
      emit(CounterValueState(currentState.value + 1));
    });
```

```
    on<DecrementCounter>((event, emit) {
      final currentState = state as CounterValueState;
      emit(CounterValueState(currentState.value - 1));
    });
  }
}
```

Этот код работает следующим образом:

1. Мы наследуем наш класс *CounterBloc* от Bloc библиотеки *flutter_bloc*, указывая тип событий (*CounterEvent*) и тип состояния (*CounterState*).

2. В конструкторе устанавливаем начальное состояние как *CounterState()*.

3. Мы определяем обработчики для каждого типа события с помощью методов *on<EventType>*. Эти методы принимают два аргумента:

• первый аргумент — событие, которое было отправлено;

• второй аргумент — функция *emit*, которая позволяет нам генерировать новые состояния.

4. В каждом обработчике мы можем выполнить какие-либо действия перед обновлением состояния.

5. После выполнения действий мы используем метод *emit()* для генерации нового состояния. В нашем случае это новое состояние типа *CounterValueState* с обновленным значением счетчика.

6. Мы также добавили проверку на отрицательное значение счетчика в обработчике *DecrementCounter*, чтобы избежать получения отрицательного значения.

3.4. Интеграция с интерфейсом

Сделаем интеграцию реализованного BLoC в интерфейс приложения:

```
BlocBuilder<CounterBloc, CounterState>(
  builder: (context, state) {
    final counterValue = (state as CounterValueState).value;
    return Text('Counter: $counterValue');
  },
);
```

Этот фрагмент кода представляет собой ключевую часть внедрения паттерна BLoC в пользовательский интерфейс Flutter приложения. Он использует *BlocBuilder*, который является компонентом Flutter для отображения изменений состояния в BLoC.

1. *BlocBuilder<CounterBloc, CounterState>*:

• первый аргумент (*CounterBloc*) указывает тип BLoC, с которым мы работаем;

• второй аргумент (*CounterState*) указывает тип состояния, которое мы ожидаем от BLoC.

2. Внутренняя функция-строитель (*builder*):

• эта функция вызывается каждый раз, когда состояние BLoC изменяется;

• она принимает два аргумента: текущий контекст и новое состояние;

• внутри функции мы извлекаем текущее значение счетчика из состояния.

3. Отображение значения:

• используем *Text* виджет для отображения текущего значения счетчика;

• форматирование строки включает в себя переменную *$counterValue*, которая динамически обновляется при изменении состояния BLoC.

4. Суть работы Cubit

Cubit (Command Unit Business Logic Template) — это упрощенная версия паттерна BLoC, которая предоставляет более легкий способ управления состоянием во Flutter-приложениях.

Основные отличия Cubit от BLoC:

1. *Простота*. Cubit требует меньше кода для реализации, что делает его более подходящим для небольших проектов или приложений с относительно простой бизнес-логикой.

2. *Отсутствие обработки событий*. В Cubit нет отдельного слоя событий. Вместо этого, все действия выполняются напрямую через методы класса Cubit.

Рассмотрим пример структуры Cubit для приложения-счетчика.

4.1. Реализация Cubit

```
import 'package:flutter_bloc/flutter_bloc.dart';
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);
  void increment() => emit(state + 1);
  void decrement() => emit(state - 1);
}
```

Этот код реализует *CounterCubit* — класс для управления состоянием счетчика с помощью *Cubit* из библиотеки *flutter_bloc*. Начальное состояние (*state*) установлено в *0*. Методы increment и decrement изменяют состояние, увеличивая или уменьшая его на единицу с помощью метода *emit*.

4.2. Интеграция с интерфейсом

```
BlocBuilder<CounterCubit, int>(
  builder: (context, state) {
    return Text('Counter: $state');
  },
);

FloatingActionButton(
  onPressed: () => context.read<CounterCubit>().increment(),
  child: Icon(Icons.add),
);
```

Этот код демонстрирует использование *BlocBuilder* и кнопки для работы с *CounterCubit*.

BlocBuilder отображает текстовый виджет *Text*, который обновляется каждый раз, когда состояние *CounterCubit* меняется. В *builder* передается текущее состояние (*state*).

FloatingActionButton при нажатии вызывает метод *increment* у *CounterCubit* через *context.read<CounterCubit>()*, увеличивая значение счетчика.

5. Области применения BLoC и Cubit

1. BLoC:

a. Подходит для сложных приложений с множеством событий.

b. Требуется четкая архитектура с разделением ответственности.

c. Необходима масштабируемость и тестируемость.

2. Cubit:

a. Подходит для простых приложений и прототипов.

b. Требуется простая бизнес-логика.

c. Приложение содержит небольшое количество состояний.

Заключение

В данной работе были проанализированы и сравнены два подхода к управлению состоянием в приложениях на Flutter: BLoC и Cubit. Рассмотрены ключевые особенности каждого подхода, их преимущества и области применения.

BLoC (Business Logic Component) представлен как более сложный и гибкий механизм управления состоянием. Он основан на концепции однонаправленного потока данных и обеспечивает четкое разделение между событиями, состояниями и бизнес-логикой. BLoC особенно эффективен для сложных приложений с множеством взаимодействий и состояний, требующих масштабируемой и тестируемой архитектуры.

С другой стороны, Cubit предлагается как упрощенная версия BLoC, более подходит для менее сложных сценариев использования. Он предоставляет более простую структуру и требует меньше кода для реализации, что делает его предпочтительным для небольших проектов или прототипов. Cubit также отличается отсутствием отдельного слоя событий, вместо этого используя прямые методы для обновления состояния.

В дальнейшем результаты данного обзора будут использованы при реализации CRM-системы для центров заботы о животных. Это позволит учесть преимущества и недостатки подходов BLoC и Cubit и выбрать наиболее подходящий.

Литература

1. Bloc State Management Library // Bloc : [сайт]. – 2024. – URL: https://bloclibrary.dev/ (дата обращения: 05.11.2024).

2. Flutter Bloc package // Bloc : [сайт]. – 2024. – URL: https://pub.dev/packages/flutter_bloc (дата обращения: 04.11.2024).

3. Flutter Bloc concepts // Bloc : [сайт]. – 2024. – URL: https://bloclibrary.dev/flutter-bloc-concepts/ (дата обращения: 04.11.2024).

4. Flutter Counter // Bloc : [сайт]. – 2024. – URL: https://bloclibrary.dev/tutorials/flutter-counter/ (дата обращения: 05.11.2024).

УДК 004.056.55

МНОГОЧЛЕННОЕ ШИФРОВАНИЕ

А. М. Джабаров, А. Ф. Сауляк, И. Н. Попов

Северный (Арктический) федеральный университет имени М. В. Ломоносова

Аннотация. Разложения многочленов по системе многочленов применяются в решении различного рода задач, как для выяснения свойств самих многочленов, так и их применения. Разложения многочленов, в частности, используются при многочленном шифровании сообщений. Если под разложением понимается запись многочлена в виде линейной комбинации многочленов данной системы со скалярами из коммутативного кольца с единицей в качестве коэффициентов, то многочленное шифрование может выражаться в том, что коэффициенты исходного многочлена соответствуют символам исходного сообщения, коэффициенты разложения — символам шифрованного сообщения. При многочленном шифровании используются разложения, которые способствуют дешифрованию сообщений. Точнее, при выборе кольца, к которому принадлежат коэффициенты исходного многочлена и коэффициенты разложения, есть возможность осуществить как шифрование, так и дешифрование. Одним из примеров разложений является разложение многочлена по степеням линейного многочлена, реализация которого осуществляется с помощью ступенчатой схемы Горнера. Такое разложение как раз обладает указанными требованиями к многочленному шифрованию сообщений. В работе рассматривается именно такое разложение с точки зрения многочленного шифрования, реализация которого осуществляется с использованием языка программирования Python.
Ключевые слова: многочлен, стандартная запись многочлена, линейная комбинация многочленов, разложение многочлена, алгоритм разложения, схема Горнера, Python, программирование, шифрование, многочленное шифрование.

Введение

В данной работе под разложением многочлена понимается представление многочлена, записанного в стандартной форме, в виде линейной комбинации неотрицательных целых степеней линейного приведенного (со старшим коэффициентом, равным единице) многочлена со скалярами из коммутативного кольца с единицей в качестве коэффициентов. Разложение многочлена имеет значение в решении задач, как связанных с выяснением свойств самого многочлена, так и прикладного характера. К первому случаю можно отнести задачи о нахождении корней многочлена (добиваясь равенства нулю того или иного коэффициента в разложении, что требуется, например, для применения формул Кардана при решении кубического уравнения) или доказательства его неприводимости (например, с использованием критерия Эйзенштейна для многочлена с целыми коэффициентами), о вычислении значений самого многочлена и всех его производных при данном значении аргумента, о приближенных вычислениях значений многочлена (например, при табулировании значений многочлена в окрестности данного значения), о раскрытии скобок и приведении подобных слагаемых в выражениях определённого вида и другие. К прикладным задачам, в которых играет разложение многочлена, относится, например, задача о представлении дробно-рациональной функции со степенью линейного многочлена в качестве знаменателя в виде суммы элементарных дробей (что, в частности, используется при интегрировании дробно-рациональных функций).

Основным алгоритмом для получения разложения многочлена является схема Горнера, которая основывается на теореме Безу. Схема Горнера применяется для возврата десятичной записи натурального числа, указанного в определенной системе счисления (в двоичной, троичной и так далее), используется в построении эффективного алгоритма возведения числа в на-

туральную степень и так далее. Само разложение многочлена получается посредством многократного применении схемы Горнера и сводится к построению ступенчатой схемы Горнера [1].

Схема Горнера применяется в шифровании сообщений, если в основе шифрования используется многочлен и вычисление символов шифрованного сообщения ведётся вычислением этого многочлена при данных значениях переменной [2].

Разложение многочлена можно рассматривать и как способ шифрования сообщений. В этом случае речь идёт о многочленном шифровании, суть которого в том, что набор коэффициентов исходного многочлена связывается с символами исходного сообщения, набор же коэффициентов разложения — с символами шифрованного сообщения.

1. Теоретические аспекты

Разложение многочлена может рассматриваться в качестве одного из способов многочленного шифрования — способа шифрования сообщений (как преобразование сообщения с целью защиты его конфиденциальности), основанного в общем случае на разложении многочлена по многочленам из определённого наперед множества. Суть многочленного шифрования сводится к тому, что символам сообщения приписываются по некоторому правилу числа, которые выступают в роли коэффициентов многочлена, записанного в стандартной форме, затем полученный многочлен раскладывается по данным многочленам и коэффициенты этого разложения считаются преобразованными числами, по которым уже строится зашифрованное сообщение. Естественно требовать, чтобы была и возможность дешифрования. Формально ставится задача об определении формул для вычисления коэффициентов разложения по коэффициентам исходного многочлена, и ей обратная.

Отметим, что рассматриваемое в работе разложение при использовании ступенчатой схемы Горнера позволяет как шифровать, так и дешифровать в произвольных коммутативных кольцах с единицей, в частности, в кольце классов вычетов целых чисел по модулю.

Разложение многочлена зависит от двух параметров: степени многочлена и точки разложения. Эти числа являются в шифровании данным способом секретными.

Схема шифрования

А) Определяется алфавит, из символов которого составляются сообщения. Используется русский алфавит из 33 букв с добавлением символов «пробела» и «*» (всего — 35 символов). Исходное сообщение может содержать символы, отличные от символа «*». Символам алфавита сопоставляются целые числа.

Б) Сообщение делится на блоки определенной длины (первое секретное число). Если длина сообщения не кратна длине блока, то к последнему «короткому» блоку добавляется определённое число символов «*».

В) Числа, сопоставленные символам блока, рассматриваются в качестве коэффициентов многочлена, который с помощью ступенчатой схемы Горнера раскладывается по степеням линейного многочлена в кольце классов вычетов по модулю 35 при выбранной точке разложения (второе секретное число). По числам-коэффициентам разложения создается блок символов алфавита. Так поступают с каждым блоком из п. б).

Г) Полученные блоки складываются в сообщение, являющееся шифрованным.

Схема дешифрования

А) Сообщение делится на блоки, длина которых совпадает с длиной блоков из п. б) схемы шифрования.

Б) Числа, сопоставленные символам блока, рассматриваются в качестве коэффициентов многочлена, который с помощью ступенчатой схемы Горнера раскладывается по степеням линейного многочлена с точкой разложения, равной разности числа 35 и точки разложения из

п. в) схемы шифрования, в кольце классов вычетов по модулю 35. По числам-коэффициентам разложения создается блок символов алфавита. Так поступают с каждым блоком из п. а).

В) Полученные блоки складываются в сообщение, удалив символы «*». Получается дешифрованное сообщение.

2. Практическая реализация

Предлагаются программы на Python [3] для шифрования и дешифрования текста с использованием многочленного шифрования на основе разложения многочлена, записанного в стандартной форме, по степеням линейного многочлена. В качестве алгоритма разложения многочлена используется ступенчатая схема Горнера.

Программа-шифратор

```
def alphabet_index(symbol):
    if symbol in alphabet:
        return alphabet.index(symbol) + 1
    else:
        raise ValueError(f"Символ '{symbol}' не найден в алфавите.")
def symbol_from_index(index):
    alphabet = 'абвгдеёжзийклмнопрстуфхцчшщьыъэюя *'
    return alphabet[index - 1]
def polynomial_encryption(message, n, a, m):
    # Приведение сообщения к нижнему регистру
    message = message.lower()
    # Преобразование сообщения в числа
    numbers = []
    for symbol in message:
        try:
            numbers.append(alphabet_index(symbol))
        except ValueError as e:
            print(e)
            return None
    # Дополнение сообщения до кратной длины n
    while len(numbers) % n != 0:
        numbers.append(alphabet_index('*'))
    # Разбиение на блоки длины n
    blocks = [numbers[i:i + n] for i in range(0, len(numbers), n)]
    # Шифрование каждого блока
    encrypted_blocks = []
    for block in blocks:
        encrypted_block = horner_scheme(block, a, m)
        encrypted_blocks.extend(encrypted_block)
    # Перевод зашифрованного сообщения в текст
    encrypted_message = ''
    for i in range(len(encrypted_blocks)):
        encrypted_blocks[i] = symbol_from_index(encrypted_blocks[i])
    for i in encrypted_blocks:
        encrypted_message += i
    return encrypted_message
```

```
def horner_scheme(block, a, m):
    result =[]
    n = len(block)
    b = [0] * n
    encrypted_block = []
    b[0] = block[0]
    for s in range(1, n+1):
        for i in range(1, n):
            b[i] = (b[i - 1] * a + block[i]) % m
        block = b
        encrypted_block.append(b[-s])
    for i in range(1,len(encrypted_block)+1):
        result.append(encrypted_block[-i])

    return result

# Пример использования
alphabet = 'абвгдеёжзийклмнопрстуфхцчшщьыъэюя *'
message = input('Введите сообщение: ')
n = 5
a = 7
m = len(alphabet)

encrypted_message = polynomial_encryption(message, n, a, m)
if encrypted_message is not None:
    print(encrypted_message)
else:
    print("Шифрование не удалось из-за недопустимых символов в сообщении.")
```

Пример:
исходное сообщение: многочленное шифрование сообщения (33 символа)
шифрованное сообщение: мфбйбчл нжо шпфгбпфниелшозб щнияёё (35 символов)

Программа-дешифратор

```
def symbol_from_index(index):
    return alphabet[index - 1]

def alphabet_index(symbol):
    if symbol in alphabet:
        return alphabet.index(symbol) + 1
    else:
        raise ValueError(f"Символ '{symbol}' не найден в алфавите.")

def polynomial_decryption(encrypted_message, n, a, m):
    # Перевод зашифрованного сообщения в индексы
    encrypted_index = []
    for i in encrypted_message:
        encrypted_index.append(alphabet_index(i))

    # Разбиение на блоки длины n
    blocks = [encrypted_index[i:i + n] for i in range(0, len(encrypted_index), n)]

    # Дешифрование каждого блока
```

```
    decrypted_blocks = []
    for block in blocks:
        decrypted_block = inverse_horner_scheme(block, a, m)
        decrypted_blocks.extend(decrypted_block)

    # Преобразование чисел в символы
    message = ''.join(symbol_from_index(num) for num in decrypted_blocks)

    # Удаление дополнительных символов '*'
    message = message.rstrip('*')

    return message

def inverse_horner_scheme(block, a, m):
    result = []
    n = len(block)
    b = [0] * n
    encrypted_block = []
    b[0] = block[0]
    for s in range(1, n + 1):
        for i in range(1, n):
            b[i] = (b[i - 1] * a + block[i]) % m
        block = b
        encrypted_block.append(b[-s])
    for i in range(1, len(encrypted_block) + 1):
        result.append(encrypted_block[-i])

    return result

# Пример использования
alphabet = 'абвгдеёжзийклмнопрстуфхцчшщьыъэюя *'
n = 5
a = 7
m = len(alphabet)
encrypted_message = input(‘Введите зашифрованное сообщение: ‘)
# Замена a на m - a
a = m - a

decrypted_message = polynomial_decryption(encrypted_message, n, a, m)
print("Дешифрованное сообщение:", decrypted_message)
```

Пример:
исходное сообщение: мфбйбчл нжо шпфгбпфниелшозб щнияёё
шифрованное сообщение: многочленное шифрование сообщения

Заключение

Итогом работы является компьютерная реализация на Python многочленного шифрования сообщений на основе разложения с применением ступенчатой схемы Горнера. Используя обобщения схемы Горнера [4, с. 40–42], можно предложить другие схемы многочленного шифрования, аналогичные рассмотренному.

Литература

1. *Винберг Э. Б.* Алгебра многочленов / Э. Б. Винберг. – Москва : Просвещение, 1980. – 176 с.

2. Введение в криптографию / Под общей ред. В. В. Ященко. – Санкт-Петербург : Питер, 2001. – 288 с.

3. *Лутц М.* Изучаем Python / М. Лутц. – Санкт-Петербург : Символ-Плюс, 2011. – 1280 с.

4. *Гриншпон С. Я.* Многочлены над областями целостности (теория и приложения) / С. Я. Гриншпон, И. Э. Гриншпон. – Томск : Издательский Дом Томского государственного университета, 2016. – 152 с.

УДК 004.94

МОДЕЛИРОВАНИЕ СУБЪЕКТИВНОГО ИГРОВОГО ОПЫТА ПРИ ПОСТРОЕНИИ ЛАБИРИНТОВ В КОМПЬЮТЕРНЫХ ИГРАХ

В. Ю. Евсеенко, Е. В. Трофименко

Воронежский государственный университет

Аннотация. В статье рассматривается моделирование субъективного игрового опыта в дизайне игр. В отличие от традиционного игрового дизайна, который опирается на фиксированную механику и статическую сложность, данный подход направлен на создание персонализированного, динамичного опыта, основанного на том, как игроки чувствуют и реагируют на игру. В статье рассматриваются ключевые компоненты субъективного моделирования и проводится анализ применения моделирования на практике.
Ключевые слова: субъективный опыт, моделирование, персонализация, генерация лабиринтов, вовлечение, доступность, поведение игрока.

Введение

Игровой дизайн часто рассматривается как единое целое, основанное на механиках игр, сценарии и графике. Хотя эти аспекты имеют решающее значение, часто упускается из виду еще одно важное измерение — субъективный опыт игрока. Концепция моделирования субъективного игрового опыта — это предметная область, которая стремится понять и смоделировать то, как отдельные игроки воспринимают и взаимодействуют с игрой, смещая фокус с универсальных профилей игроков на более персонализированный, динамичный опыт.

Субъективный игровой опыт представляет восприятие игроком игрового процесса. Этот опыт включает в себя спектр реакций, таких как эмоциональные и психологические, которые испытывает игрок при взаимодействии с игровыми элементами. В отличии от объективных показателей (время, проведённое за игрой, или процент её завершения), субъективный опыт отличается для каждого отдельного игрока, на него влияют такие факторы как настроение человека, личные предпочтения, предыдущий игровой опыт и общее эмоциональное состояние [1].

По мере того, как игры становятся всё более сложными в своём устройстве и требуют всё больших бюджетов, разработчикам приходится находить новые подходы к привлечению более широкой аудитории. Одним из таких подходов и является моделирование субъективного опыта игроков. Принимая во внимание субъективный опыт отдельных игроков, разработчики могут:

1. **Персонализировать игровой процесс:** игры могут динамически адаптироваться предпочтениям игрока. Например, если игрок совершает много ошибок, в игре может быть реализована возможность предложения игроку более простого сценария прохождения игрового уровня на основе анализа игрового процесса.

2. **Увеличить удержание игроков:** игры могут поддерживать интерес в течение более длительного времени, подстраиваясь под предпочтения и поведение игроков [2].

3. **Улучшить доступность:** субъективное моделирование игры позволяет адаптировать сложность и настраивать интерфейсы, делая игры более доступными для игроков с разным уровнем навыков и предпочтениями.

1. Ключевые компоненты моделирования субъективного игрового опыта

1.1. Профилирование игрока и персонализация

Игры, в которых используется моделирование субъективного опыта, зачастую пытаются в первую очередь понять игрока отслеживая такие параметры как:

• **предпочитаемый стиль игры** (например, агрессивный или стратегический)

• **внутриигровые решения** (то, как игрок выполняет задачи, поставленные игрой)

Анализируя эти параметры, алгоритм игра может подстраиваться под игровой стиль пользователя. Например, такие игры, как The Witcher 3 и Mass Effect, позволяют игрокам делать выбор, который влияет на сюжетную линию, персонализируя опыт на основе индивидуальных решений (рис. 1).

Рис. 1. Пример сюжетного выбора в The Witcher 3

Данные о поведении игрока:

• **принятие решений игроком:** в играх с разветвленным повествованием или множественным выбором объективные данные могут использоваться для понимания того, какие пути выбираются чаще и почему;

• **прямая конфронтация против скрытности:** в играх, которые предлагают несколько стилей игры (например, Dishonored), анализ субъективных данных может помочь определить, какой подход предпочитают игроки, и соответствующим образом адаптировать баланс игры;

• **взаимодействие с игровыми системами:**

a) **использование инвентаря:** отслеживание того, как игроки используют ресурсы, такие как зелья, улучшения или специальные предметы;

b) **использование навыков и способностей:** на какие способности игроки полагаются чаще всего, и какие механики используются большинством игроков;

1.2. Измерение показателей удержания игроков

Отслеживание того, как долго игроки остаются вовлеченными в игру и когда они прекращают играть, имеет решающее значение для понимания того, работоспособности различных компонент. Также в эту категорию входит замер того, как много игроки проводят времени за игрой каждый отдельный сеанс, что помогает измерить, насколько увлекательной остаётся игра со временем [3].

Инструменты для моделирования объективного игрового опыта:

• **телеметрические системы:** в игровом дизайне телеметрические системы собирают данные о том, как игроки взаимодействуют с игрой. Популярные движки, такие как Unity и Unreal Engine, имеют телеметрические системы, которые позволяют разработчикам собирать показатели производительности, поведения и прогресса игроков:

• **тепловые карты:** это визуальное представление данных [4], обычно используемое для того, чтобы показать, где игроки проводят больше всего времени или сталкиваются с наибольшим количеством смертей (рис. 2). Красным цветом обозначены места, в которых персонажи

игры погибают чаще всего, зелёным — места, в которых концентрация гибели персонажей меньше, но всё ещё превышает средний показатель на карте. Как видно на этом примере, использование тепловых карт может помочь в анализе того, какие области игры нуждаются в переработке и балансировке.

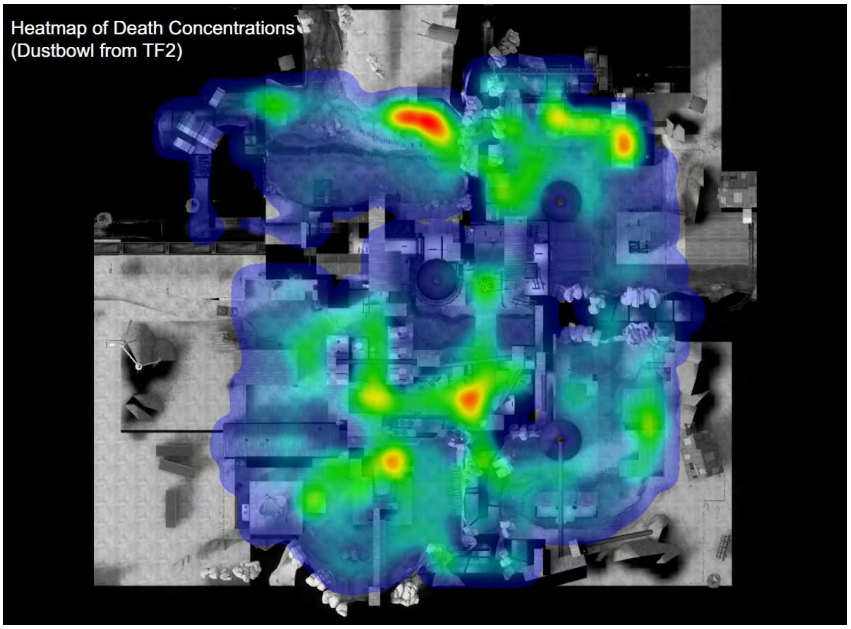


Рис. 2. Пример тепловой карты для Team Fortress 2

2. Применение моделирования субъективного опыта при генерации лабиринтов

В рамках данного исследования была реализована игра с помощью игрового движка Unity. Для создания разнообразных лабиринтов, которые смогут адаптироваться под каждого игрока в работе был использован анализ субъективного опыта, который был реализован в специальном скрипте, отслеживающем действия игрока. В нём отслеживается количество побеждённых врагов и собранное игроком золото, а также количество посещённых комнат. Далее эти данные обрабатываются генератором, который выбирает комнаты, подходящие под предпочтения игрока.

В случае, если игрок победил большое количество врагов, выбирается комната с врагами, а если он собрал много золота, выпадает комната с сокровищами. Если игрок не проявил особых предпочтений, генератор выбирает комнату случайно. В зависимости от процента посещённых комнат лабиринт может увеличиться в размере на следующей итерации.

На графике представлена зависимость генерации различных комнат от анализа поведения игрока.

На примере этого графика видно, что в каждом уровне лабиринта игра учитывает поведение игрока и старается подстроиться к нему.

Такой подход обладает следующими особенностями:

1.Так как каждый новый лабиринт отличается от предыдущего, у игрока возникает стимул исследовать новые уровни, пробовать новые пути и возможности;

2.Лабиринт создаётся согласно действиям игрока на предыдущем уровне, таким образом игрок не столкнётся со слишком сложным или слишком простым для него лабиринтом;

Альтернативой данному подходу является анализ в реальном времени. При такой реализации данные так же собираются на протяжении уровня, но их учёт и изменение переменных лабиринта происходит в реальном времени. Данный алгоритм так же был реализован на Unity. Алгоритм учитывает победу над врагами и собранное золото в реальном времени и изменя-

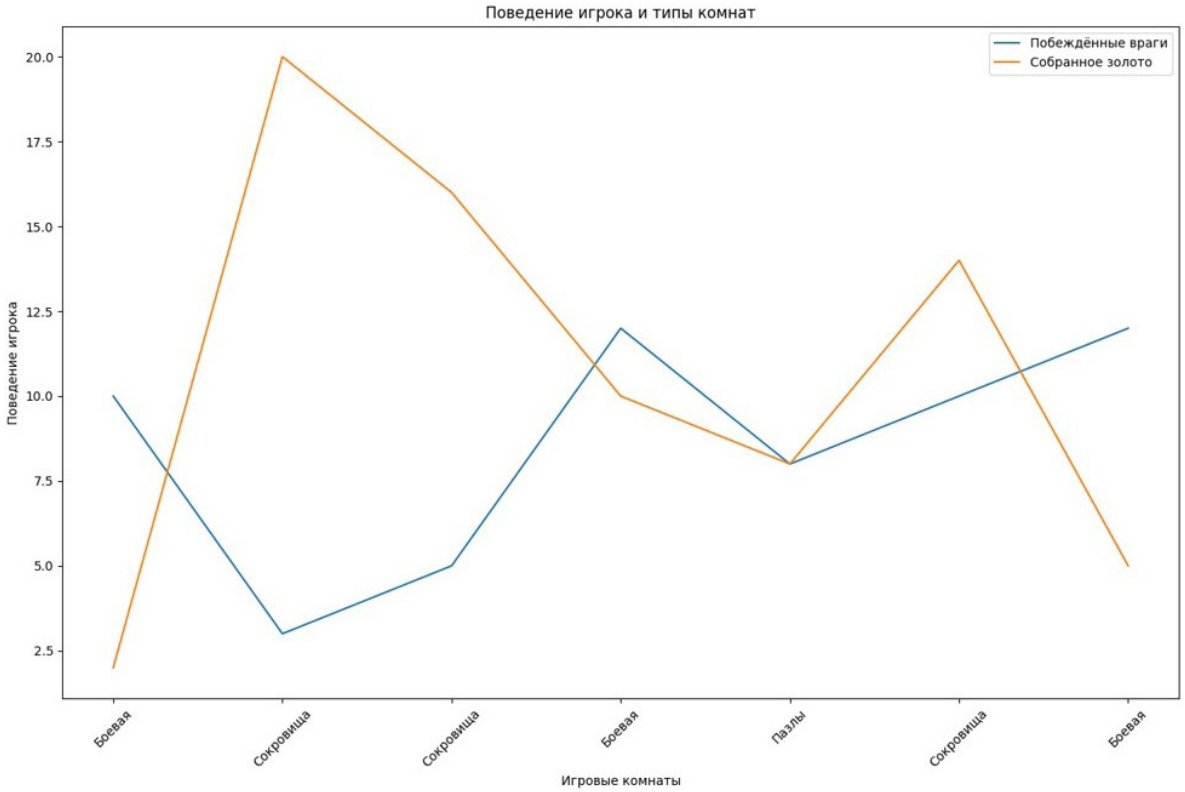


Рис. 3. Пример работы генератора

ет показатели для подстраивания под действия игрока. При использовании такого подхода вырастает нагрузка на устройство, но для небольшого количества переменных такой вариант может подойти лучше. Пример разницы в производительности можно увидеть в табл. 1.

Таблица 1

	Анализ в конце уровня	Анализ в реальном времени
Среднее число кадров в секунду	312	276
Нагрузка на ОЗУ	562 Mb	658 Mb

Исходя из данных таблицы видно, что преимуществом анализа в конце уровня является его производительность и стабильность, ведь лабиринт меняется только после перехода на следующий уровень и сам анализ использует меньшее количество ресурсов. В то же время анализ в реальном времени позволяет более динамически реагировать на действия игрока, что позволяет внести в игру элемент неожиданности и увеличить вовлечённости игрока.

Подводя итог, можно сказать, что выбор правильного подхода к отслеживанию зависит от целей игры, технических ограничений и желаемого игрового опыта.

Заключение

Использование моделирования субъективного опыта в игровом дизайне позволяет создавать более захватывающие и персонализированные впечатления, адаптируясь к поведению игроков. Этот подход основан на анализе данных, что позволяет разработчикам не только тонко настраивать игровую механику, но и формировать эмоциональное путешествие игроков, повышая удержание. Используя данный тип моделирования игрового опыта, разработчики лучше понимают игроков, что помогает им создавать более увлекательные игры, которые находят отклик на индивидуальном уровне.

Литература

1. Development and validation of the player experience inventory – URL: https://www.sciencedirect.com/science/article/pii/S1071581919301302 (дата обращения 19.11.2024)

2. Too Many Questionnaires: Measuring Player Experience Whilst Playing Digital Games -URL: https://www-users.york.ac.uk/~pc530/pubs/Nordin_YDS2014.pdf (дата обращения 19.11.2024)

3. Game metrics without players – URL: https://www.kmjn.org/notes/analytical_metrics.html (дата обращения 19.11.2024)

4. A Heatmap guide for game level analysis– URL: https://medium.com/@dariarodionovano/a-heatmap-guide-for-game-level-analysis-68cb6a7bcb2b (дата обращения 19.11.2024)

5. Unity Game Engine – URL: https://unity.com/ru (дата обращения 19.11.2024)

6. AwesomeUnityCommunityLibrary–URL:https://github.com/sickq/AwesomeUnityCommunity (дата обращения 19.11.2024)

7. Matplotlib: Visualization with Python – URL: https://matplotlib.org/ (дата обращения 19.11.2024)

УДК 004.01

ОСНОВЫ, ПРИНЦИПЫ И ОСОБЕННОСТИ LIQUIBASE

Н. В. Желтиков

Воронежский государственный университет

Аннотация. В статье рассмотрены ключевые аспекты работы liquibase — подключение, настройка и выполнение миграций. Описаны принципы работы инструмента, организация «ченджлога» (*changelog*) и управление «ченджсетом» (*changeset*). Разобраны интеграция liquibase в CI/CD-контейнеры, автоматическая генерация changelog, роллбеки, использование «Formatted SQL» и проверка синхронизации базы данных. Исследование направлено на систематизацию знаний о liquibase и демонстрацию его роли в оптимизации процессов разработки и развёртывания.
Ключевые слова: liquibase, миграции, changelog, changeset, rollback, база данных, контроль изменений, автоматизация, синхронизация, databasechangelog, databasechangelock.

Введение

Liquibase — инструмент для управления миграциями базы данных, обеспечивающий отслеживание, версионирование и автоматизацию изменений. Его основная цель — синхронизация схемы базы данных с кодовой базой и упрощение процесса развёртывания. Liquibase широко используется благодаря поддержке нескольких форматов описания изменений (XML, YAML, SQL и JSON), возможности отмены изменений на уровне отдельных релизов, возможности работы с конкретными SQL-диалектами, интеграции с популярными СУБД и современными CI/CD-платформами.

Таблица 1

Основные характеристики liquibase

Характеристика	Описание
Форматы	XML, YAML, SQL, JSON
Типы изменений	Структура базы данных, данные
Совместимость	PostgreSQL, MySQL, Oracle, SQLite и др.
Интеграция	Возможность внедрения в CI/CD инструменты (Jenkins, GitLab CI, GitHub Actions)
Универсальность	Поддержка работы за пределами Java-экосистемы
Доступность	Открытый исходный код

1. Подключение liquibase

Liquibase легко интегрируется с проектами на Java и другими экосистемами благодаря поддержке стандартных инструментов сборки и конфигурации. Для запуска liquibase преимущественно используются Maven или Gradle.

1.1. Добавление зависимости

Для проектов на Java добавление зависимости происходит через инструменты сборки:

• Maven:

```
<dependency>
    <groupId>org.liquibase</groupId>
```

```
        <artifactId>liquibase-core</artifactId>
        <version>4.x.x</version>
    </dependency>
```
• Gradle:
```
implementation 'org.liquibase:liquibase-core:4.x.x'
```

Для других языков (Python, .NET, Node.js) доступны плагины, которые можно использовать в DevOps-конвейерах.

1.2. Настройка подключения к базе данных

Liquibase рассматривает и, при наличии, использует файл liquibase.properties для конфигурации подключения:

```
url = jdbc:postgresql://localhost:5432/your_database
username = username
password = password
changelogFile = classpath:db/changelog/changelog.yaml
liquibase.hub.mode=off
```

Файл описывает URL базы данных, учётные данные, основной changelog-файл, где описаны миграции и интеграцию с Liquibase Hub.

В Spring-приложениях для конфигурации liquibase может быть использован файл application.properties или application.yaml. Пример application.yaml файла:

```
spring:
  datasource:
    password: password
    username: username
    url: jdbc:postgresql://localhost:5432/ your_database
  liquibase:
    change-log: classpath:db/changelog/changelog.xml
    defaultSchema: public
```

2. Принципы работы Liquibase

2.1. Changelog и Changeset

Liquibase работает с двумя основными концепциями:

• changelog — файл, описывающий последовательность изменений базы данных;

• changeset — блок изменений, содержащий уникальный идентификатор, имя автора и инструкции.

Пример XML-формата changeset:

```
<databaseChangeLog xmlns=»http://www.liquibase.org/xml/ns/dbchangelog»
    xmlns:xsi=»http://www.w3.org/2001/XMLSchema-instance»
    xsi:schemaLocation=»http://www.liquibase.org/xml/ns/dbchangelog
    http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.0.xsd»>

    <changeSet id=»1» author=»user»>
        <createTable tableName=»users»>
            <column name=»id» type=»int» autoIncrement=»true»>
                <constraints primaryKey=»true»/>
            </column>
```

```
            <column name=»username» type=»varchar(255)»/>
            <column name=»created_at» type=»timestamp»/>
        </createTable>
    </changeSet>
</databaseChangeLog>
```

Пример XML-формата changelog:

```
<?xml version=»1.0» encoding=»UTF-8»?>
<databaseChangeLog
        xmlns=»http://www.liquibase.org/xml/ns/dbchangelog»
        xmlns:xsi=»http://www.w3.org/2001/XMLSchema-instance»
        xsi:schemaLocation=»http://www.liquibase.org/xml/ns/dbchangelog
            http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.2.xsd»>

    <include file=»migrations/V2024_22_11_18_00_init_db.sql» relativeToChangelogFi
le=»true»/>
</databaseChangeLog>
```

Считывается changelog-файл, далее происходит проверка, какие изменения применены и какие необходимо выполнить. Одновременно с выполнением в таблицу databasechangelog записывается контрольная сумма (MD5-хэш) этого ченджсета.

Перед запуском liquibase пересчитывает контрольные суммы для всех ченджсетов и сравнивает их с ранее сохранёнными значениями в таблице. Если содержимое файла было изменено, новая контрольная сумма не совпадёт с исходной, что приведёт к остановке выполнения приложения.

2.2. Архитектура работы

При первом запуске liquibase автоматически создаёт две специальные таблицы — databasechangelog и databasechangelock, которые используются для управления миграциями:

• liquibase сохраняет данные о выполненных ченджсетах в таблице databasechangelog;

• при выполнении команды обновления (liquibase update), liquibase анализирует содержимое таблицы databasechangelog и применяет только те изменения, которые ещё не были зафиксированы;

• уникальные идентификаторы каждого ченджсета гарантируют, что миграции не будут выполнены повторно;

• таблица databasechangelock отвечает за блокировку, обеспечивая работу только одного экземпляра liquibase одновременно.

Одновременное использование нескольких экземпляров liquibase для одной базы данных может приводить к конфликтам. Для их предотвращения используется таблица databasechangelock, которая содержит поле locked (boolean). При запуске liquibase проверяет значение этого поля — если оно установлено в true, приложение ожидает, пока флаг не изменится на false. Однако в случае принудительного завершения программы на раннем этапе работы поле может остаться в состоянии true, что блокирует дальнейшие операции. Для устранения этой проблемы значение поля необходимо вручную изменить в базе данных на false.

3. Выполнение миграций

3.1. Базовые команды

Liquibase CLI предоставляет следующие команды:

• *update* — применяет изменения, описанные в changelog:

```
liquibase update
```

• *rollback* — отменяет одно или несколько изменений:
`liquibase rollbackCount 1`
• *status* — отображает список изменений, которые ещё не применены:
`liquibase status`

3.2. Интеграция в CI/CD

Liquibase интегрируется с CI/CD-платформами (Jenkins, GitHub Actions, GitLab CI). Пример выполнения миграций в Jenkins:

```
pipeline {
    stages {
        stage('Migration') {
            steps {
                sh 'liquibase update'
            }
        }
    }
}
```

4. Преимущества использования Liquibase

4.1. Контроль версий базы данных

Liquibase предоставляет встроенный механизм контроля версий базы данных, который исключает возможность повторного применения одного и того же ченджсета, позволяет отслеживать историю изменений, а также выявлять конфликтующие изменения.

4.2. Модульность

Liquibase поддерживает модульную структуру миграций, это позволяет сильно упростить управление изменениями в многокомпонентных системах. При помощи директивы include можно разделить общий changelog на несколько файлов. Это полезно в следующих случаях:

1. Работа с несколькими модулями или компонентами. В каждом модуле может храниться определённый changelog.

2. Читаемость. Крупные changelog-файлы трудно поддерживать, поэтому их разделение на модули улучшает структуру проекта и облегчает работу с ним.

3. Переиспользование. Отдельные миграции можно включать в разные проекты или компоненты.

4.3. Rollback

Liquibase предоставляет механизм отката изменений (rollback). Основные сценарии использования rollback:

• Исправление ошибок. Если новая миграция вызывает сбои, есть возможность её откатить, вернув базу данных в стабильное состояние.

• Тестирование миграций. На этапе разработки rollback позволяет многократно применять и откатывать изменения, проверяя их корректность.

Способы отката изменений:

• Откат определённого количества изменений:
`liquibase rollbackCount 2`

• Откат до конкретной точки во времени:

```
liquibase rollbackToDate 2024-11-22T18:00:00
```

• Откат изменений до определённой версии:

```
liquibase rollback version_tag_1.0
```

5. Расширенные возможности

5.1. Генерация начального Changelog

Liquibase может автоматически сгенерировать changelog, это полезно для внедрения liquibase в проекты с уже существующими базами данных:

```
liquibase generateChangeLog
```

5.2. Динамическое управление миграциями

Liquibase позволяет использовать динамические параметры для миграций, переменные заменяются значениями из файла конфигурации или окружения:

```
databaseChangeLog:
  - changeSet:
      id: 1
      author: user
      changes:
        - createTable:
            tableName: ${tableName}
```

5.2. Использование «Formatted SQL»

Liquibase поддерживает форматирование миграций непосредственно в SQL-файлах с использованием особого синтаксиса, называемого *Formatted SQL*. Это позволяет писать миграции в привычной форме SQL-запросов, добавляя специальные комментарии для управления миграциями.

Форматированный SQL использует комментарии для обозначения metadata ченджсетов. Пример:

```
--liquibase formatted sql

--changeset nzheltikov:1
CREATE TABLE users (
id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(255) NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

--changeset nzheltikov:2
ALTER TABLE users ADD COLUMN email VARCHAR(255);
```

• `--liquibase formatted sql` — указывает, что файл содержит формализованные SQL-миграции.

• `--changeset <author>:<id>` — определяет уникальный идентификатор миграции.

Liquibase читает SQL-файл и интерпретирует специальные комментарии как метадату для управления миграциями. Комментарии обрабатываются liquibase как инструкции для опреде-

ления ченджсетов. Все изменения после тега `--changeset` считаются принадлежащими этому ченджсету до следующего такого тега или до конца файла.

Заключение

Liquibase — универсальный инструмент для контроля изменений базы данных, сочетающий простоту использования, удобство управления и функциональную гибкость. Благодаря поддержке различных форматов описания изменений, гибкости работы с SQL-диалектами и интеграции с CI/CD, liquibase является важной частью современных процессов разработки. Использование liquibase помогает снизить риски, повысить производительность команд и обеспечить согласованность между базой данных и приложением.

Литература

1. Liquibase Documentation. – URL: https://www.liquibase.org/documentation.

2. Миграции схемы базы данных с Liquibase. – URL: https://struchkov.dev/blog/ru/get-started-liquibase.

3. Using Liquibase. – URL: https://docs.oracle.com/en/database/oracle/sql-developer-command-line/23.4/sqcug/using-liquibase.html#GUID-4CA25386-E442-4D9D-B119-C1ACE6B79539.

4. Use Liquibase to Safely Evolve a Database Schema. – URL: https://www.baeldung.com/liquibase-refactor-schema-of-java-app.

5. Ambler W. Refactoring Databases: Evolutionary Database Design / W. Ambler, J. Sadalage. – Addison Wesley Professional, 2006. – 384 p.

УДК 004.42, 519.683

РАЗРАБОТКА ПРИЛОЖЕНИЯ «РАСПИСАНИЕ ВГУ». РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ И ИНТЕРФЕЙСА

Е. А. Загородных

Воронежский государственный университет

Аннотация. В работе рассматривается разработка мобильного приложения «Расписание ВГУ», предназначенного для обеспечения удобного доступа студентов и абитуриентов к актуальной информации о расписании занятий и других важных данных университета. Описаны этапы проектирования, включая анализ существующих решений и формулировку технического задания. Представлена структура приложения, функциональные возможности и особенности реализации пользовательского интерфейса.
Ключевые слова: программирование, мобильное приложение, расписание, пользовательский интерфейс, проектирование, разработка, студенты, информационные технологии, оптимизация, взаимодействие с сервером.

Введение

На сегодняшний день все больше людей предпочитают мобильный и быстрый доступ к информации. С каждым годом процент пользователей мобильных версий сайтов растет, а, следовательно, полные версии становятся все менее популярны.

Согласно информации сайта I-tech.ru, количество различных мобильных устройств в 2024 году достигло уровня 5,6 миллиарда, что составило 69,4 % от общего населения планеты. Статистика сайта Cossa.ru говорит о том, что мобильным телефоном в России владеет 98 % населения. Следовательно, всё более перспективной представляется адаптация информации именно под смартфоны, а не под персональные компьютеры.

В современном мире быстрый доступ к информации играет ключевую роль в образовательном процессе. Студенты и абитуриенты стремятся получать нужные данные в удобной и оперативной форме. Однако на данный момент мобильного приложения для расписания ВГУ не существует (или оно мне неизвестно), что создаёт неудобства для пользователей, которым приходится искать информацию через браузер или полные версии сайта. В связи с этим, разработка мобильного приложения является актуальным путем для повышения доступности информации об университете.

Поэтому, было принято решение разработать мобильное приложение для расписания ВГУ, содержащее в себе часто используемую информацию с сайта.

1. Анализ существующих решений

Анализ предметной области выявил значительные преимущества мобильного приложения в сравнении с традиционной верстkой сайтов, такие как улучшенное восприятие информации на мобильных устройствах и сокращение времени на получение нужных данных.

При разработке приложения был использован современный инструментарий, включая Android Studio и язык программирования Kotlin, что позволило создать эффективный и удобный продукт. Jetpack Compose был отобран как главный инструмент для создания пользовательского интерфейса не только из-за его удобства в интеграции с другими системами, но и благодаря более простому и интуитивно понятному декларативному подходу.

Анализ аналогичных приложений показал, что они имеют схожую структуру ключевых элементов, реализация которой необходима в программном продукте для его корректного и быстрого взаимодействия с пользователем.

2. Техническое задание

В соответствии с алгоритмом разработки, перед началом процесса сбора данных и написания кода было сформулировано техническое задание на создание мобильного приложения, состоящее из следующих пунктов:

Цели и задачи приложения. Целью разработки приложения является оперативное обеспечение студентов необходимой информацией и привлечение абитуриентов за счет удобства и простоты доступа к информации университета. Приложение должно содержать данные о расписании занятий, расположении корпусов университета, контактах и ссылках на официальные электронные ресурсы.

Язык реализации. Для приложения используется русский язык. Кроме того, предусмотрена возможность реализации интерфейса на английском языке для иностранных студентов и абитуриентов.

Ориентация экрана. Предполагаемая ориентация экрана при работе приложения на мобильных телефонах – портретная, но переключение на альбомную ориентацию не должно вызывать критических ошибок.

Стартовый экран. Стартовый экран содержит информацию о расписании выбранной группы, загружаемую с удаленного сервера.

Операционная система и устройства, обеспечивающие работу приложения. Приложение должно функционировать с операционной системой Android версии 8.1 и выше.

Языки программирования. При разработке приложения используются следующие языки программирования: Kotlin, Python. Взаимодействие с сервером осуществляется путем HTTP-запроса.

Оформление. Дизайн создаваемого приложения должен соответствовать корпоративному стилю университета.

3. Структура приложения

Для разработки приложения продумана и разработана схема взаимного расположения его частей. Приложение «Расписание ВГУ» состоит из четырех экранов, переходы между которыми осуществляются посредством взаимодействия с управляющим элементом (рис. 1). При такой структуре приложение показывает наилучшие результаты производительности и удобства. От схемы напрямую зависит, сможет ли пользователь найти интересующий его материал.

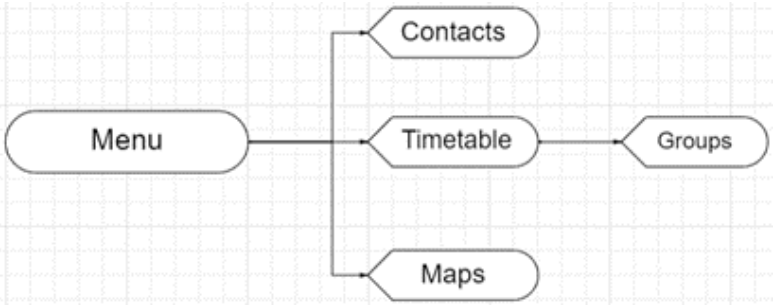


Рис. 1. Структура приложения

Приложение содержит следующие разделы, доступные для всех пользователей:

- «Расписание» — мобильная версия расписания занятий;
- «Группы» — список доступных для выбора групп;
- «Контакты» — раздел, содержащий основные контактные данные, такие как телефон, сайт, адрес;
- «Корпуса» — раздел с информацией о корпусах ВУЗа.

4. Описание разработанного приложения

Общим элементом для всех экранов является панель навигации с тремя элементами (Расписание, Корпуса, Контакты), каждый из которых имеет иконку и текстовую метку. При нажатии на элемент происходит навигация к соответствующему маршруту, а цвет иконки и текста изменяется в зависимости от того, выбран элемент или нет.

Экран расписания включает в себя верхнюю панель, проверку состояния подключения и отображение расписания с возможностью прокрутки. Заполнение экрана расписания производится данными, полученными с удаленного сервера. Экземпляры расписаний хранятся на нем в виде файлов с разрешением XML.

Для удобства отображения данных расписания в сети, на сервере размещены несколько документов HTML, формирующих таблицу архива расписаний и предоставляющих возможность просмотра данных в понятном пользователю виде с помощью технологии Github Pages.

Для загрузки файла с расписанием используется технология Retrofit 2. Важно отметить, что при наличии нескольких файлов на сервере, загружается актуальный экземпляр расписания. После загрузки файл преобразуется функцией в формат JSON. Полученный результат становится хранилищем неструктурированных данных. Затем информация заполняется в локальное хранилище с использованием классов. Это позволяет структурировать данные и оптимизировать доступ к ним. Номера групп хранятся в упорядоченном списке класса, который отличается от других тем, что содержит только номера групп для выбора, но не для самого расписания.

Верхнюю панель оборудована инструментами для выбора четности недели, обновления информации по расписанию, навигации к текущему дню и переходу к экрану для выбора учебной группы. В работе основной части экрана предусмотрены различные состояния отображения информации (рис. 2):

– При отсутствии данных или соединения с интернетом выводится уведомление о невозможности получить расписание из-за проблемы с JSON файлом или подключением к сети.

– Во время загрузки данных пользователю показывается индикатор загрузки и элементы для выбора группы.

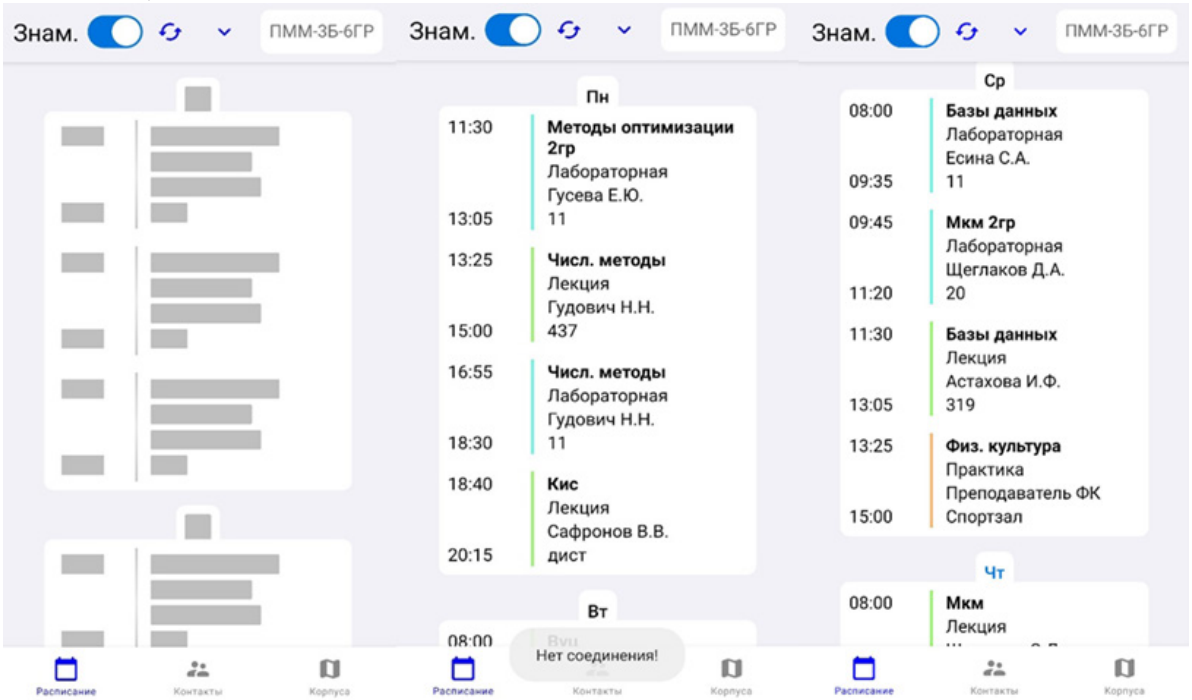


Рис. 2. Различные состояния экрана «Расписание»

– После загрузки данные фильтруются и на экран выводится расписание для конкретной группы и указанной недели.

Отдельно создается экран для выбора группы. Обеспечивает возможность поиска и фильтрации групп. В верхней панели задаются кнопка для возвращения на экран с расписанием и поле с поиском группы. Если список групп пуст, пользователю будет показано сообщение «No groups found».

Для отображения экрана со списком контактов (рис. 3) используется инструмент LazyColumn. С его помощью элементы списка загружаются по мере прокрутки, что уменьшает нагрузку на процессор. Также на некоторые контакты добавлены ссылки, чтобы пользователь смог быстро перейти на нужный ему сервис.

Для быстрой валидации полученных данных используется сущность, состоящая из названия контакта, его описание и ссылки (опционально).

Для отображения экрана с карточками корпусов ВГУ (рис. 4) реализована Composable-функция. Чтобы отобразить список корпусов также используется инструмент LazyColumn. Для создания карточки с информацией создается сущность, состоящая из названия корпуса, его фотографии, адреса и ссылки на геолокацию.

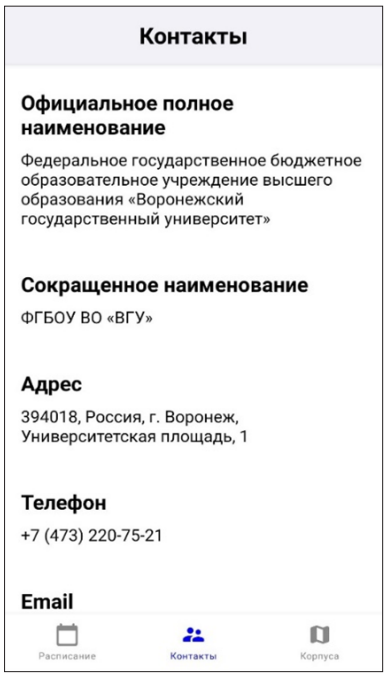


Рис. 3. Экран «Контакты»

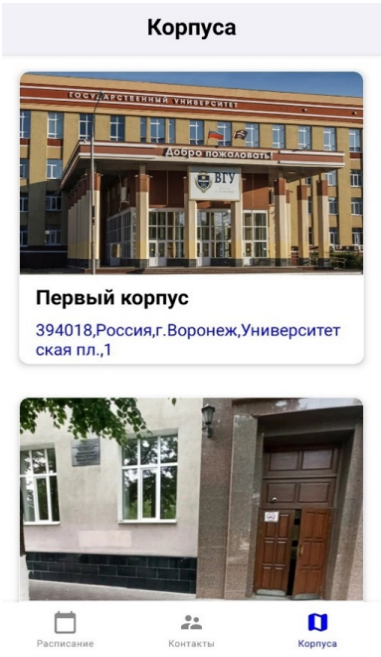


Рис. 4. Экран «Корпуса»

Заключение

Разработанное мобильное приложение «Расписание ВГУ» представляет собой мощный инструмент, обеспечивающий студентов оперативным доступом к необходимым данным. Учитывая его структуру и функционал, пользователи могут легко находить информацию о расписании, корпусах и контактных данных университета. Использование современных языков программирования и технологий, таких как Kotlin и Jetpack Compose, позволило добиться высокой производительности и удобства взаимодействия.

Приложение не только упрощает процесс получения информации, но и создает положительный имидж университета, способствуя привлечению новых абитуриентов. В дальнейшем планируется развитие функционала и интеграция дополнительных возможностей для улучшения пользовательского опыта.

Литература

1. Рынок мобильных приложений в России и мире. – (URL: http://www.json.ru/poleznye_materialy/rynok_mobilnyh_prilozhenij_v_mire) (дата обращения: 15.05.2024).

2. Приложения в Google Play – NinjaMirea. – (URL: https://play.google.com/store/apps/details?id=ninja.mirea.mireaapp&hl=ru) (дата обращения: 15.05.2024).

3. *Дэрси Л.* Разработка приложений для Android-устройств. Т. 1. Базовые принципы / Л. Дэрси, Ш. Кондер. – Москва : Эксмо, 2014. – 598 с.

4. Официальный сайт ВГУ. – (URL: https://www.vsu.ru/) (дата обращения: 15.05.2024).

5. *Майер Р.* Программирование приложений для планшетных компьютеров и смартфонов / Р. Майер. – Москва : Эксмо, 2013. – 816 с.

6. *Гриффитс Р. Д.* Head First. Программирование для Android / Р. Д. Гриффитс. – Санкт-Петербург : Питер, 2016. – 704 с.

УДК 004.424.5

АНАЛИЗ ВЫБОРА ОПОРНОГО ЭЛЕМЕНТА ДЛЯ БЫСТРОЙ СОРТИРОВКИ СПИСКА

И. А. Клюев, М. С. Ефремов

Воронежский государственный университет

Аннотация. Данная работа посвящена анализу алгоритма быстрой сортировки с использованием списковых структур данных. В работе проводится анализ возможных случаев деградации скорости работы алгоритма. Подымается проблема выбора необходимых структур данных при использовании нестандартных распределителей памяти. Предлагается возможный вариант выбора опорного элемента для быстрой сортировки при последовательном доступе к данным.
Ключевые слова: сортировка, упорядочивание, быстрая сортировка, быстрая сортировка списка, сортировка списка, quick sort, qsort.

Введение

В работе любой информационной системы может возникнуть задача упорядочивания данных, поэтому скорость выполнения данной операции является важной характеристикой. Для решения данной задачи разработано множество алгоритмов сортировок.

Самые распространённые способы упорядочивания основаны на непосредственном сравнении двух записей. Такие алгоритмы легко обобщать на разные типы данных, использовав специальную функцию сравнения — компаратор. Такие подходы в основном используют в встроенных средствах языков программирования, так как заранее неизвестно с какими данными необходимо работать программисту.

Одним из самых известных алгоритмов упорядочивания является быстрая сортировка или сортировка Хоара (qsort). В большинстве практических случаев данный способ упорядочивания показывает хорошие результаты. Так, в исследовании [1] авторы производят сравнение сортировки слиянием и быстрой сортировки, где вторая показывает более высокую скорость работы.

Хотя быстрая сортировка обычно демонстрирует хорошую производительность, также возможна и её деградация в некоторых случаях. Это хорошо рассмотрено в пособии Никлауса Вирта [2] в соответствующем разделе.

В большинстве программных пакетов алгоритм быстрой сортировки реализован только для непрерывных данных, таких как массивы, где возможно быстро осуществить доступ к произвольному элементу. Однако, не во всех информационных системах можно представить непрерывный набор данных. Как правило это касается систем, использующие нестандартные распределители памяти, которые в пользу некоторых оптимизаций накладывают ограничения на их использование. В работе [3] рассматривается сложность представления непрерывной памяти в одной из таких систем.

В таком случае известные алгоритмы надо адаптировать под определенную архитектуру программного обеспечения. В описанном ранее случае одним из вариантов решения является использование списковых структур данных, в которых выделяемые блоки памяти ограничены в наборе возможных значений. В данной работе предлагается метод быстрой сортировки с использованием односвязного списка, который обладает хорошей устойчивостью к возможным деградациям скорости.

1. Метод быстрой сортировки

В начале работы один из элементов исходного набора выбирается в качестве «опорного элемента». Далее происходит деление исходного набора на две части, так чтобы в одной из частей все элементы были не больше опорного, а в другой не меньше. Такая операция рекурсивно повторяется для каждого полученного набора пока не останется одна запись.

Как правило, для уменьшения глубины рекурсии используется гибридный подход с переключением на другой метод сортировки при достижении определенных условий. Например, данный подход лежит в основе сортировки Introsort, реализованной в стандартных библиотеках некоторых языков программирования [4].

Одной из самых известных оптимизация алгоритма является деление упорядочиваемого набора на три части. Отдельно выбирается набор с данными, которые равны опорному элементу. Такой набор по определению является отсортированным, поэтому достаточно рекурсивно обработать другие два набора. Данный подход хорошо подходит в случае, когда сортируется выборка с большим числом одинаковых элементов.

При использовании массивов данных для разделения на наборы производится обмен необходимых элементов «на месте». То есть, полученные наборы представляют независимые отрезки данных в пределах входного массива. Таким образом, алгоритм потребляет память только для обеспечения рекурсивного прохода.

В рассматриваемом случае предлагается каждый набор представлять как отдельный односвязный список, хранящий указатель на начальный и конечный узлы. Узлы входного списка будут распределяются по необходимым наборам. После упорядочивания каждого такого набора, предлагается их объединить в итоговый список. Именно для быстрого выполнения данной операции и требуется хранение указателя на конечный узел. Как и в случае сортировки массивов, алгоритм требует дополнительную память только для обеспечения рекурсии.

2. Проблема выбора опорного элемента

Худший случай работы быстрой сортировки — это когда каждый раз в качестве опорного будет выбираться минимальный или максимальный элемент. В такой ситуации за один проход по набору будет происходить упорядочивание всего одной записи, а алгоритмическая сложность всей сортировки будет квадратичной.

Для того чтобы предотвратить упадок скорости очень важно правильно выбирать опорный элемент. Для лучшего распределения по полученным наборам выгоднее всего выбирать медиану — средний элемент из всего входного потока. Однако, точный поиск медианы затрачивает слишком много дополнительных операций.

Для уменьшения сопутствующих затрат медиану можно выбирать только из нескольких записей сортируемой выборки. Даже при таком раскладе риски выбрать опорный элемент неудачно будут минимальны. Чем больше сортируемый набор, тем больше элементов стоит взять для нахождения медианы. При сортировке массива можно выбрать медиану из случайных элементов или равномерно распределенных.

При упорядочивании списка выбор опорного элемента затрудняется. Список поддерживает только последовательный доступ к данным. Получается, что без дополнительного прохода по списку невозможно выбрать опорный элемент без рисков падения скорости сортировки. Однако, дополнительный просмотр списка неприемлем с точки зрения скорости работы. Для решения этой проблемы далее предлагается один из возможных способов выбора опорного элемента.

3. Способ нахождения опорного элемента

Предлагается выбирать опорный элемент как медиану из 9 элементов входного потока. Далее всё будет анализироваться для этого случая. При необходимости алгоритм можно обобщить и на другие размеры наборов. Основная идея – выборка опорного элемента в процессе сортировки. Во время разделения исходного набора уже делается проход по списку. Во время выполнения данного процесса можно выбрать опорный элемент и использовать его при дальнейшей работе алгоритма.

Для этого следует использовать вспомогательный массив из 9 элементов, где один элемент — это указатель на запись в сортируемом списке. Необходимо аккумулировать ссылки на упорядочиваемые записи в этот массив с определённым шагом. В качестве начального шага предлагается взять значение 4. То есть каждая четвертая запись должна быть помещена в рассматриваемый массив. Для обеспечения этого процесса каждому элементу списка будет присвоен порядковый номер. Номера в списке и индексы в массиве будут начинаться с нуля.

После того как следующая запись не может поместиться в массив, осуществляется удаление элементов из массива с нечётными номерами, а также смещение элементов с чётными номерами вперёд. Получается свободное пространство в конце массива, где на первое освободившиеся место добавляется новый элемент. После того как произойдёт такая процедура, шаг увеличивается в 2 раза. То есть, после первого выполнения данной процедуры, будет аккумулироваться каждый восьмой элемент, потом каждый шестнадцатый и так далее.

После просмотра входного потока данных опорный элемент выбирается среди записей из рассматриваемого массива. Так как рекурсивная сортировка происходит для нескольких полученных наборов, то процесс выбора опорного элемента следует провести для каждого из них.

Однако, до сих пор неизвестно каким образом выбирать опорный элемент перед первым проходом алгоритма. В таком случае опорный элемент можно выбрать из первых записей списка, хоть это и не гарантирует удачность выбора. Если же данные приходят извне из некоторого потока, то выбор можно произвести в период чтения этого потока.

4. Ленивая сортировка

Одним из примечательных достоинств быстрой сортировки в том, что можно откладывать упорядочивание неиспользуемых данных.

Во время процесса сортировки исходный набор данных делится на наборы меньшего объема. Далее каждый такой набор упорядочивается независимо друг от друга. Сортировку некоторых полученных наборов можно откладывать, пока данные из них не будут запрошены. Также это ускоряет получение первой порции упорядоченных данных. Примечательно, что использование данной особенности не увеличивает время работы для получения полного результата.

Эта позволяет отдавать данные порционно в потоковом режиме Такая возможность довольно привлекательна в большом ряде задач. Например, при представлении данных с помощью пользовательского интерфейса, где записи загружаются по мере их просмотра.

Заключение

В результате данной работы был исследован алгоритм быстрой сортировки с использованием односвязного списка. В ходе работы было проанализировано возможное падение скорости работы алгоритма в случае неудачного выбора опорного элемента. Была предложена идея по выбору данного элемента в случае последовательного доступа к данным. Данный под-

ход возможно использовать в системах с нестандартными распределителями памяти, где есть ограничение на используемые структуры данных.

Литература

1. *Nunes-Harwitt A.* Quick-Sort: A Pet Peeve / A. Nunes-Harwitt, M. Gambogi, T. Whitaker // SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education (Baltimore Maryland, February 21-24, 2018). – New York : Association for Computing Machinery, 2018 – P. 547–549.

2. *Вирт Н.* Алгоритмы и структуры данных. Новая версия для Оберона. / Н. Вирт ; пер. с англ. В. Ф. Ткачева – Москва : ДМК Пресс, 2016. – 272 с.3. *Клюев И. А.* Поразрядная сортировка строк для систем с дискретно-страничной организацией памяти / И. А. Клюев, М. С. Ефремов // Сборник трудов Международной научной конференции Актуальные проблемы прикладной математики, информатики и механики (Воронеж, 4–6 декабря 2023 г). – Воронеж : Издательство «Научно-исследовательские публикации», 2024. – С. 864–869.

4. *David R. M.* Introspective sorting and selection algorithms / R. M. David // Software: Practice and Experience. – 1997. – Vol. 27, № 8. – P. 983–993.

УДК 004.415.2

ПРОЕКТИРОВАНИЕ WEB-ПРИЛОЖЕНИЯ ДЛЯ ПРЕДСТАВЛЕНИЯ ДАННЫХ СПОРТСМЕНОВ

О. О. Корнева, М. В. Матвеева

Воронежский государственный университет

Аннотация. В работе рассматривается проектирование веб-приложения для представления данных спортсменов. В статье сформулированы требования к приложению, а также приводится спроектированная модель данных для хранения информации и структура серверной части веб-приложения.
Ключевые слова: программирование, веб-приложение, веб-разработка, java, проектирование веб-приложения, легкая атлетика.

Введение

В условиях повышенной конкуренции и необходимости оптимизации тренировочного процесса спортсмены сталкиваются с ситуациями, требующими доступа к структурированной информации о результатах соперников и предстоящих соревнованиях. В связи с этим возникает потребность в разработке специализированного веб-приложения, которое позволит спортсменам получать актуальные данные, которые могут поспособствовать улучшению результатов за счет более точного подхода к подготовке на основании представленной информации.

В статье будет описано проектирование приложения, предоставляющего возможность просмотра данных о спортсменах легкоатлетах, предстоящих соревнованиях и рейтинговых списках. Доступ к этим данным позволит пользователям самостоятельно анализировать информацию о результатах завершившихся спортивных мероприятий, строить тренировочные планы на основе календаря соревнований и оценивать свои силы, ориентируясь на рейтинги других спортсменов.

1. Анализ существующих решений

На данный момент существует только один ресурс, предоставляющий доступ к данным и результатам легкоатлетов VFLA [1].

При анализе существующего решения были выявлены следующие недостатки:

• Отсутствие возможности поиска спортсмена по комбинации «Имя, Фамилия». Данная функциональность создает ограничения поиска и затрудняет доступ к информации, что делает взаимодействие пользователя с системой менее удобным.

• Отсутствие разграничения функций для спортсменов и судей. Большая часть пользователей веб-приложения является спортсменами, для которых информация для судей является не актуальной.

• Некорректное отображение данных в столбцах личного кабинета спортсмена. При взаимодействии с системой данные представлены не структурированно, что затрудняет восприятие информации пользователями.

• Главная страница не отображает сущности веб-приложения. Кратко не изложена основная задача системы, обилие вкладок затрудняет работу с приложением.

Исходя из вышесказанного, возникла необходимость с в разработке веб-приложения, которое бы учитывало все недочеты рассмотренного ресурса и развивало уже имеющуюся функциональность, повысив удобство взаимодействия пользователей с системой.

2. Требования к приложению

Веб-приложение должно предоставлять следующие возможности:
• регистрация в системе;
• аутентификация в системе;
• просмотр личных данных спортсмена:
– ФИО;
– дата рождения;
– разряд;
– тренерский штаб спортсмена;
– соревнования, в которых было принято участие с указанием результата;
• просмотр данных о предстоящих соревнованиях и итогах уже прошедших:
– название мероприятия;
– дата проведения;
– итоговые протоколы соревнований;
• просмотр топ-листов:
– сортировка по возрастной группе и дистанции;
– сортировка по очкам соревновательного сезона.

Основная функциональность приложения доступна только авторизованным пользователям. Неавторизованный пользователь должен пройти регистрацию или войти в систему, если был зарегистрирован раннее.

3. Модель данных

Для хранения пользовательской информации, сведений о спортивных мероприятиях, разрядах и тренерском составе была разработана база данных [2]. На рис.1 изображена логическая модель.

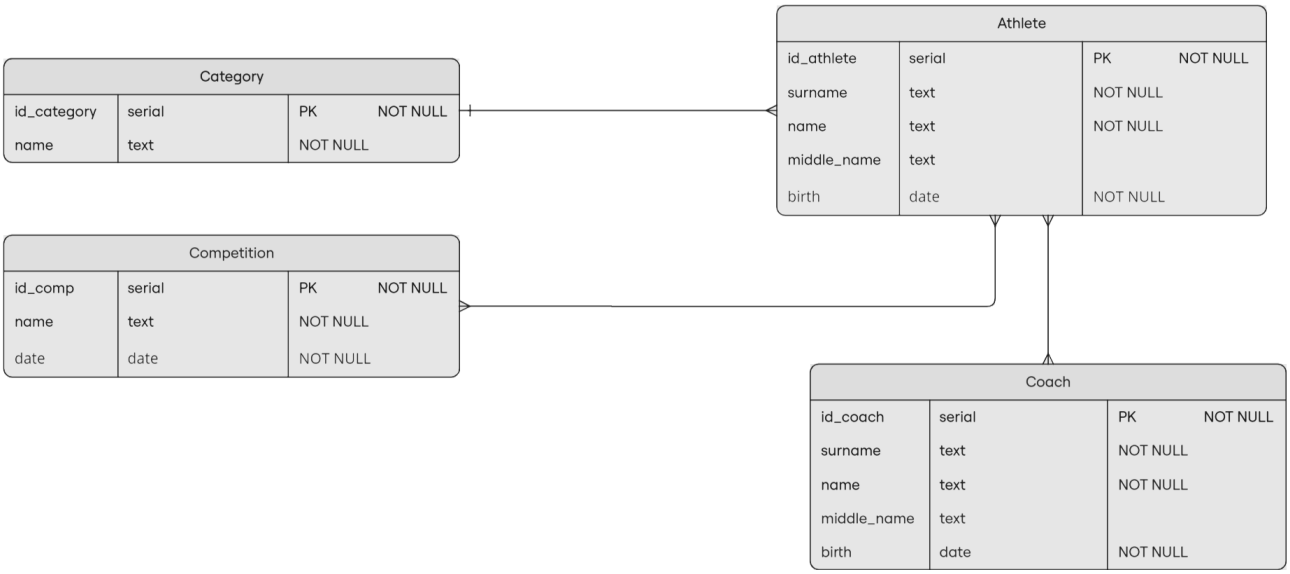


Рис. 1. Логическая модель данных

Сущность Athlete содержит личные данные спортсмена, ее структура представлена в табл. 3.1.

Сущность *Coach* содержит данных о тренерском составе и состоит из повторяющегося набора атрибутов таблицы *Athlete*.

Таблица 3.1

Сущность Athlete

Имя атрибута	Тип данных	Назначение	Ограничения
id_athlete	serial	Уникальный идентификатор. Новые значения автоматически генерируются при вставке данных в таблицу	Первичный ключ, NOT NULL
surname	text	Фамилия спортсмена	NOT NULL
name	text	Имя спортсмена	NOT NULL
middle_name	text	Отчество спортсмена	Допускается NULL значение
birth	date	Дата рождения спортсмена	NOT NULL

Coach соединены с отношением *Athlete* связью «многие-ко-многим». Это спроектировано данным образом с целью учесть наличие у тренера нескольких спортсменов, а также большого тренерского штаба, работающего с одним спортсменом.

Сущность *Category* содержит сведения о разрядах, ее структура приведена в табл.3.2.

Category соединена связью «один-ко-многим» с сущностью *Athlete*, так как специфика легкой атлетики подразумевает наличие только одного действующего разряда у спортсмена.

Таблица 3.2

Сущность Category

Имя атрибута	Тип данных	Назначение	Ограничения
id_category	serial	Уникальный идентификатор. Новые значения автоматически генерируются при вставке данных в таблицу	NOT NULL, Первичный ключ
name	text	Название разряда	NOT NULL

Сущность *Competition* содержит информацию о соревнованиях, ее структура представлена в табл.3.3.

Competiton соединена с сущностью *Athlete* связью «многие-ко-многим» для того, чтобы учесть многочисленные участия спортсмена в соревнованиях.

Таблица 3.3

Сущность Competition

Имя атрибута	Тип данных	Назначение	Ограничения
id_comp	serial	Уникальный идентификатор соревнования. Новые значения автоматически генерируются при вставке данных в таблицу	Первичный ключ, NOT NULL
name	integer	Название соревнования	NOT NULL
date	date	Дата проведения соревнования.	NULL значение допускается

4. Архитектура серверной части веб-приложения

Под архитектурой приложения понимается структура проекта и его компоненты. Разрабатываемое веб-приложение включает в себя следующие части:

- services (сервисы) — содержат бизнес-логику, выполняют *CRUD* операции [3];
- controllers (контроллеры) — отвечают за обработку HTTP-запросов;
- repository (репозиторий) — обеспечивают единый интерфейс доступа к данным для других компонентов;

• entity (сущности) — представляют собой объекты данных, которые используются в приложении для хранения информации;

• dto (англ. data transfer objects) — используются для передачи информации между компонентами, оптимизируют процесс за счет исключения избыточности [4];

• mappers (мапперы) — управляют преобразованием объектов между различными форматами;

• exceptions (исключения) — отвечают за обработку различных ошибок, возникающих при валидации и авторизации;

• security (безопасность) — обеспечивают защиту от атак, управление правами доступа к ресурсам приложения.

Структура java-проекта представлена на рис. 2.

Такой набор используемых компонентов позволяет наиболее гибко и с минимальными затратами подойти к разработке веб-приложения для представления данных спортсменов.

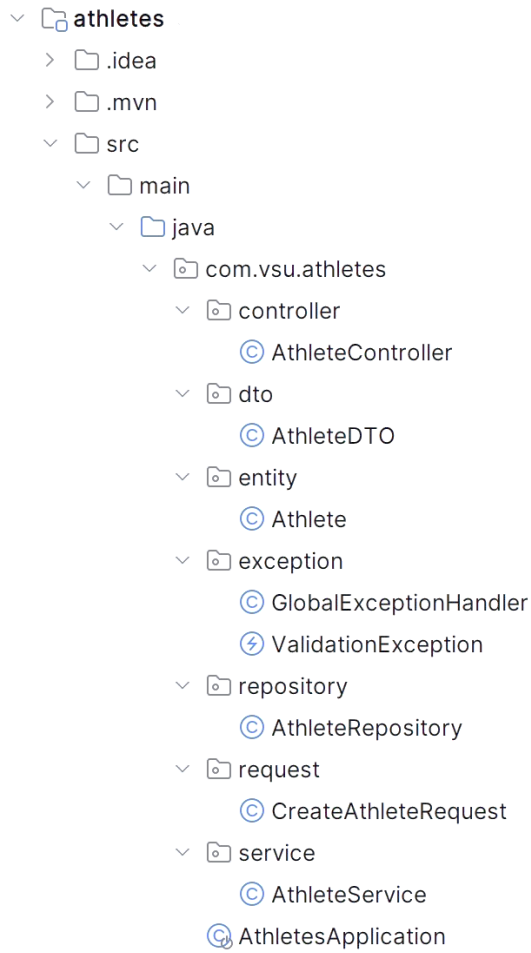


Рис. 2. Структура серверной части проекта

5. RESTful-сервис

REST API — набор правил взаимодействия веб-приложения с сервером, которые применяются в тех случаях, когда пользователю нужно предоставить данные с сервера, а также они отвечают за унификацию и стандартизацию взаимодействия компьютеров в одной сети.

В ходе разработки были использованы следующие принципы *REST API*:

• Обмен данными осуществляется с использованием стандартных HTTP-методов: GET, PUT, POST, DELETE.

• Унификация интерфейса. Для каждого ресурса был выбран индивидуальный идентификатор (URI).

•Использование набора гиперссылок. Сервер направляет клиент на следующие возможные шаги.

•При взаимодействии используются статус-коды ответов *API*. На положительный ответ от сервера указывают статус-коды 200, при ошибке обработки запроса используются коды 400, внутренняя ошибка обладает статус-кодом 500.

•При передаче данных использовался формат JSON. Преимуществом такого формата является легковесность и простота читаемости данных.

6. 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 интуитивно понятным и отражать основную идею приложения.

При создании интерфейса использовался уже существующий логотип «ВФЛА», так как федерация является узнаваемой общественной общероссийской организацией.

В структуре интерфейса были определены основные страницы:

– Страница просмотра ближайших соревнований. Отображает спортивные мероприятия, которые сформированы в соответствии с календарем.

– Страница отображения топ-листа. Предоставляет пользователю рейтинговый список по заданным критериям.

– Страница поиска спортсменов. Отображает окно поиска спортсменов, для дальнейшего ознакомления с его личными данными и результатами по итогам прошедших соревнований.

Самой частой по запросам спортсменов является страница просмотра ближайших соревнований, которая представлена на рис.3. В верхней части страницы расположено меню, которое помогает пользователю сориентироваться в разделах.

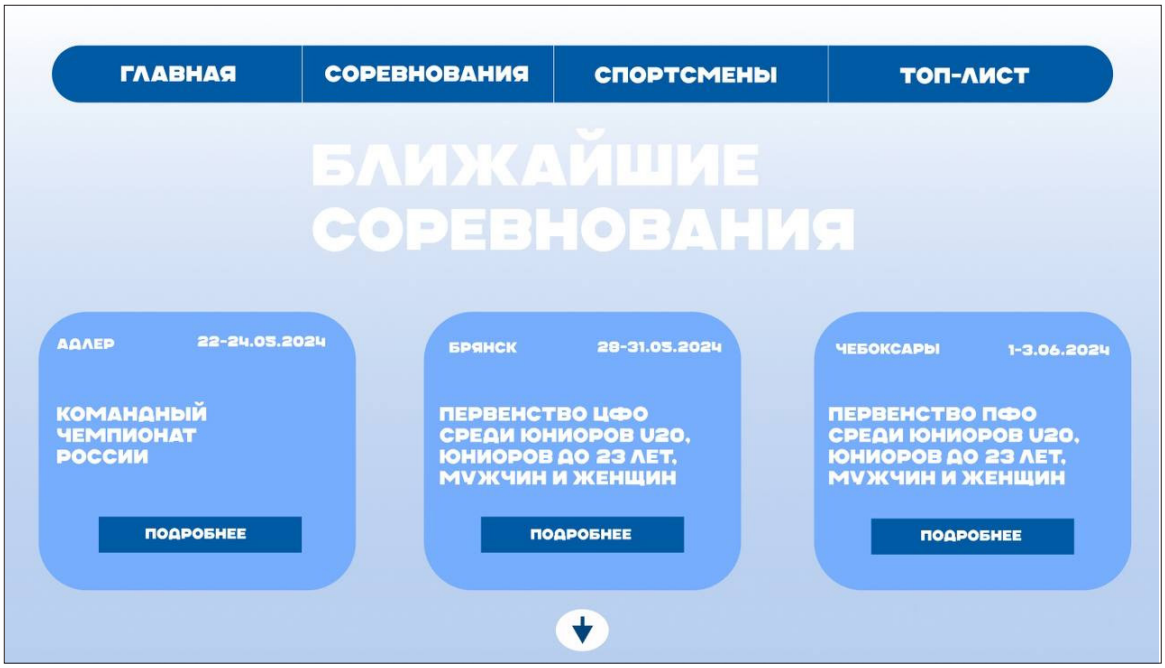


Рис. 3. Страница просмотра ближайших соревнований

На рис.4 представлена страница авторизованного пользователя. Данная страница отображает основные задачи веб-приложения, тем самым повышая удобство при взаимодействии пользователя с системой.

Рис. 4. Страница авторизованного пользователя

Заключение

Результатом работы является проект веб-приложения для представления данных спортсменов. Были разработаны требования к приложению, база данных и структура его серверной части. На данный момент приложение находится на стадии реализации. Планируется реализовать клиентскую часть, разработать улучшенную функциональность для серверной части, а также перенести ресурс на хостинг и ввести его в эксплуатацию.

Литература

1. Vfla – URL: https://vfla.lsport.net (дата обращения: 17.11.2024).
2. *Конноли Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Конноли, К. Бегг пер. с англ. – М. : Издательский дом «Вильяме», 2003. – 1440 с.
3. *Блох Д.* Java Эффективное программирование / Д. Блох. – Москва : Лори, 2016. – 440 с.
4. *Шилдт Г.* Java: Руководство для начинающих. – М. : Эксмо, 2018.

УДК 004.428.4

РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ВИДЕОХОСТИНГА НА ЯЗЫКЕ JAVA

М. С. Кравцова

Воронежский государственный университет

Аннотация. В данной работе рассматривается проектирование веб-приложения видеохостинга, а также реализация серверной части данного приложения на языке программирования Java. В статье кратко рассмотрена функциональность приложения, приведен анализ библиотек в Java, предназначенных для backend-разработки, представлена модель базы данных видеохостинга, а также рассмотрены функциональные слои серверной части веб-приложения и их взаимодействие между собой.

Ключевые слова: веб-приложение, программирование, язык Java, backend-разработка, Spring Framework, Java Servlet API, серверная часть приложения, проектирование веб-приложения, видеохостинг.

Введение

На протяжении всей истории человек нуждается в получении различной информации. В наше время источниками информации служат не только газеты и книги, а также статьи в Интернете, различные аудио и видеоматериалы. Очень большую популярность сейчас набирают именно видеоматериалы, потому что они несут как звуковую, так и визуальную информацию. Поэтому в наше время появились такие сервисы, как видеохостинги, на которых каждый человек может найти необходимый материал по интересующей теме. Также каждый человек может загрузить свой видеоматериал, который окажется полезным или развлекательным для огромного числа других пользователей системы.

На данный момент существует немалое количество видеохостингов, но, в основном, они являются зарубежными ресурсами. Некоторые из них не доступны в нашей стране. Видеохостингов, разработанных в России, сейчас очень мало. По данным причинам очень актуально создать сервис такого рода, имеющий происхождение и использование именно в нашей стране.

1. Функциональность приложения видеохостинга

Веб-приложение должно обладать следующей функциональностью:

– регистрация, создание или удаление существующего канала;

– обзор страницы существующего канала, своего или чужого;

– обзор видео, загруженных пользователем;

– обзор видео, просмотренных пользователем;

– обзор видео, оцененных пользователем;

– обзор видео, создателями которых являются пользователи, на которых подписан данный пользователь;

– обзор каналов пользователей, на которых подписан данный пользователь;

– обзор плейлистов пользователя;

– обзор видео, находящихся в определенном плейлисте;

– обзор комментариев, оставленных под конкретным видео;

– поиск видео по его названию;

– поиск видео по названию его создателя;

– обзор видео, принадлежащих к определенной категории;

– загрузка видео на свой канал или удаление видео с канала;

– просмотр видео;
– оценка видео или удаление своей оценки;
– создание или удаление комментария к видео;
– оценка комментария, оставленного к видео или удаление оценки;
– подписка на канал или осуществление отписки;
– создание или удаление плейлиста с видео;
– добавление видео в плейлист или исключение его из плейлиста.

2. Анализ библиотек в Java, предназначенных для backend-разработки

2.1. Анализ технологии Java Servlet API

В языке программирования Java существует множество библиотек, предназначенных для реализации серверной части приложений. Для анализа были взяты такие технологии, как Java Servlet API и Spring Framework.

Java Servlet API — это набор интерфейсов и классов, который позволяет разработчикам создавать серверные приложения на Java, динамически генерирующие контент в ответ на HTTP-запросы [1]. Существуют несколько типов сервлетов, такие как GenericServlet — базовый класс для всех сервлетов, HttpServlet — класс, который используется для обработки HTTP-запросов, Filter — класс, который используется для перехвата и обработки HTTP-запросов и ответов.

Плюсы данной технологии: полный контроль над серверными ресурсами и создание приложения любой сложности, обеспечение высокой производительности, возможность работать на любой платформе, поддерживающей Java Servlet API, комбинация с другими технологиями Java, большое сообщество разработчиков, множество библиотек, инструментов и учебных материалов.

Минусы данной технологии: большое количество кода, ручная обработка многих деталей, нет многих функций, таких как аутентификация, авторизация и маршрутизация URL, работа только в web-контейнере, устаревшая технология, новые функции и обновления не добавляются.

Можно сделать вывод, что Java Servlet API — это мощный инструмент для создания веб-приложений Java. Он обеспечивает гибкость и контроль над созданием динамического контента. Но данная технология уже устарела. Сейчас существует множество новых технологий, обеспечивающих удобную и гибкую разработку, например, Spring Framework.

2.2. Анализ технологии Spring Framework

Spring Framework — это открытый, объектно-ориентированный фреймворк для разработки корпоративных Java-приложений [2]. Он предоставляет набор функций, которые упрощают разработку, развертывание и управление такими приложениями. Данная технология основывается на принципах, таких как: Inversion of Control — Spring управляет жизненным циклом объектов приложения и их зависимостями [3], Dependency Injection — объекты получают свои зависимости через их внедрение, а не создают их самостоятельно, Aspect-Oriented Programming — Spring позволяет внедрять поперечные сечения, такие как безопасность, логирование и транзакции, в код приложения без изменения самого кода.

Очень важную роль в данной технологии имеют аннотации — это мощный инструмент, который позволяет разработчикам добавлять метаданные к классам, методам и полям, что делает более удобным разработку.

Плюсы данной технологии: сокрытие сложных деталей разработки, разработчики могут сосредоточиться на бизнес-логике, обеспечение высокой производительности приложений,

модульная структура и внедрение зависимостей, возможность разработки масштабируемых приложений, большое и активное сообщество разработчиков, возможность найти множество библиотек, инструментов и учебных материалов, поддержка широкого спектра технологий, таких как JPA, Hibernate, JMS и веб-сервисы.

Минусы данной технологии: много функций и концепций, что может усложнить освоение для начинающих разработчиков.

Можно сделать вывод, что Spring Framework — это мощный и популярный фреймворк с широким набором функций, которые могут помочь разработчикам создавать надежные, масштабируемые и безопасные приложения.

2.3. Сравнение двух проанализированных технологий

После рассмотрения каждой из технологий можно сравнить их по различным критериям (табл. 1).

Таблица 1

Сравнение Java Servlet API и Spring Framework

Критерий	Java Servlet API	Spring Framework
Функциональность	Только базовые возможности для создания веб-приложений.	Широкий набор функций, таких как AOP, IoC, Dependency Injection, транзакции, безопасность и тестирование.
Объем разработки	Рукописный код для многих задач, таких как обработка запросов, маршрутизация и MVC.	Сосредоточение разработчиков на бизнес-логике
Масштабируемость разработанных приложений	Не подходящий выбор для создания сложных, масштабируемых веб-приложений.	Отличный выбор для разработки масштабируемых приложений.
Тестируемость	Низкая тестируемость	Высокая тестируемость
Поддержка и сообщество	Не обновляемая документация, малое количество учебных материалов.	Множество документации и учебных материалов, которые со временем добавляются.

В итоге, можно сказать, что в настоящее время Java Servlet API уже не обладает достаточным количеством функциональности, необходимой для удобной разработки. Spring Framework будет более удобным и простым для разработки. По многим анализируемым параметрам он превосходит Java Servlet API. Поэтому данная технология является наиболее удачной для использования в проекте.

3. Модель данных видеохостинга

Для работы веб-приложения необходима база данных для хранения различной информации. Модель данных видеохостинга представлена на рис. 1. В табл. 2 описывается назначение таблиц базы данных, а также перечень их столбцов.

Таблица 2

Таблицы базы данных видеохостинга

Таблица	Столбцы	Описание таблицы
Категория видео	Id (PK), название категории	Справочная таблица, хранит категории видео

Видео в категории	Id категории (PK, FK), id видео (PK, FK)	Таблица содержит данные о том, какое видео к какой категории относится
Видео	Id (PK), id пользователя (FK), название, продолжительность, описание, дата выхода	Таблица содержит информацию о видео
Просмотр видео	Id (PK), id пользователя (FK), id видео (FK), дата просмотра	Таблица содержит данные о просмотре пользователями видео
Оценка видео	Id (PK), id пользователя (FK), id видео (FK), дата оценки, наличие положительной оценки	Таблица содержит информацию об оценке пользователями видео
Видео в плейлисте	Id (PK), id плейлиста (FK), id видео (FK), дата добавления	Таблица содержит информацию о том, какие видео в каких плейлистах содержатся
Комментарий	Id (PK), id пользователя (FK), id видео (FK), текст комментария, время создания комментария	Таблица содержит данные о комментарии к видео
Пользователь	Id (PK), электронная почта, название канала, описание, дата регистрации, пароль	Таблица содержит данные о пользователе, зарегистрированном на видеохостинге
Плейлист	Id (PK), id пользователя (FK), название плейлиста, дата создания плейлиста	Таблица содержит информацию о плейлисте
Оценка комментария	Id (PK), id пользователя (FK), id комментария (FK), наличие положительной оценки	Таблица содержит информацию об оценке пользователями комментариев
Подписка	Id пользователя, осуществившего подписку (PK, FK), id пользователя, на которого подписались (PK, FK)	Справочная таблица. Показывает какой пользователь имеет подписку на других пользователей

4. Функциональные слои серверной части видеохостинга

Backend-часть веб-приложения содержит в себе три функциональных слоя: репозиторий, сервис и контроллер.

4.1. Слой репозитория

Репозиторий нужен для работы с хранилищем данных. В классах-репозиториях содержатся методы, отправляющие запросы в базу данных и возвращающие ответ на слой сервиса.

В реализуемом приложении репозитории представлены интерфейсами.

На данном слое используются классы-сущности (entity), предоставляющие поля, которые являются атрибутами соответствующих отношений в базе данных. Примеры таких классов:

1. **Класс User.** Данный класс предоставляет информацию о пользователе и содержит поля: private Long idUser, private String email, private String channelName, private String description, private Timestamp dateOfRegistration, private String password, private List<User> subscriptions, private List<Video> videos, private List<Playlist> playlists.

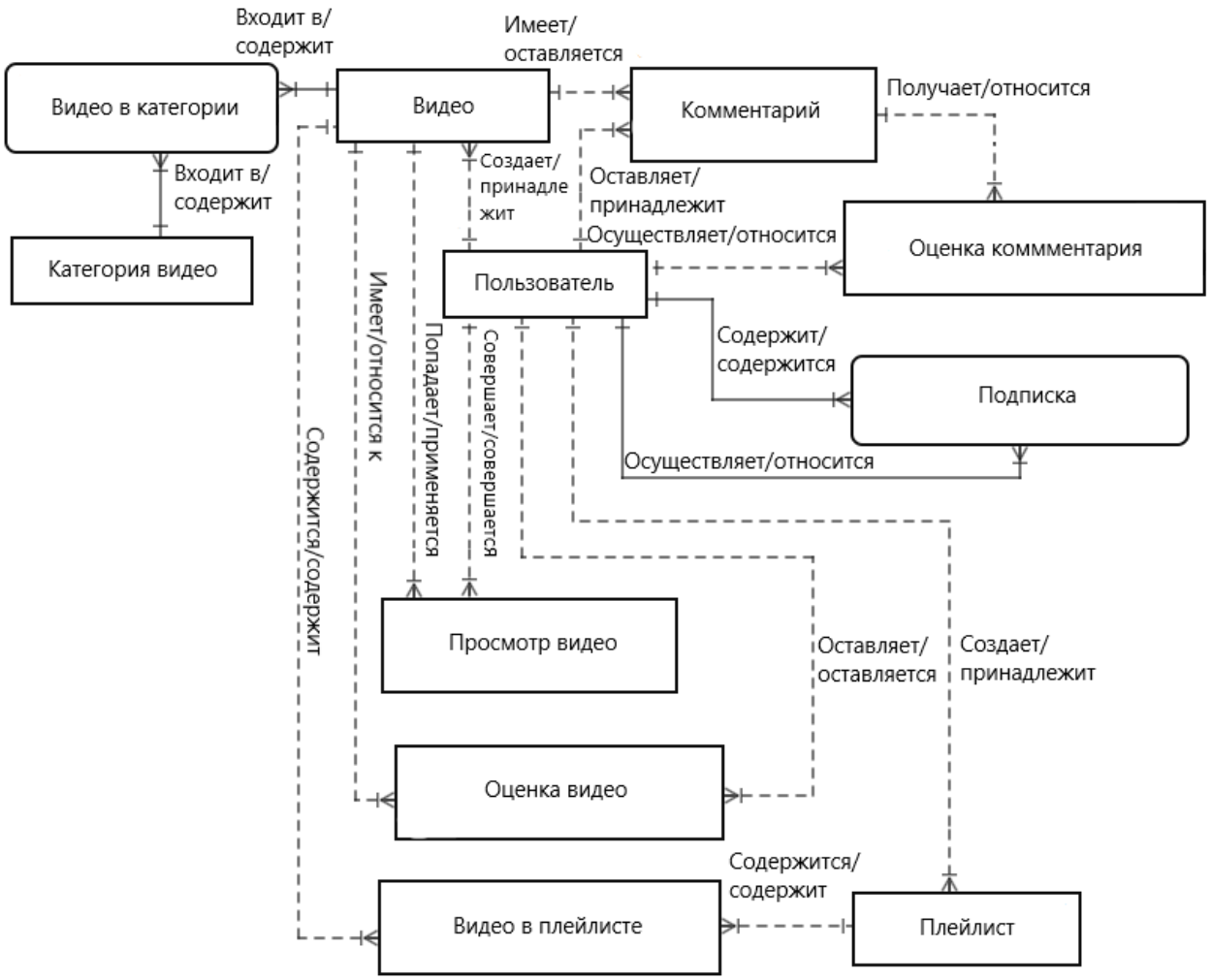


Рис. 1. Модель данных видеохостинга

2. **Класс Video.** Данный класс предоставляет информацию о видео и содержит поля: private Long idVideo, private User user, private String name, private Long duration, private String description, private Timestamp releaseDateTime, private List<Category> categories.

Примеры репозиториев:

1. **Интерфейс UserRepository.** Данный интерфейс наследует интерфейс JpaRepository<User, Long> и имеет методы:

– Long getSubscribersCountByIdUser(Long idUser) — возвращает количество подписчиков для пользователя с id, равным параметру idUser.

– User getUserByEmail(String email) — возвращает пользователя по его email, который равен параметру email.

Помимо указанных методов репозиторий использует методы родительского класса: save (User entity), deleteById (ID id), findById (ID id).

2. **Интерфейс VideoRepository.** Данный интерфейс наследует интерфейс JpaRepository<Video, Long> и имеет методы:

– List<Video> getVideosBySubscriptions(Long idUser) — возвращает список видео, принадлежащих пользователю с id, равным параметру idUser.

– List<Video> findByNameContaining(String videoName) — возвращает список видео, в название которых входит строка из параметра videoName.

– List<Video> findByUser_ChannelNameContaining(String name) — возвращает список видео, у которых в названии канала автора присутствует строка из параметра name.

– List<Video> findDistinctByCategories_NameIn(List<String> categories) — возвращает список видео, у которых хотя бы одна из категорий входит с список категорий из параметра categories.

Также, помимо указанных методов, репозиторий использует методы родительского класса: save (Video entity), deleteById (ID id), findById (ID id).

4.2. Слой сервиса

Сервис реализует бизнес-логику приложения. В классах-сервисах выполняется проверка данных на соответствие бизнес-правилам, установленным для приложения. Данные классы используют объекты классов-репозиториев, преобразуют ответы их методов, а далее отправляют преобразованные ответы на слой контроллера.

Слой сервиса использует классы (model), имеющие такие же поля, как и в соответствующих entity-классах. Также имеются дополнительные поля, которые требует бизнес-логика приложения. Примеры таких классов:

1. Класс UserModel. Данный класс содержит такие же поля, как и класс User, но имеет дополнительное поле: private Long countSubscribers.

2. **Класс VideoModel.** Данный класс содержит такие же поля, как и класс Video, но имеет дополнительные поля: private Long countViewing, private Long countLikes, private Long countDislikes.

Примеры классов-сервисов:

1. **Класс UserService.** Данный класс содержит поля: private final UserRepository userRepository, private final ViewedVideoRepository viewedVideoRepository, private final PlaylistWithVideosRepository playlistWithVideosRepository, private final UserMapper userMapper, private final PasswordEncoder passwordEncoder, private final MediaService mediaService, private final Logger logger.

Класс содержит методы:

– public UserModel insert(UserModel userModel, MultipartFile imageHeader, MultipartFile imageIcon) — метод для сохранения нового пользователя, а также его файлов;

– public UserModel update(UserModel userModel, MultipartFile imageHeader, MultipartFile imageIcon) — метод для изменения данных о пользователе, а также его файлов;

– public UserModel addSubscription(UserModel userModel) — метод для добавления подписки пользователя на другой канал;

– public UserModel deleteSubscription(UserModel userModel) — метод для удаления подписки пользователя на другой канал;

– public void delete(Long id) — метод для удаления пользователя по его id;

– public UserModel findUserById(Long id) — метод для поиска пользователя по его id.

2. **Класс MediaService.** Данный класс содержит методы, которые производят загрузку видео и картинок. Класс имеет поля: private String mediaRoot — корневой путь, по которому сохраняются все видео и картинки, private final Logger logger.

Методы класса:

– public void saveMedia(MultipartFile file, String filePath) — сохранение файла из параметра file. Путь сохранения находится в параметре filePath;

– public Resource getMedia(String filePath) — получение файла по пути filePath;

– public void deleteMedia(String filePath) — удаление файла по пути filePath;

– public Long getDuration(String videoPath) — получение продолжительности видео в секундах. Путь к видео находится в параметре videoPath.

4.3. Слой контроллера

Контроллер отвечает за прием и обработку HTTP-запросов от клиентов, а также возврат ответа клиенту, помещая в тело ответа необходимые данные. Классы данного слоя используют объекты классов слоя сервиса. Классы-контроллеры определяют конечные точки приложения и связывают запросы с соответствующими методами сервиса для обработки.

Слой контроллера использует классы (DTO), которые несут информацию о данных запросов (request) и ответов (response). Примеры таких классов:

1. **Класс UserResponse.** Данный класс содержит информацию о пользователе, которую необходимо отправить на сторону клиента. Класс имеет поля: private Long idUser, private String email, private String channelName, private String description, private Timestamp dateOfRegistration, private Long countSubscribers.

2. **Класс CreateVideoRequest.** Данный класс содержит информацию, которая нужна для создания видео и соответствующей записи в базе данных. Класс имеет поля: private Long idUser, private String name, private String description, private List<String> categories.

Примеры классов-контроллеров:

1. **Класс UserController.** Данный класс имеет адрес "/user" и содержит поля: private final UserService userService, private final UserMapper userMapper, private final VideoService videoService, private final VideoMapper videoMapper, private final ViewedVideoMapper viewedVideoMapper, private final PlaylistMapper playlistMapper.

Некоторые методы класса:

– public ResponseEntity<UserResponse> putUser(@Valid @RequestPart(value = "request", required = true) String request, @RequestPart(value = "icon", required = true) MultipartFile icon, @ RequestPart(value = "header", required = true) MultipartFile header, Authentication authentication) — HTTP-метод PUT. Данный метод принимает запрос на изменение данных о пользователе. В теле ответа метод возвращает информацию о пользователе, с учетом изменений.

– public ResponseEntity<UserResponse> postSubscription (@Valid @RequestBody UpdateSubscriptionsRequest request, Authentication authentication) — HTTP-метод POST, эндпоинт "/subscription". Данный метод принимает запрос на добавление подписки пользователя. В теле ответа метод возвращает информацию о пользователе, который совершил подписку.

2. **Класс VideoController.** Данный класс имеет адрес "/videos", а также содержит поля: private final VideoService videoService, private final VideoMapper videoMapper, private final AssessmentVideoMapper assessmentVideoMapper, private final ViewedVideoMapper viewedVideoMapper, private final CommentService commentService, private final CommentMapper commentMapper.

Некоторые методы класса:

– public ResponseEntity<VideoResponse> putVideo(@Valid @RequestPart(value = "request", required = true) String request, @RequestPart(value = "preview", required = true) MultipartFile preview) — HTTP-метод PUT, эндпоинт "/updateVideo". Данный метод принимает запрос на изменение данных о видео. Метод возвращает данные о видео, с учетом измененной информации.

– public ResponseEntity<List<CommentResponse>> getCommentsOnTheVideo (@PathVariable Long idVideo) — HTTP-метод GET, эндпоинт «"/{idVideo}/comments". Данный метод принимает запрос на получение информации о комментариях, оставленных к конкретному видео. В теле ответа метод возвращает список необходимых комментариев.

Заключение

В результате работы была разработана функциональность приложения видеохостинга, также проанализированы технологии для backend-разработки, существующие в языке программирования Java, спроектирована модель данных для видеохостинга, реализованы функциональные слои серверной части веб-приложения.

Литература

1. Servlet-technology // oracle.com : [сайт]. – 2024. – URL: https://www.oracle.com/java/technologies/servlet-technology.html (дата обращения: 22.11.2024)

2. Spring Framework Overview // docs.spring.io : [сайт]. – 2024. – URL: https://docs.spring.io/spring-framework/reference/overview.html (дата обращения: 22.11.2024)

3. Introduction to the Spring IoC Container and Beans // docs.spring.io : [сайт]. – 2024. – URL: https://docs.spring.io/spring-framework/reference/core/beans/introduction.html (дата обращения: 22.11.2024)

УДК 004.67:004.932.2

ПРЕОБРАЗОВАНИЕ ДАННЫХ СПЕЦИАЛЬНОГО ВИДА, ПОЛУЧЕННЫХ ИЗ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ ОПАСНЫХ ПРИРОДНЫХ ЯВЛЕНИЙ

Б. А. Крынецкий[1], А. В. Калач[1,2], А. А. Парамонов[1]

[1]РТУ МИРЭА
[2]Воронежский институт ФСИН России

Аннотация. При обработке графических данных исследователь сталкиваться с проблемами, вытекающими из представления данных в виде строго структурированной матрицы, столбцы и строки которой соответствуют координатам пикселя изображения, выраженными в том, что исследуемые объекты могут носить закономерности, не синхронизированные с прямоугольной системой координат. Для преобразования систем координат существуют типовые анаморфозы, однако для частных случаев необходимо применение обобщенного подхода, способного выполнять спрямления произвольных форм.
Ключевые слова: анализ изображений, обработка изображений, анализ лавин, анализ оползней, анализ циклонов.

Введение

При анализе изображений с применением почтипериодического анализа закономерно определение изображения как наборов рядов строк и столбцов системы координат, описывающей изображение. Так, было показано, что при исследовании структуры пространственно-временных характеристик в изображениях тропических циклонов имеет место применение декартовой и полярной систем координат [1–4]. Полученные результаты определяют фундаментальные структурные шаблоны, формирующие чрезвычайные явления циклонов.

Среди чрезвычайных явлений, обладающих фиксируемыми с помощью фото и видеосъемки процессами протекания и последствиями, можно выделить явления, связанные с движением масс разного рода — такие, как селевые потоки, лавины, оползни. Специфика этих явлений заключается в коротком времени жизни, а также в зависимости пространственной зависимости формы развития процесса от окружающей среды. Так, в случае прохождения лавины в горах массы могут сходить как по открытому склону, так и быть ограниченными рамками естественных неровностей структуры возвышенности — хребтами и расщелинами. Пример прогнозируемой области развития лавины из [5] приведён на рис. 1.

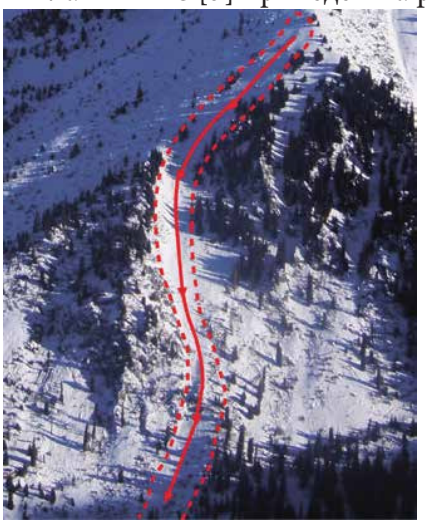


Рис. 1. Модельный лавинный очаг в долине реки Малая Алматинка

Описание алгоритма

Таким образом, очевидно, что применение почтипериодического анализа в декартовых координатах может содержать априорные потери информации, связанные со специальным видом структуры исследуемого объекта — возникает нужда в алгоритме, реализующем преобразование координат, выражающее характер данных и формы произвольной природы.

Преобразование данных предлагается проводить с применением представления исследуемого пространства в виде набора линейных полигонов, которые внутри будут разбиты на мелкую сетку. Выдвигаемое требование к формируемым внутри полигонов сеткам заключается в сохранении одной из размерностей — количества разбиений по ширине. Это условие необходимо для того, чтобы в спрямлении данных по сетке сохранялась возможность конкатенации сеток. Пример структуры полигонов и сеток, которая ожидается в результате произвольного построения, приведён на рис. 2.

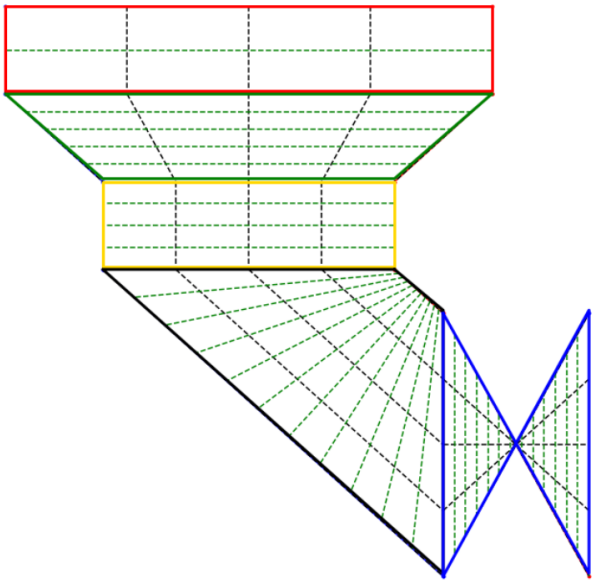


Рис. 2. Пример структуры полигонов с разметкой внутренних узлов интерполяции

Полигоны на изображении определяются последовательностью узлов, задаваемых координатами в системе изображения — шириной и высотой. Форма полигонов может быть любой плоской фигурой, обладающей 4 вершинами — так, на рис. 2 красный и жёлтый полигоны имеют прямоугольную форму, но количество поперечных делений в них отличаются. Зелёный и чёрный полигоны носят трапециевидную форму, однако основаниями одной трапеции являются стороны полигона, сонаправленные с последовательностью, а другого — направлены поперёк. Синий полигон демонстрирует, что в общем случае не исключено и самостоятельных пересечений структуры.

После разметки полигонов на плоскости изображения определяется общая величина — количество разделений полигонов в направлении вдоль последовательности — пунктирные чёрные линии на рис. 2. Количество поперечных делений может быть определено уникально для каждого полигона, и определяться либо произвольно, либо пропорционально длине полигона, либо из соображений эксперта — в целях более подробного исследования конкретных сегментов изображения.

Для дальнейшего описания предполагается обработка изображения в его представлении в виде матрицы значений яркости чёрно/белой гаммы — обобщение до представления в тензорном виде с учётом RGB составляющих усложняет вычисления кратно размерностям анализа данных. Таким образом, алгоритм преобразования и его математическое сопровождение

требует дополнительных разработок в области применения целевого метода и приводится в следующем виде.

Для матрицы $\boldsymbol{A}$ размерности $m \times n$ определить набор узлов полигонов — тензор

$$P_{i,j,r},\ i \in [1, \dim(K)],\ j \in [1..2],\ r \in [1..\dim(A)],$$

где i — номер полигона,

$\dim(K)$ — количество полигонов,

j — указатель на сторону полигона (левая/правая),

r — номер координаты пространства матрицы A (в рассматриваемом случае принимает значения от 1 до 2)

Так же из соображений эксперта определяется вектор K — содержащий количество индивидуальных разделений полигонов и общее значение продольных разделений полигонов L. Тогда после преобразований область изображения, описанная структурой тензора P, представляется в виде матрицы $\boldsymbol{B}$ размерности

$$L \times \sum_{i=1}^{\dim(K)} K_i.$$

Далее матрица B заполняется с применением метода билинейной интерполяции по узлам сетки, организованной полигонами. Координаты точки в пространстве изображения в зависимости от номера строки и столбца матрицы $\boldsymbol{B}$ можно определить по алгоритму, приведённому на блок-схеме на рис. 3.

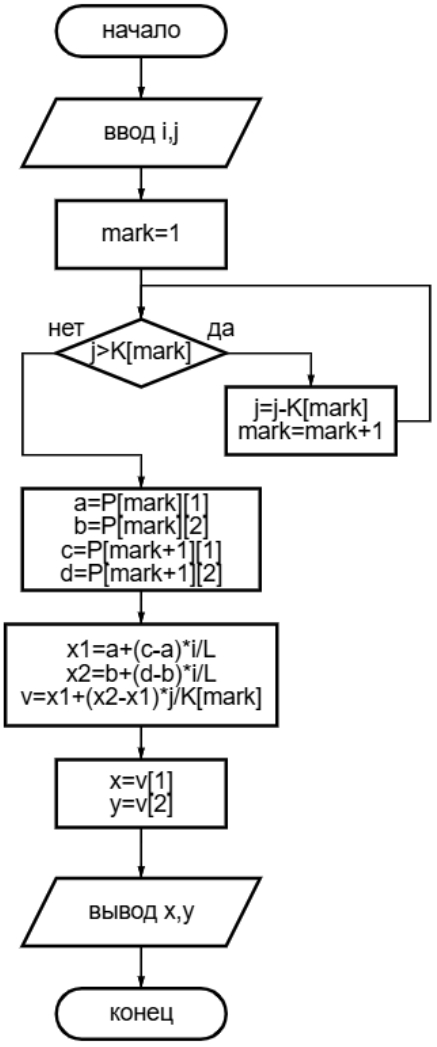


Рис. 3. Схема алгоритма расчёта координат

Таким образом, в результате преобразования для каждого элемента матрицы $\boldsymbol{B}$ рассчитывается интерполированное значение яркости изображения в точке с координатами, рассчитываемыми по алгоритму, приведённому на рис. 3.

В качестве демонстрации преобразования предлагается обработка фотографии последствия оползня, приведенное на рис. 4.

Определим последовательность полигонов, перекрывающих исследуемую область. На рис. 5 приводится выделение последовательности полигонов.

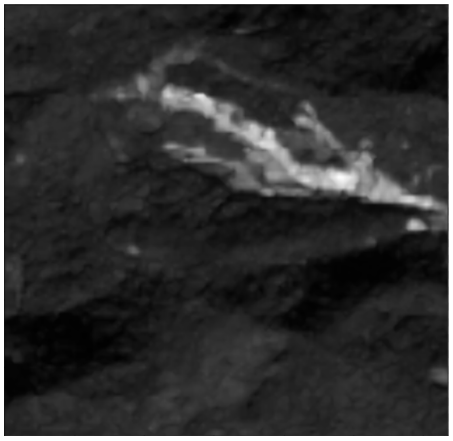


Рис. 4. Исходные данные. Последствия схода оползня (светлая область)

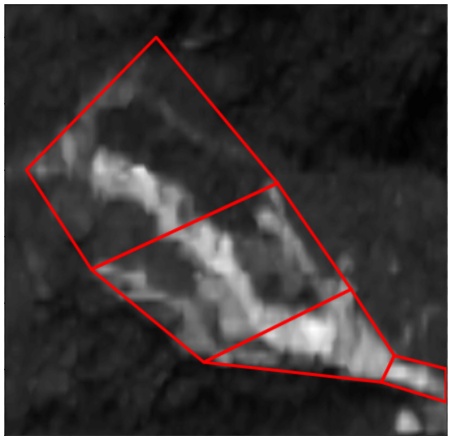


Рис. 5. Выделение анализируемой области

После приведения данных к спрямлённым координатам системы полигонов получается матрица значений интерполированной яркости. Её графическое представление приводится на рис. 6 — пунктирная красная линия разделяет соответствующие полигоны:

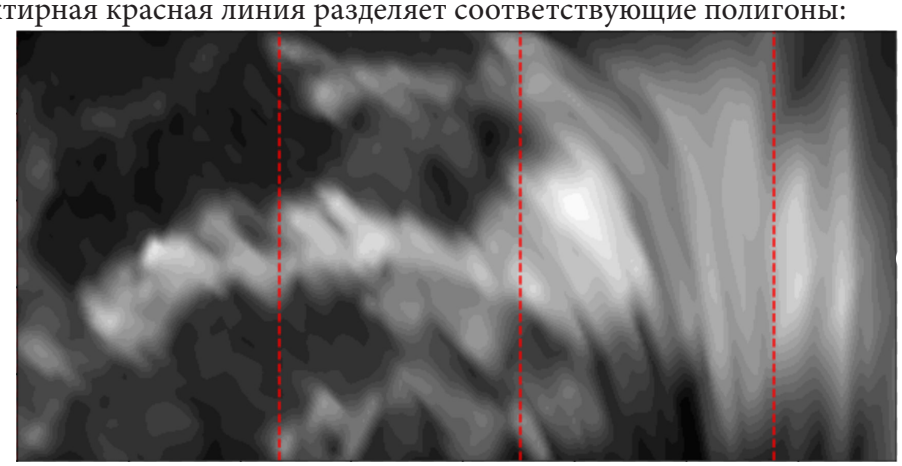


Рис. 6. Результат преобразования данных

Таким образом, удалось перейти к новым спрямлённым координатам. Такой набор данных становится пригодным для дальнейшего анализа за счёт лучшей синхронизации наглядных направлений развития явления с новой системой координат. Искажения на рис. 6 в правой части можно связать с существенной разностью в ширине полигонов в структуре, исследование их влияния на результаты дальнейшего анализа является предметом дальнейших научных работ.

Заключение

В статье приводится алгоритм, предлагаемый к использованию в решении задач анализа графических данных в геоинформатике. Продемонстрированы результаты работы алгоритма,

реализованного на языке программирования Python. Определены области дальнейших исследований.

Литература

1. *Парамонов А. А.* Практическая значимость почти-периодического анализа / А. А. Парамонов, Б. А. Крынецкий, Т. Е. Смоленцева // Труды международного симпозиума «Надежность и качество». – 2024. – Т. 1. – С. 227–230. – EDN ACEALJ.

2. *Калач А. В.* Применение почти-периодического анализа изображений динамики активности циклонов в целях выработки управленческих решений по повышению устойчивости территорий в чрезвычайных ситуациях / А. В. Калач, А. А. Парамонов // Теория и практика повышения устойчивости урбанизированных территорий в чрезвычайных ситуациях : Сборник материалов международного круглого стола (в рамках проведения XV Международного салона средств обеспечения безопасности «Комплексная безопасность-2024»), Конгрессно-выставочный центр «Патриот», г. Кубинка, Московская обл., 30 мая 2024 года. – Москва : Всероссийский научно-исследовательский институт по проблемам гражданской обороны и чрезвычайных ситуаций МЧС России, 2024. – С. 106–112. – EDN UUIGZQ.

3. *Малыгин И. Г.* Почти-периодический анализ накопленной энергии тайфунов при исследовании развития чрезвычайных ситуаций, обусловленных активностью тропических циклонов / И. Г. Малыгин, А. В. Калач, А. А. Парамонов // Проблемы управления рисками в техносфере. – 2024. – № 3(71). – С. 55-62. – DOI 10.61260/1998-8990-2024-3-55-62. – EDN IBPLCG.

4. *Крынецкий Б. А.* Анализ моделей периодических структур пространственно-временных процессов / Б. А. Крынецкий // Актуальные проблемы прикладной математики, информатики и механики : сборник трудов Международной научной конференции, Воронеж, 04–06 декабря 2023 года. – Воронеж : Общество с ограниченной ответственностью «Вэлборн», Издательство «Научно-исследовательские публикации», 2024. – С. 497–501. – EDN VBSJNL.

5. *Благовещенский В. П.* Калибровка математических моделей лавин по данным о реальных лавинах в Иле (Заилийском) Алатау / В. П. Благовещенский, М. Э. Эглит, В. В. Жданов, Б. Б. Аскарбеков // Лёд и снег. – 2017. – Т. 57, № 2. – С. 213–220. – DOI 10.15356/2076-6734-2017-2-213-220. – EDN YPMGGL.

УДК 539.3:519.6

РАЗРАБОТКА АЛГОРИТМА И ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ МОДЕЛИРОВАНИЯ ОДНОМЕРНОЙ ДИНАМИКИ ДЕФОРМИРОВАНИЯ РАЗНОМОДУЛЬНЫХ СРЕД

А. А. Лаптева, О. В. Дудко, В. Е. Рагозина

Институт автоматики и процессов управления ДВО РАН

Аннотация. Разработан программный комплекс, предназначенный для решения краевых задач ударного деформирования разномодульной полуограниченной среды при различных режимах нестационарного одноосного нагружения. В основу вычислительного модуля комплекса заложен оригинальный алгоритм, использующий кусочно-линейную аппроксимацию функции граничного перемещения, изначально заданной в гладкой форме. Алгоритм позволяет получить обобщенное решение нестационарной краевой задачи деформирования разномодульной среды с учетом всех эффектов столкновения и отражения разнотипных фронтов сильных разрывов, не обращаясь при этом к приближенным методам малого параметра.

Ключевые слова: математическое моделирование, программный комплекс, упругость, полуограниченная разномодульная среда, нестационарное одноосное нагружение, квазилинейное уравнение движения, нелинейное краевое условие, кусочно-линейная аппроксимация, сильный разрыв, ударная волна, столкновение и отражение волн.

Введение

Материалы с переменными упругими свойствами, зависящими от типа деформированного состояния, в механике сплошных сред называют разносопротивляющимися или разномодульными. Подобные материалы с сингулярностью механического поведения существуют как в естественной природной форме (грунты, горные породы, связные сыпучие среды, кость, дерево и др.), так и среди современных конструкционных металлических сплавов, неоднородных и волокнистых композитов и т. п. Учитывая практическую ценность и широкую распространенность разномодульных материалов во многих отраслях человеческой деятельности (от горнодобывающей промышленности и тяжелого машиностроения до медицины и микроэлектроники), изучение особенностей их механического поведения в различных условиях является актуальной задачей. Наиболее существенно разномодульность проявляется при динамическом деформировании таких сред, когда их механические параметры могут скачком измениться, например, при смене направления граничной нагрузки.

Проведение динамических экспериментов для исследования физико-механических свойств разномодульных материалов часто требует слишком больших финансовых и временных затрат. Традиционным решением данной проблемы является математическое и компьютерное моделирование, но даже в этом случае могут возникнуть существенные затруднения, связанные, в частности, с сингулярностью модельных соотношений. В [1, 2] положено начало разработке эффективного подхода к решению нестационарных краевых задач одномерной динамики разномодульных и пористых сред. Предложенный подход, основанный на одновременной кусочной линеаризации модельных соотношений и краевых условий, позволяет воспользоваться формализмом и известными решениями линейной теории упругости, однако его «ручное применение» по-прежнему остается достаточно трудоемкой задачей. Закономерным развитием такого подхода является его алгоритмизация и разработка вычислительного комплекса, автоматизирующего процесс формирования систем краевых условий на возникающих волновых фронтах и проведение вычислительных экспериментов при различных нестационарных режимах граничного нагружения.

1. Упрощающие положения модели

Для описания изотропной упругой разномодульной среды используем физически нелинейную модель [3], где модули упругости сингулярным образом зависят от типа деформаций. В одномерном случае малых деформаций модельные соотношения разномодульной упругой среды [3] максимально упрощаются до кусочно-линейной формы с особенностью в точке свободного состояния. С одной стороны, такое упрощение не мешает описывать различные нелинейные эффекты, характерные для динамики деформировании разномодульной среды [4]: «быстрые» волны сжатия и «медленные» волны растяжения, ударные волны и движущиеся жесткие слои. С другой стороны, линеаризация модельных соотношений может оказаться недостаточной для отказа от приближенных методов (типа метода малого параметра) при учете столкновений и отражений разнотипных волновых фронтов, которые движутся в разномодульной среде с разными скоростями. В задачах нестационарного одноосного деформирования эту проблему можно устранить дополнительным упрощением — заменой гладкого нестационарного краевого условия на его кусочно-линейную аппроксимацию [1]. В этом случае нелинейное решение исходной динамической задачи аппроксимируется связной последовательностью локальных линейных решений, каждое из которых определено в своих интервалах пространства и времени.

2. Постановка нестационарной задачи с кусочно-линейным краевым условием и алгоритм ее решения

Рассмотрим разномодульную полуограниченную среду (полупространство или стержень), изначально недеформированную. Положим, что с нулевого момента времени на границу среды начинает действовать нестационарная нормальная нагрузка, при этом точки полупространства (до этого момента неподвижные) начинают двигаться с ненулевой скоростью. Необходимый режим нестационарного нагружения задаем функцией граничного перемещения, зависящей от времени. Такая функция может быть изначально кусочно-линейной или гладкой. Во втором случае предварительно (до решения задачи) строим ее кусочно-линейное приближение, располагая основные узловые точки в «критических» моментах времени изменения направления и интенсивности граничной нагрузки. Дополнительные точки разбиения можно добавлять из соображений повышения точности аппроксимации.

При описанном режиме нагружения решение краевой задачи принимает обобщенную форму, когда искомая функция перемещений оказывается почти всюду гладкой, дважды непрерывно дифференцируемой и удовлетворяющей дифференциальным законам сохранения за исключением конечного числа фронтов сильных разрывов, где выполняются интегральные законы сохранения [5]. Разнотипные фронты сильных разрывов делят всю область деформирования на локальные зоны постоянства деформаций, в каждой из которых разномодульная среда с кусочно-линейными определяющими соотношениями ведет себя линейным образом. Таким образом, обобщенное решение нашей задачи есть объединение цепочки локальных линейных решений, которые могут появляться как при изменении режима движения граничных точек, так и в результате попутных и встречных столкновений волновых фронтов, движущихся в разномодульной среде с различными скоростями. Узловые точки граничного перемещения и моменты времени столкновения волновых фронтов между собой и с границей среды в процессе решения задачи включаются в множество точек, разбивающих непрерывный полуоткрытый интервал времени на отдельные этапы, где начало каждого этапа соответствует появлению нового локального решения. Для «склеивания» локальных решений на фронтах сильных разрывов используем условия Гюгонио [5].

Теперь опишем алгоритм построения последовательности локальных решений подробнее.

1) Начиная с нулевого момента времени, поэтапно строим локальные решения первичной волновой картины. Каждое скачкообразное изменение скорости граничного перемещения порождает первичный волновой фронт, уходящий от границы среды. Тип каждого первичного сильного разрыва (медленная волна растяжения, быстрая волна сжатия или ударная волна) определяем по уже известной предварительной деформации перед ним и неизвестному состоянию за ним, которое определяется по актуальной части граничного перемещения. Действуем таким образом до тех пор, пока в решении не появится хотя бы одна пара сильных разрывов, состоящая из убегающей медленной волны растяжения и догоняющей быстрой волны сжатия. Именно с этого момента начинают действовать эффекты взаимодействия и отражения волновых фронтов.

2) Анализируем актуальную волновую картину и находим все пары подходящих для столкновения фронтов (таких пар может оказаться больше одной). Если в актуальной волновой картине нет подходящих для столкновения волн, то процесс решения останавливаем.

3) Приравнивая координаты пары взаимодействующих фронтов, вычисляем время и координату столкновения для каждого возможного инцидента, обнаруженного на шаге 2. Затем из всех возможных инцидентов выбираем единственный с наименьшим значением времени столкновения. Прогнозируем его результат (т.е. типы расходящихся сильных разрывов и деформированное состояние в новой растущей зоне между ними). При прогнозе учитываем свойства взаимодействующих волн, диктуемые условиями Гюгонио, а также известные решения слева и справа от координаты столкновения.

4) Находим новое локальное решение, возникающее в результате выбранного инцидента. Неизвестные параметры этого решения вычисляем из системы условий Гюгонио, записанных на подвижных границах его интервала определения. Затем проверяем эволюционность этих границ [6]. При нарушении эволюционности какой-либо из границ корректируем прогноз, заменяя неэволюционный сильный разрыв на жесткий слой, ограниченный быстрым передним и медленным задним фронтами (можно показать, что такой слой является бездиссипативным процессом, удовлетворяющим требованию неубывания энтропии). Если обе границы интервала определения нового локального решения эволюционны, то возвращаемся к шагу 2 и выбираем следующий инцидент в обновленной волновой картине.

3. Описание программного комплекса

Предложенный алгоритм решения одномерных краевых задач динамики деформирования разномодульной упругой среды включен в вычислительный модуль программного комплекса [7]. В качестве средства реализации программного комплекса выбрана свободная система компьютерной алгебры Maxima, позволяющая не только выполнять численные расчеты, но и автоматизировать выбор систем уравнений для каждого локального решения, а также сохранять результаты вычислений, в том числе и в графической форме.

Программный комплекс состоит из трех основных модулей:

1) модуль обработки исходных данных;

2) модуль обмена данными;

3) вычислительное ядро (модуль – «решатель»).

В модуле обработки исходных данных пользователю предоставляется возможность задавать механические параметры нагружаемой среды и координаты узлов кусочно-линейного приближения функции перемещения граничных точек. Результатом работы модуля является график аппроксимации функции граничного перемещения (рис. 1). Введенные исходные данные и координаты узловых точек функции граничного перемещения сохраняются во внешнем файле, который передается в модуль обмена данными.

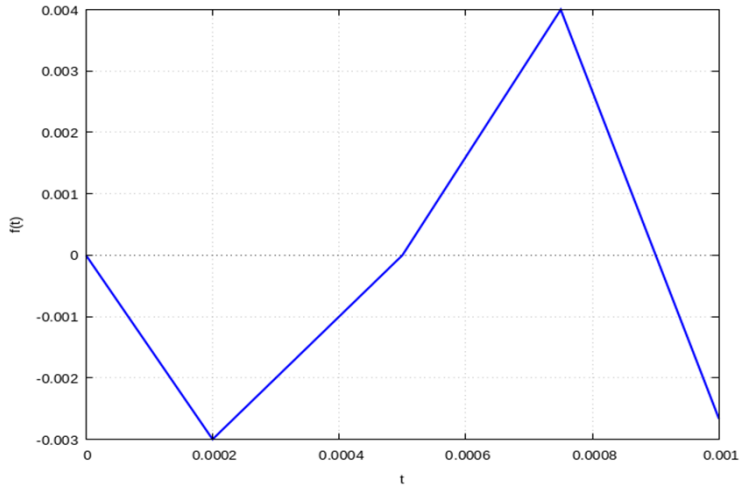
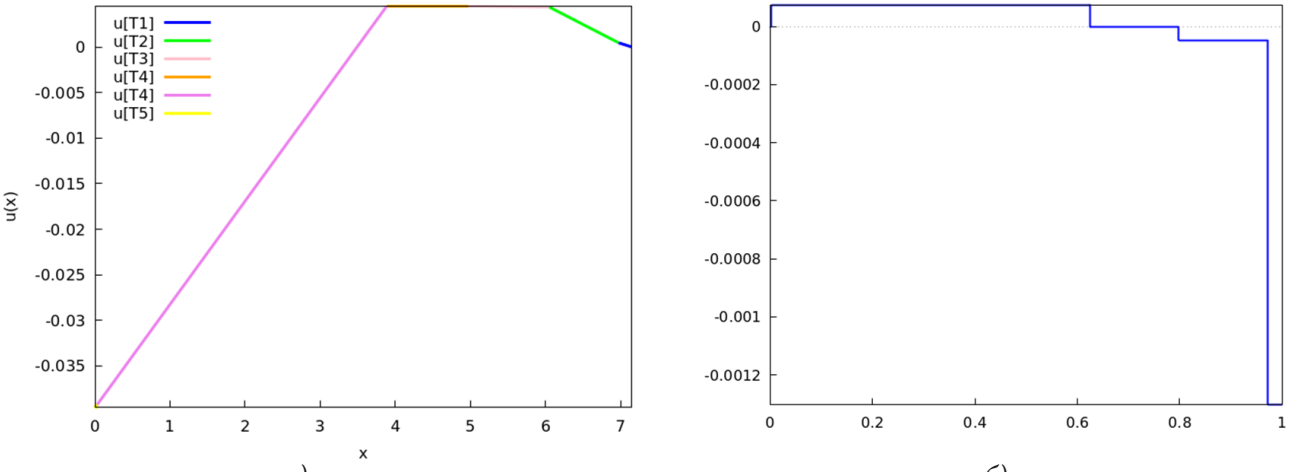


Рис. 1. Пример работы модуля обработки исходных данных

Модуль обмена данными управляет поэтапной передачей необходимой информации «решателю», а также принимает от него готовые решения в численной и графической форме. На каждом такте работы модуль обработки данных передает «решателю» только ту часть информации, которая необходима для построения очередного локального решения. Обратный поток данных от «решателя» к модулю обмена данными состоит из вычисленных значений деформаций, параметров функции перемещений очередного локального решения, а также пакета форматированных данных для построения мгновенных диаграмм полей перемещений (рис. 2а) и деформаций (рис. 2б). Модуль обмена данными накапливает результаты работы «решателя», на основе которых поэтапно строит и возвращает пользователю графическую форму обобщенного решения задачи — изображение характеристической плоскости с диаграммами сильных разрывов и областей различных деформированных состояний (рис. 3).



а) *б)*

Рис. 2. Примеры работы модуля обмена данными: мгновенные диаграммы полей перемещений и деформаций

Вычислительное ядро (модуль — «решатель») работает автономно от пользователя и содержит набор функций, необходимых для автоматизации решения задачи:

– вычисление скоростей ударных волн;

– вычисление времени и координаты столкновения волновых фронтов;

– вычисление скоростей волн, возникающих в результате столкновений, определение их количества и типов (включая проверку эволюционности и корректировку волновой картины по необходимости);

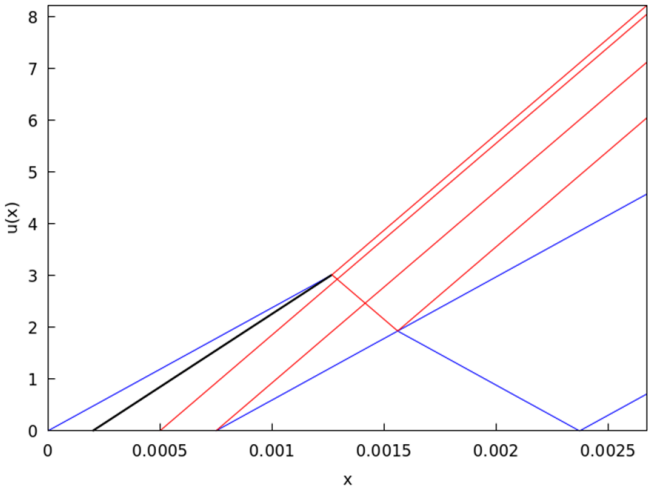


Рис. 3. Пример работы модуля обмена данными: характеристическая плоскость с диаграммами сильных разрывов

– вычисление параметров функции перемещений, вычисление деформаций в определенные моменты времени;

– сохранение результатов работы в численном виде;

– формирование пакета форматированных данных и передача их в модуль обмена данными.

Все три модуля являются самостоятельными программными единицами, которые при необходимости можно модифицировать независимо друг от друга.

Преимуществом использования СКА Maxima для реализации описанного алгоритма является ее способность кешировать результаты символьных вычислений и за счет этого существенно сокращать время получения обобщенных решений выбранного класса краевых задач.

Заключение

Итак, результатом нашей работы является программный комплекс для решения краевых задач нестационарного одноосного деформирования полуограниченной разномодульной среды при различных режимах граничного нагружения. В настоящий момент реализованы случаи с кусочно-линейными функциями граничного перемещения, соответствующими нестационарной нагрузке с растяжением на первом этапе: «растяжение-сжатие» [1], циклическое знакопеременное нагружение «растяжение-сжатие-растяжение» [8], принудительное удержание границы после импульса растяжения-сжатия [9]. Перечень режимов нагружения может быть легко дополнен путем минимальной модификации вычислительного модуля.

Разработанное программное обеспечение автоматизирует проведение масштабных вычислительных экспериментов и позволяет пользователю отказаться от трудоемких аналитических вычислений, тем самым сокращая время получения качественных результатов в исследованиях особенностей динамики одномерного деформирования разномодульных упругих сред.

Литература

1. *Дудко О. В.* Нестационарные одномерные динамические задачи разномодульной упругости с кусочно-линейной аппроксимацией краевых условий / О. В. Дудко, А. А. Лаптева, В. Е. Рагозина // Вестник Пермского национального исследовательского политехнического университета. Механика. – 2019. – № 4. – С. 37–47.

2. *Дудко О. В.* Эволюция волновой картины кусочно-линейного растяжения и сжатия разномодульного упругого стержня / О. В. Дудко, А. А. Лаптева, В. Е. Рагозина // Сибирский журнал индустриальной математики. – 2022. – Т. 25, № 4. – С. 54–70. DOI: 10.33048/SIBJIM.2022.25.405

3. *Ляховский В. А.* О поведении упругой среды с микронарушениями / В. А. Ляховский, В. П. Мясников // Известия АН СССР. Физика Земли. – 1984. – № 10. – С. 71–75.

4. *Дудко О. В.* О распространении плоских одномерных волн и их взаимодействии с преградами в среде, по-разному сопротивляющейся растяжению и сжатию / О. В. Дудко, А. А. Лаптева, К. Т. Семенов // Дальневост. матем. журн. – 2005. – Т. 6, No 1-2. – С. 94–105.

5. *Маслов В. П.* Общая теория уравнений движения разномодульной среды / В. П. Маслов, П. П. Мосолов // Прикладная математика и механика. – 1985. – Т. 49, вып. 3. – С. 419–437.

6. *Куликовский А. Г.* Нелинейные волны в упругих средах / А. Г. Куликовский, Е. И. Свешникова. – Москва : Московский Лицей, 1998. – 412 с.

7. Bimodular Dynamics — вычислительный комплекс для моделирования динамики одномерного деформирования разномодульных сред: свидетельство о государственной регистрации программ для ЭВМ № 2023667828 Российская Федерация / А. А. Лаптева, О. В. Дудко; правообладатель ИАПУ ДВО РАН. – № 2023666274/69; заявл. 02.08.23; опубл. 18.08.23, Бюл. № 8. – 1 с.

8. *Dudko O. V.* Solving nonstationary boundary value problem with piecewise linear boundary condition for one-dimensional dynamics of deformable heteromodular elastic solid / O. V. Dudko, A. A. Lapteva, V. E. Ragozina // AIP Conference Proceedings. – 2021. – V. 2448, Iss. 1, Paper 020003. – P. 1–5.

9. *Лаптева А. А.* Нестационарное одномерное деформирование разномодульного упругого полупространства в циклическом режиме / А. А. Лаптева, О. В. Дудко, В. Е. Рагозина // Актуальные проблемы прикладной математики, информатики и механики : сборник трудов Международной научной конференции, Воронеж, 12-14 декабря 2022 г. – Воронеж, 2023. – С. 1066–1070.

УДК 004.4’233

РЕАЛИЗАЦИЯ НОВОГО ПОДХОДА ИНСТРУМЕНТИРОВАНИЯ КОДА, ОСНОВАННОГО НА ГРАФЕ ЗАВИСИМОСТЕЙ ДАННЫХ

М. В. Любопытнов, М. С. Ефремов

Воронежский государственный университет

Аннотация. В данной статье рассматривается реализация нового метода инструментирования кода основанного на использовании графа зависимостей данных для программного комплекса (ПК) *AFL-VIZIR* с целью увеличения процента покрытия ветвей и повышении эффективности обнаружения ошибок.
Ключевые слова: инструментирование, фазз-тестирование, динамический анализ, покрытие кода, покрытие ветвей, граф зависимостей, граф потока управления, битовая карта, базовый блок, LLVM.

Введение

Инструментирование — это процесс модификации исходного или скомпилированного кода приложения с целью добавления специальной логики, которая позволяет наблюдать за его поведением во время выполнения. Инструментирование используется при фазз-тестировании [1].

Фазз-тестирование — это техника динамического анализа, которая заключается в подаче на вход программы различных вариаций входных данных с целью выявления и обнаружения потенциальных проблем, обеспечивая максимальное покрытие кода [2].

Покрытие кода — это метрика, измеряющая, какие части программного кода были выполнены или протестированы. Это важное понятие, поскольку позволяет оценить степень тестирования программы и указать на участки кода, которые требуют дополнительного внимания [3].

Для измерения покрытия кода в фаззере *AFL-VIZIR* используется покрытие ветвей.

Покрытие ветвей — это метрика, используемая для оценки того, насколько хорошо тесты, либо тестовые данные покрывают все возможные ветви (условия перехода) в программном коде [4].

С целью повышении эффективности обнаружения ошибок в коде различных программ было принято решение использовать граф зависимостей данных совместно с уже существующим решением ПК *AFL-VIZIR*, а именно — графа потока управления, по той причине, что граф зависимостей позволяет получать дополнительную обратную связь для фаззера, в отличие от графа потока управления.

Граф зависимостей — это структура данных, которая моделирует зависимости между различными операциями или элементами данных в программе. Граф зависимостей полезен для анализа зависимостей данных в программном коде, что позволяет понять, какие данные используются или модифицируются в различных частях программы [5].

В данной работе представлено описание основных шагов реализации нового подхода инструментирования кода.

1. Реализация алгоритма

1.1. Построение графа зависимостей

Основная идея состоит в том, чтобы анализировать только часть переменных *LLVM*, учитывая их определение и использование в байт-коде, и восстанавливать зависимости данных,

относящиеся исключительно к этой части. Это требует определения, какие инструкции будут считаться определениями (*def*), а какие — использованиями (*use*).

На уровне машинного кода, передача данных между различными частями системы происходит только тогда, когда память читается или записывается. На уровне промежуточного представления *LLVM IR* это привело к использованию инструкций *load* в качестве возможного источника и *store* в качестве возможного приёмника потоков данных, инструкции вызова функций *call*, связанной с отслеживанием зависимостей параметров вызова функции и инструкции *alloca* в качестве источника связи между определением (*def*) и использованием (*use*). Хотя оптимизационные проходы компилятора удаляют большинство переменных, определённых через *alloca*, отслеживание зависимостей этих переменных может быть полезным, когда они остаются в сгенерированном байт-коде.

На рис. 1 отображен итоговый алгоритм.

```
1  function BuildDDG(module)
2      blocks ← GetBasicBlocks(module)
3      DDG ← {}, ∀ b ∈ blocks
4      DFT ← {}
5      for b ∈ blocks do
6          instructions ← GetInstructions(b)
7          for i ∈ instructions do
8              if IsDefinition(i) then
9                  val ← GetValue(i)
10                 DFT ← {}
11             if IsGeneric(i) then
12                 val ← GetValue(i)
13                 ops ← GetOperands(i)
14                 InsertDataFlow(DFT, val, ops)
15             for u ∈ GetUses(i) do
16                 def ← QueryDataFlow(DFT, u)
17                 src ← GetParentBlock(def)
18                 DDG ← AddToSet(src)
```

Рис. 1. Итоговый алгоритм графа зависимостей данных

Для построения графа используются две основные структуры:

• *DDG* — граф зависимостей данных;

• *DFT* — структура отслеживания потока данных, которая представляет собой карту множеств, где ключом является *LLVM*-значение, и каждый ключ получает набор *LLVM*-значений, зависящих от этого ключа.

Описание алгоритма:

1. Получить базовый блок, а также инициализировать структуры графа зависимостей данных (*DDG*) и структуры отслеживания потока данных (*DFT*).

2. Перебрать все полученные базовые блоки во внешнем цикле.

3. Извлечь инструкции для каждого полученного базового блока.

4. Перебрать все полученные инструкции, полученные из базового блока, и выполнить ряд проверок:

– если встретилась определяющая инструкция (*def*), например, *alloca* и *load*, то добавить записи в *DFT*;

– если встретилась инструкция, которая не является ни источником, ни приемником, то извлечь операнды и возвращаемое значения инструкции, а также вызвать процедуру *InsertDataFlow*, принимающую на вход *DFT*, значение инструкции и операнд, и отвечает за вставку зависимости данных между операндом и результатом инструкции.

5. Перебрать в последнем вложенном цикле все инструкции использования *u*. Для каждого использования *u*, вызвать функцию *QueryDataFlow*, принимающую на вход *DFT* и инструкцию использования *u*. Эта функция вернет место в коде, где переменная *u* была определена и присвоит его *def*. Вызвать функцию *GetParentBlock* принимающую на вход определение *def* и вернуть родительский блок кода, в котором это определение находится, также сохранить результат в *src*. Последним шагом добавить родительский блок *src* в множество *DDG*, тем самым восстановить зависимости.

1.2. Фильтрация зависимостей

На этапе фильтрации происходит отбрасывание зависимостей, которые не предоставляют дополнительную обратную связь, поскольку связанные с ними переходы уже включены в покрытие ребер. Зависимости внутри одного и того же блока кода не учитываются, а также несколько зависимостей, соединяющие два базовых блока не считаются значимыми, так как в дальнейшем будут объединяться в один.

Именно поэтому граф зависимостей работает совместно с стандартным механизмом покрытия краев, что позволяет получать два разных сигнала:

1. Первый сигнал полезен для тестирования различных зависимостей.
2. Второй сигнал необходим для дальнейшего изучения кода приложения.

Однако это также означает о возможных пересечениях покрытий графа зависимостей и графа потока управления, что в свою очередь может привести к дублированию обратной связи.

С целью устранения избыточных зависимостей данных были внедрены два ключевых правила:

1. Исключать те потоки данных в графе потока управления, где существует зависимость между двумя связанными базовыми блоками, где один базовый блок является преемником второго базового по той причине, что в данном случае зависимость между базовыми блоками не добавит никакой дополнительной информации, которая не была бы уже захвачена покрытием ребер, а также увеличит накладные расходы.

2. Второе правило расширяет первое и включает другие случаи, когда зависимость от данных уже охвачена покрытием ребер.

На рис. 2. изображен алгоритм фильтрации зависимостей на примере определений *Def1* и *Def2*.

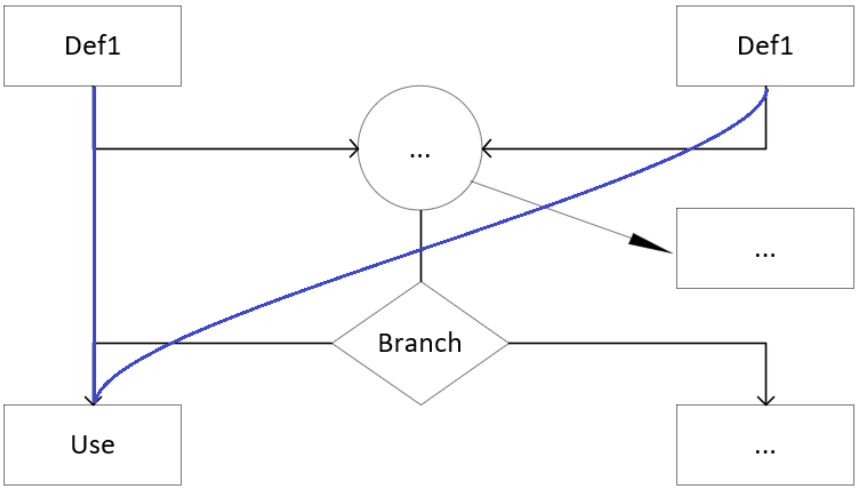


Рис. 2. Схема фильтрации зависимостей

На данном рисунке черные стрелки представляют ребра графа потока управления, а синие стрелки — ребра графа зависимостей данных. В этом случае фаззер может достичь 100-процентного покрытия ребер и при этом сработать только на одну из двух пар *def-use*.

В результате этот тип зависимости является единственным типом потока, который следует сохранять, с целью получения нового способа обратной связи для фаззера по сравнению со стандартным механизмом ПК *AFL-VIZIR*.

1.3. Инструментирование цели

Первый инструментальный слой сохраняется подход, который уже реализован и имеется в ПК *AFL-VIZIR*. Данный подход включает в себя регистрацию посещенных ребер, где вычисляется *XOR* между идентификаторами текущего и предыдущего местоположения. Полученное значения используется в качестве индекса для доступа к битовой карте, которая хранит количество посещений каждого конкретного ребра [6].

Битовая карта — структуру данных, которая состоит из массива байтов, где каждый байт представляет информацию о выполнении определенного участка кода, например, базового блока или ребра в графе и используется для отслеживания покрытия кода во время выполнения программы [7].

Однако в контексте графа зависимостей это решение не работает, так как два места, которые необходимо использовать в качестве входа для *XOR*, не являются последовательными, и выполнение может пройти через множество промежуточных базовых блоков, прежде чем достигнет того, который содержит использование, зависящее от данных.

Для решения данной проблемы было принято решение добавить дополнительную вспомогательную переменную, первоначально установленную в 0 для каждого базового блока, содержащего определение (*def*), которое необходимо отслеживать. В последствии, при достижении блока, значение маркера будет изменяться по средствам инструментирования на идентификатор этого блока.

В дальнейшем происходит инструментирование каждого базового блока, содержащего использование (*use*), создавая новый индекс битовой карты, полученный путем выполнения операции *XOR* над соответствующими переменными-маркерами и идентификатором конечного блока. Поскольку при выполнении операции *XOR* между *N* и 0 будет *N*, то результат будет учитывать только определение (*def*), которое было фактически выполнено, что приведет к разным значениям, когда один и тот же базовый блок достигается разными путями потока данных.

Пример алгоритма инструментирования представлен на рис. 3.

Описание алгоритма, представленного на рис. 3:

1. Создание вспомогательных переменных. Инициализировать вспомогательные переменные block_A и block_B значением ноль, и вставить для каждого блока определения (def), который необходимо отследить.

2. Установка вспомогательных переменных в блоках определения (def). Установить для идентификации блока соответствующую переменную (в данном примере это A и B соответственно) в базовых блоках, содержащих определения.

3. Инструментирование блока использования (use). Базовый блок, содержащий использование (use) переменной, инструментировать для вычисления хэша переменных, связанных с переменной val, и идентификатором текущего блока. В данном примере это C.

4. Использование хэша. Полученный хэш использовать в качестве индекса в битовой карте ПК AFL-VIZIR.

Заключение

В ходе работы был создан инструмент, направленный на повышение безопасности программного обеспечения. Этот инструмент учитывает зависимости данных и улучшает возможности тестирования, что позволяет выявлять уязвимости и ошибки на ранних этапах разработки.

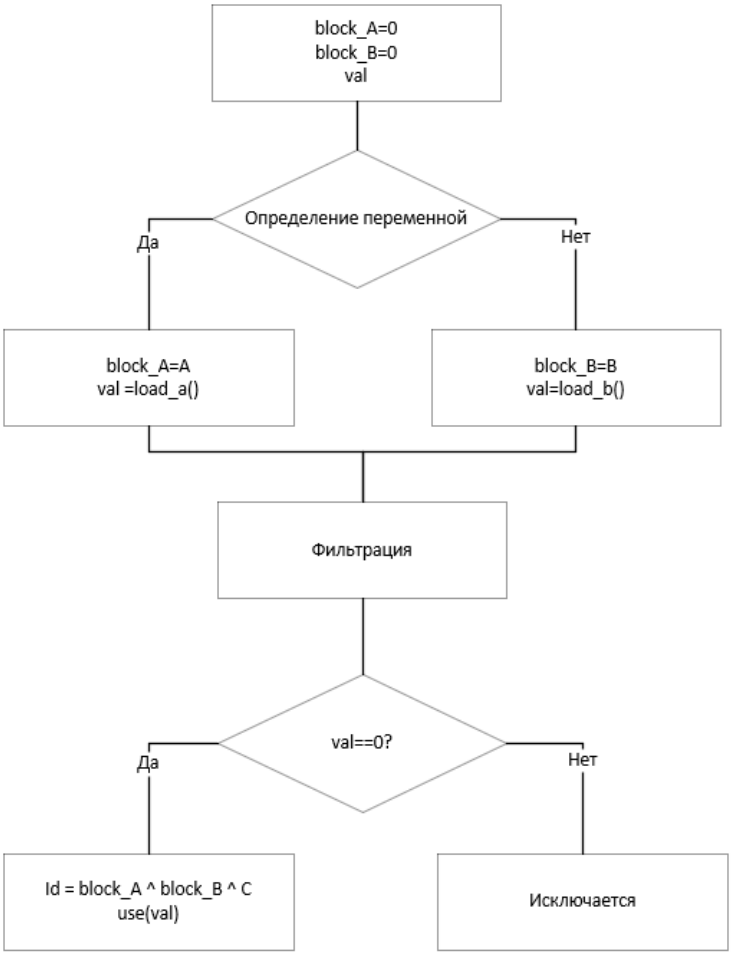


Рис. 3. Схема алгоритма инструментирования

Литература

1. Инструментирование кода // Википедия : [сайт]. – 2024. – URL: https://en.wikipedia.org/wiki/instrumentation (дата обращения: 22.11.2024)

2. *Саттон М., Грин А., Амини П.* Fuzzing: Исследование уязвимостей методом грубой силы, «Символ Плюс». – СПб. – Москва, 2009. – 560 с.

3. Покрытие кода // Википедия : [сайт]. – 2024. – URL: https://en.wikipedia.org/wiki/code_coverage (дата обращения: 22.11.2024)

4. Покрытие ветвей // TestEngineer : [сайт]. – 2024. – URL: https://testengineer.ru/code-coverage/ (дата обращения: 22.11.2024)

5. *Cytron R. K., Ferrante J., Rosen B. K., Wegman M. N., Zadeck F. K.* Efficiently computing static single assignment form and the control dependence graph // ACM Trans. on Programming Languages and Systems. – 1991. – Vol. 13, N 4. – P. 451–490.

6. Операция Xor // LoginRadius : [сайт]. – 2024. – URL: https://www.loginradius.com/blog/engineering/how-does-bitwise-xor-work/ (дата обращения: 24.11.2024)

7. Битовая карта // Википедия : [сайт]. – 2024. – URL: https://en.wikipedia.org/wiki/bitmap (дата обращения: 25.11.2024)

УДК 004.9

ВЕБ-СЕРВИС ДЛЯ ИНФОРМАЦИОННОГО МОНИТОРИНГА КРИМИНОГЕННОЙ ОБСТАНОВКИ

А. М. Мельников, И. Е. Воронина

Воронежский государственный университет

Аннотация. Рассматривается задача разработки веб-сервиса для мониторинга криминогенной обстановки и обеспечения взаимодействия между пользователями, государственными органами и правоохранительными структурами. Веб-сервис предоставляет доступ к актуальным новостям о преступности, информации о зонах повышенного риска, а также возможность обмениваться сообщениями в защищенных чатах, выполнять мониторинг статистических данных. Используются прогнозирующие алгоритмы на базе машинного обучения.

Ключевые слова: веб-сервис, безопасность, мониторинг, криминогенная обстановка, ролевой доступ, предсказание преступлений, чат, карта горячих точек, интеграция, тестирование, аналитика данных, бот, машинное обучение.

Введение

Цифровизация привнесла в повседневную жизнь новые механизмы взаимодействия в различных областях. В частности, задачи мониторинга криминогенной обстановки может стать доступным для использования различными группами населения. Однако среди пользователей таких систем остаются два ключевых запроса: защищенность информации и точность аналитики [1]. Важным шагом для построения открытой и безопасной среды могла бы стать разработка веб-сервиса, который мог бы объединить в себе как функциональность для информирования, так и интерактивные возможности взаимодействия. Кроме того, интеграция алгоритмов прогнозирования позволила бы не только получать сводку текущей криминогенной ситуации, но и заранее предсказывать потенциальные зоны риска, помогая в принятии превентивных мер.

Предлагаемый веб-сервис для мониторинга безопасности ориентирован на ряд задач, решающих как индивидуальные, так и общественные потребности. Во-первых, это систематизированный доступ к актуальным новостям и криминальной статистике, что повысит уровень информированности граждан. Во-вторых, пользователи будут получать доступ к статистическим данным в режиме реального времени через интерактивную карту горячих точек, позволяющую оценить ситуацию в конкретных регионах. В-третьих, система может предоставить защищенные каналы для обмена сообщениями, что особенно важно для органов правопорядка. Ролевая модель системы, в рамках которой каждому пользователю присваивается определенный уровень доступа, позволит надежно защитить конфиденциальные данные и оптимизировать взаимодействие между участниками сервиса [2].

Эффективность работы сервиса можно обеспечить за счет внедрения алгоритмов машинного обучения, анализирующих данные и формирующих прогнозы на основе большого массива информации. Это позволит не только ретроспективно оценивать ситуацию, но и предсказывать её развитие. Подобная технология полезна для служб, занимающихся вопросами безопасности, так как предсказание зон потенциального риска позволяет быстрее реагировать на угрозы, планировать меры профилактики и снижать вероятность правонарушений [3].

1. Цель и задачи разработки

Основная цель разработки заключается в создании доступного и удобного веб-сервиса для пользователей, нуждающихся в оперативной информации о безопасности и обмене данными с органами правопорядка. Сервис ориентирован на несколько групп пользователей: граждане, сотрудники государственных органов и правоохранительных структур, что требует создания ролевой модели с разграничением прав доступа [2].

В ходе разработки необходимо было решить ряд задач:

1. Разработать ролевую модель доступа, обеспечивающую безопасность данных и контроль за действиями пользователей в соответствии с их ролью [1, 2].

2. Создать разделы для мониторинга новостей, актуальных данных о преступности, чат для обмена сообщениями, а также администраторский интерфейс для управления пользователями.

3. Интегрировать сервис с внешними источниками новостей и данных по криминогенной обстановке [4, 5].

4. Разработать прогнозирующие алгоритмы на базе машинного обучения для анализа и прогнозирования криминогенной активности [3].

5. Обеспечить защиту данных через систему сессий и JWT токенов, а также создать гибкую систему уведомлений [1].

2. Функциональность веб-сервиса

Главная страница веб-сервиса представлена разделами для ознакомления и регистрации. Пользователи могут просмотреть описание сервиса и часто задаваемые вопросы. Процесс регистрации предусматривает создание личного аккаунта, который впоследствии позволяет получить доступ к расширенной функциональности сервиса.

Кабинет администратора предоставляет функции управления пользователями, назначение ролей и прав, а также возможность контролировать доступ к различным функциям сервиса. Администраторы могут создавать учетные записи для новых сотрудников, изменять пароли и настраивать доступ к разделам.

Пользователи могут управлять личными данными в разделе профиля. В настройках можно включить уведомления о новых сообщениях, новостях, обновлениях сервиса и других событиях, что повышает удобство использования

Раздел помощи включает часто задаваемые вопросы и ответы, руководство пользователя, а также форму для обращения в службу технической поддержки. Для каждой роли предусмотрены отдельные руководства, которые можно скачать и использовать для работы.

Раздел новостей служит для предоставления пользователям актуальной информации по теме преступности и безопасности. Новости поступают из государственных источников, через API интеграцию с фильтрацией по ключевым словам. Новостные карточки содержат краткое описание и изображение, а также возможность раскрыть полное содержание новости для ознакомления с деталями [4].

Существуют чаты для всех пользователей, а также закрытые чаты для госорганов и правоохранительных структур. Закрытые чаты обеспечивают защиту шифрованием данных и конфиденциальное общение, а также позволяют оперативно реагировать на ситуации, требующие внимания.

Журнал преступлений предоставляется в рамках расширенной функциональности и имеет ограниченный доступ. Работникам правоохранительных органов предоставлена возможность добавлять, изменять и удалять записи о преступлениях. Данные представляются в виде таблицы с функцией фильтрации и поиска. Для удобства пользователей предусмотрены экспорт и импорт данных в стандартных форматах, а также возможность загрузки данных для офлайн-работы.

Существует возможность просмотра карты, так называемой «горячих точек», которая отображает статистику по криминогенной ситуации в различных регионах. Карта позволяет визуально оценить уровень преступности в разных областях, а также отслеживать динамику по категориям преступлений. Данные доступны за интересующий период, что делает анализ простым и доступным [5].

Сервис оснащен ботом, который обучен на массиве данных и способен прогнозировать риск совершения преступлений по заданным критериям. Бот предоставляет пользователям информацию, основываясь на актуальной статистике, что позволяет использовать его в качестве вспомогательного инструмента для правоохранительных органов [3].

3. Примеры использования веб-сервиса

Рассмотрим функциональность страницы «Журнал преступлений» веб-сервиса. Основные функции страницы дополняются визуальными иллюстрациями.

На главной странице «Журнала преступлений» (рис. 1) представлена информация, содержащая записи о преступлениях. Каждая запись имеет раскрывающийся слой с детальными данными, такими как описание инцидента, местоположение, информация о преступнике, жертве и других деталях. Для удобства пользователей реализована система фильтрации записей по типу преступления, дате и статусу. Нажатие кнопки «Очистить фильтры» позволяет сбросить ранее установленные параметры фильтрации, возвращая таблицу к исходному состоянию.

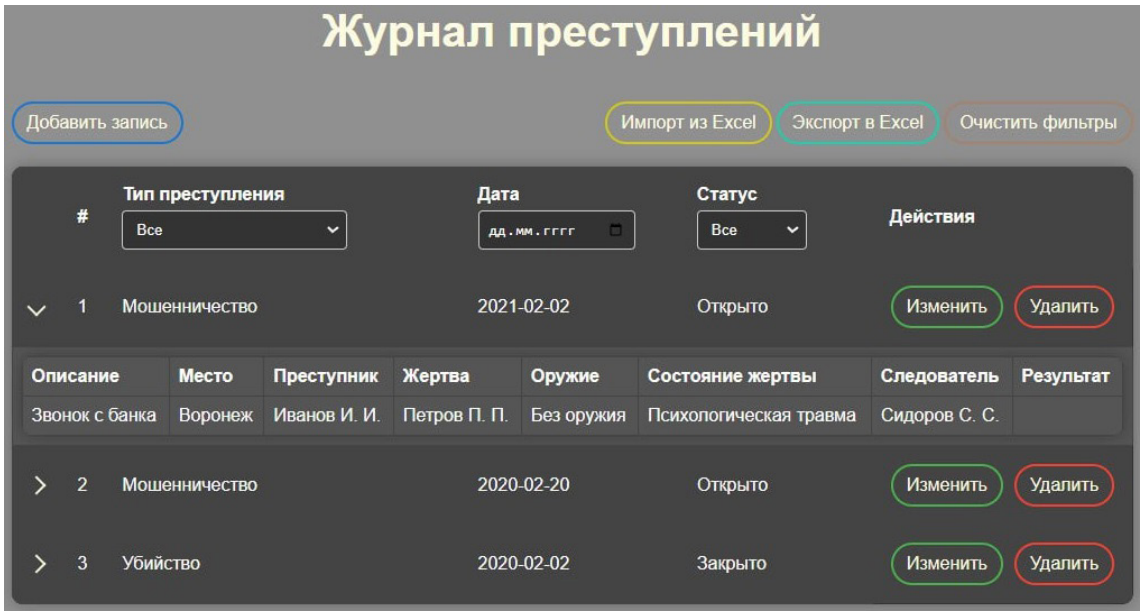


Рис. 1. Главная страница «Журнала преступлений»

Пользователи могут добавлять новые записи, а также редактировать или удалять существующие. Для создания и редактирования записей разработаны интуитивно понятные формы, которые включают обязательные поля для ввода информации. Пример интерфейсов добавления и редактирования записей приведен на рис. 2.

Данные из «Журнала преступлений» могут быть экспортированы в формате Excel для дальнейшего анализа или архивного хранения. Выгружаемый файл содержит структурированную таблицу с актуальными данными из базы. Для полей с ограниченным числом вариантов предусмотрены выпадающие списки, автоматически генерируемые на основе базы данных. Пример экспортируемой таблицы представлен на рис. 3.

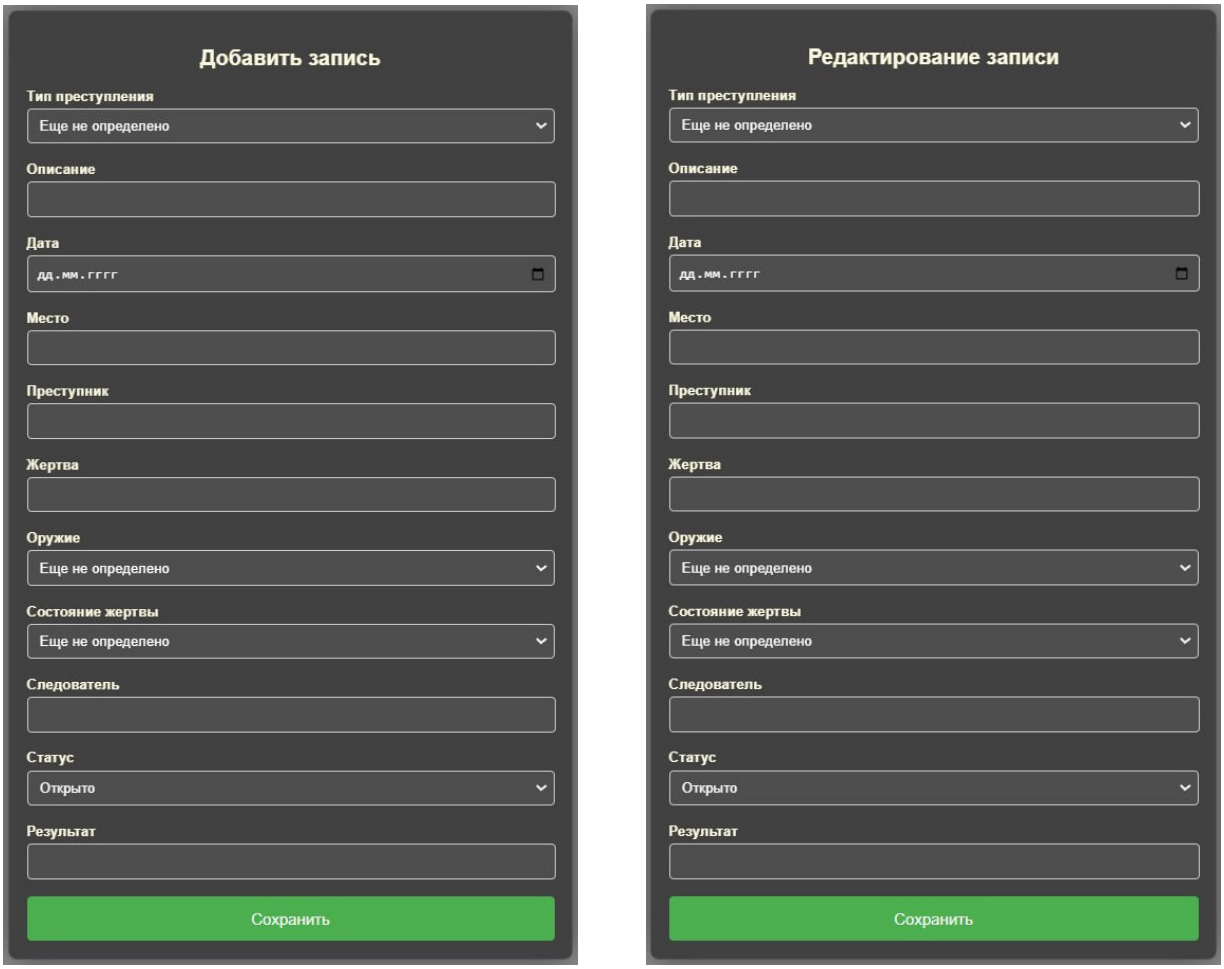


Рис. 2. Добавление и редактирование записей

	A	B	C	D	E	F	G	H	I	J
1	Тип преступления	Описание	Дата	Место	Преступник	Жертва	Оружие	Состояние жертвы	Следователь	Результат
2	Мошенничество	онок с банка	2021-02-02	Воронеж	Иванов И. И.	Петров П. П.	Без оружия	Психологическая травма	Сидоров С. С.	
3	Еще не определено	ман в магазине	2020-02-20	Липецк	Щестуков В. П.	Евкопин Р. Л.	Без оружия	Без травм	Костомалов А. П.	
4	Кража	ийство розочкой	2020-02-02	Ростов	Гудилин Р. В.	Куполин Е. Д.	Бутылка	Летальный исход	Никулин А. В.	
5	Убийство									
6	Мошенничество									
7	Нарушение ПДД									
	Грабеж									

Рис. 3. Таблица данных в Excel

Кроме того, система поддерживает импорт данных из шаблонного файла Excel. Данная функция значительно упрощает массовую загрузку записей и работу в офлайн-режиме. Для импорта данных используется специальная форма (рис. 4).

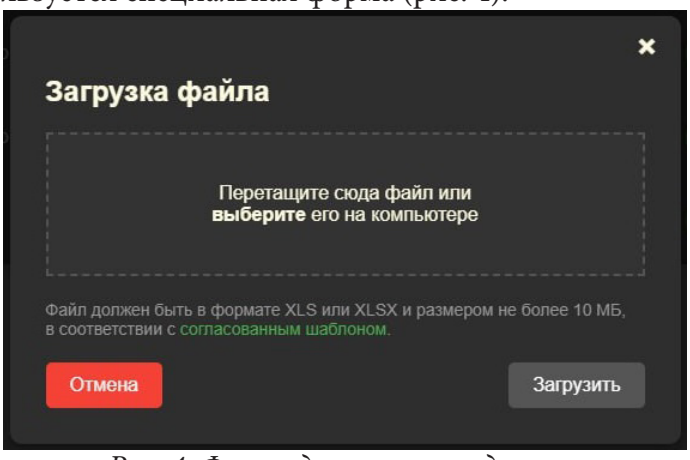


Рис. 4. Форма для импорта данных

Реализованные функции обеспечивают интуитивно понятное управление данными и способствуют повышению эффективности работы с информацией о преступлениях.

4. Технические аспекты разработки

Проект выполнен в монолитной архитектуре, что позволяет снизить сложность поддержки и разворачивания системы. Основные инструменты включают Java 17, Apache Tomcat, PostgreSQL для хранения данных, а также сервлеты и JSP для обработки данных на стороне сервера.

Ролевая система сервиса обеспечивает доступ пользователей к функциям на основании их роли (граждане, сотрудники спецслужб, сотрудники госорганов). Для защиты данных используются токены сессий, реализованные на базе JWT (access и refresh токены), что обеспечивает безопасность и контроль за действиями пользователей [1].

Тестирование API проводилось с использованием Postman: были разработаны коллекции и моковый сервер для проверки функциональности API и обработки ошибок. Веб-сервис также прошел нагрузочное тестирование, тестирование на обратную совместимость и регрессионное тестирование, оптимизированное с помощью автоматизированных тестов [6]. В рамках тестирования были выполнены проверки безопасности, UI/UX тестирование, а в некоторых модулях использовалась методология разработки через тестирование (TDD).

Заключение

Разработанный веб-сервис обеспечивает эффективное и комплексное решение для мониторинга криминогенной обстановки, обмена информацией и поддержки взаимодействия между гражданами и государственными структурами. Реализация многоуровневой ролевой системы, позволяющей разграничивать права доступа для разных категорий пользователей, обеспечивает защиту данных. Правоохранительные органы получают рабочий инструмент для мониторинга криминогенной ситуации с целью своевременного реагирования, а обычные пользователи — аналитический инструмент.

Формирование прогнозов и выявление потенциальных зон риска открывает возможности для превентивных мер. Кроме того, сервис может способствовать улучшению взаимодействия между гражданами и органами правопорядка, создавая безопасную информационную среду для всех участников.

Литература

1. *Ховард М.* Защищенный код / М. Ховард, Д. Леблан. – 2-е изд., испр. – М. : Русская Редакция, 2005. – 704 с.

2. *Хрущев Д. Г.* Ролевые модели доступа в информационных системах / Д. Г. Хрущев // Форум молодых ученых. – 2017. – № 3. – С. 634–636.

3. *Флэч П.* Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных / П. Флэч. – М. : ДМК Пресс, 2015. – 400 с.

4. *Иванов А. М.* Интеграция новостных API в веб-приложения / А. М. Иванов // Вестник информационных технологий. – 2022. – № 3. – С. 45–52.

5. *Клишков В. В.* Геоинформационные системы в криминологии / В. В. Клишков. – Санкт-Петербург : Санкт-Петербургский университет МВД России, 2010. – 256 с.

6. *Савин Р.* Тестирование dot com / Р. Савин. – М. : Дело, 2007. – 312 с.

УДК 004.42

АДАПТАЦИЯ ПРИМЕНЕНИЯ КОНКУРЕНТНОСТИ В ЗАДАЧЕ РЕИНЖИНИРИНГА ДЕСКТОПНОГО ПРИЛОЖЕНИЯ В ВЕБ-ПРИЛОЖЕНИЕ

И. Ю. Полеводин, И. Е. Воронина

Воронежский государственный университет

Аннотация. Рассматривается задача реинжиниринга десктопного приложения в веб-приложение, а именно адаптация исходного кода, использующего при выполнении своих функций конкурентность. Решение основано на адаптации применения асинхронности и параллелизма.
Ключевые слова: реинжиниринг, веб-приложение, десктопное приложение, конкурентность, асинхронность, параллелизм, многопоточность.

Введение

Группа компаний «Экологический центр» (ГК «ЭКО ЦЕНТР») — крупнейший проектный институт, практическая деятельность которого связана с разработкой и сопровождением природоохранной документации, а также специализированных программных комплексов. Одним из таких программных продуктов является десктопный программный комплекс «Методики», позволяющий рассчитывать выбросы загрязняющих веществ в атмосферу во время работы определенного завода или предприятия. В расчете учитываются особенности производственных процессов и нюансы технологических операций предприятия. Программный комплекс разработан на основе соответствующих нормативных документов. Примерами модулей, которые входят в комплекс, являются:

- «Котельная» (расчет выбросов в атмосферный воздух загрязняющих веществ при сжигании твердого, жидкого и газообразного топлива);
- «Ферма» (расчет выбросов загрязняющих веществ от объектов животноводства и птицеводства).

Анализ пользовательского опыта, особенностей областей экологического и гидрогеологического проектирования, проведения изыскательских работ, а также опыта разработки и сопровождения программного комплекса «Методики» показал, что:

1. Часто пользователям требуется доступ к системе вне их рабочего места с установленным программным обеспечением (ПО). Удобно решать некоторые задачи, находясь непосредственно на предприятии, а иногда требуется выполнить (доделать) определенные работы в дороге, дома или в любом другом месте.

2. Узкая специализация программного комплекса требует регулярной обратной связи от пользователей. Для того, чтобы ПО помогало решать их задачи корректно и эффективно, часто выпускаются обновления, которые пользователям необходимо каждый раз устанавливать. На частоту обновлений комплекса также влияет публикация новых и обновление существующих нормативных документов. Поэтому необходимо решить задачу автоматического получения новых версий ПО пользователями.

Наличие доступа к системе через браузер, установленный на устройстве пользователя, позволит решить вышеперечисленные проблемы. Таким образом, актуальна задача реинжиниринга программного комплекса «Методики» из десктопного приложения в веб-приложение.

Одной из подзадач реинжиниринга десктопного приложения в веб-приложение является задача адаптации исходного кода, использующего при выполнении своих функций конкурентность, так как принцип работы десктопного приложения, установленного на устройстве

пользователя, отличается от веб-приложения, запущенного на сервере и обрабатывающего входящие запросы.

1. Особенности решения задачи реинжиниринга

Десктопное приложение устанавливается локально на устройство пользователя. Этот пользователь является единственным для данной программы. Веб-приложение развертывается на сервере и эксплуатируется многими пользователями, отправляющими запросы с помощью браузера на своем устройстве. Таким образом, в распоряжении нескольких пользователей оказывается одни и те же ресурсы сервера: центральный процессор, память и др. Серверная часть кода приложения должна учитывать это, имея некоторые ограничения на написание компонентов кода и пользуясь возможностями, которые предлагает выбранная платформа для улучшения производительности.

Реинжиниринг десктопного приложения в веб-приложение должен включать в себя адаптацию кода, использующего конкурентность. Рассмотрим необходимые определения [1].

1. Конкурентность — выполнение сразу нескольких действий в одно и то же время.

2. Многопоточность — форма конкурентности, использующая несколько программных потоков выполнения.

3. Параллельная обработка — выполнение большого объема работы за счет распределения ее между несколькими потоками, выполняемыми одновременно.

4. Асинхронное программирование — разновидность конкурентности, использующая обещания или обратные вызовы для предотвращения создания лишних потоков.

Десктопное приложение часто может свободно использовать параллельную обработку, поскольку все потоки, необходимые для нее, не могут потребоваться для задач другого пользователя (все происходит локально). Их, разумеется, не должно быть слишком много, чтобы не снижалась производительность. Но для десктопного приложения при создании новых потоков такое снижение менее вероятно, чем для веб-приложения, которое выделяет потоки для обработки отдельных запросов разных пользователей. Поэтому хорошей рекомендацией при написании веб-приложения является сведение использования параллелизма к минимуму там, где это возможно [2].

При этом, асинхронность может быть более востребована в веб-приложении, чем в десктопном, поскольку она позволяет освобождать потоки во время выполнения определенных операций (например, ввода-вывода), что не влияет на скорость ответа текущему пользователю, но положительно сказывается на общей производительности, так как освобожденный поток может заняться обработкой другого запроса. В десктопном приложение тоже может потребоваться освободить поток, чтобы он, например, будучи UI-потоком, продолжил реагировать на действия пользователя, но не всегда это необходимо.

Также при реинжиниринге десктопного приложения в веб-приложение следует соответствующим образом скорректировать работу со статическими компонентами (классами, членами), поскольку они будут являться общими для нескольких (возможно, всех) пользователей, и при обращении к ним возникнет конкурентность. Для таких компонентов можно использовать потокобезопасные структуры.

Необходимо учесть, что память сервера, на котором развернуто веб-приложение, ограничена и используется для множества пользователей, поэтому следует минимизировать загрузку и хранение данных в памяти [3].

2. Общие сведения о технологиях программного комплекса «Методики»

Программный комплекс «Методики» разработан на платформе .NET Framework на языке C# и является приложением Windows Forms. Пользователь скачивает установщик программы с сайта компании и устанавливает ПО на устройство. Далее программа активируется и используется для расчета выбросов загрязняющих веществ, создания отчетов, экспорта в различные форматы и т. п. Пользователь указывает данные о своем предприятии (площадки, цеха, источники загрязнения, их режимы работы и многое другое), и приложение выполняет подробный расчет для всех или для определенных данных. Основным значением в расчете является скорость выброса загрязняющего вещества, измеряющаяся в г/с (грамм в секунду) или в т/г (тонн в год).

Веб-версия программного комплекса должна обладать той же функциональностью, но с доступом из браузера и системой аккаунтов, в рамках которых будет осуществляться работа. В качестве платформы будет использоваться ASP.NET Core (язык C#), в качестве ORM — Entity Framework Core.

3. Асинхронность в программном комплексе «Методики»

Программный комплекс «Методики» является десктопным приложением, то есть рассчитан на одного пользователь в каждый момент времени, поэтому наличие асинхронности при выполнении различных операций не является фактором, критично влияющим на производительность или на пользовательский опыт. Большинство операций выполняется синхронно, то есть поток выполнения не освобождается на время каких-либо длительных операций, в которых он не принимает участие. Также не предусмотрена возможность отмены каких-либо действий во время их выполнения или любое другое взаимодействие с интерфейсом, пока не закончится текущая обработка.

В веб-приложении асинхронность более востребована, так как пользователей одновременно несколько. Поэтому при реинжиниринге программного комплекса «Методики» необходимо добавить асинхронность в соответствующих местах.

Примерами операций, для которых требуется асинхронность в веб-приложении, являются чтение и запись в базу данных (БД). Как правило, это происходит почти при каждом запросе пользователя. В десктопной версии программного комплекса «Методики» такие операции выполняются синхронно.

Рассмотрим добавление в БД данных о площадке (объединяющей цеха предприятия).

```
public void AddArea(Area area)
{
    _db.Add(area);
    _db.SaveChanges();
}
```

Для работы с БД используется Entity Framework. Сущность добавляется в контекст, представляющий сеанс с БД, и затем выполняется сохранение всех внесенных в контекст изменений. Операция сохранения (записи в БД) является относительно длительной по времени и не требующей участия выполняемого потока, поэтому если она будет выполняться на сервере веб-приложения, то желательно сделать ее асинхронной.

Entity Framework Core поддерживает асинхронные операции. У многих методов есть асинхронные аналоги. В том числе, у синхронного SaveChanges есть асинхронный аналог SaveChanesAsync.

Рассмотрим добавление в БД данных о площадке, адаптированное под выполнение на сервере веб-приложения.

```
public async Task AddArea(Area area)
{
    _db.Add(area);
    await _db.SaveChangesAsync();
}
```

Когда приходит запрос от пользователя для добавления площадки, начинает выполняться асинхронный обработчик этого запроса (с модификатором async), который вызывает (с помощью выражения await) асинхронный метод добавления площадки в БД, внутри которого вызывается асинхронный метод записи в БД. И при непосредственном начале записи выполнение кода возвращается в вызывающий метод, из него — в его вызывающий метод, и т. д. И в конце поток, выделенный для обработки этого запроса, возвращается в пул потоков для обработки других входящих запросов [4]. Когда запись в БД закончится, какой-нибудь поток из пула продолжит выполнение кода, и в итоге пользователю вернется ответ об успешном добавлении площадки (либо информация об ошибке).

Таким образом, производительность приложения повышается за счет того, что потоки не простаивают, а освобождаются для выполнения других задач.

4. Параллелизм в программном комплексе «Методики»

Ключевой функцией программного комплекса «Методики» является расчет выбросов загрязняющих веществ. Расчет выполняется на основе введенных пользователем данных. Таких данных может быть очень много, при этом значительную часть алгоритма можно выполнять, обрабатывая фрагменты данных независимо друг от друга. Поэтому в десктопной версии приложения для расчета выбросов применяется параллелизм.

В веб-приложении параллелизм, предполагающий выделение нескольких потоков для одного пользователя, может оказаться неэффективным в контексте того, что приложение также обрабатывает запросы других пользователей с помощью потоков из того же пула потоков.

Рассмотрим расчет выбросов загрязняющих веществ (методы находятся в разных классах).

```
public CalcResult Calc()
{
    var calcInput = GetCalcInput();
    return _calc.Calc(calcInput);
}
public CalcResult Calc(CalcInput calcInput)
{
    // Реализация алгоритма, в рамках которой данные для расчета
    // обрабатываются параллельно
}
```

Помимо того, что при расчете используется параллелизм, данные для расчета загружаются из БД синхронно. Необходимо адаптировать код так, чтобы, выполняясь на сервере веб-приложения, расчет проводился достаточно быстро, и при этом не снижалась общая производительность приложения.

Особенности эксплуатации программного комплекса таковы, что активность использования может сильно различаться в разные периоды, поскольку клиентам необходимо формировать отчетность примерно в одни и те же сроки, и происходит это, как правило, ближе к окончанию этих сроков. Соответственно, количество одновременных пользователей веб-приложения также будет сильно различаться. Опираясь на это количество, можно принимать решение, выделять дополнительные ресурсы для обработки запроса или нет. Если пользователей мало, то для расчета допустимо применить параллелизм. Если много — выделять потоки не стоит.

Рассмотрим фрагмент компонента для подсчета пользователей веб-приложения, активных в данный момент.

```
private static readonly ConcurrentDictionary<string, bool> _allKeys = new();
public Task InvokeAsync(HttpContext context, IMemoryCache memoryCache)
{
    if (!context.Request.Cookies.TryGetValue(CookieName, out var userGuid))
    {
        userGuid = Guid.NewGuid().ToString();
        // Добавление cookie к ответу пользователю
    }
    memoryCache.GetOrCreate(userGuid, cacheEntry =>
    {
        if (!_allKeys.TryAdd(userGuid, true))
        {
            cacheEntry.AbsoluteExpiration = DateTimeOffset.MinValue;
        }
        else
        {
            cacheEntry.SlidingExpiration =
                TimeSpan.FromMinutes(LastActivityMinutes);
            cacheEntry.RegisterPostEvictionCallback(RemoveKeyWhenExpired);
        }
        return string.Empty;
    });
    return _next(context);
}
public static int Count() => _allKeys.Count(p => p.Value);
```

Для каждого пользователя (при его первом запросе) генерируется GUID, который хранится на устройстве пользователя и отправляется вместе с каждым запросом. На сервере хранится список таких ключей, который периодически обновляется. Если пользователь определенное время не отправлял запросы серверу, то его ключ удаляется, соответственно, пользователь не учитывается в подсчете. Для хранения ключей используется потокобезопасная коллекция, потому что она, являясь статическим членом класса, может использоваться из разных потоков одновременно. Класс ConcurrentDictionary позволяет обеспечить корректную работу с объектом.

Теперь рассмотрим расчет выбросов загрязняющих веществ, адаптированный под выполнение на сервере веб-приложения (методы находятся в разных классах).

```
public async Task<CalcResult> Calc(int onlineUsersCount)
{
    var calcInput = await GetCalcInput();
    return _calc.Calc(calcInput,
        useParallelism: onlineUsersCount < MaxOnlineUsersCountForParallelism);
}
public CalcResult Calc(CalcInput calcInput, bool useParallelism)
{
    if (useParallelism)
    {
        // Реализация алгоритма, в рамках которой данные для расчета
        // обрабатываются параллельно
    }
    else
    {
```

```
        // Реализация алгоритма, в рамках которой данные для расчета
        // обрабатываются последовательно
    }
}
```

При расчете учитывается количество пользователей, активных в данный момент (как показатель загруженности сервера). Если это количество меньше определенного значения, то применяется параллелизм, иначе данные обрабатываются последовательно. Также данные для расчета загружаются из БД асинхронно.

Таким образом, производительность приложения повышается за счет того, что количество потоков, выполняемых на сервере, сохраняется достаточно небольшим.

Заключение

В рамках решения задачи реинжиниринга десктопного приложения в веб-приложение был проведен анализ подзадачи адаптации исходного кода, использующего при выполнении своих функций конкурентность. Определены основные аспекты подзадачи, для каждого из которых рассмотрены конкретные примеры из программного комплекса «Методики», и для каждого предложено решение, обеспечивающее корректную работу веб-приложения.

Литература

1. *Стивен Клири.* Конкурентность в C# / Стивен Клири. – Санкт-Петербург : Питер, 2020. – 17 с.

2. Джеффри Рихтер. CLR via C# / Джеффри Рихтер. – Санкт-Петербург : Питер, 2022. – 250 с.

3. CodeGuru. WinForms to Web – URL: https://www.codeguru.com/csharp/winforms-to-web (дата обращения: 10.10.2024).

4. *Эндрю Лок.* ASP.NET Core в действии / Эндрю Лок. – Москва : ДМК Пресс, 2021. – 122 с.

УДК 004.413.5

АНАЛИЗ МЕТОДОВ ОЦЕНКИ ДЛЯ ЗАДАЧИ ОПРЕДЕЛЕНИЯ СТОИМОСТИ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

К. Е. Полянская, И. Е. Воронина

Воронежский государственный университет

Аннотация. Рассматривается нетривиальная задача определения стоимости разработанного программного обеспечения. Представлен анализ существующих алгоритмических методов оценки стоимости разработки: рассмотрены преимущества и недостатки методик, а также возможность их применения для оценки стоимости уже разработанного программного обеспечения.
Ключевые слова: оценка, стоимость, трудозатраты, трудоёмкость, размер программного продукта, LOC, функциональные точки, модель оценки, методика.

Введение

Оценка стоимости разработки программного обеспечения (ПО) является одной из ключевых задач в управлении проектами и планировании ресурсов. Стоимость программного обеспечения определяется множеством факторов: уровнем сложности и уникальностью разработки, объёмом кода, качеством документации, возможностями для поддержки и адаптации, а также рядом других показателей, которые часто бывают трудно поддающимися точной оценке.

Особую сложность представляет собой оценка стоимости уже разработанного ПО. С развитием технологий задача оценки стоимости уже существующего ПО становится весьма актуальной. Ситуация, когда требуется такая оценка, возникает, например, в случаях судебных разбирательств по государственным контрактам, когда конечный продукт не соответствует ожиданиям заказчика, или возникает подозрение в завышении его стоимости. При решении данной задачи требуется адаптация традиционных методов оценки к новым условиям, поскольку зачастую отсутствуют подробные данные о процессе разработки, использованных методах и затратах на промежуточных этапах. Во многих случаях оценка стоимости может основываться в большей мере только на анализе исходного кода, что значительно усложняет сам процесс оценивания и понижает точность результата, так как программный код не всегда может полностью отражать затраты на проектирование, тестирование, интеграцию и другие важные аспекты разработки.

Поскольку сама по себе задача оценки стоимости разработанного ПО относится к разряду нетривиальных, необходимо проанализировать существующие алгоритмические методы оценки стоимости разработки программного обеспечения, их преимущества и недостатки, а также возможности их применения для оценки стоимости уже разработанных систем.

1. COCOMO II

Методика COCOMO [1] позволяет оценить трудоемкость и время разработки программного продукта.

Различаются две стадии оценки проекта: предварительная оценка на начальной фазе и детальная оценка после проработки архитектуры. Формула оценки трудоемкости проекта в чел.*мес. имеет следующий вид:

$$PM = A \times SIZE^{E} \times \sum_{i=1}^{n} EM_i, \;\; E = B + 0.01 \times \sum_{j=1}^{5} SF_j.$$

Здесь $SIZE$ — размер продукта в KSLOC; EM_i — множители трудоемкости; SF_j — факторы масштаба; $A = 2{,}94$; $B = 0{,}91$; $n = 7$ — для предварительной оценки; $n = 17$ — для детальной оценки.

В контексте решаемой задачи имеет смысл акцентировать внимание на модели детальной оценки, хотя обе модели используют один и тот же подход к определению размеров продукта и коэффициентов масштабирования.

Основной метрикой объёма продукта, применяемой в данной методике, является размер программного продукта в тысячах строках исходного кода (KSLOC, Kilo Source Lines Of Code). Определение термина «строка кода» имеет некоторые сложности, поскольку подсчёт исполняемых элементов в разных языках программирования сильно различается в связи с особенностями объявления переменных, конструкций языка и т. п. В модели COCOMO II в качестве единицы строки кода были выбраны логические строки (т. е. строки кода, которые содержат конкретную операцию или выражение, что позволяет учесть особенности языка, в отличие от физических строк без разбиения на операции).

Для минимизации проблем, связанных с оценкой количества строк кода, Институт программной инженерии (Software Engineering Institute — SEI) разработал специальный анкетный лист, по результатам которого можно оценить размер кода. На основе данной информации становится проще выделить логические строки и подсчитать их общее количество.

Альтернативным способом подсчёта размера продукта может быть применение метода функциональных точек, основная идея которого заключается в подсчёте количества реализованных функций, которые система предоставляет пользователям. В таком случае подсчитанные функциональные точки конвертируются в KSLOC с помощью исторических данных организации по уже выполненным проектам, либо среднестатистическим показателям по отрасли.

Одним из аспектов, учитываемых в методике COCOMO II, является подсчёт повторно используемого, адаптированного и автоматически генерируемого кода. Эффективный размер таких компонентов корректируется таким образом, чтобы он был эквивалентен новому коду. Для этого используются такие показатели, как процент автоматически генерируемого кода, фактор трудоёмкости перевода компонентов в повторно используемые, процент модифицируемых для повторного использования проектных моделей, процент модифицируемого для повторного использования кода, процент затрат на интеграцию и тестирование повторно используемых компонентов, фактор понятности повторно используемого кода.

В методике используются пять факторов масштаба SF_j которые определяются следующими характеристиками проекта: прецедентность, наличие опыта аналогичных разработок; гибкость процесса разработки; архитектура и разрешение рисков; сработанность команды; зрелость процессов.

В рамках пост-архитектурной модели методики необходимо провести оценку 17-ти множителей трудоёмкости: надёжность продукта, сложность продукта, размер базы данных разрабатываемого приложения, требуемый уровень повторного использования, требуемый уровень документированности, опыт работы команды в данной предметной области, опыт работы команды с используемыми платформами и др.

COCOMO II является мощным инструментом для оценки стоимости разработки ПО, так как позволяет учитывать не только объём кода, но и такие параметры, как качество программных процессов, зрелость технологий, опыт команды и требования к надёжности продукта. Однако его применение ограничено в случаях, когда для анализа представлен только исходный код. Значительным преимуществом данной методики можно считать наличие достаточно тщательного подхода к оценке размера продукта, позволяющего скорректировать размер повторно использованного и автоматически генерируемого кода. Но качество подсчёта бюджета в рамках модели COCOMO II сильно зависит от адекватности количественных показателей её факторов, соответственно от опыта и информированности ответственного за расчёт.

В рамках задачи определения стоимости разработанного ПО возникает трудность в оценке данных показателей, т. к. как уровень зрелости команды, сложность требований, взаимодействие с заказчиком и многие другие факторы, которые учитываются в COCOMO II, не могут быть оценены на основе только исходного кода, а усреднённые значения данных коэффициентов могут дать значительную погрешность в оценке. Кроме того, стоит сказать об ограниченной применимости для современных методологий: несмотря на то, что COCOMO II поддерживает многие современные подходы, её применение в рамках Agile и других гибких методологий может быть затруднительным из-за отсутствия фиксированных требований и итерационного характера разработки. Хотя модель обновлялась для учёта новых технологий, некоторые её предположения могут быть неактуальны для проектов, использующих самые современные инструменты и практики (например, DevOps, CI/CD).

2. Метод функциональных точек (IFPUG FPA)

Одной из ключевых особенностей моделей семейства COCOMO является зависимость их результата от размера самого программного комплекса, выраженного в строках исходного кода. Стоит понимать, что количество строк зависит от технологий и синтаксических особенностей конкретного языка программирования, применяемого при разработке приложения, а также стиля программирования, характерного для конкретной команды разработчиков, поэтому такая метрика не может в полной мере отражать количество трудозатрат при написании заданного объёма кода.

Альтернативным способом подсчёта размера продукта, который применяется во многих других моделях оценки трудозатрат, является метод функциональных точек [2].

Суть метода состоит в том, что размер компонентов программного обеспечения можно оценивать в терминах количества и сложности функций, реализованных в данном программном коде. Оценка числа функциональных точек для программного продукта выводится на основе данных, которые определяются в результате анализа информационной области программного продукта и изучения особенностей его будущего функционирования.

При анализе методом функциональных точек надо выполнить следующую последовательность шагов: определение типа оценки; определение области оценки и границ продукта; подсчёт функциональных точек, связанных с данными; подсчёт функциональных точек, связанных с транзакциями; определение суммарного количества не выровненных функциональных точек (UFP); определение значения фактора выравнивания (FAV); расчёт количества выровненных функциональных точек (AFP).

Общее число функциональных точек программы зависит от количества элементарных процессов пяти типов:

1) входящие транзакции (External inputs (EI)) — получают данные от пользователя;

2) исходящие транзакции (External outputs (EO)) — передают данные пользователю;

3) взаимодействия с пользователем (External inquiries (EQ)) — интерактивные диалоги с пользователем;

4) файлы внутренней логики (Internal logical files (ILF)) — файлы (логические группы информации), использующиеся во внутренних взаимодействиях системы;

5) файлы внешних взаимодействий (External interface filese (EIF)) участвуют во внешних взаимодействиях с другими системами. В данной терминологии транзакция — элементарный неделимый замкнутый процесс, имеющий значение для пользователя и переводящий продукт из одного консистентного состояния в другое.

Каждому числовому значению параметра присваивается соответствующий весовой коэффициент, который учитывает, насколько сложно реализовать рассматриваемый параметр. Очевидно, что подобная оценка сложности является экспертной и достаточно субъективной.

Для окончательного подсчёта количества функциональных точек, требуется дополнительно учесть 14 факторов, для которых установлена шкала весовых коэффициентов, отражающих степень влияния того или иного фактора при функционировании программного изделия, таких как: связи данных (как много связей используется для передачи данных внутри системы), распределённая обработка данных, требования по производительности, требования по аппаратной части и др.

Метод функциональных точек — это полезный инструмент для оценки размеров и усилий разработки программного обеспечения на основе функциональности системы. Он особенно эффективен для крупных систем и на ранних стадиях планирования, когда требуется оценка затрат без учёта деталей внутренней реализации. С одной стороны, при оценке размера продукта данный метод позволяет абстрагироваться от конкретной технологической платформы, на которой разрабатывается продукт, и обеспечивает единообразный подход к оценке его размера. Однако нельзя утверждать, что при наличии исходного кода программного продукта, стиль написания кода которого соответствует стандартам языка, подсчёт его функциональных точек даст более точный результат, чем определение количества логических строк кода. Стоит понимать, что метод анализа функциональных точек ничего не говорит о трудоёмкости разработки оцененного продукта, он позволяет определить размер программного продукта, а временные затраты оцениваются исходя из статистики реализации аналогичных функциональных точек, либо при помощи соответствующих моделей оценки. Кроме того, данный метод достаточно трудоёмкий в использовании, т. к. трудозатраты при его использовании пропорциональны объёму оцениваемого программного продукта и количеству функций, реализуемых в нём, а также точность оценки сильно зависит от правильности классификации функций и опыта оценщика.

3. SLIM

Модель Путнэма (SLIM) — это алгоритмическая модель оценки трудоёмкости разработки ПО, предложенная в 1978 г. и основанная на распределении Рэлея [3]. Модель основывается на утверждении, что затраты на разработку ПО распределяются согласно кривым Рэлея, которые являются графиками функции, представляющей распределение рабочей силы по времени. Эта модель строится на основании исторической базы данных (БД) по выполненным в данной организации проектам и поэтому полнее учитывает её специфику.

Трудоёмкость в данной модели рассчитывается по следующей формуле:

$$PM = \left(\frac{Size}{P \times T^{4/3}} \right)^3 \times B.$$

Здесь $Size$ — размер программного обеспечения, P — технологический фактор/продуктивность, способность организации разрабатывать ПО требуемого объёма и качества, T — ограничения на время разработки, B — масштабируемый коэффициент.

Технологический фактор включает в себя характеристику проекта в следующих аспектах: методы управления и понимание процесса, качество используемых методов инженерии ПО, уровень используемых языков программирования, уровень развития среды, навыки и опыт команды разработчиков, сложность приложения.

Одним из этапов применения данной модели является определение значения производительности. Если предположить, что в стабильном коллективе разработчиков, работающих по налаженному технологическому процессу, производительность практически не меняется от проекта к проекту, то можно выписать $n-1$ уравнений для уже выполненных проектов. Подставляя в эти уравнения значения $Размер\_кода_i$, $Срок_i$ и $Трудоёмкость_i$ из исторической базы данных уже выполненных n проектов, можно получить значения множителя B в n точках

Размер_кода$_i$ $(1 \le i \le n)$, по которым его можно аппроксимировать для нового $n+1$ проекта. Вместе с этим можно получить и значение производительности, после чего по уже известным значениям B и *Производительности* можно получить уравнение, связывающее оценки значений *Срок* и *Трудоёмкость* для нового проекта.

К преимуществам модели SLIM можно отнести широкие возможности по калибровке модели, простая зависимость, связывающая затраты на программное обеспечение и трудоёмкость на его разработку, а также то, что для расчёта используется меньше параметров по сравнению с моделями семейства COCOMO. К недостаткам модели SLIM можно отнести чувствительность её оценок к изменениям в параметрах разработки, а также слабую применимость этой модели к малым проектам (не рекомендуется использование данной модели для проектов объёмом меньше 70 000 строк кода).

В условиях решения поставленной задачи использование модели SLIM затруднительно, поскольку модель строится на основании исторической БД по выполненным в данной организации проектам, однако доступ к историческим данным может отсутствовать.

4. PERT

PERT — метод оценки трудоёмкости проекта, основанный на анализе задач, необходимых для его выполнения [4]. PERT был разработан главным образом для упрощения планирования и составления графиков больших и сложных проектов, давая возможность разработать рабочий график проекта без точного знания деталей и необходимого времени для всех его составляющих.

Входом для данного метода оценки служит список элементарных пакетов работ. Диапазон неопределённости характеризуется тремя оценками: M_i — наиболее вероятная оценка трудозатрат; O_i — минимально возможные трудозатраты на реализацию пакета работ; P_i — пессимистическая оценка трудозатрат (реализация всех рисков).

Средняя трудоёмкость по каждому пакету:

$$E_i = (P_i + 4M_i + O_i) / 6.$$

Далее подсчитываются суммарная трудоёмкость, среднеквадратичная ошибка оценки каждого пакета, среднеквадратичная ошибка суммарной оценки и гарантированная оценка с вероятность 95 %.

Применение метода PERT после завершения разработки и при наличии исходного кода не является типичным. Однако, можно использовать некоторые аспекты PERT для сравнения запланированных и фактических затрат и сроков, что может помочь в улучшении будущих оценок. Например, замена единицы измерения реализации компонентов в PERT (время) на количество строк кода (соответственно оптимистическое, пессимистическое, наиболее вероятное) может быть альтернативным способом определения размера программного продукта, однако при наличии исходного кода данная задача не является актуальной.

Стоит понимать, что при использовании метода PERT необходимо предъявлять строгие требования к статистической независимости оценок, компетентности эксперта, т. к. чрезвычайный оптимизм, или, наоборот, необоснованный пессимизм оценок серьёзно влияет на результирующую оценку.

5. SEER-SEM

SEER-SEM — автоматизированная система расчета стоимости разработки программного обеспечения с помощью параметрического моделирования [5].

Одним из этапов использования данной модели является определение размера программного продукта. Данная метрика трансформируется во внутренний программный показатель

S_e (1), который позволяет провести анализ процесса разработки нового или уже готового к коммерческому использованию кода.

$$S_e = NewSize + ExistingSize * (0.4 * Redesign + 0.25 * Reimpl + 0.35 * Retest) \quad (1)$$

S_e увеличивается прямо пропорционально количеству нового ПО, а также зависит от объёма работ по перепроектированию, повторному использованию и тестированию.

Измерение количества функциональных точек может намного повысить точность прогноза. Модель SEER-SEM не просто вносит поправки в соответствии с особенностями языка программирования, но также учитывает такие факторы, как фазу оценки, операционное окружение, тип приложения и его сложность с использованием показателя энтропии в зависимости от типа разрабатываемого ПО.

Модель учитывает взаимную связь трудоёмкости и сроков проекта по формуле (2).

$$K = D^{0.4} * \left(\frac{S_e}{C_{te}} \right)^E . \quad (2)$$

Здесь S_e — размер, приведенный ранее; C_{te} — эффективность технологии — составной показатель, который отражает продуктивность или эффективность людей, процессов, вовлечённых в развитие проекта; D — степень интеграции сотрудников, показатель сложности проекта с точки зрения количества вовлечённых человек; E — энтропия.

Модель SEER-SEM является частью коммерческого программного обеспечения, алгоритмы вычислений которого не раскрыты, в связи с чем модель невозможно использовать в независимых программных продуктах. Из-за недоступности алгоритмов SEER-SEM возникает неясность, касающаяся поддержки некоторых типы входных данных и определения значений используемых параметров. А также нет опубликованных независимых исследований, касающихся использовании этой модели.

6. Методика расчёта планируемой стоимости работ по созданию информационных систем Московской области

В методике расчёта планируемой стоимости работ по созданию, развитию и сопровождению информационных систем (ИС) Московской области расчёт затрат на создание ИС представлен следующими последовательно выполняемыми этапами [6]: оценка функционального размера ИС, оценка базовой трудоёмкости создания ИС, оценка итоговой трудоёмкости работ по созданию ИС, расчёт средней стоимости человеко-месяца работы разработчика ИС, расчёт затрат на создание ИС.

Оценка функционального размера ИС производится на основе функциональных требований ИС, содержащихся в технических требованиях. Функциональный размер задается набором из шести элементов:

1) количество подсистем ИС (Subsystem) — S;
2) количество функций подсистем ИС (Function) — F;
3) количество микрофункций (возможностей функций) ИС (Tool) — T;
4) количество уникальных информационных объектов ИС (Entity) — E;
5) количество атрибутов информационных объектов ИС (Attribute) — A;
6) количество свойств атрибутов информационных объектов ИС (Properties) — P.

Далее оценивается базовая трудоёмкость разработки ИС (Basic labor intensity — B) как сумма произведений единиц измерения функционального размера и значений коэффициентов трудоёмкости с учётом поправочных коэффициентов трудоёмкости по формуле:

$$B = w_1 * (S * i_1 + F * i_2 + T * i_3) + w_2 * (E * i_4 + A * i_5 + P * i_6), \text{ где}$$

$\{S, F, T, E, A, P\}$ — функциональные единицы размера ИС, определённые на основании анализа функциональных требований к ИС, содержащихся в технических требованиях; $i_1 - i_6$ —

коэффициенты трудоёмкости — функциональные единицы измерения, определенные на основании анализа функционального размера разработанных ИС Московской; w_1 и w_2 — поправочные коэффициенты трудоемкости — функциональные единицы измерения, определённые на основании анализа функционального размера разработанных ИС Московской области.

Процесс создания и развития ИС, согласно методике, состоит из следующих видов работ (type of work – tw), представленных в табл. 1.

Таблица 1

Виды работ по созданию ИС

Наименование вида работ по созданию ИС	Условное обозначение	Коэффициент трудоёмкости работ
Разработка технического задания на создание ИС	tw_1	0,05
Разработка документации технического проекта ИС	tw_2	0,07
Разработка СПО	tw_3	0,7
Разработка рабочей документации на ИС	tw_4	0,05
Пусконаладочные работы	tw_5	0,05
Проведение предварительных испытаний ИС	tw_6	0,01
Подготовка (обучение) персонала ИС	tw_7	0,02
Проведение опытной эксплуатации ИС	tw_8	0,03
Проведение приемочных испытаний ИС	tw_9	0,02

Каждый из перечисленных в табл. 1 типов работ рассчитывается по определённым в рамках методике формулам с учётом поправочных коэффициентов.

Главным преимуществом методики является то, что подсчёт трудозатрат каждого типа работ по созданию ИС основывается на базовой трудоёмкости, которая, в свою очередь, определяется на основе размера ИС. В условиях поставленной задачи данный подход обладает значительным преимуществом, так как функциональные единицы размера ИС могут быть подсчитаны при наличии исходного кода. Недостатком данной методики является то, что значения всех поправочных коэффициентов, используемых в рамках методики, определялись на основе анализа разработанных ИС Московской области, что накладывает ограничения на применение данного метода для любых информационных систем.

Заключение

Проведённый анализ алгоритмических методов оценки стоимости ПО с точки зрения их применения для оценки разработанных программных продуктов показал, что на данный момент не существует универсальной модели, позволяющей получить оценку затрат, основанную исключительно на анализе исходного кода.

Модель COCOMO II предоставляет широкие возможности для оценки размера продукта, в т. ч. с точки зрения наличия в нём повторно используемого и автоматически сгенерированного кода. Однако в рамках поставленной задачи возникает трудность в оценке факторов масштаба и множителей трудоёмкости., т. к. они не могут быть оценены на основе исходного кода, а усреднённые значения данных коэффициентов могут дать значительную погрешность в оценке.

Метод функциональных точек является альтернативным способом определения размера ПО, который может повысить точность оценки при использовании его в других методиках в случаях, когда исходный код программного продукта не отражает трудозатраты на его создание.

Метод PERT не может использоваться в данной задаче как самостоятельная модель, однако в комбинации с другими методиками может помочь улучшить оценку посредством сравнения запланированным и фактических трудозатрат и сроков.

Метод SLIM ограничивает своё применение использованием данных из исторической базы данных организации, разработавшей программный продукт.

Модель SEER-SEM является частью коммерческого программного обеспечения, алгоритмы вычислений которого не раскрыты, в связи с чем модель невозможно использовать в независимых программных продуктах.

Методика оценки, утверждённая постановлением Правительства Московской области, основывается на подсчёт базовой трудоёмкости ПО, которая в свою очередь определяется на основе размера ИС. В условиях поставленной задачи данный подход обладает значительным преимуществом, но используемые в методике поправочные коэффициенты накладывают ограничения на применения данного метода для любых информационных систем.

Задача оценка стоимости разработанного ПО является нетривиальной и не имеет простого решения. Сравнительный анализ алгоритмических методов подчеркивает важность создания гибридного метода, либо принципиально нового подхода в оценке существующих программных продуктов.

Литература

1. *Boehm B.* Software Cost Estimation with COCOMO II. / B. Boehm, C. Abts. – New Jersey : Prentice Hall Press, 2009. – 544 p. – ISBN 978-0-13-702576-3.

2. Function Point Analysis (FPA). – URL: https://ifpug.org/ifpug-standards/fpa (дата обращения: 30.09.2024).

3. *Hamayoon G.* The review of software cost estimation model: SLIM / G. Hamayoon, A. Faqeed S. // International Journal of Advanced Academic Studies. – 2020. – Vol. 2. – P. 511–515.

4. *Алиев Х. Р.* Модель планирования и управления разработкой сложных программных систем на основе комбинированной методики оценки трудозатрат : автореф. дис. ... канд. экон. наук : 08.00.13 / Алиев Хаджимурад Расулович; СПбГУ. – Санкт-Петербург, 2010. – 25 с.

5. SEER for Software (SEER-SEM) – Cost Estimation. – URL: https://galorath.com/cost-estimation/seer-sem-software/ (дата обращения: 05.10.2024).

6. Об утверждении Методики расчета планируемой стоимости работ по созданию, развитию и сопровождению информационных систем Московской области : постановление Правительства МО от 11.11.2022 № 1251/38.

УДК 004.415.2

ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ИЗУЧЕНИЯ ТЕХНИЧЕСКОГО АНГЛИЙСКОГО ЯЗЫКА

С. М. Проняева

Воронежский государственный университет

Аннотация. В статье представлено описание процесса создания веб-приложения для изучения технического английского языка. Рассматриваются основные этапы разработки приложения — проектирование, реализация, тестирование. Проводится анализ требований к приложению, выбираются технологии для разработки, описывается архитектура приложения и реализуется бизнес-логика.
Ключевые слова: веб-приложение, архитектура приложения, ER-модель, база данных, миграция, интерфейс JDBC, внедрение зависимостей, REST-архитектура, API.

Введение

В настоящее время английский язык встречается во многих сферах жизни, и область информационных технологий — не исключение. Так как в русском языке среди IT-терминов встречаются заимствованные слова и синтаксис всех популярных языков программирования написан на английском языке, владение соответствующей терминологией помогает быстрее изучать новые технологии и лучше ориентироваться в языке при написании программ. При изучении английского языка IT-специалисты обычно прибегают к общедоступным ресурсам, таким как онлайн-курсы, лекции на видео-платформах, словари и переводчики и т. д. Данные инструменты, как правило, предназначены для изучения общего языка, а не терминов той или иной области. Поэтому становится актуальным вопрос о создании приложения для изучения технического английского, позволяющего IT-специалистам в короткие сроки освоить необходимую им терминологию.

Статья посвящена проектированию и реализации серверной части веб-приложения для изучения технических терминов английского языка в области информационных технологий.

1. Постановка задачи

1. Спроектировать веб-приложение, предназначенное для изучения технических терминов английского языка в области информационных технологий, обладающее следующей функциональностью:

– регистрация;

– выполнение заданий для усвоения терминологии (выбор перевода слова, вставка слова в предложение, образование фраз с помощью соединения слов, кроссворд);

– сохранение прогресса и отслеживание его в профиле;

– поиск перевода и значения слов во встроенном словаре и формирование на их основе собственного.

2. Реализовать серверную часть приложения.

2. Сравнительный анализ разрабатываемого приложения и существующих решений

Существует несколько приложений-аналогов, например, «Английский для айтишников», «Английский для IT», проект «Разработка веб-приложения для изучения английского языка

IT-специалистами». Последний проект находится на стадии разработки, из-за чего не представляется возможным оценить его преимущества и недостатки.

Обзорная оценка предоставляемых приложениями функций и некоторых других критериев представлена в табл. 1.

Таблица 1

Сравнение приложений-аналогов с предлагаемыми решениями

Приложение / Критерий	«Английский для айтишников»	«Английский для IT»	Разрабатываемый продукт
Наличие интерактива с пользователем	+/–	+/–	+
Выбор перевода	+	+	+
Вставка слова в предложение	+	–	+
Решение кроссворда	–	–	+
Образование фраз	–	–	+
Общий словарь	–	+	+
Словарь пользователя	–	+	+
Отслеживание прогресса	–	+	+
Платное	+	–	–

Таким образом, можно сделать вывод о том, что разрабатываемый продукт обладает рядом преимуществ, среди которых основными являются большой выбор заданий, наличие словаря, а также то, что приложение является полностью бесплатным.

3. Проектирование приложения и реализация его серверной части

Данный раздел посвящен описанию процесса проектирования веб-приложения и реализации его серверной части.

3.1. Средства реализации

В качестве средств реализация были использованы: Figma — приложение для создания UX-дизайна, СУБД PostgreSQL, язык программирования Java, приложение Postman — HTTP-клиент для тестирования API.

3.2. Архитектура приложения

Для данного проекта была выбрана многослойная архитектура. Она работает по методу разделения ответственностей. Программное обеспечение состоит из слоёв (слой представления, слой бизнес-логики, слой передачи данных), которые накладываются один на другой и у каждого из которых есть своя обязанность.

3.3. Проектирование и создание базы данных приложения

Для структурирования данных на этапе логического проектирования строится ER-модель, описывающая то, какие объекты в приложении можно выделить, какими свойствами они обладают и как связаны между собой [1].

В результате, была построена модель данных (рис. 1).

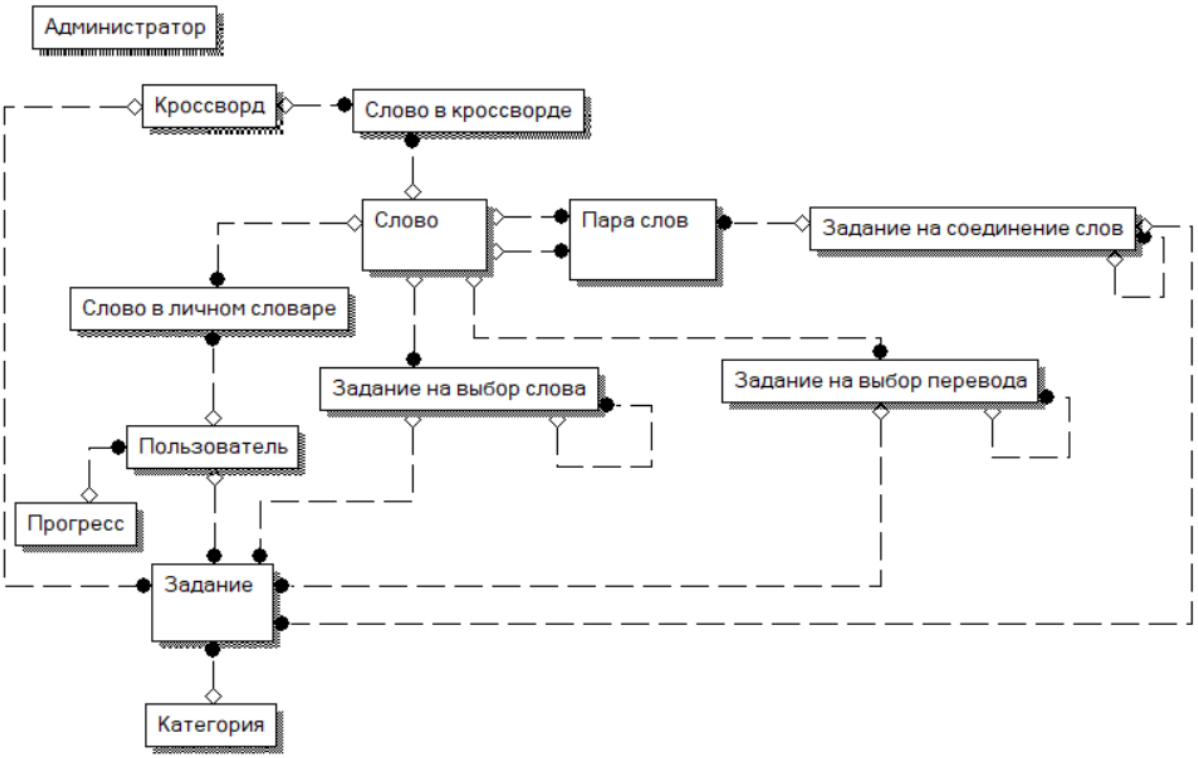


Рис. 1. Модель данных

В данной системе «Пользователь» кроме личных данных (имя, почта, пароль и др.), имеет свой «Прогресс», содержащий информацию о количестве пройденных заданий каждого типа.

Всего существует четыре типа заданий: задание на выбор слова, на выбор перевода, на соединение слов и кроссворд.

«Задание на выбор слова» содержит предложение с пропущенным словом, которое нужно подобрать, и 4 варианта ответа.

«Задание на выбор перевода» содержит слово, необходимое перевести, и 4 варианта ответа.

«Задание на соединение слов» содержит несколько «Пар слов», которые требуется составить из предложенных в задании слов.

Все описанные выше задания содержат номер (ссылку) следующего задания соответствующего типа.

В «Кроссворде» есть список «Слов».

У «Пользователя» есть список выполненных «Заданий», которые могут относиться к той или иной «Категории» (в данном случае под «категорией» понимают определенный раздел информационных технологий, например, «Компьютерная безопасность»).

Кроме того, «Пользователь» имеет свой личный словарь, состоящий из «Слов».

Само же «Слово» имеет основной и дополнительный переводы, значение на английском, транскрипцию и примеры использования в предложении.

Этап физического проектирования предполагает создание описания реализации базы данных. Для этой задачи обычно используют SQL-скрипты, которые создают базу данных один раз при проектировании. Однако в связи с тем, что определить конечную структуру базы данных в начале разработки приложения — задача непростая, целесообразно использовать миграции, благодаря которым проще при необходимости вносить изменения в модель. Одним из инструментов для настройки миграций является Liquibase. Это открытая независимая от базы данных библиотека для отслеживания, управления и применения изменений схемы, поддерживаемая различными СУБД и работающая с такими форматами, как XML, YAML, JSON и SQL.

Для каждой таблицы был создан отдельный файл со скриптом Liquibase.

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
www.liquibase.org/xml/ns/dbchangelog
https://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd">

<changeSet id="create-words-table" author=»postgres">
      <preConditions onFail="WARN">
            <sqlCheck expectedResult="0">SELECT COUNT(*) FROM words</sqlCheck>
      </preConditions>
      <createTable tableName="words">
            <column autoIncrement="true" name="word_id" type="serial">
                  <constraints primaryKey="true" nullable="false"/>
            </column>
            <column name="word" type="text">
                  <constraints nullable="false» unique="true"/>
            </column>
            <column name="main_translation" type="text">
                  <constraints nullable="false"/>
            </column>
            <column name="additional_translation" type="text">
            </column>
            <column name="meaning" type="text">
                  <constraints nullable="false"/>
            </column>
            <column name="examples" type="text">
            </column>
            <column name="transcription" type="text">
                  <constraints nullable="false" unique="true"/>
            </column>
      </createTable>
</changeSet>
</databaseChangeLog>
```

Затем в корневой файл с настройками базы данных db.changelog-master.xml были импортированы миграции.

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
https://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd">
      <include file="changeset/create-changeset-words-table.xml"
               relativeToChangelogFile="true"/>
</databaseChangeLog>
```

3.4. Реализация слоя доступа к данным

В первую очередь для последующей реализации слоя доступа к данным для каждой таблицы базы данных был создан соответствующий ей класс с теми же столбцами (полями) [2].

Далее был задействован центральный класс JdbcTemplate прикладного интерфейса JDBC (Java Database Connectivity API), предназначенного для взаимодействия Java-приложений с

различными СУБД [3]. JdbcTemplate выполняет SQL-запросы, отвечает за высвобождение ресурсов, открывает и закрывает соединение с базой данных.

В данном приложении был создан IRepository<T> — интерфейс, описывающий методы обращения к данным, присущие всем объектам системы (например, поиск всех записей таблицы или обновление какой-либо записи).

Для каждого объекта системы был написан класс, имплементирующий этот интерфейс и привносящий некоторые специфические методы, присущие только тому объекту, для которого он написан.

Так как выполненный запрос возвращает данные в виде объекта типа ResultSet, необходим метод, конвертирующий полученные строки в объекты предметной области. С целью решения данной задачи для каждой модели данных системы был создан класс, имплементирующий интерфейс RowMapper<T>. Класс содержит единственный метод, который переводит запись базы данных в объект модели данных:

```
public class WordRowMapper implements RowMapper<Word> {

    @Override
    public Word mapRow(ResultSet rs, int rowNum) throws SQLException {

        Word word = new Word();
        word.setId(rs.getLong("word_id"));
        word.setWord(rs.getString("word"));
        word.setMainTranslation(rs.getString("main_translation"));
        word.setAdditionalTranslation(rs.getString("additional_translation"));
        word.setMeaning(rs.getString("meaning"));
        word.setExamples(rs.getString("examples"));
        word.setTranscription(rs.getString("transcription"));
        return word;
    }
}
```

3.5. Реализация слоя бизнес-логики

В данном блоке расположена практически вся бизнес-логика приложения. Для каждого класса модели был создан интерфейс сервисов и его реализация, которая использует объект Repository-класса, описанного ранее и отвечающего за взаимодействие с базой данных. Внедрение зависимостей [4] осуществляется с помощью аннотации @Autowired:

```
@Service
public class WordService {
    private static final Logger logger = LogManager
                          .getLogger(WordService.class);
    @Autowired
    private WordRepository repository;

    public Word findWordById(Long id) {
        logger.info("Вызвана функция поиска слова по id");
        return repository.findById(id).orElseThrow(
                    () -> new ResourceAccessException("Word not found"));
    }
    /* реализация других методов */
}
```

3.6. Реализация слоя контроллеров

Концепция REST-архитектуры заключается в том, что сервис получает HTTP-запросы от клиента, обрабатывает их и отправляет обратно ответы. Для этого используются REST-контроллеры — специальные классы для обработки клиентских запросов, которые обращаются к методам классов сервисов. Каждый контроллер возвращает объект типа ResponseEntity — HTTP-ответ, включающий статус, тело и заголовки. При этом в качестве тела ответа может выступать любой тип.

```
@Controller
@RequestMapping("/words")
public class WordController {
    private static final Logger logger = LogManager
                                .getLogger(WordController.class);
    @Autowired
    private WordService service;

    @GetMapping("/translation")
    public ResponseEntity<List<Word>> findByTranslation(
@RequestParam("translation") String translation) {
        logger.info("End point for word: GET (by translation)");
        var words = service.findByTranslation(translation);
        return new ResponseEntity<>(words, HttpStatus.OK);
    }
    /* реализация других методов */
}
```

Заключение

Таким образом, в ходе данной работы:

1. Было спроектировано веб-приложение, предназначенное для изучения технических терминов английского языка в области информационных технологий, обладающее следующей функциональностью:

– регистрация;

– выполнение заданий для усвоения терминологии (выбор перевода слова, вставка слова в предложение, образование фраз с помощью соединения слов, кроссворд);

– сохранение прогресса и отслеживание его в профиле;

– поиск перевода и значения слов во встроенном словаре и формирование на их основе собственного.

2. Была реализована серверная часть приложения.

Литература

1. *Коннолли Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. – 3-е изд. : Пер. с англ. – Москва : Издательский дом «Вильямс», 2003. – 1440 с.

2. *Фаулер М.* Архитектура корпоративных программных приложений / М. Фаулер. : Пер. с англ. – Москва : Издательский дом «Вильямс», 2006. – 544 с.

3. *Хорстманн Кей С.* Java. Библиотека профессионала. В 2 т. Т. 2. Расширенные средства / Кей С. Хорстманн, Г. Корнелл. – 9-е изд. : Пер. с англ. – Москва : Издательский дом «Вильямс», 2014. – 1008 с.

4. *Нейгард М.* Release it! Проектирование и дизайн ПО для тех, кому не всё равно / М. Нейгард. – Санкт-Петербург : Питер, 2016. – 320 с.

УДК 004.9

ПРОГРАММИРУЕМЫЙ АППАРАТНЫЙ МИКШЕР ГРОМКОСТИ С ИСПОЛЬЗОВАНИЕМ ESP32 И .NET

Е. В. Проценко, И. Е. Воронина

Воронежский государственный университет

Аннотация. Рассматривается задача создания физического микшера громкости на базе микроконтроллера ESP32 и программы на .NET для управления звуком на уровне ОС. Микшер позволяет изменять громкость различных аудиосессий и уровня мастер-громкости. Применяются OLED дисплеи для визуализации громкости в реальном времени, а также отслеживание подключений через USB для автоматического обнаружения устройства. Разработка может быть полезна для упрощения и автоматизации управления звуком на компьютере.
Ключевые слова: физический микшер, громкость, ESP32, .NET, аудиосессии.

Введение

Физический микшер громкости — это устройство, которое позволяет управлять уровнем громкости разных источников звука, объединяя их в одном компактном устройстве. В отличие от программных решений, физический микшер обеспечивает интуитивное и быстрое управление.

Микшер особенно полезен для людей, работающих с мультимедийным контентом (стримеров, музыкантов, звукорежиссеров, геймеров и др.), поскольку позволяет легко и точно регулировать баланс звука.

Однако, использование стандартного микшера часто порождает ряд неудобств, например, пользователи сталкиваются с трудностями при настройке громкости нескольких источников звука, таких как программы для общения, медиаплееры, игры и т.д., поскольку переключение между ними и настройка громкости каждой программы через системные настройки требует времени и неудобна. Таким образом, актуальна задача создания собственного микшера, реализующим более удобное управление звуком, которое обходит ограничения, связанные с использованием стандартных программных регуляторов громкости.

1. Особенности реализации

Рассмотрим задачу создания компактного физического устройства для управления громкостью на уровне операционной системы, которое предоставляет интуитивный и простой способ контроля звука на компьютере. Такое решение должно поддерживать настройку громкости, как для каждой аудиосессии, так и для общей, так называемой мастер-громкости.

Рассмотрим требования к физическому микшеру громкости:

1. Аппаратное управление:

• использование OLED дисплеев: для визуализации громкости каждого аудиоканала в реальном времени. Это позволит пользователю быстро оценить уровень громкости каждого источника звука;

• набор физических элементов управления (фейдеры): для удобной настройки громкости с интуитивной тактильной обратной связью, которая позволяет менять настройки быстро и не отвлекаясь от других задач.

2. Программная часть:

• программа на .NET: взаимодействует с операционной системой и обрабатывает команды с ESP32 для управления громкостью;

• программное обеспечение (микроконтроллера): основной цикл, выполняющий чтение данных слайдеров и обновление значений на дисплеях. Код также включает функции для обработки графики на дисплеях и вывода числовых значений.

Пользователь получит возможность:

• управлять громкостью отдельных программ или устройств, и общей громкостью с помощью физического микшера;

• быстро подключить устройство без дополнительных настроек;

• получить доступ к удобной визуализации уровня громкости в реальном времени.

Для решения задачи предлагается использовать микроконтроллер ESP32. Основание: низкая стоимость, возможность, достаточно простым способом, осуществить перепрошивку. Такое решение позволит объединить аппаратное и программное управление и реализовать необходимую функциональность.

2. Аппаратные компоненты

На рис. 1 приведена простая наглядная схема подключения компонентов к модулю микроконтроллера ESP32.

1. Модуль микроконтроллера ESP32 — используется для обработки данных с потенциометров (фейдеров) и вывода информации на дисплеи. Обеспечивает связь с ПК.

2. Модуль расширения интерфейса I2C — плата для коммутации и подключения нескольких I2C-устройств.

3. Монохромный OLED-дисплей 0,96 дюйма (4 шт.) — I2C-дисплей с разрешением 128x64 для вывода текстовой и графической информации.

4. Скользящий потенциометр (фейдер) 10K (4 шт.) — линейный потенциометр, используемый для регулировки громкости.

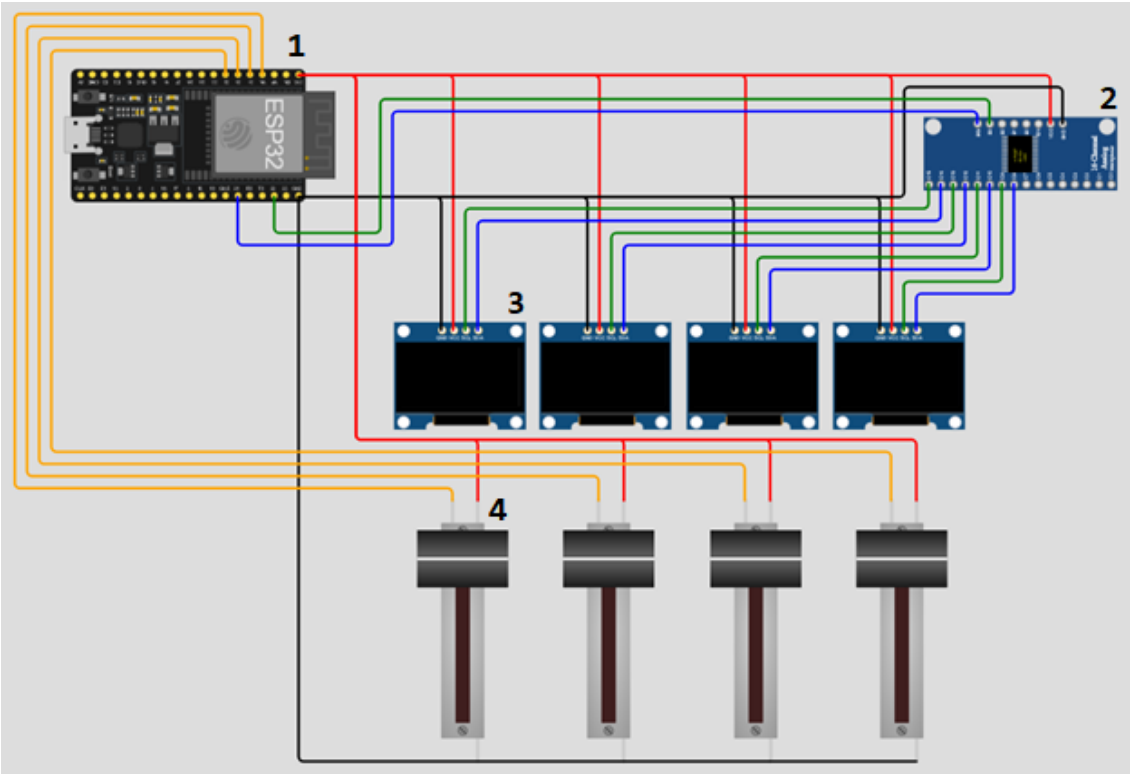


Рис. 1. Схема подключения компонентов

3. Реализация для микроконтроллера ESP32

В каждой прошивке (скетче) для микроконтроллера ESP32 обязательно должны быть определены две функции: setup() и loop(). Функция setup() вызывается однократно при запуске устройства. Функция loop(), напротив, выполняется в бесконечном цикле и отвечает за основную логику работы программы. В функции setup() выполняются следующие действия:

1. Инициализация последовательного порта и шины I2C:

```
Serial.begin(115200);
Wire.begin();
```

2. Инициализация OLED-дисплеев, с помощью библиотеки Adafruit_SSD1306 [2], по I2C с указанием канала и адреса экрана:

```
setupDisplayOled(oled1, OLED_CHANNEL_1, OLED_ADDRESS_C);

void setupDisplayOled(Adafruit_SSD1306 &pDisplay, int8_t pChannel, uint8_t
pScreenAddress)
{
    selectI2CChannel(pChannel);
    if (!pDisplay.begin(SSD1306_SWITCHCAPVCC, pScreenAddress)) {
        Serial.println(F(«SSD1306 allocation failed»));
        for (;;); // Бесконечный цикл при ошибке
    }
    pDisplay.fillScreen(SSD1306_BLACK); // Очистка экрана
}
```

В функции loop() выполняются следующие действия:

1. Обновляем массив, хранящий значения фейдеров

`previousSliderValues` — массив хранящий значения потенциометров;
`NUM_SLIDERS` — число потенциометров;

```
updateSliderValues(previousSliderValues, NUM_SLIDERS);
```

2. Отправляем данные на ПК и выводим значение на дисплей:

```
sendSliderValues(previousSliderValues, NUM_SLIDERS);
updateNumber(oled1, OLED_CHANNEL_1, previousSliderValues[0], 5, 3,
globalHeaderHeight);
```

4. Интеграция с .NET приложением

Код использует библиотеку NAudio для доступа к аудиосессиям Windows, управляя громкостью через MMDevice и AudioSessionManager [1].

Обработчик данных из последовательного порта: данные с ESP32 принимаются на последовательном порте, парсятся и преобразуются для установки уровня громкости на различных аудиосессиях.

На рис. 2 представлен интерфейс пользователя, пользователю выводятся все приложения, у которых активна аудиосессия, для каждого приложения есть трекбар, для регулировки громкости, а также кнопка, которая заглушает звук. Приложение также включает автоматическое обнаружение USB-устройства для быстрого подключения микшера.

Таким образом, пользователю в простой, удобной форме доступно управление громкостью приложений и выбор COM-порта, к которому подключено устройство.

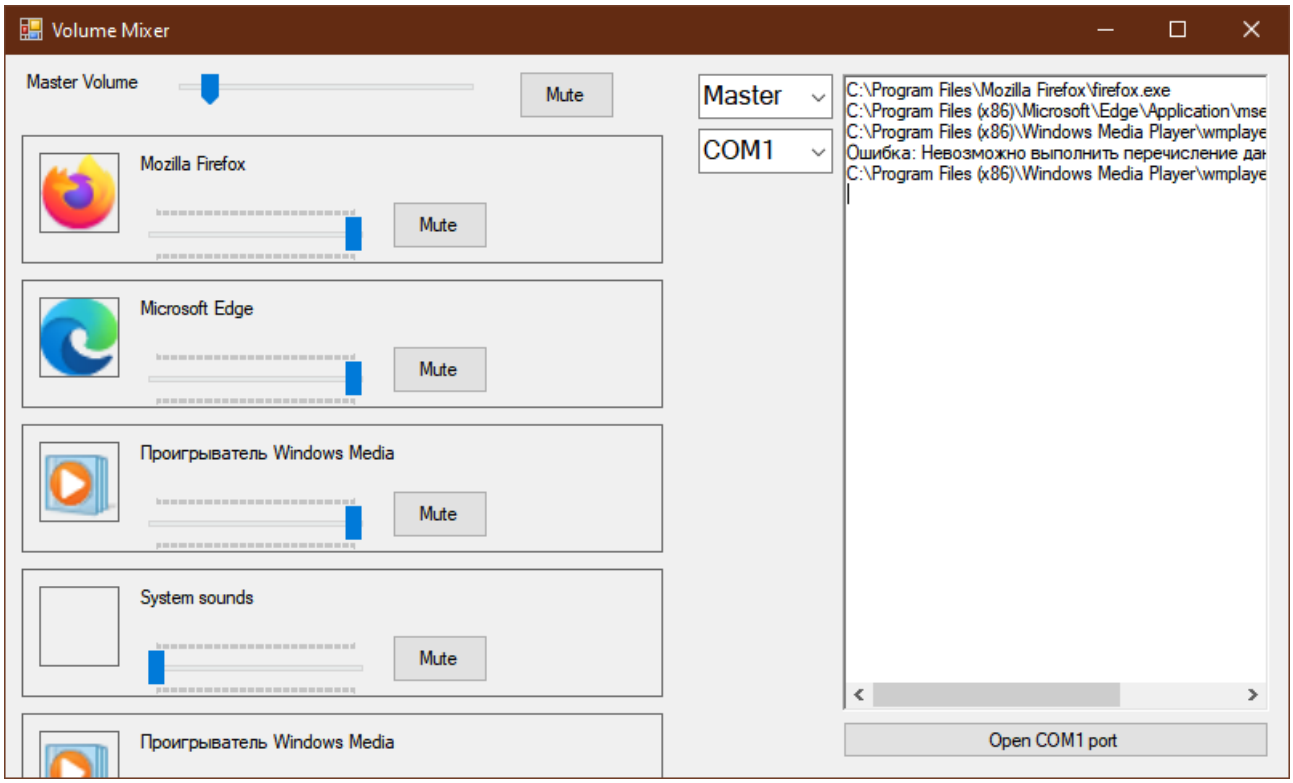


Рис. 2. Интерфейс пользователя

Заключение

Разработанный микшер громкости на базе микроконтроллера ESP32 с интеграцией в операционную систему через программное обеспечение на платформе .NET позволяет простым и удобным способом управлять уровнем громкости, позволяя регулировать громкость как общей аудиосистемы, так и отдельных приложений. Использование OLED дисплеев обеспечивает наглядную визуализацию громкости в реальном времени.

Перспективы дальнейшего развития устройства включают добавление новых функций, таких как настраиваемые профили громкости, улучшенная совместимость с различными операционными системами и расширение функциональности программного обеспечения для еще более гибкого управления аудиосистемой.

Литература

1. Аудио и MIDI-библиотека для .NET : NAudio. – URL: https://github.com/naudio/NAudio (дата обращения: 31.10.2024).

2. Библиотека для монохромных OLED-дисплеев на базе драйвера SSD1306 : Adafruit_SSD1306. – URL: https://github.com/adafruit/Adafruit_SSD1306 (дата обращения: 31.10.2024).

УДК 004.92

АНАЛИЗ ЗАДАЧИ ИНТЕГРАЦИИ ДВУМЕРНЫХ ВЕКТОРНЫХ ОБЪЕКТОВ В ТРЁХМЕРНУЮ СЦЕНУ

И. Б. Рахимов, О. Г. Корольков

Воронежский государственный университет

Аннотация. В работе рассматривается задача корректной интеграции двумерных векторных объектов в трёхмерную сцену, что представляет особый интерес в условиях активного использования комбинированной графики в современной мультипликации. Приводится анализ основных трудностей, возникающих при визуализации векторной графики с учётом глубины и перспективных искажений. Описаны два подхода к решению данной задачи: использование полигональных моделей с текстурированием и применение буфера глубины для векторных объектов. Проведён сравнительный анализ преимуществ и недостатков каждого метода, а также обсуждены минимальные требования к графическим системам. Сделаны выводы о применимости подходов в зависимости от требований к визуализации и производительности.

Ключевые слова: векторная графика, комбинированная графика, трёхмерная графика, визуализация трёхмерных сцен, текстурирование, тест глубины, мультипликация.

Введение

Классическая двумерная анимация, в которой каждый кадр создаётся художниками вручную, — достаточно трудоёмкая задача, которая ушла в прошлое из-за своей дороговизны, длительности производства и развития более быстрых и дешёвых инструментов трёхмерной компьютерной графики. Даже такие гигантские студии, как «DreamWorks Animation» и «Walt Disney Animation Studios», которые, казалось бы, имеют неограниченные запасы ресурсов для производства мультфильмов, перешли на использование трёхмерной графики. Последний раз они выпускали фильмы в двумерной технике в 2003 и 2011 году соответственно.

В настоящее время двумерная анимация всё же применяется, но для её создания используются более дешёвые инструменты и техники, основным из которых является векторная графика. Векторная графика имеет множество преимуществ: одним из главных является гибкость: векторные объекты могут легко изменены — трансформированы, масштабированы и т. д. — без потери качества. Это позволяет вносить быстрые и эффективные изменения на любом этапе производства, сокращая время и ресурсы. Также эти трансформации будут более плавными в сравнении с растровой графикой.

Но векторная графика имеет и недостатки, основной из которых — проблемы с вращением камеры вокруг произвольной оси, кроме направленной от экрана к зрителю (такой поворот называют «креном»). Пытаясь реализовать любой простейший поворот, который не является креном (рысканье, тангаж или любые комбинации этих трёх поворотов), например, облёт камеры вокруг персонажа, художники возвращаются к классической покадровой анимации, так как векторная графика не позволяет аффинными преобразованиями создавать такие проекции камеры.

Также векторная графика не учитывает глубину сцены: очень часто векторные объекты визуализируются с использованием алгоритма художника, когда новые объекты накладываются поверх старых. В таких случаях для упрощения работы художников используется подход, объединяющий визуализацию трёхмерных и векторных плоских объектов, когда покадрово анимировать необходимо только некоторые двумерные объекты, трёхмерные же проецируются для произвольной позиции камеры. В последнее время такой подход используется во многих

анимационных фильмах и мультфильмах. Например, в трёхмерной сцене могут использоваться двумерные эффекты (такие, как искры, взрывы и т. д.), а также двумерные персонажи.

В контексте сказанного выше интерес представляет задача разработки методов и алгоритмов интеграции векторных объектов в трёхмерную сцену, сохраняющих преимущества векторной графики. Важным требованием к таким алгоритмам являются корректный тест глубины для векторных плоских и трёхмерных объектов любой формы, а также корректная перспективная и ортогональная проекции.

В работе предлагается два подхода к решению поставленной задачи интеграции векторных объектов в трёхмерную сцену, проводится анализ их достоинств и недостатков, а также анализ минимальных требований к графическим системам векторной и трёхмерной визуализации.

1. Подход, основанный на текстурировании полигональных моделей и использовании теста глубины трёхмерной сцены

В основе первого подхода лежит приём, использующий наложение заранее подготовленных векторных текстур на полигональные модели. При этом во время этой подготовки все искажения камеры игнорируются, а векторная текстура, соответственно, визуализируется без искажений и перспективы. Искажения же камеры учитываются на следующем этапе, при визуализации полученных текстурированных моделей трёхмерным полигональным рендером. Таким образом, данный подход позволяет достаточно легко решить проблему глубины и проекции камеры за счёт трёхмерной визуализации.

Этапы предложенного подхода представлены на рис. 1. Векторный рендер представлен как «чёрный ящик», трёхмерный полигональный рендер изображён, для наглядности, с использованием стандартных шагов графического конвейера OpenGL (впрочем, допустимо применение любого трёхмерного полигонального рендера, для которого выполняется единственное требование к полигональной визуализации — поддержка текстурирования полигональных объектов).

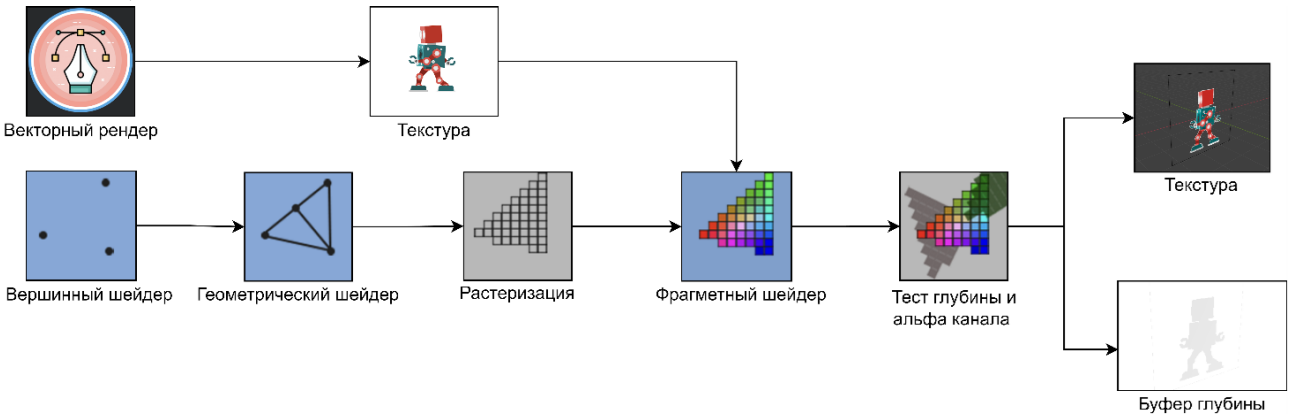


Рис. 1. Конвейер трёхмерного полигонального и векторного рендеринга с использованием текстурирования плоскостей

Заметим, что подход, основанный на текстурировании полигональных моделей, имеет несколько недостатков, которые связаны со сложностью выбора как размера (в пикселях) выходного буфера векторного рендеринга, так и размера (в единицах масштаба трёхмерной сцены) полигональной плоскости, на которую накладывается текстура.

Необходимость изменения размера выходного буфера векторного рендеринга обусловлена условием сохранения качества изображения при масштабировании и других трансформациях полигональной модели, на которую накладывается векторная текстура. Следовательно, процесс подготовки текстуры должен учитывать такие факторы, как размер полигональной

модели, а также её удалённость от камеры. Однако решение данной задачи весьма неочевидно. Во-первых, постоянное выделение памяти для буфера изображения — весьма трудоёмкий процесс [4]. Во-вторых, подбор алгоритма, определяющего размер векторного буфера, — достаточно сложная и неоднозначная задача, которая должна учитывать большое количество факторов, включая ограниченность оперативной памяти или видеопамяти.

Также существует проблема сложности выбора размера полигональной плоскости в координатах сцены. Для векторного объекта могут быть применены различные эффекты или модификаторы, которые потребуют пересчёта ограничивающего прямоугольника и создания новой полигональной плоскости подходящего размера (или увеличение текущей) для визуализации всей видимой части векторного изображения. Например, увеличение толщины векторных линий (рис. 2) или применение аффинного преобразования поворота не к векторной плоскости, а к векторной текстуре, изменяет ограничивающий объём модели — и для сложных векторных объектов расчёт может занимать значительное время.

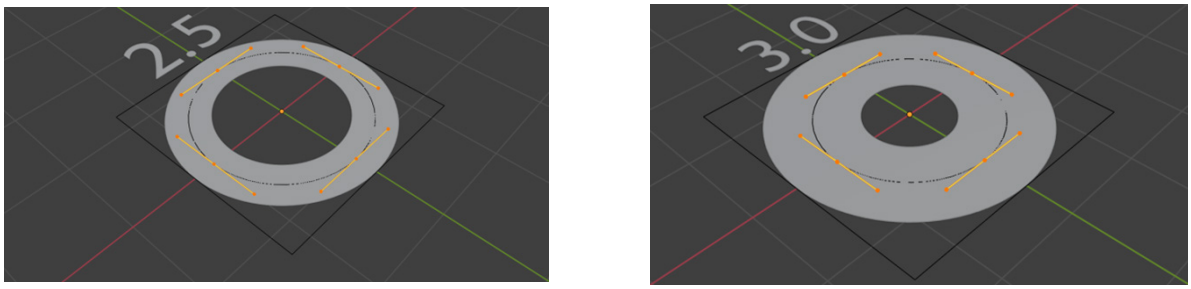


Рис. 2. Пример, демонстрирующий изменение ограничивающего прямоугольника при изменении ширины векторных линий.
Слева — векторная окружность с толщиной линии 0.25 и ограничивающим квадратом со стороной 2.5.
Справа — векторная окружность с толщиной линии 0.5 и ограничивающим квадратом со стороной 3.0

2. Подход, основанный на вычислении буфера глубины для векторных объектов и его применении для теста глубины

Как было сказано в предыдущем разделе статьи, важным недостатком первого подхода являются трудности, связанные с постоянными изменениями размеров буферов, используемых для хранения векторных текстур, которые могут быть вызваны, например, изменением положения камеры. Данную проблему позволяет решить использование единого буфера постоянного размера (в пикселях), равного размеру кадра при итоговой визуализации. Данный подход вполне корректен, поскольку при создании любых фильмов размер кадра всегда постоянен.

Использование постоянного буфера, с одной стороны, решает проблемы, имевшиеся в первом подходе: исчезает необходимость дополнительных выделений памяти, а также необходимость вычислять ограничивающий объём векторного объекта и создание полигональной плоскости. С другой стороны, возникает необходимость вычисления перспективы и глубины векторных объектов.

Данную задачу предлагается решать в два этапа:

- предварительная подготовка цветного векторного изображения с корректными трёхмерными преобразованиями, но без учёта глубины;
- формирование буфера глубины для векторных объектов и его применение для теста глубины.

Предварительная подготовка векторного изображения не вызывает затруднений: большинство современных движков для визуализации векторных изображений поддерживают любые трёхмерные трансформации, но даже если такой возможности нет, мы можем эмулировать их

с помощью добавления третьей координаты, равной нулю, применения к новым координатам трансформаций модели, вида и проекции, и, наконец, отбрасывания третей координаты.

На данном этапе мы имеем три буфера с корректными трёхмерными преобразованиями:

- буфер с итогом визуализации векторных объектов;
- буфер с итогом визуализации трёхмерной сцены;
- буфер глубины трёхмерной сцены.

Для корректной визуализации векторных объектов в трёхмерной сцене необходимо вычисление буфера глубины для векторных объектов. В этом случае задача вычисления буфера глубины и визуализации решается при помощи следующего алгоритма:

1. На вход алгоритма подаются:

- результат визуализации трёхмерной сцены (буфер с цветом и буфер глубины);
- положение векторной плоскости в пространстве сцены, заданное, например, точкой и нормалью;
- подготовленное цветное векторное изображение, учитывающее перспективные искажения камеры.

2. Для каждого непрозрачного пикселя векторного изображения вычисляется глубина как глубина плоскости, занимающей всё изображение. Её без труда можно вычислить как пересечение луча с началом в позиции камеры в трёхмерной сцене и направлением от камеры до текущего пикселя и векторной плоскости [5].

3. Вычисленная на втором шаге глубина сравнивается с глубиной трёхмерного изображения и, в зависимости от неё, определяется цвет итогового пикселя.

Графический конвейер, соответствующий данному алгоритму, представлен на рис. 4.

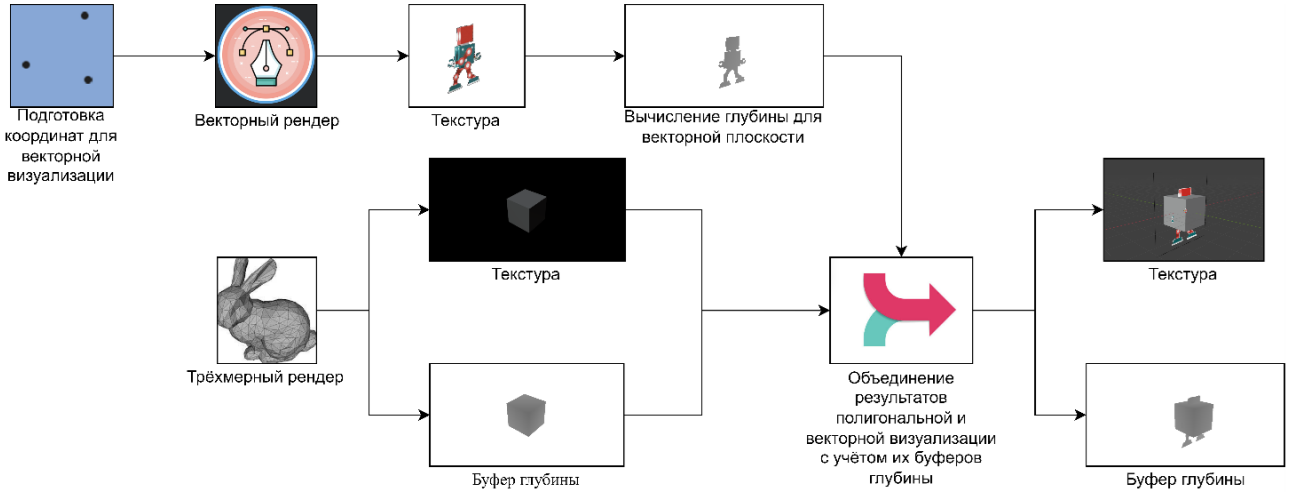


Рис. 4. Конвейер трёхмерного и векторного рендеринга, основанного на вычислении буфера глубины для векторных объектов и его применении для теста глубины

3. Анализ предложенных подходов

Перед анализом предложенных подходов кратко перечислим их достоинства и недостатки. К достоинствам первого подхода относятся:

- простота реализации: зачастую трёхмерная визуализация уже поддерживает текстурирование, и использовать для этого дополнительно созданную текстуру не составит труда;
- скорость визуализации: текстурирование обычно осуществляется через графический процессор, который заточен под быстрое решение данной задачи.

К недостаткам первого подхода относятся:

- дополнительные требования для трёхмерной визуализации: поддержка рендера полигональных моделей и их текстурирования;

• неочевидный размер выходного буфера векторного рендеринга в пикселях;

• неочевидный размер векторной плоскости, на которую накладывается текстура в координатах сцены;

• вероятные дополнительные аллокации при устранении двух предыдущих недостатков.

Достоинства второго подхода:

• отсутствие требований для видов визуализации;

• отсутствие дополнительных аллокаций за счёт одного размера векторного буфера;

Недостатки второго подхода:

• дополнительные операции над векторными координатами, в случае, если векторный рендер не поддерживает трёхмерные преобразования;

• поскольку описанный алгоритм вычисления буфера глубины применяется к каждому векторному объекту отдельно, это замедляет скорость визуализации: каждая операция отрисовки является ресурсоёмкой и значительно увеличивает нагрузку на процессор.

При анализе двух предложенных подходов можно отметить, что первый отличается простотой реализации и высокой скоростью визуализации благодаря использованию возможностей графического процессора. Однако он имеет существенные ограничения, связанные с дополнительными требованиями для трёхмерной визуализации, сложностями в определении размеров выходных данных и возможными дополнительными аллокациями.

Второй подход, напротив, лишён этих ограничений, так как не предъявляет специфических требований к визуализации и не требует дополнительных аллокаций. Однако он требует дополнительных операций с векторными координатами и отличается высокой затратностью ресурсов, что может существенно замедлять процесс визуализации, особенно при большом числе вызовов операции отрисовки.

Таким образом, выбор подхода зависит от конкретных задач: первый подходит для случаев, где важна скорость и простота, а второй — для универсальности и гибкости в работе с векторной графикой.

Заключение

В рамках работы были предложены подходы для решения задачи визуализации векторных объектов в трёхмерной сцене, сохраняющие преимущества векторной графики, такие как гибкость трансформаций и отсутствие потерь качества при масштабировании, а также проведён их анализ.

Комбинированное использование двумерных и трёхмерных элементов в сценах значительно упрощает процесс анимации и предоставляет новые возможности для аниматоров. Этот подход уже успешно применяется в современных мультфильмах, и его развитие открывает перспективы для дальнейшего совершенствования анимационных технологий, объединяющих лучшие стороны двумерной и трёхмерной графики.

Литература

1. *Watt M.* Multithreading for Visual Effects / M. Watt, E. Coumans, G. ElKoura, R. Henderson, M. Kraemer, J. Lait, J. Reinders. – CRC Press, 2014. – 255 p.

2. *Engel K.* Real-Time Volume Graphics / K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, D. Weiskopf. – Taylor & Francis, 2006. – 497 p.

3. *Pharr M.* Physically Based Rendering: From Theory to Implementation / M. Pharr, W. Jakob, G. Humphreys. – The MIT Press, 2023.

4. Gregory J. Game Engine Architecture / J. Gregory. – CRC Press, 2017. – 1052 p.

5. *Laszlo M. J.* Computational Geometry and Computer Graphics in C++ / M. J. Laszlo. – Prentice Hall, 1996. – 266 p.

УДК 004.056.53

ПРИМЕНЕНИЕ TOKEN HANDLER PATTERN ДЛЯ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ И УПРАВЛЕНИЯ ТОКЕНАМИ В SPA

Д. А. Рудаков

Воронежский государственный университет

Аннотация. Статья посвящена анализу и применению Token Handler Pattern для обеспечения безопасности взаимодействия между клиентом и сервером в веб-приложениях. В статье рассматриваются основные проблемы, связанные с безопасностью хранения и использования токенов (access и refresh токены) в современных SPA (Single Page Application). Предложенный подход позволяет централизованно управлять токенами, минимизируя их хранение на стороне клиента. Статья анализирует архитектуру Token Handler, её преимущества, а также рассматривает взаимодействие с внешними сервисами аутентификации.
Ключевые слова: веб-приложения, безопасность веб-приложений, Token Handler Pattern, Single Page Application, авторизация, OAuth2, JWT, Identity Provider, HTTP-only cookies, Proof of Key for Code Exchange.

Введение

На сегодняшний день SPA (Single Page Applications) стали стандартом для разработки современных веб-приложений. Они обеспечивают высокую производительность, интерактивность и плавность работы за счёт загрузки единственной загрузки HTML страницы и динамического обновления её содержимого. Основная логика приложения исполняется на стороне клиента, а данные запрашиваются с сервера через API.

Для защиты взаимодействия между SPA и серверной частью чаще всего применяются токены, которые используются для проверки прав доступа к ресурсам. Наиболее популярными стандартами для управления такими токенами являются OAuth2, OpenID Connect и JSON Web Token (JWT). На рис. 1 изображена схема взаимодействия при использовании JWT токенов для микросервисной архитектуры.

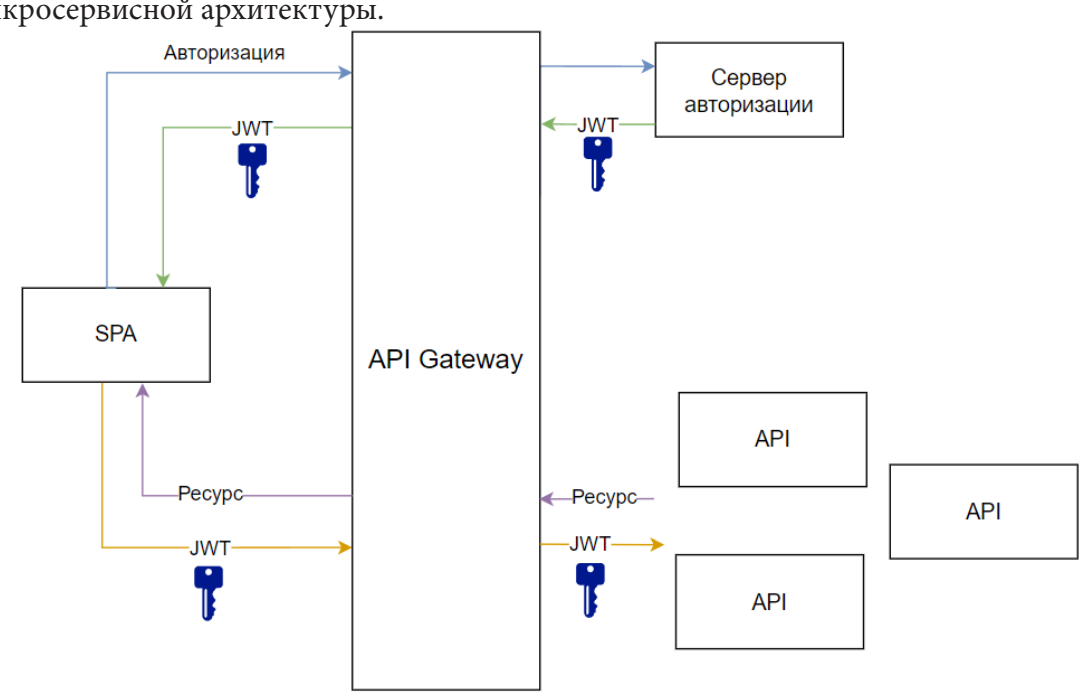


Рис. 1. Схема взаимодействия SPA при использовании access токена

Однако, такая схема взаимодействия имеет слабые места. Основной проблемой является то, что SPA работает в небезопасной среде — браузере пользователя. Помимо access токена, в браузере часто хранится refresh токен, который необходим для обновления access токен после истечения срока его действия. Это делает приложение уязвимым для XSS-атак (Cross-Site Scripting). Злоумышленник, внедрив вредоносный код, может получить доступ к обоим токенам, что позволит ему выполнять любые действия от имени пользователя.

Даже тщательная защита от XSS-атак не гарантирует полной безопасности, поскольку сторонние библиотеки или зависимости могут содержать уязвимости. В качестве альтернативы вышеописанному подходу может быть предложен Token Handler Pattern.

1. Token Handler Pattern

Token Handler Pattern обеспечивает централизованное управление токенами, исключая необходимость обработки токенов непосредственно в SPA. Общая схема взаимодействия для этого подхода представлена на рис. 2.

Схема отображает лишь ключевые аспекты взаимодействия между компонентами, подробные детали опущены для упрощения дальнейшего анализа. Важно отметить, что в данной архитектуре SPA больше не хранит JWT токены. Вместо этого после успешной авторизации пользователю в браузере устанавливается HTTP-only, SameSite=strict куки. Такой подход исключает возможность доступа к токенам через JavaScript в контексте SPA, значительно повышая защиту от XSS-атак.

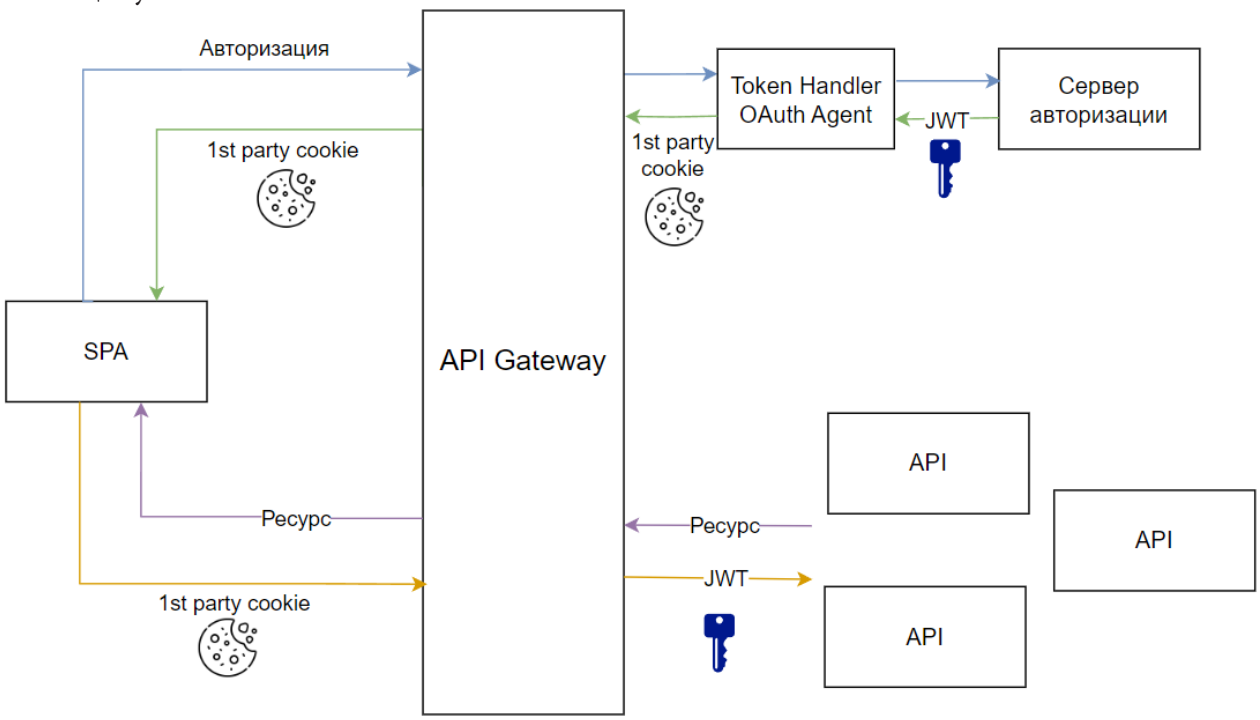


Рис. 2. Общая схема взаимодействия Token Handler Pattern

Token Handler OAuth Agent работает здесь, как BFF компонент (Backend for Front-end) — компонент, который является прокси-сервисом для клиентского приложения, он занимается отправкой запросов за клиента и преобразует ответ в вид, понятный для фронтенда.

Таким образом, вместо того чтобы SPA напрямую отправляло запрос на сервер авторизации для входа или выполнения других операций, запросы направляются в сервис Token Handler OAuth Agent, который обрабатывает их, добавляет необходимую информацию о пользователе, если это необходимо, и уже затем перенаправляет запросы на сервер авторизации.

Основной API, предоставляемый сервисом Token Handler OAuth Agent, включает следующие эндпоинты:

• POST /login/start — инициирует процесс авторизации, устанавливая временный куки для SPA;

• POST /login/end — завершает процесс авторизации, устанавливая в браузере зашифрованный HTTP-only куки для пользователя;

• POST /refresh — обновляет access токен;

• POST /logout — удаляет куки и завершает сессию на сервере авторизации.

Следует отметить, что процесс авторизации реализован в два этапа. Это позволяет после ввода учетных данных на стороне сервера авторизации корректно вернуть зашифрованный JWT токены в виде зашифрованного HTTP-only куки, который устанавливается в браузере пользователя. Временные куки, созданный на первом этапе, связывает пользователя с началом его сессии и обеспечивает дальнейшую безопасную обработку запросов.

Рассмотрим данный процесс с конкретной реализацией, где в качестве сервера авторизации будет выступать Keycloak. Диаграмма авторизации представлена на рис. 3.

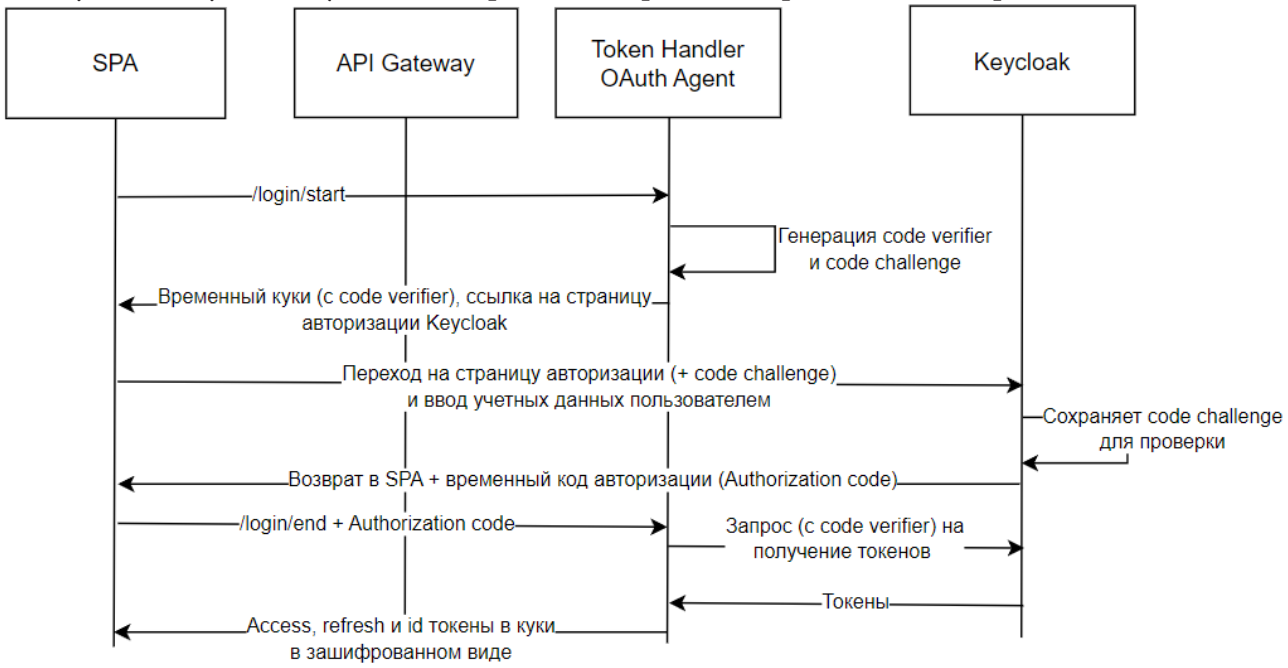


Рис. 3. Диаграмма авторизации

Следует отметить, что представленная диаграмма соответствует механизму PKCE (Proof of Key for Code Exchange), который является расширением Authorization Code Flow. Основная задача PKCE — защита от атак, таких как CSRF (Cross-Site Request Forgery) и Code Injection. PKCE добавляет дополнительный уровень безопасности за счёт использования уникального кода подтверждения (code verifier) и его хэша (code challenge). Этот код генерируется OAuth Agent, и проверяется Keycloak на этапе обмена временного кода авторизации на токены. Таким образом, даже если злоумышленник перехватит временный код авторизации, он не сможет использовать его без уникального кода подтверждения, известного только OAuth Agent. При этом данный подход является всё ещё stateless, т. к. code verifier хранится во временных куки пользователя.

После успешного получения токенов процесс доступа к защищённым ресурсам в Token Handler Pattern мало чем отличается от стандартной схемы использования access токена. Основное различие заключается в том, что обработка и использование токенов сосредоточены на стороне OAuth Agent, минимизируя риски, связанные с их хранением и передачей в клиентском приложении.

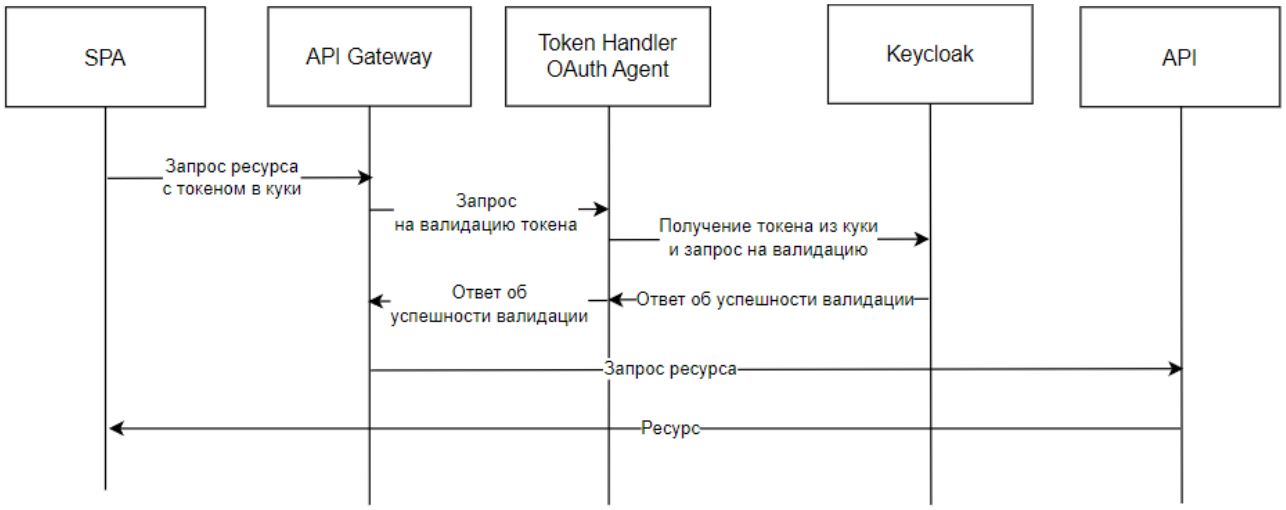


Рис. 4. Диаграмма получения защищенного ресурса

Заключение

Хранение access и refresh токенов в SPA представляет значительную угрозу безопасности. Token Handler Pattern решает эту проблему, убирая токены из SPA и передавая управление ими в отдельный сервис Token Handler OAuth Agent. Этот подход делает взаимодействие между клиентом и сервером безопасным, сохраняя удобство для пользователей и надёжно защищая их данные.

Литература

1. RFC 6749. The OAuth 2.0 Authorization Framework. – 2012. – URL: https://datatracker.ietf.org/doc/rfc6749/ (дата обращения: 20.11.2024).

2. OpenID Connect Core 1.0. – 2014. – URL: https://openid.net/specs/openid-connect-core-1_0.html (дата обращения: 20.11.2024).

3. RFC 7519. JSON Web Token (JWT). – 2015. – URL: https://datatracker.ietf.org/doc/rfc7519/ (дата обращения: 20.11.2024).

4. OWASP Foundation. Cross Site Scripting Prevention Cheat Sheet. – URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата обращения: 21.11.2024).

5. OWASP Foundation. Cross-Site Request Forgery (CSRF). – URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet (дата обращения: 21.11.2024).

6. Документация Auth0. – URL: https://auth0.com/docs (дата обращения: 21.11.2024).

7. The Backend for Frontend Pattern. – URL: https://auth0.com/blog/the-backend-for-frontend-pattern-bff/ (дата обращения: 21.11.2024).

8. RFC 6265. Cookies: HTTP State Management Mechanism. – URL: https://datatracker.ietf.org/doc/draft-ietf-httpbis-rfc6265bis/ (дата обращения: 21.11.2024).

9. RFC 7636. Proof key for Code Exchange by OAuth Public Clients. – 2015. – URL: https://datatracker.ietf.org/doc/html/rfc7636 (дата обращения 21.11.2024).

УДК 004.056

ПРОГНОЗИРОВАНИЕ РИСКОВ ИНФРАСТРУКТУРНОГО ДЕСТРУКТИВИЗМА НА ОСНОВЕ АНАЛИЗА ЖУРНАЛОВ СОБЫТИЙ ОБЛАЧНОЙ ПЛАТФОРМЫ OPENSTACK

А. М. Русаков

МИРЭА – Российский технологический университет

Аннотация. В статье рассмотрен способ прогнозирование рисков инфраструктурного деструктивизма на основе анализа журналов событий облачной платформы OpenStack. Описан эффект инфраструктурного деструктивизма как деструктивное воздействие на инфраструктуру, в результате которого проявляется непредвиденное и(или) нежелательное событие неконтролируемого саморазрушения инфраструктуры. Проводится экспериментальное исследования данных журналов событий для обнаружения эффектов инфраструктурного деструктивизма. Показана пропорциональная зависимость между «отрицательным» межсервисным взаимодействием и вероятностью появления эффектов инфраструктурного деструктивизма.
Ключевые слова: инфраструктурный деструктивизм, деструктивные воздействия инфраструктурного генеза, анализа журналов событий, облачная платформа OpenStack.

Введение

Постоянно растущие объёмы информации и необходимость её обработки в режиме реального времени предъявляют всё новые требования к производительности и эффективности инфраструктур организаций [1–3]. В современных условиях многие организации сталкиваются с тем, что постоянно увеличивавшийся объём обрабатываемой информации требует использование дополнительных сложных технологических решений, связанных с распределением и масштабирование нагрузки на инфраструктуру. Для этого применяются различные балансировщики и оптимизаторы запросов, средства кеширования и виртуализации, горизонтальное и вертикальное масштабирование, а также другие средства, которые приводят к увеличению сложности мониторинга и управления инфраструктурой [1, 3]. Возрастающая сложность инфраструктур многократно повышает риски информационной безопасности организации.

Наиболее востребованными методами обеспечения кибербезопасности инфраструктур являются методы, связанные с обнаружением и реагированием на события безопасности, антивирусная защита конечных точек, различные системы обнаружения вторжений, системы реагирования на сложные угрозы и целевые атаки [2]. Повсеместно применяются интеллектуальные методы анализа поведенческой активности пользователей и сущностей (UEBA). Одним из перспективных направлений поведенческой аналитики и обеспечения кибербезопасности инфраструктур является исследование эффектов инфраструктурного деструктивизма [4].

Современные исследователи характеризуют эффект инфраструктурного деструктивизма как деструктивное воздействие на инфраструктуру, в результате которого проявляется непредвиденное и(или) нежелательное событие, вызванное совокупностью факторов и условий инфраструктурного генеза, создающих опасность неконтролируемого саморазрушения инфраструктуры и нарушения информационной безопасности [4]. Также исследователи необходимость исследования межобъектных взаимосвязей инфраструктуры, как источников деструктивных воздействий, способных привести к эффекту инфраструктурного деструктивизма, т. е. к саморазрушению инфраструктуры. При наличии эффектов инфраструктурного деструктивизма появляются серьезные аномалии поведенческой активности объектов инфраструктуры (замедленнее или ускорение информационных процессов), которые нуждаются в

прогнозировании и оценке. Поведенческие особенности объектов инфраструктуры с различными уязвимостями могут оказывать синергетический эффект совместного ухудшения уровня информационной безопасности. Для поведенческой аналитики применяются статистические методы, методы прогнозирования и анализа временных рядов, а также методы машинного обучения, которые позволяют по-новому оценивать риски возникновения инфраструктурного деструктивизма и снижать риски кибератак и саморазрушения инфраструктуры.

Однако, недостаточное рассмотрение эффектов инфраструктурного деструктивизма в научных трудах, а также отсутствие на рынке готовых программных продуктов, делают актуальной разработку моделей и алгоритмов оценки динамики рисков инфраструктурного деструктивизма в качестве средства повышения уровня информационной безопасности инфраструктуры.

1. Антропоморфическая модель оценки деструктивных воздействий инфраструктурного генеза

В основе антропоморфической модели оценки деструктивных воздействий инфраструктурного генеза, лежит предположение о возможности описания процесса взаимодействия элементов инфраструктуры с позиции «живых» микроорганизмов [5, 6]. Опишем данную модель использую теоретико-множественное представление. Пусть: $O_1, O_2, ..., O_n$ — взаимодействующие элементы инфраструктуры; $S_1, S_2, ..., S_n$ — взаимосвязанные моногенные подсистемы инфраструктуры; D_1 и D_2 — источники деструктивных воздействий; R_1 и R_2 — результаты воздействий D_1 и D_2 соответственно;. Информационное взаимодействие между элементами инфраструктур является важным процессом, прямо отражающимся на процессе функционирования инфраструктуры в целом. Охарактеризуем виды имеющихся взаимодействий и воздействий между элементами инфраструктуры.

Под негативным (отрицательным) воздействием на инфраструктуру будем понимать различные уязвимости, вирусы, кибератаки, и другие негативные воздействия, которые могут вывести информационную инфраструктуру из строя, обозначим данные воздействия через источники деструктивных воздействий D_1 и D_2, в количественном отношения будем обозначать как $-1d_1...0$ и $-1d_2...0$.

Под позитивным (положительным) воздействием на инфраструктуру будем понимать различные эффекты от деструктивных воздействий D_1 и D_2, которые приводят к положительным эффектам на информационную инфраструктуру в целях информационной безопасности при этом данные воздействия в количественном отношения будем обозначать как $-1d_1...0$ и $-1d_2...0$ соответственно.

Модель основывается на допущении о том, что взаимодействие деструктивных воздействий инфраструктурного генеза принципиально схоже со взаимодействием биологических микроорганизмов. Согласно широко распространенному в науке делению отношений организмов известны следующие типы их взаимодействий: симбиоз (облигатный и факультативный симбиоз, комменсализм, паразитизм, хищничество) — когда хотя бы один из организмов получает выгоду, антибиоз (аменсализм, аллелопатия, конкуренция) — когда один из организмов ограничивает возможности другого, и нейтрализм — сосуществования организмов без взаимного влияния [7, 8]. В этой работе приведем два крайних случая, когда деструктивные воздействия образуют факультативный симбиоз и аллелопатию, которые дают максимальный и минимальный антропоморфический эффект.

Рассмотрим факультативный симбиоз деструктивных воздействий, который в природе характеризуется взаимной выгодой от совместного сосуществования организмов, но без необхо-

димости как таковой. То есть деструктивные воздействия влияют на информационную инфраструктуру, но при этом деструктивный эффект от их совместного взаимодействия возрастает.

Положим, что эффект возрастает со знаком минус $-2d_1$ при условии наличия деструктивных воздействий $-1d_2$ и $-2d_2$ при условии $-1d_1$ соответственно. Таким образом общий эффект от факультативного симбиоза составит $-2d_1 - 2d_2$, что соответствует следующей формуле: $-2(d_1 - d_2)$, которая подтверждается механизмами существования микроорганизмов в живой природы. Схематически отобразим данное явление на рис. 1.

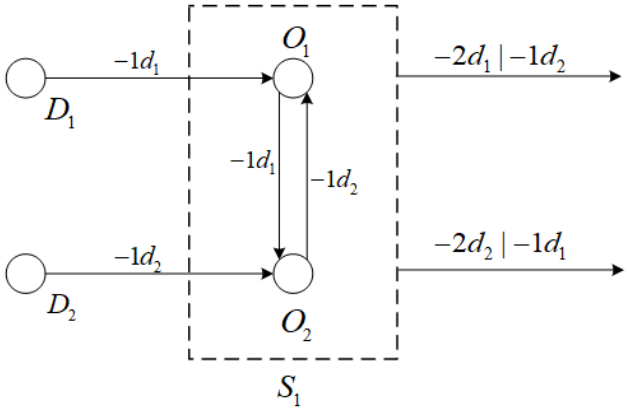


Рис. 1. Схема факультативный симбиоз деструктивных воздействий

Рассмотрим аллелопатию деструктивных воздействий. С позиции живой природы данный тип взаимодействия биологических организмов характеризуется их взаимно-вредным влиянием друг на друга. То есть деструктивные воздействия влияют на информационную инфраструктуру, но при этом они сами себя уничтожают.

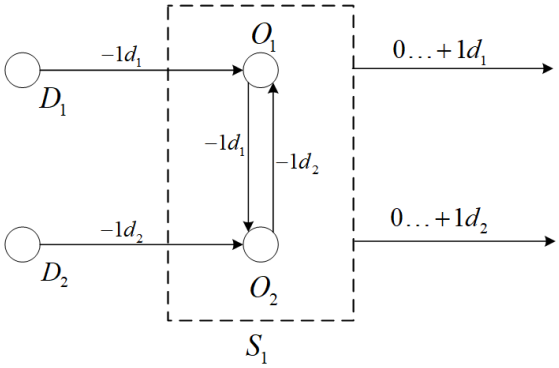


Рис. 2. Схема аллелопатия деструктивных воздействий

Таким образом общий эффект от аллелопатии деструктивных воздействий составит в худшем случае 0 и в лучшем случае составит $1d_1 + 1d_2$, что также подтверждается механизмами существования живой природы. Описав данные процессы для всех антропоморфических механизмов взаимодействия деструктивных воздействий стало возможным качественно и количественно оценить каждый из них как по отдельности, так и в совокупности. Это позволяет оценить взаимное влияние сервисов инфраструктур и оценить их по антропоморфическим типам.

2. Описание облачной платформы OpenStack

Облачная платформа OpenStack представляет собой комплекс проектов свободного программного обеспечения для создания инфраструктурных облачных сервисов, центров обработки данных, облачных хранилищ, разработки и предоставления приложений. Облачная платформа OpenStack может использоваться как для публичных, так и частных облачных

сервисов. В данной исследовании используется данные журналов событий системы фрагмент представлен на рис. 3.

StartTime	EndTime	URL	SpanType	Service	SpanId	TraceId	Peer	ParentSpan	Component	IsError
1628884697515	1628884697523	SpringAsync	Local	ts-cancel-service	a8ac2bc13e5c423d83743ee2d661482b.58.16288846975150062.0	a8ac2bcl3e5c423d83743ee2d661482b42.16288846975100029	ts-cancel-service	a8ac2bc13e5c423d83743ee2d661482b.42.16288846975100028.0	SpringAsync	False
1628884697515	1628884697523	/api/v1/inside_pay_service/inside_payment/drawback/4d2a46c7-71cb-4cf1-b5bb-b68406d9da6f/	Exit	ts-cancel-service	a8ac2bc13e5c423d83743ee2d661482b.58.16288846975150062.1	a8ac2bcl3e5c423d83743ee2d661482b42.16288846975100029	ts-inside-payment-service	a8ac2bcl3e5c423d83743ee2d661482b.58.16288846975150062.0	SpringRestTemplate	False
1628884697518	1628884697524	{GET}/api/v1/inside_pay_service/inside_payment/drawback/{userId}	Entry	ts-inside-payment-service	e011366013074feda29701dc14e41305.44.16288846975180094.0	a8ac2bcl3e5c423d83743ee2d661482b42.16288846975100029	ts-inside-payment-service	a8ac2bc13e5c423d83743ee2d661482b.58.1628 8846975150062.1	SpringMVC	False
1628884697520	1628884697520	MongoDB/FindOperation	Exit	ts-inside-payment-service	e011366013074feda29701dc14e41305.44.16288846975180094.0	a8ac2bcl3e5c423d83743ee2d661482b42.16288846975100029	ts-inside-payment-mongo	e011366013074feda29701dc14e41305.44.16288846975180094.0	mongodb-driver	False
1628884697523	1628884697524	MongoDB/UpdateOperation	Exit	ts-inside-payment-service	e011366013074feda29701dc14e41305.44.16288846975180094.0	a8ac2bcl3e5c423d83743ee2d661482b42.16288846975100029	ts-inside-payment-mongo	e011366013074feda29701dc14e41305.44.16288846975180094.0	mongodb-driver	False
1628884697515	1628884701520	SpringAsync	Local	ts-cancel-service	a8ac2bc13e5c423d83743ee2d661482b.59.16288846975150062.0	a8ac2bcl3e5c423d83743ee2d661482b42.16288846975100029	ts-cancel-service	a8ac2bc13e5c423d83743ee2d661482b.58.16288846975100028.0	SpringAsync	False
1628884701516	1628884701520	/api/v1/orderservice/order	Exit	ts-cancel-service	a8ac2bc13e5c423d83743ee2d661482b.59.1 6288846975150062.1	a8ac2bcl3e5c423d83743ee2d661482b42.16288846975100029	ts-order-service	a8ac2bc13e5c423d83743ee2d661482b.58.16288846975150062.0	SpringRestTemplate	False

Рис. 3. Фрагмент журналов событий облачной платформы OpenStack

В эксперименте использованы открытые данные «DeepTraLog» из лаборатории разработки программного обеспечения университета Фудань (Китай, Шанхай). Данные «DeepTraLog» использовались для проведения соревнований по обнаружению аномалий в журналах событий для инфраструктур, построенных на сервисах с использованием облачной платформы OpenStack.

3. Экспериментальное исследование и метода прогнозирования рисков инфраструктурного деструктивизма на основе анализа журналов событий

Основной целью экспериментального исследования является прогнозирование рисков и обнаружение аномалий в журналах событий инфраструктуры построенной облачной платформе OpenStack схожих по своей природе с описанной в данной работе эффектами инфраструктурного деструктивизма. Предложенная модель реализована в программном обеспечении [11]. Разработанное программное обеспечение визуализирует динамику работы процессов как показано на рис. 4.

В эксперименте использовались данные с 05.08.2021 по 26.12.2021 для 32 сервисов [10]. Проводилась оценка динамики взаимного влияния сервисов на основе антропоморфических моделей поведения. Вычислялись оценки положительного, нейтрального и отрицательного взаимного влияния сервисов как показано на рис. 5 и 6.

Проанализировав отображение множества и построив линейные тренды — красные, синие и зеленые линии на рис. 5 и 6. Сделан вывод, что эффекты инфраструктурного деструктивизма наиболее вероятны в момент после 10.12.2021. Данный вывод подтверждается практически, так как инфраструктура престала функционировать после 27.12.2021 без видимых на это причин [10].

Заключение

Полученные результаты еще раз подтверждают, что имеется пропорциональная зависимость между «отрицательным» межсервисным взаимодействием и вероятностью появления эффектов инфраструктурного деструктивизма. Вероятнее всего эффект инфраструктурного деструктивизма будет себя проявлять, когда возрастает показатель отрицательного» межсервисным взаимодействием (см. рис. 5). Предложенный подход прогнозирования инфраструктурного деструктивизма, возможно также использовать как метрику (показатель) «здоровья» инфраструктуры для прогнозирования и оценки рисков. Используя данную метрику возмож-

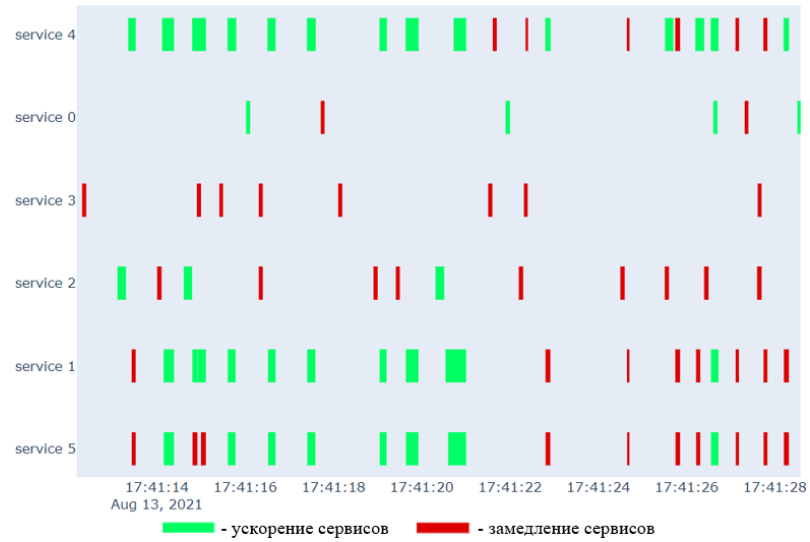


Рис. 4. Фрагмент визуализации динамики работы процессов

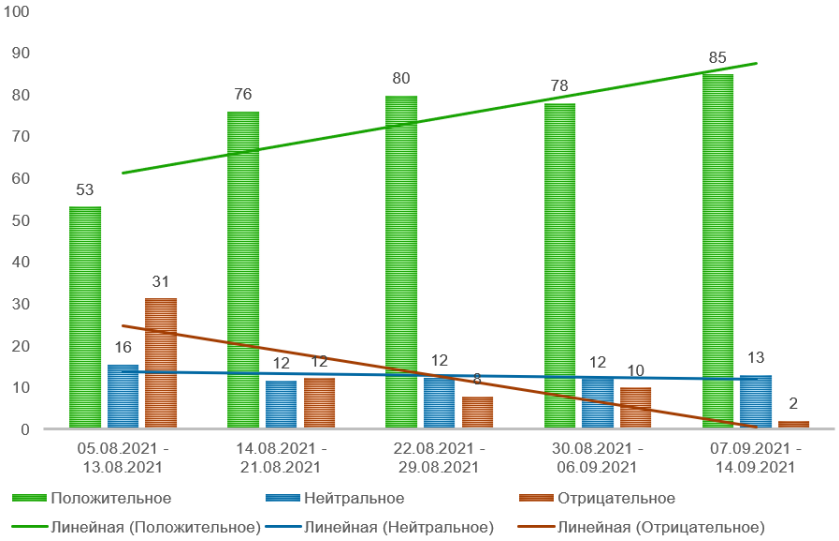


Рис. 5. Динамика взаимного влияния сервисов на основе антропоморфических моделей поведения за период с 05.08.2021 по 14.09.2021

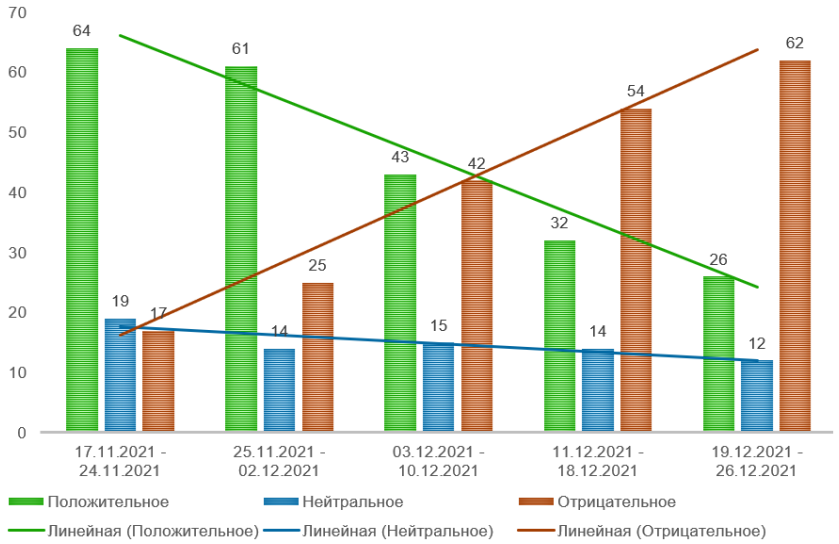


Рис. 6. Динамика взаимного влияния сервисов на основе антропоморфических моделей поведения за период с 17.11.2021 по 26.12.2021

но заранее прогнозировать эффекты возникновения инфраструктурного деструктивизма для сервисных архитектур облачных платформ таких как платформа OpenStack. Разработанная антропоморфическая модель является одним из перспективных подходов для анализа поведенческой активности сервисов в инфраструктурах.

Литература

1. *Головина Е. Ю.* Оценка состояния безопасности ИТ-инфраструктуры в организации / Е. Ю. Головина, А. В. Журавлева, Л. И. Татарникова // Молодежный вестник ИрГТУ. – 2022. – Т. 12, № 2. – С. 266–272.

2. *Качуров Е. И.* Подход к разработке модели угроз информационной безопасности для ИТ-инфраструктуры / Е. И. Качуров // Научная дискуссия: вопросы технических наук. – 2017. – № 6(46). – С. 74–80.

3. *Хрошин А. А.* Безопасность баз данных и алгоритмы анализа Big Data для мониторинга ИТ инфраструктуры предприятия / А. А. Хрошин // Научный аспект. – 2024. – Т. 21, № 5. – С. 2819–2823.

4. *Максимова Е. А.* Методы выявления и идентификации источников деструктивных воздействий инфраструктурного генеза / Е. А. Максимова // Электронный сетевой политематический журнал «Научные труды КубГТУ». – 2022. – № 2. – С. 86–99.

5. *Русаков А. М.* Анализ динамики рисков деструктивного воздействия инфраструктурного генеза / А. М. Русаков // Кибербезопасность: технические и правовые аспекты защиты информации: Сборник научных трудов I Национальной научно-практической конференции, Москва, 24–26 мая 2023 года. – Москва : МИРЭА - Российский технологический университет, 2023. – С. 85–87.

6. Anthropomorphic Model of States of Subjects of Critical Information Infrastructure Under Destructive Influences / E. A. Maksimova, A. M. Rusakov, M. A. Lapina, V. G. Lapin // Lecture Notes in Networks and Systems. – 2022. – Vol. 424. – P. 569–580.

7. *Буйневич М. В., Израилов К. Е.* Антропоморфический подход к описанию взаимодействия уязвимостей в программном коде. Часть 1. Типы взаимодействий // Защита информации. Инсайд. – 2019. – № 5 (89). – С. 78–85.

8. *Буйневич М. В., Израилов К. Е.* Антропоморфический подход к описанию взаимодействия уязвимостей в программном коде. Часть 2. Метрика уязвимостей // Защита информации. Инсайд. 2019. № 6 (90). С. 61-65.

9. *C. Zhang [et al.]* DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning, 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA. – 2022. – P. 623–634.

10. DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning. – 2024. – URL: https://github.com/FudanSELab/DeepTraLog (дата обращения: 26.10.2024)

11. Свидетельство о государственной регистрации программы для ЭВМ №2023683118 Российская Федерация. Антропоморфическая система моделирования деструктивных воздействий инфраструктурного генеза на объектах критической информационной инфраструктуры: № 2023682500: заявл. 24.10.2023 : опубл. 03.11.2023 / А. М. Русаков.

УДК 004.415.2.052.03

ПАТТЕРНЫ ОБРАБОТКИ ОШИБОК МЕЖСЕРВИСНОЙ КОММУНИКАЦИИ В МИКРОСЕРВИСНЫХ АРХИТЕКТУРАХ

С. Р. Сидельникова, Н. А. Каплиева

Воронежский государственный университет

Аннотация. В статье рассматриваются часто используемые паттерны обработки ошибок в микросервисных архитектурах. Проанализированы причины возникновения сбоев при взаимодействии между сервисами и их влияние на устойчивость системы. Описаны принципы работы таких паттернов, как временное отключение (timeout), повторные попытки (retry), размыкатель цепи (circuit breaker), изоляция ресурсов (bulkhead) и ограничитель скорости (rate limiter). Особое внимание уделено проблемам, с которыми можно столкнуться при использовании паттернов и комбинированию паттернов между собой.
Ключевые слова: микросервисы, распределенные системы, отказоустойчивость, изоляция, обработка сетевых ошибок, паттерны устойчивости, retry, timeout, circuit breaker, bulkhead, rate limiter.

Введение

Современные программные системы стремительно переходят от монолитной архитектуры к распределённой, где микросервисы выступают отдельными строительными блоками цельной системы. Такой подход позволяет достичь масштабируемости, гибкости разработки и быстрой доставки новых функций. Однако вместе с преимуществами микросервисная архитектура привносит и новые сложности, в первую очередь — в области обработки ошибок.

Одним из ключевых источников сбоев становятся ошибки, возникающие в процессе коммуникации между сервисами. Даже кратковременная недоступность одного из компонентов может привести к сбоям в цепочке вызовов, каскадным отказам и снижению общей устойчивости системы. Причины таких ошибок могут быть самыми разными: от перегрузки и сетевых задержек до частичной деградации инфраструктуры. Несмотря на свою, как правило, временную природу, без должной обработки подобные сбои могут вызывать серьёзные последствия — от потери согласованности данных до снижения безопасности.

В данной статье рассматриваются некоторые современные подходы к обеспечению отказоустойчивости микросервисных систем, а также приводятся практические рекомендации по выбору и комбинированию этих решений.

1. Причины возникновения ошибок сетевого взаимодействия в микросервисных архитектурах

1.1. Сетевая нестабильность

Часто ошибки в микросервисных системах возникают из-за распределенной природы архитектуры, где взаимодействие сервисов зависит от надежности сетевой инфраструктуры. Каждый вызов между сервисами требует установления сетевого соединения, что подвержено различным видам сбоев. Эти проблемы могут включать в себя потерю пакетов, высокую задержку, снижение пропускной способности, а также полное отключение части сетевой инфраструктуры.

Потери пакетов могут происходить из-за перегрузки сетевых интерфейсов, ошибок на маршрутизаторах, проблем с буферизацией или недостаточной пропускной способности ка-

налов. Даже минимальная потеря пакетов может привести к повторной передаче данных, дополнительной задержке или ошибкам при восстановлении соединения.

Джиттер (нестабильная задержка) — случай, когда запрос выполняется разное количество времени, в зависимости от нагрузки сети. Это разрушает предсказуемость поведения системы и может привести к превышению установленных лимитов ожидания на клиентской стороне.

Разрывы соединений до завершения передачи данных могут быть связаны с перегрузкой сети, перезапуском инфраструктурных компонентов, (например, балансировщиков нагрузки), перезапуском контейнеров или миграцией виртуальных машин.

1.2. Недоступность узлов или их частичный отказ

Сервисы в микросервисной архитектуре взаимодействуют друг с другом через сеть, и недоступность одного из них может привести к каскадным последствиям для всей системы. Под недоступностью понимается не только полный отказ сервиса, но и его частичная деградация, когда он отвечает с большой задержкой, нестабильно или с ошибками. Такие сбои могут возникать по логическим причинам.

Бывают ситуации, когда нагрузка на определённый сервис может резко возрасти. Это может происходить из-за резких изменений в пользовательском трафике. Если сервис не масштабируется должным образом или не защищён от перегрузки (например, отсутствуют механизмы ограничения числа одновременных запросов), он начинает деградировать. В лучшем случае это проявляется в увеличении времени отклика, в худшем — в полной недоступности.

Особенно опасны ситуации, когда зависимые сервисы продолжают направлять запросы к деградировавшему компоненту, не распознавая его состояние. В таком случае вызывающий сервис продолжает ожидать ответа, занимая свои ресурсы и блокируя последующие вызовы других сервисов к себе. Таким образом, даже один проблемный сервис может вызвать «цепную реакцию», в результате которой вся система становится нестабильной или недоступной.

1.3. Гетерогенность протоколов и технологий

Ещё одной важной причиной сбоев на сетевом уровне является технологическая разнородность микросервисной среды. Микросервисы проектируются как изолированные компоненты, и команды разработчиков часто выбирают для каждого сервиса индивидуальные подходящие инструменты. В результате в рамках одной системы может одновременно использоваться множество форматов передачи данных (JSON, XML, Protocol Buffers), транспортных протоколов (HTTP/1.1, HTTP/2, gRPC, AMQP), а также библиотек сериализации и фреймворков.

Такое разнообразие увеличивает гибкость системы, но в то же время порождает множество потенциальных точек несовместимости. Даже незначительные отличия в реализации протоколов, обработке ошибок могут привести к некорректному обмену данными, ошибкам десериализации и потере связи между сервисами.

Например, сервис на базе GraphQL может вернуть ошибку логики в теле ответа с HTTP статусом 200 OK, что нарушает ожидания клиента, ориентированного на REST и стандартные коды ошибок.

Особенно остро проявляется эта проблема при использовании асинхронных средств коммуникации — очередей сообщений, брокеров (RabbitMQ, Kafka и др.), где необходимо соблюдать строгую совместимость между схемами сообщений и их обработкой на стороне получателя.

1.4. Ограниченная гибкость системы

Одной из распространённых причин сбоев в микросервисных системах является недостаточная гибкость инфраструктуры при изменении нагрузки. В условиях динамичного трафика

и высокой распределённости система должна уметь оперативно адаптироваться — масштабироваться, перераспределять ресурсы, перенастраивать маршруты вызовов.

Масштабируемость — это способность системы адаптироваться к изменению объёма нагрузки. Система должна уметь правильно реагировать на увеличение или снижение трафика, например, регулируя количество инстансов. Но даже при наличии автоматического масштабирования новые инстансы сервисов могут запускаться с опозданием, что не избавит работающие сервисы от перегрузки.

Даже если система контролируется балансировщиком нагрузки, равномерно распределяющим трафик среди нескольких серверов или сервисов, она не застрахована от проблем. Если схема балансировки не учитывает реальную загрузку, то все узлы получают равное количество запросов, независимо от своей производительности. Это особенно критично, если инстансы работают, например, в разных регионах. Также медленная реакция балансировщика на изменения может быть причиной того, что запросы будут продолжать направляться на уже неработающие какое-то время узлы.

Результат этих проблем — образование «горячих точек» в системе, где одни инстансы перегружены и возвращают ошибки, а другие недогружены и простаивают. Это, опять же, может привести к каскадным отказам, рассмотренным выше.

2. Паттерны обработки ошибок

Микросервисные архитектуры подвержены множеству факторов, способных нарушить устойчивость сетевого взаимодействия. Для повышения устойчивости таких систем применяются особые архитектурные паттерны. Эти паттерны не устраняют первопричины — например, не могут предотвратить отключение сервиса или перегрузку сети — но обеспечивают корректную реакцию на сбои, минимизируют их влияние на другие компоненты и помогают системе продолжать работу.

2.1. Timeout

Паттерн Timeout (интервал ожидания) является одним из самых базовых паттернов. Его основная цель — ограничить время ожидания ответа от удалённого сервиса или компонента, чтобы избежать «зависания» вызывающей стороны и высвобождать ресурсы в случае сбоев или чрезмерных задержек.

Паттерн работает следующим образом: для каждого вызова задается максимальное время ожидания ответа (например, 2 секунды). Если ответ не получен за это время, операция отменяется. Клиент получает уведомление (например, HTTP 504 Gateway Timeout) и может самостоятельно решать, какие дальнейшие действия предпринимать. Таким образом, клиент не окажется заблокированным, если сервис не доступен.

Можно выделить следующие рекомендации по работе с паттерном:

1. По возможности использовать адаптивные таймауты — динамически изменяющие свое значение в зависимости от текущих характеристик системы. Если значение таймаута жесткое и фиксированное, при временных всплесках задержки все рабочие запросы будут отклонены, хотя они могли бы быть выполнены, но немного медленнее.

2. Разделять таймауты по контексту. Разные типы операций требуют разного уровня терпимости к задержкам. Универсальный таймаут для всех вызовов — грубая ошибка.

3. Ставить таймаут не на вызов конкретного метода, а на логически объединенную операцию. Необходимо прекратить бессмысленные действия внизу цепочки вызовов, если общий таймаут уже истекает;

4. Нет единого «правильного» значения таймаута — его нужно подбирать опытным путем индивидуально для каждого сервиса и сценария.

2.2. Retry

Паттерн Retry (повторная попытка) — это стратегия обработки ошибок, при которой система автоматически повторяет неудачный запрос. Этот подход эффективен, когда кратковременная недоступность или перегрузка сервисов могут носить эпизодический характер из-за сетевых проблем. Он часто используется совместно с паттерном Timeout.

Сначала система определяет, что запрос завершился неудачей (например, HTTP 503 Service Unavailable). Решение о повторе принимается только для ошибок, которые могут не возникнуть при повторном запросе (как правило, запросы с кодом 5xx). Если ошибочный запрос является таким, то операция выполняется повторно через заданный интервал. Также устанавливаются ограничения на количество повторных попыток, чтобы избежать бесконечных циклов. По достижении максимального количества попыток система фиксирует окончательную неудачу выполнения запроса.

Существует несколько распространённых стратегий задержек между попытками при использовании Retry. Некоторые из них:

1. Фиксированная задержка

$$delay(n) = \text{const}.$$

Каждая попытка выполняется через одинаковый интервал времени.

2. Линейная задержка

$$delay(n) = \text{const} * n.$$

Интервал увеличивается линейно с каждой попыткой.

3. Экспоненциальная задержка

$$delay(n) = \text{const} * (2^{n-1})).$$

Интервал увеличивается экспоненциально с каждой попыткой.

Рекомендации по использованию паттерна:

1. Не все операции можно безопасно повторить. Повторный вызов метода, который изменяет состояние (например, списание средств, создание заказа), может привести к дублирующим транзакциям.

2. Retry имеет смысл только для временных и восстанавливаемых ошибок.

3. Массовые повторные запросы могут усугубить проблему, создавая еще большую нагрузку. Вместо восстановления, Retry может стать причиной окончательного отказа сервиса.

4. Совместное использование Retry и Timeout предполагает два варианта расчета временной задержки: Timeout устанавливается суммарно для всех попыток или для каждой попытки по отдельности. Можно не выполнять третью (или даже вторую) попытку, если общий порог тайм-аута уже превышен.

2.3. Circuit Breaker

Circuit Breaker (автоматический выключатель) — это шаблон, предназначенный для предотвращения повторных неудачных попыток взаимодействия с компонентом, находящимся в нестабильном состоянии. Он действует как автоматический «предохранитель», который прерывает поток вызовов к проблемному сервису и дает ему время на восстановление. Тем временем вместо реальных вызовов к сервису клиент может обратиться к кэшу. Блокируя вызовы к упавшему сервису, Circuit Breaker защищает и остальные компоненты от перегрузки.

Паттерн использует три состояния (рис. 1):

• closed (закрытое) — все запросы к сервису выполняются. Ошибки накапливаются. Если процент ошибок превышает заданный порог (например, 50 % за последние 60 секунд), Circuit Breaker переходит в состояние open;

• open (открытое) — вызовы к сервису блокируются. Сразу возвращается ошибка или заглушка. В этом состоянии система даёт время на восстановление. Через заданный интервал (например, 30 секунд) Circuit Breaker переходит в состояние half-open;

• half-open (полуоткрытое) — разрешается ограниченное число «пробных» вызовов. Если они успешны, предохранитель переходит в состояние closed и разрешает доступ к сервису. При повторных сбоях возврат в open и продолжение прерывания запросов.

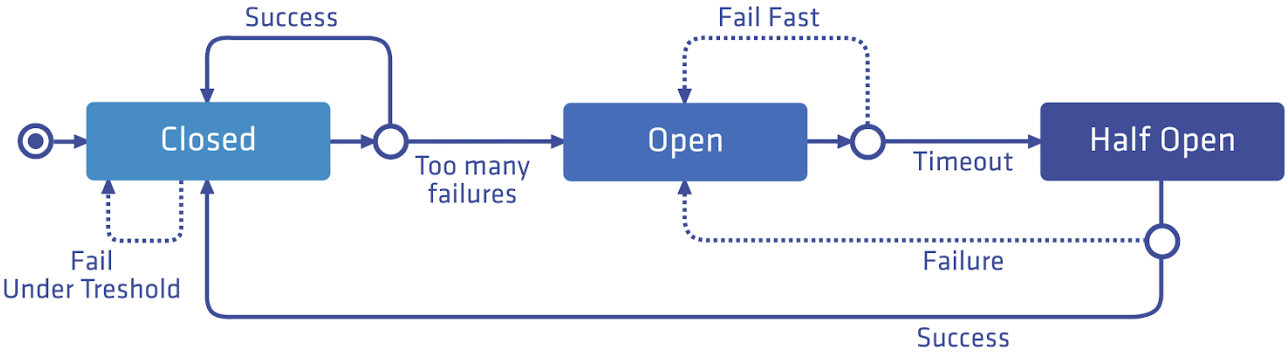


Рис. 1. Схема работы Curcuit Breaker

Основная сложность при работе с паттерном — это правильный выбор параметров, которые будут регулировать состояние Circuit Breaker. Например, если количество допустимых подряд ошибок слишком мало, то это приведёт к частым ложным срабатываниям, а если слишком высоко — к долгому реагированию на реальную проблему. Также не нужно слишком быстро отключать автоматический выключатель, и нельзя допустить, чтобы его отключение занимало слишком много времени.

Нужно координировать работу Timeout + Retry + Circuit Breaker. Например, если каждый повторный вызов будет восприниматься как отдельная ошибка, то Circuit Breaker быстро перейдёт в открытое состояние. А если таймаут будет дольше, чем время ожидания перед переходом в новое состояние предохранителя, Circuit Breaker может не увидеть реальные проблемы, потому что ошибки просто будут в ожидании и не посчитаются сбоями.

2.4. Bulkhead

Bulkhead (перегородка) — это архитектурный паттерн, направленный на локализацию сбоев внутри системы. Он предполагает разделение компонентов или ресурсов для ограничения влияния сбоев или перегрузок в одной области на остальную часть системы. Такая изоляция гарантирует, что сбой в одной части системы не выведет ее целиком из строя.

Паттерн основан на использовании отдельных пулов потоков для выполнения задач разных типов. Идентифицируются вызовы или компоненты, которые потенциально могут создавать нагрузку или быть нестабильными, каждой такой группе выделяется отдельный пул ресурсов. Примерами могут быть отдельный пул потоков для обработки входящих HTTP-запросов от пользователей и отдельный пул для запросов к внешнему API. Для каждого пула задаются индивидуальные лимиты, в зависимости от ожидаемой рабочей нагрузки. Если лимит исчерпан, новые запросы отклоняются или ставятся в очередь. Это предотвращает истощение общих ресурсов и конкуренцию за них между различными частями системы, а также всю систему от полной деградации.

Следует обратить внимание на несколько ключевых особенностей при внедрении паттерна в систему:

1. Непросто заранее определить, какие ресурсы стоит изолировать. Нужно проанализировать, какое влияние оказывает на бизнес-логику потеря каждой конкретной функциональности и стремиться к разделению ресурсов на основе логики: формировать отдельные пулы для критических и фоновых задач.

2. Паттерн может быть реализован на разных уровнях системы. Наиболее распространённый способ применения — изоляция в рамках отдельных пулов потоков приложения. Но также могут быть установлены отдельные пулы на количество подключений к базе данных или отдельные очереди или потоки обработки сообщений в очередях сообщений. Например, можно задать очередь с транзакциями, которая будет обрабатываться с высоким приоритетом, и очередь с логами, которая может быть обработана с некоторыми задержками.

2.5. Rate Limiter

Rate Limiter (ограничитель скорости) — это паттерн, направленный на управление нагрузкой и предотвращение перегрузки системы путем ограничения частоты запросов или операций, которые могут быть выполнены за определённый промежуток времени. Другими словами, он помогает контролировать пропускную способность.

Основная идея паттерна — отслеживать количество запросов от клиента/сервиса. Фиксируется каждый запрос от клиента в течение какого-то времени, и если их количество превышает допустимый лимит, то дальнейшие запросы блокируются. Как правило, возвращается ошибка HTTP 429 Too Many Requests.

Некоторые возможные реализации паттерна:

1. Алгоритм Fixed Window (фиксированное окно) — вводятся ограничения на количество запросов, которые могут быть выполнены за фиксированное время (например, 1000 запросов в минуту). Простой в реализации, но может приводить к пикам нагрузки на границе окон времени.

2. Алгоритм Sliding Window (скользящее окно) — работает по тому же принципу, но с более динамичным учётом запросов. Для каждого нового запроса система проверяет, сколько запросов было выполнено за последние N секунд с момента последнего запроса, а не в фиксированном промежутке.

3. Token-bucket — метод, при котором система выдает пользователю определенное количество токенов с фиксированной скоростью, при этом каждый запрос расходует один токен. Если токены закончились — запрос отклоняется.

Паттерн Rate Limiter используется для защиты сервисов от перегрузки. Примеры использования включают защиту API от злоупотреблений, регулирование нагрузки на серверы, а также защиту от атак типа DoS (Denial of Service), когда злоумышленники пытаются перегрузить систему запросами.

Выбор правильных параметров также является существенным, как и для других рассмотренных выше паттернов. Ограничение на количество запросов может быть настроено как для отдельных пользователей, так и для всех пользователей системы. Поэтому важно разделять глобальные лимиты для системы и индивидуальные лимиты для каждого клиента.

Также необходимо координировать работу паттерна совместно с другими паттернами. Например, Retry-запросы могут быстро исчерпать лимит, особенно при высокой нагрузке, а потому должны использовать экспоненциальная задержку.

Заключение

Микросервисная архитектура предоставляет широкие возможности для масштабирования и гибкости разработки, но одновременно предъявляет повышенные требования к надёжности взаимодействия компонентов системы между собой. Сетевые ошибки и временные сбои являются неизбежными в распределённых системах, и потому требуют системного подхода к их обработке.

Рассмотренные в статье паттерны позволяют повысить устойчивость системы к отказам и минимизировать негативные последствия временных сбоев. Однако выбор конкретных решений должен основываться на характеристиках системы, частоте и природе ошибок, а также на допустимых компромиссах между доступностью, консистентностью и производительностью.

Литература

1. *Ньюмен С.* Создание микросервисов / Сэм Ньюмен. – СПб. : Питер, 2023. – 624 с.

2. *Ричардсон К.* Микросервисы. Паттерны разработки и рефакторинга / Крис Ричардсон. – СПб. : Питер, 2019. – 554 с.

3. *Нейгард М.* Release it! Проектирование и дизайн ПО для тех, кому не все равно / Майкл Нейгард. – СПб. : Питер, 2016. – 320 с.

4. Язык шаблонов для микросервисов // Что такое микросервисы: [сайт]. – URL: https://microservices.io/patterns/index.html (дата обращения: 05.02.2025).

5. What is Circuit Breaker Pattern in Microservices? // GeeksForGeeks: [сайт]. – URL: https://www.geeksforgeeks.org/what-is-circuit-breaker-pattern-in-microservices (дата обращения: 06.02.2025).

6. Bulkhead Pattern // GeeksForGeeks: [сайт]. – URL: https://www.geeksforgeeks.org/bulkhead-pattern (дата обращения: 06.02.2025).

7. Rate Limiting Algorithms – System Design // GeeksForGeeks: [сайт]. – URL: https://www.geeksforgeeks.org/rate-limiting-algorithms-system-design (дата обращения: 06.02.2025).

УДК 004.42

АВТОМАТИЗАЦИЯ ОЦЕНКИ ТРУДОЗАТРАТ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Р. Е. Скиданов, И. Е. Воронина

Воронежский государственный университет

Аннотация. Рассматривается применение методов машинного обучения для оценки трудозатрат в проектах по разработке программного обеспечения. Традиционные методы оценки, такие как экспертные оценки и аналогии, часто оказываются недостаточно точными из-за высокой степени неопределенности и изменчивости в процессе разработки. Предлагается новый подход, основанный на анализе исторических данных о выполненных проектах с использованием алгоритмов машинного обучения для предсказания временных затрат с учетом различных факторов, таких как сложность задачи, опыт команды и используемые технологии.
Ключевые слова: машинное обучение, оценка трудозатрат, управление проектами, организация работ, автоматизация.

Введение

Оценка времени выполнения задачи в области разработки программного обеспечения (ПО) является актуальной задачей, поскольку непосредственно влияет на эффективность управления проектами и ресурсами. В условиях современного быстро меняющегося мира, где сроки выполнения задач становятся все более жесткими, точная оценка времени позволяет избежать задержек и перерасхода ресурсов. Кроме того, правильная оценка времени способствует улучшению планирования и координации между командами, что особенно важно в многозадачных и междисциплинарных проектах.

Практический опыт показывает, что ошибки в оценке времени могут привести к значительным финансовым потерям и снижению качества конечного продукта.

Кроме того, с развитием технологий и методов анализа данных возникает необходимость в разработке более точных алгоритмов для оценки времени выполнения задач. Это особенно актуально в таких областях, как программирование, строительство и научные исследования, где временные рамки имеют критическое значение.

Таким образом, исследование методов оценки времени выполнения задач не только способствует повышению общей продуктивности, но и помогает формировать более реалистичные ожидания у всех участников процесса. В конечном итоге, это может привести к более успешным проектам и улучшению качества жизни работников.

1. Важные аспекты задачи оценки трудозатрат

При разработке ПО, во многих фирмах используется система управления проектами Jira [1] или ее аналоги, созданные в том числе и в рамках программы импортозамещения. В системе управления проектами существуют задачи — информационный объект с некоторыми характеристиками, который подразумевает под собой некоторое задание, выполнение которого приблизит программу к желаемому состоянию. Оценить трудозатраты на выполнение задачи достаточно сложно даже для опытных разработчиков, но это важная часть работы, так как это позволяет оценивать приоритет разных задач с учетом приносимой ими пользы в соотношении ко времени разработки (например, может быть сложная и долгая задача без особой выгоды для компании или простая и рентабельная).

Выделим основные характеристики каждой задачи в системе управления проектами:

• описание — текст, содержащий в себе полную формулировку задачи для выполнения;

• дата создания — дата создания задачи;

• дата завершения — дата перевода задачи в терминальный статус;

• название — краткий текст для описания задачи;

• статус — для задач существует предопределенная (или пользовательская) статусная модель с допустимыми переходами между ними;

• тип задачи — предопределенный (или пользовательский) словарь различных типов задач. В основном выделяются следующие типы задач: «Ошибка», «Новая функциональность», «Эпик»;

• приоритет — предопределенный (или пользовательский) словарь различных уровней важности задачи;

• вложения — дополнительные материалы для конкретизации постановки задачи (это могут быть скриншоты, файлы с логами системы, документация системы и т.д.);

• комментарии — при необходимости конкретизировать постановку задачи пользователи могут общаться между собой для уточнения аспектов, если при этом используется комментарии, история общения остается привязанной к задаче и все договоренности будут фиксированы.

На основании представленных характеристик задачи можно ввести некоторые производные характеристики:

• количество слов в описании — в зависимости от типа задачи и принятых на проекте (или в организации практик), может быть корреляция между длиной описания и трудозатратами на выполнение задачи (например, если описание короткое, то это может означать как короткую задачу, так и сложную исследовательскую задачу без четких временных границ выполнимости);

• количество новых введенных сущностей — в методологии Domain Driven Development [2] почти каждое существительное, используемое заказчиком ПО относительно его понимания процесса, должно отражаться в коде в виде сущности, что ведет к увеличению трудозатрат. Однако в описании задачи могут встречаться не только термины бизнеса, но и общие термины разработки ПО, а также принятые внутри команды/проекта/организации шаблоны описания определенных типов задач или свойственные только конкретным авторам задач существительные;

• количество новых связей между сущностями — при употреблении нескольких существительных в рамках одного предложения/абзаца/задачи можно с некоторой долей вероятности утверждать, что в рамках выполнения задачи между этими терминами будет осуществляться добавление/редактирование/ удаление связи, что в свою очередь влияет на сложность выполнения задачи, и, как следствие, на трудозатраты.

При оценке количества новых сущностей и связей следует учитывать не только новизну этих объектов, но и как давно эти объекты были затронуты в последний раз. Известно, что человеческая память способна удерживать от 5 до 9 предметов в кратковременной памяти [3]. Похожий эффект есть и с долговременной памятью — сущности программы могут вытесняться другими сущностями, если прошло достаточно много времени. Процесс вытеснения дополнительно усугубляется тем, что над разработкой ПО обычно трудится целая команда, каждый из членов которой может внести правки в объекты, написанные другим человеком. В результате может оказаться так, что даже если прошло не так много времени с момента создания некоторых объектов, они были видоизменены настолько, что разбор их новой логики работы можно будет приравнять к реализации этого же объекта с нуля.

2. Вычислительный эксперимент

Был проведен вычислительный эксперимент по прогнозированию оценки трудозатрат на задачи с использованием методов машинного обучения, в частности многослойной нейронной сети для задачи нелинейной регрессии. Результаты эксперимента приведены на рис. 1.

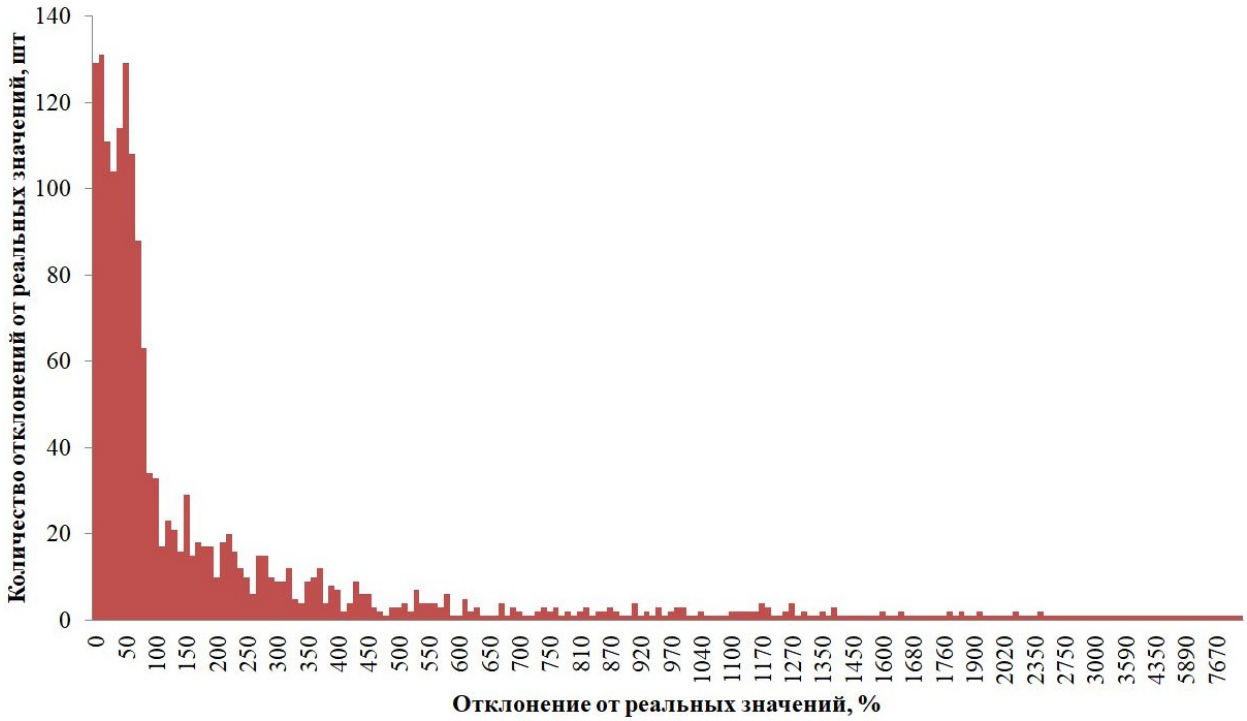


Рис. 1. Оценка количеств отклонений от реальных значений в процентах

Общий объем обучающей выборки составляет 1707 объектов. Количество объектов, на которых отклонение составило не больше 100 %, равно 1011. Максимальное отклонение от фактического результата составляет 17570 %, а минимальное — 0 %. Для практического применения предложенной модели необходимо понизить максимальное отклонение или научиться отделять такие задачи, на которых отклонение будет превышать некоторое пороговое значение, которое должно определяться на каждом проекте индивидуально или в рамках компании. В процессе проведения вычислительного эксперимента в обучающей выборке были найдены задачи, которые напрямую не относятся к разработке ПО, что негативно сказывается на качестве модели.

Выявлены следующие виды задач:

- разработка ПО;
- управление командой;
- тестирование ПО;
- подбор инструментов для разработки;
- написание документации;
- проведение совещаний;
- проведение сдачи проекта.

Для дальнейшего улучшения необходимо добавить автоматическую фильтрацию задач или очистить обучающие данные вручную.

Заключение

Результатом работы является приложение, которое позволяет провести оценку трудозатрат для задачи разработки ПО на основании исторических данных. На текущем уровне приложение не может применяться для оценки трудозатрат на практике из-за значительного отклонения от реальных значений. В дальнейшем планируется улучшить качество оценки путем добавления новых характеристик и улучшения качества данных.

Следует также заметить, что психологические аспекты, такие как оптимизм и предвзятость при оценке, также играют важную роль в данной проблеме, что делает ее интересной для междисциплинарного подхода.

Литература

1. Jira | Issue & Project Tracking Software | Atlassian. – Atlassian, 2024. – URL: https://www.atlassian.com/software/jira (дата обращения 09.09.2024).

2. Domain Driven Design на практике. – Habr, 2024. – URL: https://habr.com/ru/articles/334126/ (дата обращения 03.10.2024).

3. Семь плюс минус два (7 ± 2). Кошелек Миллера. – 4BRAIN, 2024. – URL: https://4brain.ru/blog/семь-плюс-минус-два/ (дата обращения 17.09.2024).

УДК 501,533, 004.021

МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ ОСАЖДЕНИЯ ДИСПЕРСНОЙ ФАЗЫ ДВУХФАЗНОЙ СРЕДЫ

Д. А. Тукмаков

Казанский научный центр РАН, Институт механики и машиностроения

Аннотация. В данной работе моделировались процессы гравитационного осаждения аэрозоля состоящего из частиц одинакового размера. Актуальность исследования связана с различными практическими приложениями осаждения дисперсной фазы газодисперсных сред. В работе получены математические модели, описывающие осаждение частиц на основе одномерных нестационарных уравнений диффузии с учетом и без учета конвективного слагаемого. Новизна работы заключается в том, что проводится сопоставление математических моделей, основанных на уравнения в частных производных и дискретной математической модели. Без учета, конвективного слагаемого гравитационное осаждение также происходит, но за существенно более долгий промежуток времени, чем при учете конвективного слагаемого. Также получена дискретная математическая модель осаждения частиц аэрозоля. Сопоставление демонстрирует, что наиболее интенсивно осаждение дисперсных частиц происходит по дискретной модели осаждения.
Ключевые слова: аэрозоли, гравитационное осаждение, уравнение диффузии, гравитационное осаждение, математическое моделирование.

Введение

Одним из разделов механики жидкости и газа является динамика неоднородных сред [1–14]. Частным случем неоднородных сред являются аэрозоли — взвешенные в газе твердые частицы или жидкие капли. В монографии [1] разработаны общие принципы теории динамики неоднородных сред. В монографии [2] численно иследованы одномерные течения моно и полидисперсных газовзвесей. В монографии [3] представлены различные математические модели течений запыленных сред. Монография [4] посвящена разработке теоретических основ осаждения дисперсных частиц. Результаты эксперементального наблюдения параметров седементации частиц позволяют определить свойства дисперсной системы [5]. В статье [11] математически моделируется гравитационная седементация дисперсных частиц. В исследовании [12] эксперементально изучается гравитационное осаждение аэрозоля в акустическом резонаторе. В работе [13] исследовано осаждение дисперсной фазы газовзвеси в аэродинамическом и электрическом поле.

В статье [14] рассматриваются вопросы прменения процессов диффузии в ядерной физике.

В данной работе проводится сопоставление математических моделей диффухии на освнове обыкновенного одномерного уравненя диффузии и уравнения диффузии с конвективным слагаемым, также предложена математическая модель предполагающая дискретность процесса осаждения дисперсных частиц.

Математическая модель

На рис. 1 представлено схематическое изображение гравитационного осаждения аэрозоля в емкости.

Для описания процесса изменения концентрации частиц применяется уравнение одномерной диффузии (1):

$$\frac{\partial c}{\partial t} = a\frac{\partial^2 c}{\partial x^2}. \qquad (1)$$

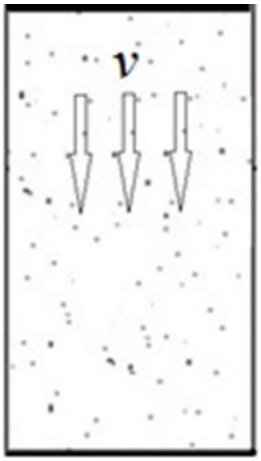


Рис. 1. Схематичное изображение емкости с осаждающимся аэрозолем

Уравнение (1) имеет частное решение (2), частное решение описывает постепенное уменьшение концентрации дисперсных частиц определяемое коэффициентом диффузии — a [15]:

$$c(x,t) = c_0 \exp\left(-ak^2 t\right)\cos(kx). \tag{2}$$

Одномерное уравнение конвективной диффузии имеет вид (3):

$$\frac{\partial c}{\partial t} = a\frac{\partial^2 c}{\partial x^2} - v\frac{\partial c}{\partial x}. \tag{3}$$

С помощью подстановки (4) [15] уравнение (3) приводится к уравнению (1) имеющему решение с учетом (4) в виде (5):

$$c^*(x,t) = \exp(2avx)\exp\left(tv^2/4a\right)c(x,t) \tag{4}$$

$$c(x,t) = c_0 \exp(-2avx)\exp\left(-tv^2/4a\right)\exp\left(-ak^2 t\right)\cos(kx) \tag{5}$$

Здесь k — подгоночный коэффициент, задаваемый из условия, что $c(x,0) \approx c_0$.

Скорость осаждения дисперсных включений определяется выражением (6) [4, 11]

$$v = \frac{gd^2(\rho - \rho_{10})}{18\mu}. \tag{6}$$

Здесь d — диаметр частицы, g — ускорение свободного падения ρ — плотность газа ρ_{10} — плотность материала частицы.

Коэффициент диффузии сферической частицы определяется выражением [5, 16]:

$$a = \frac{Tk}{3\pi\mu d}. \tag{7}$$

T — температура среды, k — постоянная Больцмана.

Общее количество частиц монодисперсной газовзвеси для емкости объема — V и объемного содержания дисперсной фазы — α можно предположить равным — N_p определяется формулой (8); время осаждение дисперсных частиц — t_s (9):

$$N_p = \frac{6\alpha V}{\pi d^3} \tag{8}$$

$$t_s = L/u. \tag{9}$$

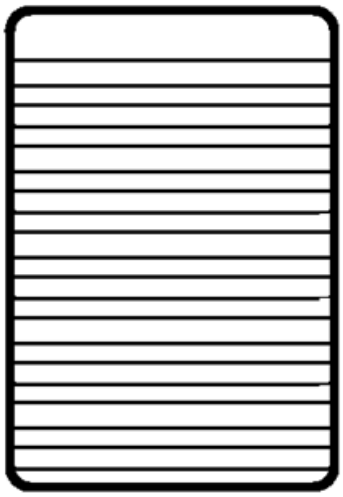


Рис. 2. Схематическое изображение слоев аэрозоля

Можно предположить, что в поперечном сечении в емкости дисперсные частицы распределены равномерно. Таким образом в емкости с аэрозолем образуется некоторое количество слоев, толщиной в диаметр одной частицы, расстояние между слоями также можно предположить равным одному диаметру частиц — рис. 2. Количество слоев — n определяется формулой (10), N_s — количество частиц в каждом слое.

$$n = L/2d \tag{10}$$

$$N_S = N_p/n. \tag{11}$$

В дискретной модели осаждения частиц предполагается, что за период времени t_{s0} (9*) осаждается N_s частиц:

$$t_{s0} = 2d/u. \tag{9*}$$

При описании процесса осаждения дисперсных частиц дифференциальными уравнениями количество осевших частиц определятся выражением (12):

$$N(t) = N_T - \alpha V \int_0^L c(x,t)\,dx. \tag{12}$$

Для моделей (1) и (3) скорости осаждения частиц определяются выражениями (13) и (14)

$$N(t) = N_T - \frac{6\alpha V e^{\left(-ak^2 t\right)} \sin(k)}{\pi d^3 k} \tag{13}$$

$$N(t) = N_T - \frac{6\alpha V}{\pi d^3}\left|\left(\frac{e^{-ak^2}}{a^2k^4+k^2}\right)\left(-ak^2\cos(k)+k\sin(k)\right)+\left(\frac{ak^2}{a^2k^4+k^2}\right)\right| e^{\left(-t\left(v^2/4a+ak^2\right)\right)}. \tag{14}$$

В расчетах предполагались следующие параметры несущей среды и дисперсной фазы — Физическая плотность дисперсной фазы $\rho_1 = 1000$ кг/м^3, диаметр частиц — $d = 5$ мкм, объемное содержание дисперсной фазы — $\alpha = 10^{-5}$. Длина области течения — $L = 1$ м, динамическая вязкость газа — $\mu = 1.72\ 10^{-5}$ Па·с, плотность несущей среды — $\rho = 1.29$ кг/м^3.

В начальный момент времени концентрации частиц, рассчитываемые по математической модели диффузии без учета конвективного слагаемого отличается несущественно, максимальное значение концентрации достигается вблизи нижней границы емкости, в последующие моменты времени распределению концентрации рассчитываемое по различным моделям существенно отличается — рис. 3 (*а–в*).

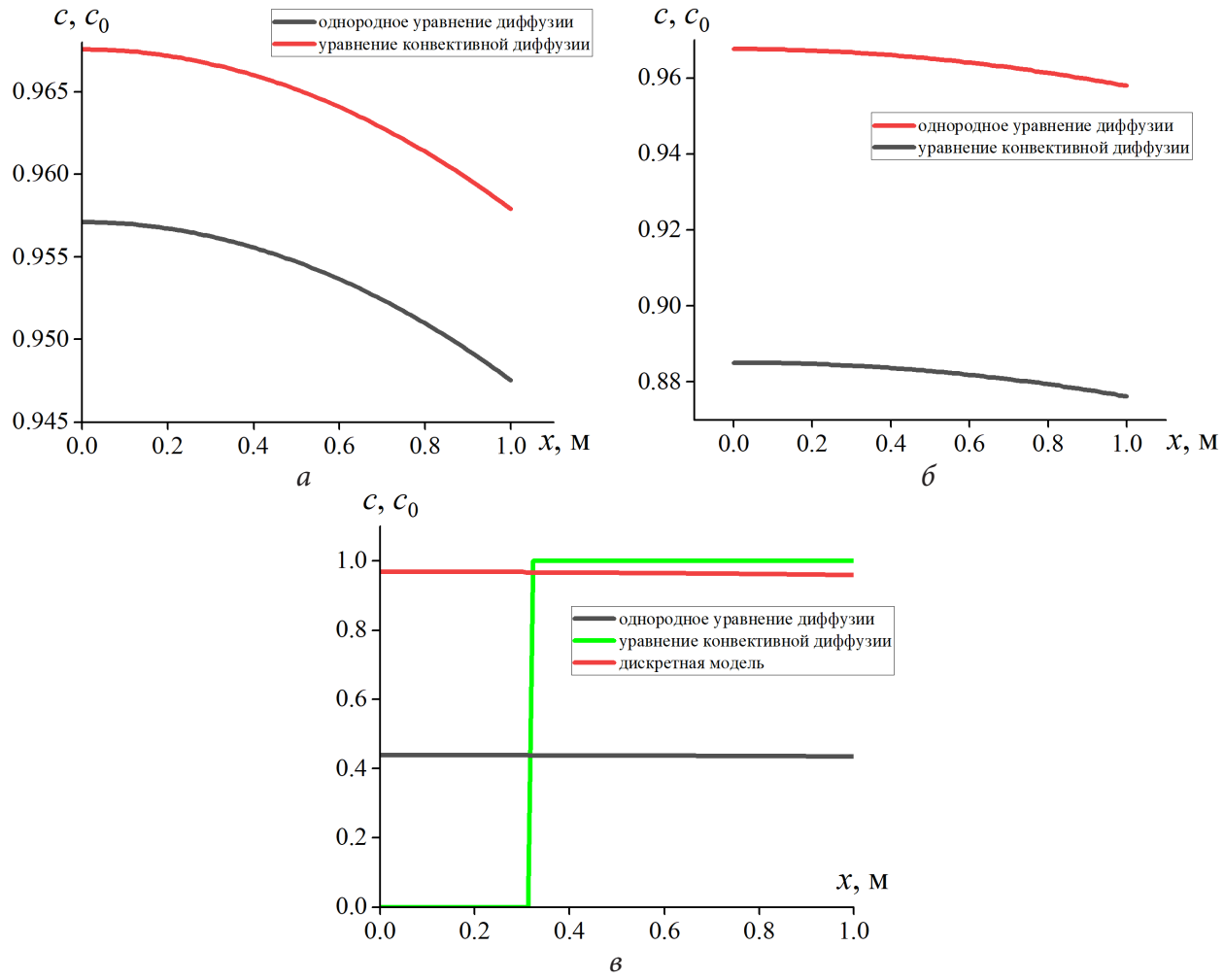


Рис. 3 Пространственные распределения концентрации частиц дисперсной фазы при расчете различными моделями в моменты времени: а — $t = 0.02t_S$; б — $t = 0.05t_S$; в — $t = 0.33t_S$

В расчетах отличие значений количества осевших частиц, за всё время осаждения при расчетах уравнение (1) и уравнением (3) составляет соответственно 90 % и 9 % количества осевших частиц по дискретной модели — рис. 4*а*. Полное осаждение описываемое однородным уравнением диффузии происходит за существенно более долгий период времни — рис. 4*б*.

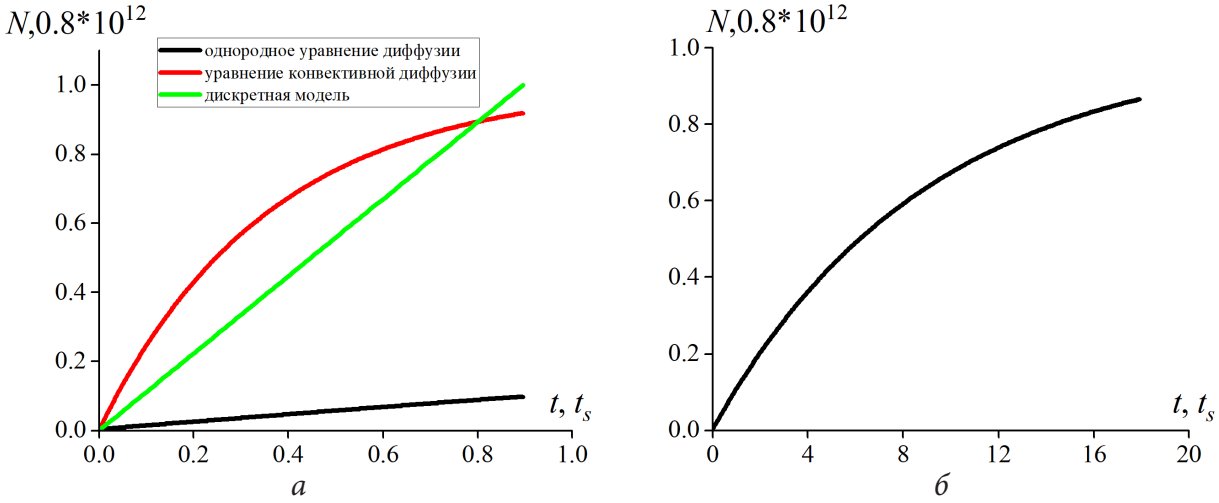


Рис. 4. Временные зависимости концентрации частиц для различных моделей расчета — а; временная зависимость для однородного уравнения диффузии — б

Заключение

В данной работе моделировались процессы гравитационного осаждения аэрозоля. Получены математические модели описывающие осаждение частиц с учетом и без учета конвективного слагаемого. Получена дискретная математическая модель осаждения частиц аэрозоля. Наиболее интенсивно осаждение частиц происходит при расчетах по дискретной математической модели.

Благодарности

Работа выполнена за счет гранта Академии наук Республики Татарстан, предоставленного молодым кандидатам наук (постдокторантам) с целью защиты докторской диссертации, выполнения научно-исследовательских работ, а также выполнения трудовых функций в научных и образовательных организациях Республики Татарстан в рамках Государственной программы Республики Татарстан «Научно-технологическое развитие Республики Татарстан.

Литература

1. *Нигматулин Р. И.* Основы механики гетерогенных сред / Р. И. Нигмутулин. – Москва : Наука, 1978. – 336 с.

2. *Кутушев А. Г.* Математическое моделирование волновых процессов в аэродисперсных и порошкообразных средах / А. Г. Кутушев. – Санкт-Петербург : Недра, 2003. – 284 с.

3. *Федоров А. В.* Волновые процессы в газовзвесях частиц металлов / А. В Федоров, В. М. Фомин, Т. А. Хмель. – Новосибирск : Параллель, 2015. – 301 с.

4. *Ходаков Г. С.* Седиментационный анализ высокодисперсных систем / Г. С. Ходаков, Ю. П. Юдкин. – Москва : Химия, 1981. –192 с.

5. *Леушина А. П.* Молекулярно-кинетические свойства дисперсных систем / А. П. Леушина, Д. Н. Данилов. – Седиментационный анализ. Киров : Издатльство ВятГУ, 2008. – 54 с.

6. *Тукмаков Д. А.* Трехмерная нестационарная математическая модель загрязнения канала осаждающейся дисперсной примесью / Д. А. Тукмаков // Экологические системы и приборы. –2022. – № 11. – С. 26–35.

7. *Тукмаков Д. А.* Исследование загрязнения водотока взвесью с помощью стационарной двухмерной математической модели / Д. А. Тукмаков // Вестник Пермского национального исследовательского политехнического университета. Прикладная экология. Урбанистика. – 2022. – Т. 45, № 1. – С. 88–98.

8. *Тукмаков Д. А.* Численная модель течения аэрозоля, обусловленного взаимодействием частиц и газа // Д. А. Тукмаков // Сложные системы. – 2021. – № 1. – С. 64–71.

9. *Тукмаков Д. А.* Аналитическая модель одномерной нестационарной динамики одиночной частицы в акустическом и электрическом полях / Д. А. Тукмаков // Лесной вестник. – 2022. – Т. 26, № 5. – С. 135–144.

10. *Тукмаков Д. А.* Сопоставление численных моделей динамики электрически заряженных газовзвесей с массовой и поверхностной плотностями зарядов для различных дисперсностей частиц / Д. А. Тукмаков // Вестник МГТУ им. Н. Э. Баумана. Серия Естественные науки. – 2022. – № 3. – С. 43–56.

11. *Студенов И. И.* Расчет гидравлической крупности взвеси при моделировании динамики концентрации взвешенных веществ в приустьевых районах арктических морей на примере Белого моря/ И. И. Студенов, Н. А. Шилова // Научные исследования в Арктике. – 2015. – № 3. – С. 40–47.

12. Осаждение дыма при нелинейных колебаниях в открытой трубе вблизи резонанса / Д. А. Губайдуллин, Р. Г. Зарипов, Л. А. Ткаченко, Л. Р. Шайдуллин // Теплофизика высоких температур. – 2019. – Т. 57, № 5. – С. 793–796.

13. *Тукмаков А. Л.* Модель движения и осаждения заряженной газовзвеси в электрическом поле / Д. А. Тукмаков // Инженерно-физический журнал. – 2014. – Т. 87, № 1. – С. 35–44.

14. Особенности формирования собственного электрического поля низкотемпературной кислород-метановой плазмы / А. В. Рудинский, А. В. Ягодников, С. В. Рыжков, В. В. Онуфриев // Письма в ЖТФ. – 2021. – Т. 47, № 10. – С. 42–45.

15. *Полянин А. Д.* Справочник по линейным уравнениям математической физики / А. Д. Полянин. – Москва : ФИЗМАТЛИТ, 2001. – 576 с.

16. *Ландау Л. Д.* Теоретическая физика. Гидродинамика / Л. Д. Ландау, Е. Ф. Лифшиц. – Москва : Наука, 1986. – 736 с.

УДК 517.9

СОЗДАНИЕ ВЕБ-ПРИЛОЖЕНИЯ «МЕССЕНДЖЕР» НА КЛИЕНТ-СЕРВЕРНОЙ АРХИТЕКТУРЕ

А. С. Украинский

Воронежский государственный университет

Аннотация. Рассматривается создание клиент-серверного приложения с помощью следующих технологий: Серверная часть: Java 17, Spring framework, а также компоненты Spring, такие как Spring Data JPA для работы с базами данных, Spring Web для работы с сетью. Клиентская часть: Javascript, ReactJS.
Ключевые слова: сервер, клиент, запрос, база данных, WebSocket, соединение, контроллер, метод, класс, аннотация.

Введение

Создание клиент-серверного приложения, такого как мессенджер, представляет собой увлекательную задачу, сочетающую в себе аспекты фронтенд- и бэкенд-разработки. В современном мире мессенджеры стали неотъемлемой частью повседневной жизни, обеспечивая быстрый и удобный способ общения. Однако их реализация требует глубокого понимания принципов работы клиент-серверной архитектуры, взаимодействия через API, обработки данных, а также обеспечения безопасности и масштабируемости.

В этой статье рассматривается процесс создания мессенджера с использованием современных технологий: Java и Spring Boot для серверной части и ReactJS для клиента. Также затрагиваются важные аспекты, такие как авторизация пользователей, организация чатов и обработка сообщений в реальном времени с помощью WebSocket. Пример поможет понять, как интегрировать эти компоненты в полноценное приложение, готовое к реальным нагрузкам.

Процесс создания приложения
Серверная часть

Первый этап создания приложения — подключение необходимых зависимостей. Java приложение имеет систему сборки Maven, поэтому для подключения зависимостей необходимо добавить в файл pom.xml следующий код:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-websocket</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
```

```
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Добавлены зависимости для создания подключения по протоколу WebSocket, для работы с базой данных на основе объектно-реляционной модели, а также для подключения серверного приложения к СУБД Postgres.

Далее необходимо создать классы, представляющие собой модели данных.

```
@Entity
@Table(name = "users")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    private String password;
}

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class FriendRequest {

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

    @ManyToOne
    private User user;

    @ManyToOne
    private User friend;

    private boolean accepted;
}

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Message {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
    @ManyToOne
    private User sender;

    @ManyToOne
    private User recipient;

    private String content;
}
```

Аннотация @Entity — то, что позволяет программе преобразовать обычный класс в запись таблицы базы данных и обратно. @Id показывает, что помеченное данной аннотацией поле является первичным ключом таблица базы данных. Аннотация @ManyToOne позволяет создать связь «многие к одному» между таблицами.

Помимо этого для работы с базой данных необходимо создать интерфейсы-репозитории.

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);

    List<User> findByUsernameContainingIgnoreCase(String username);

    @Query(value = """
        select new User(u.id, u.username, u.password) from User u
        join FriendRequest fr on fr.friend.id = u.id or
        fr.user.id = u.id
         where (fr.friend.id = ?1 or fr.user.id = ?1) and u.id != ?1 and accepted
= true""")
    List<User> findFriendsById(String id);
}
```

Методы `findByUsername()` и `findByUsernameContainingIgnoreCase()` представляют собой select-запросы к базе данных, написанные на языке JPQL. Это мощный язык запросов, который позволяет определять запросы к базе данных на основе модели объекта. JPQL является надстройкой над SQL, который позволяет более удобно работать с базой данных. Например, первый метод можно заменить на запрос:

```
select * from users u where u.username = username;
```

Второй метод — запрос для поиска пользователей, содержащих в логине заданную строку.

Третий метод помечен аннотацией @Query. Дело в том, что язык JPQL ограничен в своих возможностях и не позволяет писать длинные сложные запросы. В таком случае используется @Query. Эта аннотация связывает метод с явно заданным запросом, что позволяет создавать запросы к базе данных различной сложности. В данном случае реализован запрос с использованием join для объединения двух таблиц и поиска друзей пользователя с заданным id.

```
@Repository
public interface FriendRequestRepository extends JpaRepository<FriendRequest,
Long> {
    List<FriendRequest> findByFriendIdAndAcceptedFalse(Long id);

    List<FriendRequest> findByUserAndAcceptedTrue(User user);

    List<FriendRequest> findByUser(User user);

    Optional<FriendRequest> findByUserAndFriendAndAcceptedFalse(User user, User
friend);

    @Transactional
    void deleteAllByUserIdAndFriendId(Long userId, Long friendId);
}
```

Благодаря аннотации `@Transactional` метод `deleteAllByUserIdAndFriendId()` выполняется в транзакции.

```
@Repository
public interface MessageRepository extends JpaRepository<Message, Long> {
    List<Message> findAllBySenderIdAndRecipientId(Long senderId, Long recipientId);
}
```

Помимо приведенных выше методов, интерфейсы, расширяющие JpaRepository содержат базовые методы, необходимые для работы с базой данных, такие как: `save()`, `findById()`.

На следующем этапе для соблюдения объектно-ориентированной модели и принципа единой ответственности создаются классы-сервисы. Такие классы содержат бизнес-логику приложения и являются прослойкой между контроллерами и репозиториями.

```
@Service
@RequiredArgsConstructor
public class UserService {
    private final UserRepository userRepository;
    public User register(String username, String password) {
        if (userRepository.findByUsername(username).isPresent()) {
            throw new RuntimeException("Username is already taken");
        }
        User user = new User(null, username, password);
        return userRepository.save(user);
    }
    public User login(String username, String password) {
        return userRepository.findByUsername(username)
                .filter(u -> u.getPassword().equals(password))
                .orElseThrow(() -> new RuntimeException("Invalid credentials"));
    }
    public List<User> searchUsers(String username) {
        Return userRepository.findByUsernameContainingIgnoreCase(username);
    }
    public List<User> searchFriends(String id) {
        return userRepository.findFriendsById(id);
    }
    public Optional<User> getUser(Long id) {
        return userRepository.findById(id);
    }
}
```

Описание методов данного класса:

`register()` — поиск пользователя по логину. Если пользователь найден — вернуть ошибку, иначе создать нового пользователя.

`login()` — поиск пользователя по логину и паролю. Если пользователь не найден - вернуть ошибку.

`searchUsers()` — поиск всех пользователей, чьи логины содержат заданную строку.

`searchFriends()` — поиск пользователей, являющихся друзьями пользователя, имеющего заданное id.

`getUser()` — поиск пользователя по id.

```
@Service
@RequiredArgsConstructor
public class FriendRequestService {
```

```
    private final FriendRequestRepository friendRequestRepository;
    private final UserRepository userRepository;

    public void sendFriendRequest(Long userId, Long friendId) throws
RuntimeException {
        if (userId.longValue() == friendId.longValue()) {
            throw new RuntimeException("You can’t send friend request to
yourself");
        }

        Optional<User> user = userRepository.findById(userId);
        Optional<User> friend = userRepository.findById(friendId);
        user.orElseThrow();
        friend.orElseThrow();
        List<FriendRequest> requests = friendRequestRepository.findByUser(user.
get());
        if (requests.stream().anyMatch(fr -> fr.getFriend().equals(friend.get())
&& fr.isAccepted())) {
            throw new RuntimeException(«Already friends»);
        }
        if (requests.stream().anyMatch(fr -> fr.getFriend().equals(friend.get())
&& !fr.isAccepted())) {
            throw new RuntimeException("Friend request already sent");
        }

        FriendRequest request = new FriendRequest(null, user.get(), friend.get(),
false);
        friendRequestRepository.save(request);
    }

    public void acceptFriendRequest(Long requestId) {
        Optional<FriendRequest> request = friendRequestRepository.
findById(requestId);
        if (request.isEmpty()) {
            return;
        }

        request.get().setAccepted(true);
        friendRequestRepository.save(request.get());
    }

    public void rejectFriendRequest(Long requestId) {
        friendRequestRepository.deleteById(requestId);
    }

    public List<FriendRequest> getNonAcceptedRequests(Long id) {
        return friendRequestRepository.findByFriendIdAndAcceptedFalse(id);
    }

    public void deleteFriend(Long userId, Long friendId) {
        friendRequestRepository.deleteAllByUserIdAndFriendId(userId, friendId);
    }
}
```

Описание методов данного класса:

`sendFriendRequest()` — Отправить запрос в друзья. Если id отправителя и получателя совпадают — выдать ошибку. Если хотя бы по одному из id не найдено пользователя — выдать

ошибку. Если пользователи уже являются друзьями или для них сохранена не подтвержденная заявка — выдать ошибку.

`acceptFriendRequest()` — подтвердить заявку в друзья. Если заявка по заданному id не найдена — вернуть ошибку.

`rejectFriendRequest()` — отклонить заявку в друзья.

`getNonAcceptedRequests()` — поиск заявок в друзья, отправленных пользователю с заданным id.

`deleteFriend()` — удалить заявку в друзья между двумя пользователями.

```
@Service
public class MessageService {

    private final MessageRepository messageRepository;

    public MessageService(MessageRepository messageRepository) {
        this.messageRepository = messageRepository;
    }

    public Message saveMessage(Message message) {
        return messageRepository.save(message);
    }
}
```

`save()` — сохранить сообщение.

Теперь необходимо создать контроллеры. Контроллер - класс, содержащий методы, которые получают и обрабатывают запросы на сервер. Так как в данном случае сервер получает запросы и возвращает данные в формате JSON, классы необходимо пометить аннотацией @RestController.

```
@RestController
@RequestMapping("/api/users")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

    @GetMapping("/get")
    public ResponseEntity<?> get(@RequestParam String id) {
        Optional<User> user = userService.getUser(Long.parseLong(id));
        if (user.isPresent()) {
            return new ResponseEntity<>(user.get(), HttpStatus.OK);
        } else {
            return new ResponseEntity<>("User not found", HttpStatus.NOT_FOUND);
        }
    }
    @PostMapping("/register")
    public ResponseEntity<User> register(@RequestBody User user) {
         return ResponseEntity.ok(userService.register(user.getUsername(), user.
getPassword()));
    }
    @PostMapping("/login")
    public ResponseEntity<User> login(@RequestBody User user) {
           return ResponseEntity.ok(userService.login(user.getUsername(), user.
getPassword()));
    }
```

```
    @GetMapping("/search")
    public ResponseEntity<List<User>> search(@RequestParam String query) {
        return ResponseEntity.ok(userService.searchUsers(query));
    }
    @GetMapping("/friends")
    public ResponseEntity<List<User>> searchFriends(@RequestParam String id) {
        return ResponseEntity.ok(userService.searchFriends(id));
    }
}
```

`@RequestMapping` — задает префикс url-адреса, по которому данный класс может получать запросы.

`@PostMapping` и `@GetMapping` — показывают, что метод соответствует HTTP методу POST или GET и задают конечную точку url-адреса. Например, запрос, отправленный с локальной машины на url `http://localhost/api/users/get` будет обработан методом `get()` из этого класса.

Параметры методов, помеченные аннотацией `@RequestParam` должны передаваться в url адресе в таком виде: `http://localhost/api/users/get?id=12345`.

Параметры, помеченные аннотацией @RequestBody должны передаваться в теле запроса. Например, чтобы передать объект класса User, тело запроса должно иметь вид:

```
{
    "id": 123,
    "username": "User1",
    "password": "1Qwerty"
}

@RestController
@RequestMapping("/api/friend-requests")
@RequiredArgsConstructor
public class FriendRequestController {
    private final FriendRequestService friendRequestService;
    @PostMapping("/send")
    public ResponseEntity<?> send(@RequestParam String senderId, @RequestParam
String recipientId) {
        try {
            friendRequestService.sendFriendRequest(Long.parseLong(senderId),
Long.parseLong(recipientId));
        } catch (NoSuchElementException e) {
            return new ResponseEntity<>(«User not found», HttpStatus.INTERNAL_
SERVER_ERROR);
        } catch (RuntimeException e) {
            return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
        return new ResponseEntity<>(HttpStatus.OK);
    }
    @GetMapping("/get")
    public ResponseEntity<List<FriendRequest>> get(@RequestParam String id) {
        return ResponseEntity.ok(friendRequestService.getNonAcceptedFalseRequests
(Long.parseLong(id)));
    }
```

```
    @PostMapping("/accept")
    public ResponseEntity<?> accept(@RequestParam String requestId) {
        friendRequestService.acceptFriendRequest(Long.parseLong(requestId));
        return new ResponseEntity<>(HttpStatus.OK);
    }
    @PostMapping("/reject")
    public ResponseEntity<?> reject(@RequestParam String requestId) {
        friendRequestService.rejectFriendRequest(Long.parseLong(requestId));
        return new ResponseEntity<>(HttpStatus.OK);
    }
    @PostMapping("/delete")
    public ResponseEntity<?> delete(@RequestParam String userId, String friendId)
{
        friendRequestService.deleteFriend(Long.parseLong(userId), Long.
parseLong(friendId));
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
@RestController
@RequiredArgsConstructor
public class MessageController {
    private final SimpMessagingTemplate template;
    private final MessageService messageService;
    @GetMapping
    @ResponseBody
    public List<Message> getMessages(@RequestParam String user1, @RequestParam
String user2) {
        return messageService.getMessagesBetweenUsers(user1, user2);
    }
    @PostMapping
    @ResponseBody
    public Message sendMessage(@RequestBody Message message) {
        return messageService.saveMessage(message);
    }
    @MessageMapping("/send")
    public ResponseEntity<?> send(@Payload Message message) {
        messageService.saveMessage(message);
        System.out.println(message);
        template.convertAndSendToUser(
                message.getRecipient().toString(),
                "/messages",
                message);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Методы контроллеров получают запросы на сервер с необходимыми параметрами, после чего вызываются методы соответствующего сервиса для обработки запроса. Сервис в свою очередь при необходимости обращается к базе данных, чтобы сохранить новые данные и получить уже имеющиеся.

В серверную часть осталось добавить возможность создания соединения по протоколу WebSocket. Дело в том, что обычный протокол HTTP работает по принципу вопрос-ответ. То есть сервер может отправить данные на клиент только в ответ на запрос. Такой вариант не подходит для реализации мессенджера. Протокол WebSocket позволяет отправлять данные на клиент не дожидаясь запроса.

Класс `WebSocketConfig` является конфигурационным классом для настройки WebSocket в Spring Boot. Он использует аннотацию `@EnableWebSocketMessageBroker`, которая включает поддержку WebSocket и брокера сообщений на основе STOMP (Simple/Streaming Text Oriented Messaging Protocol).

I. `configureMessageBroker(MessageBrokerRegistry registry)`

Этот метод настраивает брокер сообщений, который управляет маршрутизацией сообщений между клиентами через сервер.

Подробности:

1. `registry.enableSimpleBroker("/topic")`

• Включает встроенный брокер сообщений (in-memory broker), который будет обрабатывать сообщения, направленные на каналы с префиксом `/topic`.

• **Пример:** Если клиент подписывается на `/topic/chat`, он будет получать все сообщения, отправленные в этот канал.

• Это удобно для рассылки сообщений нескольким клиентам одновременно (например, уведомлений или сообщений в группах).

2. `registry.setApplicationDestinationPrefixes("/app")`

• Указывает префикс для сообщений, которые отправляются с клиента на сервер.

• **Пример:** Если клиент отправляет сообщение на /app/sendMessage, сервер обрабатывает его с помощью метода, аннотированного `@MessageMapping("/sendMessage")`.

3. `registry.setUserDestinationPrefix(«/topic/{userId}/messages»)`

• Устанавливает префикс для сообщений, направленных конкретным пользователям (point-to-point messaging).

• **Пример:** Сообщение отправляется на /topic/123/messages, где 123 — идентификатор пользователя. Только пользователь с этим идентификатором получит сообщение.

• Это полезно для реализации приватных сообщений.

II. `registerStompEndpoints(StompEndpointRegistry registry)`

Этот метод регистрирует конечные точки **WebSocket**, которые клиенты используют для подключения к серверу.

Подробности:

1. `registry.addEndpoint(«/ws»)`

• Регистрирует WebSocket-эндпоинт по адресу /ws, к которому клиенты могут подключаться.

• **Пример:** Клиент будет устанавливать соединение с сервером через WebSocket на ws://localhost:8080/ws.

2. `setAllowedOriginPatterns("*")`

• Разрешает подключение с любых источников (все домены и порты). Это используется для предотвращения проблем с CORS.

3. `withSockJS()`

• Включает поддержку SockJS, который используется как резервный механизм на случай, если WebSocket недоступен (например, в старых браузерах).

• SockJS автоматически переключается на другие протоколы, такие как AJAX Long Polling или XHR Streaming, чтобы обеспечить стабильную связь.

Заключение

Изучение динамики клеточных взаимодействий при различных значениях параметра η предоставляет важные сведения о том, как изменения в микроокружении могут влиять на поведение различных типов клеток. Этот анализ имеет особое значение для понимания процессов, связанных с онкологией, регенерацией тканей и патогенезом различных заболеваний.

Параметр η, скорее всего, отражает скорость взаимодействия между клетками или уровень их агрессивности. При малых значениях η (например, 0.1) системы остаются сбалансированными, где здоровые клетки способны сохранять свою численность без значительного влияния со стороны лейкозных клеток. Это может быть связано с тем, что в условиях низкой агрессивности лейкозные клетки не имеют достаточной силы, чтобы доминировать над здоровыми клетками. В такой среде здоровые клетки могут активно делиться и поддерживать свое количество, что важно для нормального функционирования организма.

С увеличением значения η (до 0.2 и 0.4) наблюдается изменение динамики: число лейкозных клеток начинает расти, а здоровые клетки начинают подвергаться более значительному стрессу. Это подчеркивает, что даже небольшие изменения в микросреде могут значительно повлиять на здоровье организма. Высокий уровень η может означать, что клетки начинают конкурировать за ресурсы, такие как питательные вещества и пространство, что ведет к ухудшению состояния здоровых клеток.

При ($\eta = 0.4$) мы видим, что лейкозные клетки становятся основным источником угрозы для здоровых клеток. Это может объясняться активной саморазмножающейся природой лейкозных клеток, которые начинают вытеснять здоровые клетки, что приводит к угнетению нормальных гематопоэтических процессов. Такие условия могут также способствовать развитию дополнительных мутаций и активизации других механизмов, которые еще больше усиливают малигнизацию клеток.

Наконец, при высоких значениях η (как 0.8) мы можем говорить о критической ситуации, когда лейкозные клетки практически полностью доминируют. Их присутствие может подавлять иммунный ответ организма, что делает его более уязвимым к инфекциям и другим заболеваниям. В таких условиях нормальная гомеостаза нарушается, и организм не может восстановить утраченные функции. Это драматический пример того, как высоко агрессивная среда приводит к быстрой эволюции злокачественных клеток, которые становятся устойчивыми к терапии и препятствуют нормальному функционированию организма.

Таким образом, исследование влияния параметра η на динамику клеточных взаимодействий является ключевым моментом в понимании сложных биологических систем. Эти знания могут не только помочь в лечении заболеваний, но и расширить наши представления о клеточной биологии, иммунологии и онкологии. В конечном итоге, глубокое понимание этих процессов может улучшить качество жизни пациентов и принести новые подходы в медицинскую практику.

Благодарности

Выражаю особую благодарность моему научному руководителю Половинкину Игорю Петровичу, доценту, доктору физико-математических наук, за значимые замечания и важнейшие советы при проведении исследования и оформления данной статьи.

Литература

1. *Глушков В. Г., Сергеев А. А.* Численные методы решения СЛАУ: Метод сеток и его применения. – Москва : Наука, 2021. – 280 с.

2. Математическая модель лейкоза [Электронный ресурс]. URL: https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=irj&paperid=225&option_lang=rus (дата обращения: 01.10.2024).

3. Метод конечных разностей [Электронный ресурс]. URL: https://portal.tpu.ru/SHARED/a/ALEX1479/study/Matmod/Tab1/Tab/11%20%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F%20%D0%9C%D0%B0%D1%82_%D0%BC%D0%BE%D0%B4_%D1%84%D0%B8%D0%B7_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D0%B2.pdf (дата обращения 09.11.2024).

4. *Смирнова Е. Е., Кузнецов В. В.* Применение дифференциальных уравнений в биомедицинских исследованиях. — Новосибирск : Сибирское университетское издательство, 2019. – 300 с.

5. *Ковалев Д. Д.* Исследование моделей лейкемии с использованием численных методов. // Журнал прикладной математики. – 2023. – Т. 58, № 2. – С. 45–56.

6. *Глушков В. Г., Сергеев А. А.* Численные методы решения СЛАУ: Метод сеток и его применения. – Москва : Наука, 2021. – 280 с.

7. *Кузнецов И. И., Петрова Н. Н.* Методы численного решения линейных уравнений. – Санкт-Петербург : БХВ-Петербург, 2020. – 320 с.

8. *Романов В. В.* Алгоритмы и программирование методов решения СЛАУ. – Новосибирск: Сибирское университетское издательство, 2023. – 180 с.

9. *Fuentes-Garí M.* A mathematical model of subpopulation kinetics for the deconvolution of leukaemia heterogeneity / M. Fuentes-Garí, R. Misener, D. García-Munzer and others // Journal of the Royal Society Interface. – 2015. – V. 12, № 108. – 20150276. – doi:10.1098/rsif.2015.0276.

10. *Berezansky L.* Stability and Controllability Issues in Mathematical Modeling of the Intensive Treatment of Leukemia / L. Berezansky, S. Bunimovich-Mendrazitsky, B. Shklyar // Journal of Optimization Theory and Applications. – 2015. – V. 167, № 1. – P. 326–341. – doi:10.1007/s10957-015-0717-9.

11. *Rădulescu I. R.* A complex mathematical model with competition in Leukemia with immune response – An optimal control approach / I. R. Rădulescu, D. Cândea, A. Halanay // IFIP Advances in Information and Communication Technology. – 2016. – V. 494. – P. 430–441. – doi:10.1063/1.4765573

12. *Bunimovich-Mendrazitsky C.* Mathematical model of pulsed immunotherapy for bladder cancer / C. BunimovichMendrazitsky, H. Byrne, L. Stone // Bulletin of mathematical biology. – 2008. – V. 70, № 7. – P. 2055–2076. – doi:10.1007/s11538-008-9344-z.

13. *Clapp G.* A review of mathematical models for leukemia and lymphoma / G. Clapp, D. Levy // Drug Discovery Today: Disease Models. – 2015. – V. 16, № 2. – P. 1–6.

14. *Kolpak E. P.* Neoplasm Morbidity among the Population of Russia / E. P. Kolpak, I. S. Frantsuzova, K. V. Kuvshinova // International journal of advanced biotechnology and research. – 2017. – V. 8, № 3. – P. 2315–2322.

15. *Жукова И. В.* Математические модели злокачественной опухоли / И. В. Жукова, Е. П. Колпак // Вестник Санкт-Петербургского университета. Серия 10: Прикладная математика. Информатика. Процессы управления. – 2014. – № 3. – С. 5–18.

УДК 004.056.2

ПРОТОКОЛ RPC И ЕГО РЕАЛИЗАЦИИ КАК СРЕДСТВА ОБЕСПЕЧЕНИЯ УДАЛЕННОГО ВЗАИМОДЕЙСТВИЯ УЗЛОВ В СЕТИ

К. С. Ухина

Воронежский государственный университет

Аннотация. В статье рассматривается средства обеспечения удаленного вызова процедур на примере реализаций протокола RPC.
Ключевые слова: удаленный вызов процедур, RPC, gRPC, RMI, HTTP/2, заглушка, маршалинг/демаршалинг, Protocol Buffer.

Задачи протокола RPC (Remote procedure Call)

Протокол RPC позволяет программам вызывать процедуры или функции, которые находятся на удаленном узле. При этом создается иллюзия, что процедура выполняется локально. Таким образом решаются следующие задачи:

1. Взаимодействие программ, которые написаны на разных языках.
2. Распределение вычислений между несколькими узлами.
3. Возможность повторного использования кода.
4. Возможность разрабатывать приложения в виде набора распределенных модулей.
5. Распределенное тестирование приложения на различных платформах.

Проблемы, возникающие при использовании протокола RPC:

1. Задержки в сети могут снизить производительность, увеличивая время отклика.
2. Сбои сети могут привести к потере запроса или ответа.
3. Обеспечение совместимости между клиентской и серверной версиями может быть сложной задачей при обновлении или обслуживании системы

Принцип работы протокола RPC

Маршалинг — это конвертирование данных и аргументов функции в формат, подходящий для передачи по сети.

Демаршалинг — это обратный процесс, при котором переданные данные преобразуются в значимые аргументы.

Заглушка — процедура, которая вызывается вместо фактического кода, реализующего логику. Заглушка выполняет маршалинг/демаршалинг данных и отправляет запрос/ответ между узлами.

Алгоритм выполнения удаленной процедуры (рис. ниже):

1. Процесс на клиентской машине вызывает процедуру-заглушку клиента.
2. Клиентская заглушка упаковывает данные о процедуре — имя и параметры — в структуру для отправки по сети.
3. Управление передается ядру ОС клиента, которое дополняет сообщение заголовками и метками, в которых содержится информация об адресе клиента, типе сообщения (запрос/ответ), идентификатор процедуры.
4. Сообщение передается ядру ОС сервера, которое передает управление серверной заглушке.
5. Серверная заглушка выполняет демаршалинг данных, распаковывает имя процедуры и параметры.

6. Данные передаются процессу на сервере, который выполняет процедуру.

7. Результат процедуры передается в обратном направлении по цепочке от серверной заглушки до клиентской [1].

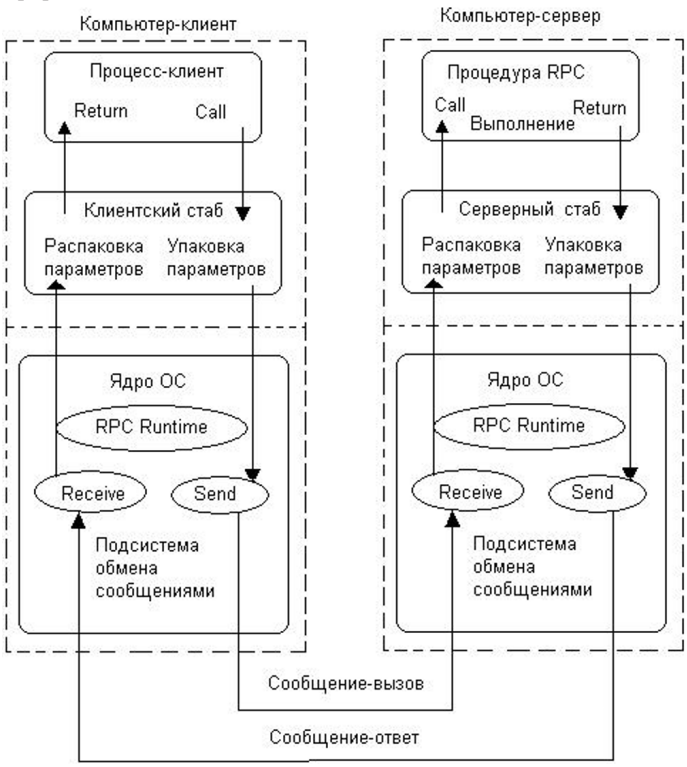


Рис. Схема выполнения удаленной процедуры

Обзор технологий для реализации удалённого взаимодействия между приложениями

• DCE/RPC — двоичный протокол на базе различных транспортных протоколов, в том числе TCP/IP и Named Pipes из протокола SMB/CIFS;

• DCOM — объектно-ориентированное расширение DCE/RPC, позволяющее передавать ссылки на объекты и вызывать методы объектов через таковые ссылки;

• Microsoft RPC;

• gRPC;

• ZeroC ICE;

• JSON-RPC — текстовый протокол на базе HTTP;

• .NET Remoting — двоичный протокол на базе TCP, UDP, HTTP;

• Java RMI — вызов удалённых методов для платформы Java;

• SOAP — текстовый протокол на базе HTTP;

• Sun RPC — двоичный протокол на базе TCP и UDP и XDR;

• XML RPC — текстовый протокол на базе HTTP.

Особенности фреймворка gRPC

GRPC — RPC-фреймворк с открытым исходным кодом, разработанный Google. Фреймворк поддерживается многими языками, что дает возможность разрабатывать клиент и сервер на разных языках.

В качестве транспортного протокола gRPC использует HTTP/2. По сравнению с HTTP/1.1 HTTP/2 повышает общую производительность, позволяя нескольким запросам и ответам передаваться параллельно в рамках одного соединения [4].

Для описания вида структурированных данных используется Protocol Buffers — язык описания механизма сериализации. Структура данных, которая будет использоваться для отправки сообщений между узлами, а так же сервисы с методами, которые будут вызываться, описываются в файлах специального типа *.proto. Из таких файлов генерируются клиентская и серверная заглушки, которые далее будут использоваться для реализации логики приложения [5]. Пример описания сервиса и метода в файле *.proto представлен ниже.

```
service SaveService{
    rpc SaveUserToDB(UserRequest) returns SaveResponse;
}
//тип параметра метода
message UserRequest{
    string name = 1;
    string age = 2;
}
//тип возвращаемого значения метода
message SaveResponse{
    boolean result = 1;
}
```

Из файлов *.proto генерируются Java файлы. В результате генерации будут получены следующие файлы:

- *UserRequest.java* — определение типа *UserRequest*;
- *SaveResponse.java* — определение типа *SaveResponse*;
- *SaveServiceImplBase.java* — абстрактный класс *SaveServiceImplBase*, который обеспечивает реализацию всех операций, которые определили в интерфейсе сервиса.

Недостатки использования фреймворка gRPC:

1. Сложность освоения из-за специфичности protobuf и HTTP/2;
2. Не поддерживается напрямую большинством браузеров, поэтому существует необходимость в использовании прокси;
3. Ограниченность форматом Protocol Buffer;
4. Необходимость настроить HTTP/2 на уровне сетевого интерфейса [5].

Особенности технологии RMI

RMI (Remote Method Invocation) — механизм взаимодействиями между двумя Java-машинами, позволяющий вызывать удаленные методы объекта.

Основные принципы RMI:

1. Объекты реализуют специальные интерфейсы (Remote), которые определяют методы, доступные для удалённого вызова.
2. Параметры метода и возвращаемое значение должны быть подвержены маршалингу.
3. Для того чтобы клиентский код мог найти удалённые объекты, RMI предоставляет реестр удалённых объектов.

Реестр RMI — это пространство имен, в котором размещаются все объекты сервера. Каждый раз, когда сервер создает объект, он регистрирует его в RMIregistry. Объекты зарегистрированы под уникальным именем (bind name) [3].

Одна из центральных и уникальных функций RMI является возможность загрузки определение class объекта, если class не определяется в виртуальной машине Java получателя. Все типы и поведение объекта, ранее доступного только в единственной виртуальной машине Java, могут быть переданы другой.

В RMI *удаленный* интерфейс — это интерфейс, который объявляет набор методов, которые могут быть вызваны с удалённой виртуальной машины Java. Удаленный интерфейс должен соответствовать следующим требованиям:

• Удаленный интерфейс должен наследовать, прямо или косвенно, интерфейс java.rmi. Remote.

• Каждое объявление метода в удалённом интерфейсе или его супер-интерфейсах должно соответствовать требованиям объявления *удалённого метода*, как показано ниже:

– Объявление удалённого метода должно включать исключение java.rmi.RemoteException;

– В объявлении удалённого метода удалённый объект, объявленный в качестве параметра или возвращаемого значения должен быть объявлен как удалённый *интерфейс*, а не как класс реализации этого интерфейса.

```
public interface HelloService extends java.rmi.Remote {
        public void sayHello(String name)
                        throws java.rmi.RemoteException;
}
```

Для клиента библиотека RMI будет динамически создавать заглушку. Далее на сервере необходимо описать реализацию удаленного интерфейса и добавить ее в реестр, чтобы клиент смог вызвать реализацию, используя заглушку. Получение из реестра заглушки и вызов метода представлен ниже [3].

```
Registry registry = LocateRegistry.getRegistry();
HelloService server = (HelloService) registry.lookup("HelloService");
server.sayHello("World");
```

Недостатки использования RMI:

1. Поддержка только одного языка программирования JAVA.
2. Собственный протокол взаимодействия.
3. Трудность интегрирования с существующими приложениями.
4. Плохая масштабируемость.

Заключение

При выборе метода и средств обеспечения удаленного взаимодействия необходимо опираться на целевую задачу. Каждый подход имеет свои особенности, которые могут пригодиться в той или иной ситуации. Так, например, RMI подходит для взаимодействия между Java-приложениями, однако если существует необходимость использовать другой язык программирования, можно рассмотреть другие подходы взаимодействия.

Литература

1. Что такое RPC удаленного вызова процедуры в ОС? : [сайт]. – 2023. – URL: https://andreyex.ru/stati/chto-takoe-rpc-udalennogo-vyzova-protsedury-v-os/ (дата обращения 10.10.2014).

2. Удалённый вызов процедур : [сайт]. – 2023. – URL:https://ru.wikipedia.org/wiki/Удалённый_вызов_процедур (дата обращения 10.10.2014).

3. *Esmond Pitt, Kathy McNiff*. Java.rmi: The Remote Method Invocation Guide . – Addison-Wesley; First Edition, 2001. – 320 с.

4. Introduction to gRPC : [сайт]. – 2024. – URL: https://www.baeldung.com/grpc-introduction (дата обращений 13.10.2024).

5. *Kasun Indrasiri, Danesh Kuruppu*. gRPC: Up and Running . – O'Reilly Media, Inc., 2020. – 204 с.

УДК 004.415.2

ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ХИМЧИСТКИ

Е. П. Хворостинина, М. В. Матвеева

Воронежский государственный университет

Аннотация. В работе представлена задача проектирования и разработки клиентской части веб-приложения для химчистки, позволяющего клиентам самостоятельно заполнить форму квитанции.
Ключевые слова: веб-приложение, приложение, химчистка, квитанция, приложение для химчистки, онлайн заказ.

Введение

В сфере химчисток оформление заказов часто сопряжено с длительными процедурами и ручным заполнением квитанций. Отсутствие удобного инструмента для самостоятельного составления и проверки шаблонов квитанций замедляет процесс обслуживания клиентов и увеличивает трудозатраты персонала.

В настоящий момент не существует ни одного веб-приложения, где пользователь мог бы сам составить шаблон квитанции, который уже впоследствии проверялся бы и корректировался сотрудником химчистки. Создание приложения, позволяющего заполнять клиентские квитанции, ускорило бы процесс приема и оформления заказа. Пользователь сможет просматривать статус заказа, а сотрудники будут тратить меньше времени на создание и заполнение клиентской квитанции.

1. Анализ существующих приложений

Был проведен анализ имеющихся веб-приложений химчисток города Воронежа, а именно София, Стиралкин и Wash of Fame [1–3].

София предоставляет базовую функциональность: просмотр предлагаемых услуг, цен на эти услуги, проводимых акций, адресов пунктов приема-выдачи, собственную контактную информацию. Существует возможность фильтрации услуг. Недостатком является то, что ресурс не предоставляет возможности просмотра отзывов клиентов.

Стиралкин сервис также имеет классический набор функциональных действий таких как: просмотр предлагаемых услуг с указанием цен на эти услуги, адресов пунктов приема-выдачи, собственную контактную информацию, возможность просмотра отзывов и примерный срок оказания услуги. Недостатком является отсутствие фильтрации услуг.

Wash of Fame сервис предоставляет возможность просмотра предлагаемых услуг с указанием цен на эти услуги, существует возможность просмотра составляющей услуги, адресов пунктов приема-выдачи, собственную контактную информацию, проводимых акций и возможность просмотра выполненных работ. Недостатком является отсутствие фильтрации услуг.

Также на представленных сайтах нет возможности оформлять заказы в режиме онлайн.

В табл. 1 представлены результаты сравнения функциональности разрабатываемого приложения и существующих аналогов.

Таблица 1

Анализ существующих приложений

Название	Перегруженность страниц	Интуитивно-понятный интерфейс	Возможность заказа	Фильтрация услуг
София	+	+	–	+
Стиралкин	+	–	–	–
Wash of Fame	–	+	–	–
Разрабатываемое веб-приложение	–	+	+	+

2. Постановка задачи

Из вышесказанного следует, что разрабатываемое приложение должно предоставлять конечному пользователю удобный интерфейс, позволяющий осуществить:

- возможность фильтровать услуги;
- добавление товаров в клиентскую квитанцию;
- добавление дополнительных услуг в заказ;
- возможность отслеживать статус заказа;
- возможность оставить комментарий к заказу;
- возможность удалить уже добавленные услуги;
- возможность оформить заказ;
- со стороны администратора — вносить изменения в статус заказа;
- со стороны администратора — вносить скидку в заказ.

Для проектирования был использован язык графического описания UML. На рис. 1. представлена диаграмма вариантов использования, которая иллюстрирует функциональность веб-приложения и возможности пользователя.

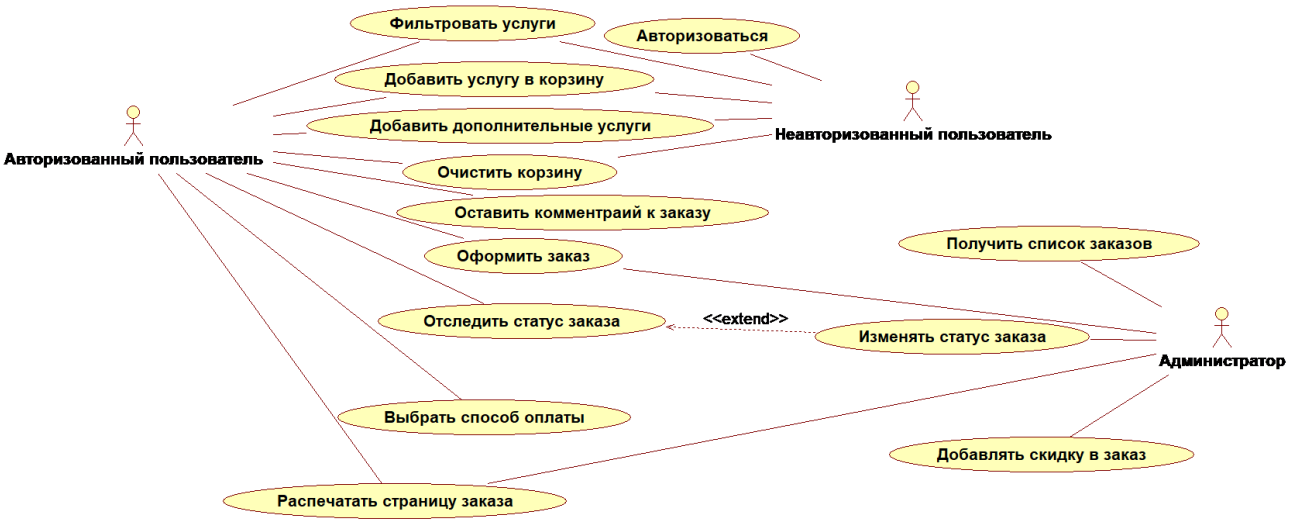


Рис. 1. Диаграмма вариантов использования

В данной системе представлено три актера.

Неавторизованный пользователь может совершать следующие действия.

1. Авторизоваться на стайте — пользователь может открыть страницу с регистрацией и создать аккаунт.

2. Фильтровать услуги — пользователь может, нажав на соответствующие кнопки, оставить только интересующие услуги.

3. Добавить услугу в корзину — пользователь может просмотреть услуги и добавить их в корзину.

4. Добавить дополнительные услуги — пользователь может добавить к заказу дополнительные услуги. Такие как: озонирование, покраска и замена бегунка.

5. Очистить корзину

Авторизованный пользователь может совершать следующие действия.

1. Фильтровать услуги.

2. Добавить услугу в корзину

3. Добавить дополнительные.

4. Очистить корзину.

5. Оставить комментарий к заказу — пользователь может написать комментарий во время оформления заказа.

6. Выбрать способ оплаты — пользователь может выбрать способ оплаты заказа.

7. Оформить заказ — пользователь может оформить заказ содержимого корзины.

8. Распечатать страницу заказа — пользователь может распечатать страницу заказа в формате PDF.

9. Отследить статус заказа — пользователь может отслеживать статус заказа в электронной квитанции.

Администратор может совершать следующие действия.

1. Получить список заказов — после входа сотрудник получает список всех оформленных пользователем заказов.

2. Изменять статус заказа — сотрудник химчистки может вносить изменения в статус заказа.

3. Добавлять скидку в заказ — сотрудник может добавить в заказ скидку.

4. Распечатать страницу заказа — сотрудник может распечатать страницу заказа в формате PDF.

3. Средства реализации

Поставленная задача будет реализована с использованием следующих средств:

• Visual Studio Code текстовый редактор для кроссплатформенной разработки веб- и облачных приложений [4];

• JavaScript, HTML, CSS [6, 7];

• СУБД PostgreSQL;

• библиотека Ulkit CSS [5].

Пример интерфейса (страница оформления заказа) представлена на рис. 2, 3.

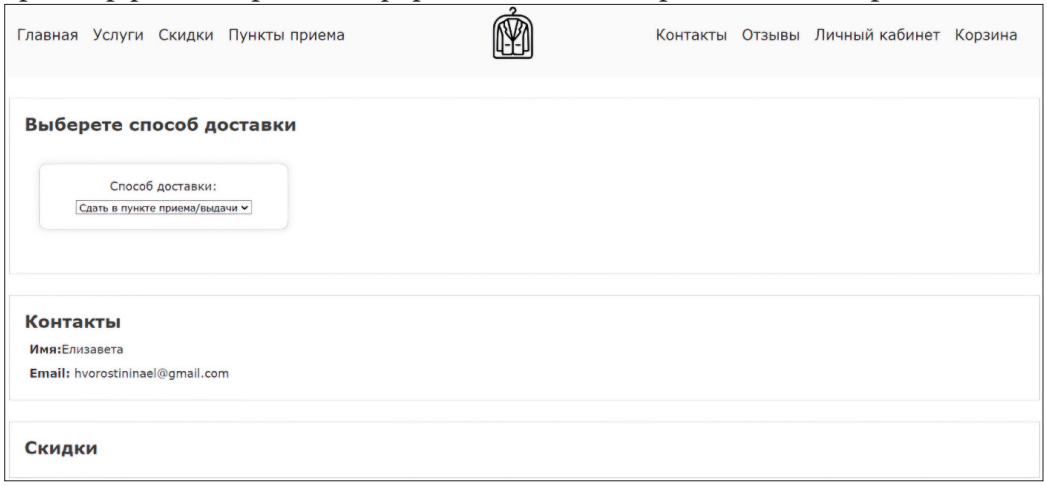


Рис. 2. Страница оформления заказа

Рис. 3. Страница оформления заказа

На рис. 4 представлен блок главной страницы, отвечающий за предоставляемые услуги. Каталог представляет собой адаптивную карусель с кнопками фильтрации сверху.

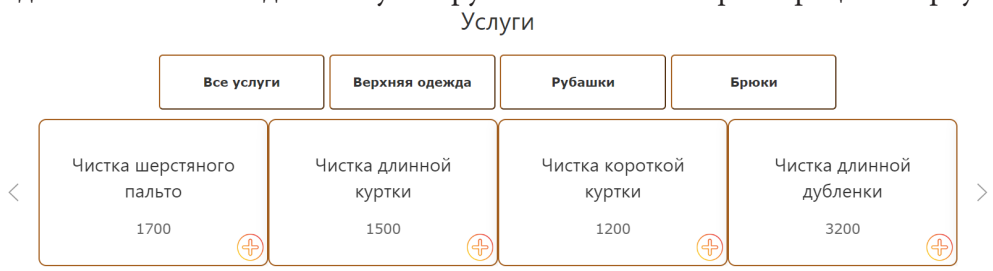


Рис. 4. Блок каталога

Пример работы фильтрации представлен на рис. 5.

Рис. 5. Пример фильтрации услуг

4. Архитектура приложения

Веб-приложение состоит из 5 основных компонентов: клиентской части, серверной части, базы данных, файлового сервера и JavaScript API Яндекс карт (рис. 6).

1. Компонент оформления заказ — компонент, отвечающий за оформление заказа клиентом.
2. Компонент контента — выводит информацию из базы данных на страницу сайта.
3. HTTP-клиент — компонент, который реализует взаимодействие с сервером системы.
4. Компонент обработки запросов — компонент, отвечающий за обработку запросов, которые пришли от веб-интерфейса системы;
5. Компонент работы с БД — компонент, отвечающий за взаимодействие с базой данных системы.
6. База данных — хранит информацию о пользователях и метаданные.

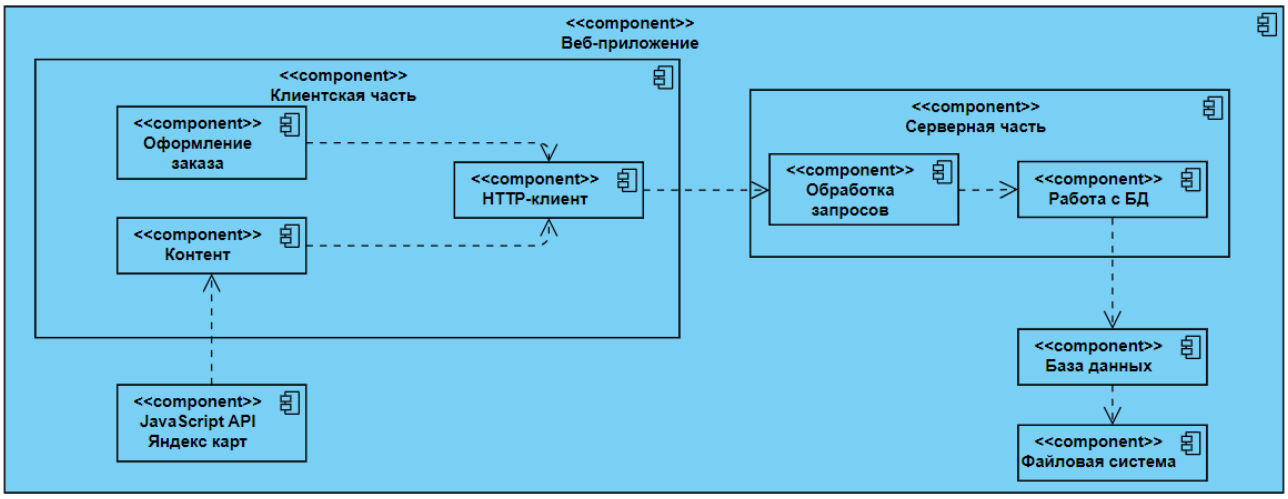


Рис. 6. Диаграмма компонентов веб-приложения

7. Файловый сервер — хранит фактические файлы, такие как изображения и большие текстовые документы.

8. JavaScript API Яндекс карт — отображает на странице сайта Яндекс карту с метками на ней.

5. Логическая модель базы данных

Логическая модель базы данных была спроектирована с помощью программы PowerDesigner и представлена представлена на рис. 7.

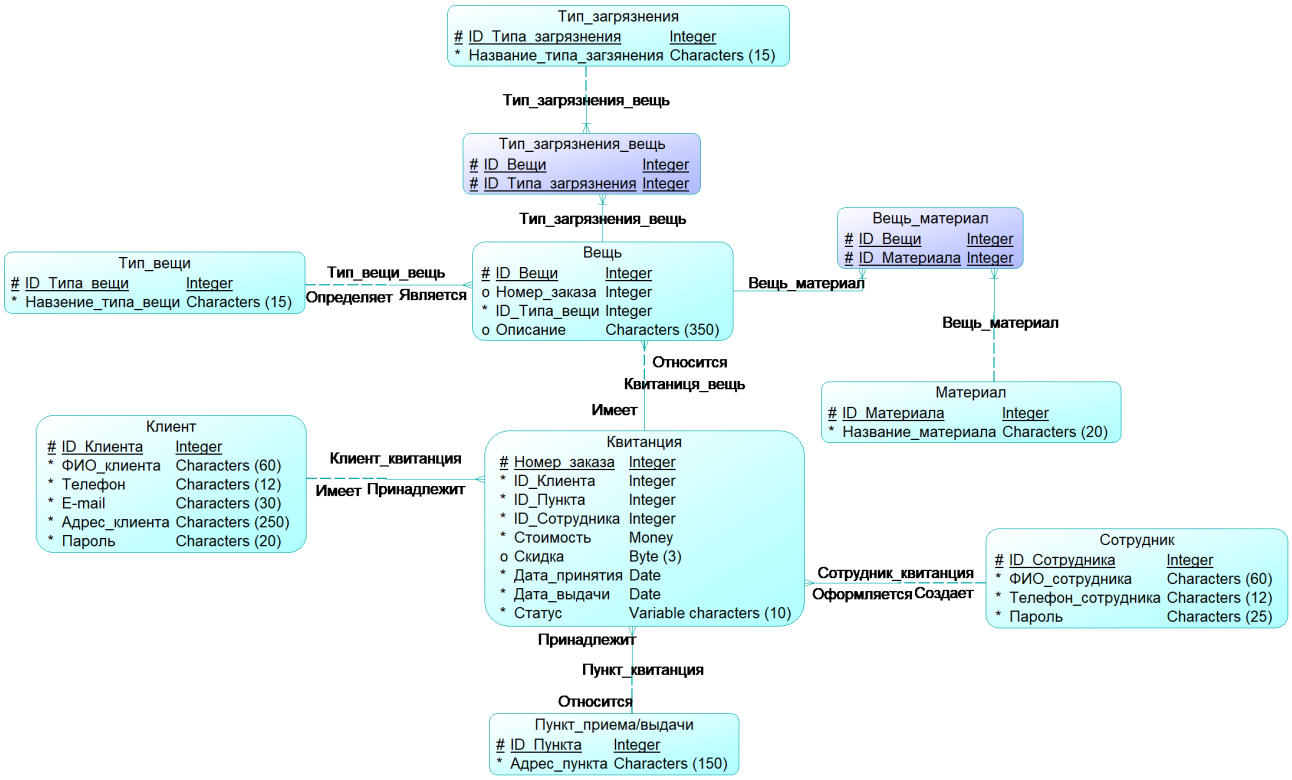


Рис. 7. Логическая модель БД

Сущность «Заказ» хранит информацию о заказе: номер_заказа, данные клиента, информацию о пункте выдачи, цену, скидку, даты приема и выдачи, а также статус заказа.

Сущность «Вещь» содержит информацию о вещи: id_типа_вещи, номер заказа и описание вещи.

Сущность «Клиент» содержит информацию о клиенте химчистки: ФИО клиента, его телефон, адрес электронной почти и адрес.

Сущность «Пункт приема/выдачи» хранит информацию о пункте приема и его адресе.

Сущность «Сотрудник» содержит информацию о сотрудниках химчистки: ФИО сотрудника, его телефон.

Сущность «Тип_вещи» хранит информацию о типе вещи и названии типа.

Сущность «Материал» хранит информацию о материале и его названии.

Сущность «Материал_вещь» хранит информацию о материале вещи.

Сущность «Тип_загрязнения» хранит информацию о загрязнении и его названии.

Сущность «Типа_загрязнения_вещь» хранит информацию о конкретном типе загрязнения на определенной вещи.

Замечание: все атрибуты, имеющие имя «ID», являются суррогатными ключами.

Заключение

В результате работы была реализована клиентская часть веб-приложения химчистки, реализующее следующую функциональность:

- добавление товаров в клиентскую квитанцию;
- возможность отслеживать статус заказа;
- возможность редактировать уже добавленный продукт.

Также были выполнены следующие задачи:

- анализ существующих решений;
- разработка требований к веб-приложению;
- разработка дизайна веб-приложения, проектирование макетов страниц;
- проектирование архитектуры веб-приложения;
- проектирование базы данных;
- реализация клиентской части веб-приложения;

В дальнейшем планируется реализовать серверную часть

Литература

1. Химчистка София. – URL: https://sofia-dryclean.ru/. (дата обращения 12.01.2024)

2. Химчистка Стиралкин. – URL: https://stiralkin36.ru/. (дата обращения 19.01.2024)

3. Химчистка Wash of Fame. – URL: https://washoffame.ru/. (дата обращения 19.01.2024)

4 *Адам Фримен, Эрик Робсон*. «HTML, CSS и JavaScript Web-разработка. Библия пользователя», 3-е изд. – Вильямс, 2020. – 635 с.

5. Адаптивная карусель для сайта. – URL: https://asuikit.com/v3/slider

6. *Васильев А. П.* JavaScript в примерах и задачах. 3-е изд. – СПб. : Эксмо, 2019. – 720 с.

6. *Дэвид Флэнаган*. «JavaScript. Подробное руководство», 6-е изд. – O'Reilly, 2017. – 1080 с.

7. Общие архитектуры веб-приложений. – URL: https://docs.microsoft.com/ru-ru/dotnet/architecture/modern-web-

8. Общий справочник веб-приложений. – URL: https://html5book.ru/

УДК 004.42

ОСНОВНЫЕ ВОЗМОЖНОСТИ UNITY DOTS НА ПРИКЛАДНОМ ПРИМЕРЕ

А. Е. Холодова, Е. В. Трофименко

Воронежский государственный университет

Аннотация. В данной статье рассматривается технологический стек Unity DOTS, который Unity предлагает, как решение для создания амбициозных и производительных приложений. Также сделан краткий обзор на составляющие этого стека, его преимущества и недостатки и сравнительный анализ привычного подхода программирования приложений в Unity с помощью компонентов и MonoBehaviour скриптов. Приведен мини-пример, сделанный с использованием двух технологий для их сравнения.
Ключевые слова: Unity, Unity DOTS, Job System, ECS, шейдеры.

Введение

Впервые Unity рассказали о Unity DOTS в октябре 2018. DOTS расшифровывается как Data Oriented Technology Stack — технологический стек, ориентированный на данные. Было показано техническое демо Megacity, запущенное на телефоне, с частотой обновления кадров равной 60 кадров в секунду.

Обновлённая версия вышла в мае 2022 года и уже в сентябре была представлена новая версия — экспериментальная 1.0. В которой была улучшена производительность и повышено удобство разработки. Летом 2023 года Unity назвали DOTS ECS официально готовым к использованию в разработках. [1]

В данной статье будут рассмотрены основные правила работы с DOTS и его главные принципы на основе мини-примера игры.

1. Unity DOTS

Unity DOTS — Это комбинация, состоящая из трех основных инструментов: Jobs, Burst и ECS.

ECS — Entity Component System. Это иная парадигма программирования, основанная на том, как работают процессоры и их память. Код, написанный в этой парадигме просто сделать мультипоточным. К нему можно также добавить векторизацию (SIMD). Данные обрабатываются целыми чанками (блоками) — это эффективно задействует процессорный кэш, так как в цикле следующие элементы чанка уже загружены в кэш. А значит, процессор задействуется полноценно и эффективнее.

Jobs — это система мультипоточности, которая была использована Unity и до DOTS. Её адаптировали под C# и создали систему потокобезопасности, позволив упростить процесс создания мультипоточности

Burst — это компилятор высокопроизводительного и оптимизированного кода для C#. Он имеет множество ограничений, но в итоге получается производительность, которая в некоторых случаях превосходит производительность игр в C++.

2. Пример работы с Unity DOTS

2.1. Подготовка к работе

Для обзора основных возможностей Unity DOTS, было реализован мини-приложение с использованием стандартного шаблона Unity 3D проекта и применением Universal Render Pipeline

(URP). Для работы с ECS нужно установить пакеты com.unity.entities и com.unity.entities.graphics с помощью Package Manager. Пакет Entities реализует паттерн Entity, Component, System, а пакет Entities Graphics использует Scriptable Render Pipeline (SRP), чтобы отрисовывать на уроне entity. Также в коде будут использоваться unmanaged коллекции из Collections и библиотека Mathematics, оптимизированные специально для Burst.

Для IDE Unity советует использовать Visual Studio 2022+ или Rider 2021.3.3. [2, 3]

На уровне мини приложения будет создано заданное количество кубов-префабов[1]. Часть кубов меняет свой цвет на каждый тик, другая — нет. Процент неизменных кубов задается заранее.

2.2. Создание саб-сцены и спавнера сущностей

Работу с ECS следует начинать с создания саб-сцены на уровне. В этой саб-сцене хранятся и отрисовываются сущности и соответствующие им префабы или меши. Для создания саб-сцены необходимо открыть нужный уровень и в иерархии сцены уровня нажать правой кнопкой мыши. Далее в меню нужно выбрать New Sub Scene –> Empty Scene. После сохранения новой саб-сцены она появится на уровне и ее можно использовать [4].

Для создания сущностей кубов-префабов сделаем отдельный префаб объект с компонентом Spawner.

Компоненты в Unity DOTS ECS могут быть managed и unmanaged:

1. Интерфейс IComponentData помечает структуры как unmanaged компоненты. Это предпочтительный вариант из двух, так как они доступны для jobs и в Burst-откомпилированном коде. В этих структурах могут быть преобразуемые типы, простые типы, коллекции из Collections.

2. Интерфейс IComponentData помечает классы как managed компоненты. Эти компоненты используются реже. Они могут содержать поля любых типов. Но эти компоненты не могут быть использованы в jobs, не оптимизируются Burst компилятором, нуждаются в garbage collection и должны иметь дефолтный конструктор для сериализации. Это все сказывается на производительности и удобстве работы с данными компонентами. Поэтому Unity советует сокращать количество managed компонент в проектах.

Компонент Spawner будет unmanaged компонентом, и следовательно, структурой с интерфейсом IComponentData. У него будут поля Entity для префаба с мешем куба-префаба, float3 для хранения позиции для создания следующего префаба, int количества кубов и float — вероятность, что куб не будет менять своего цвета.

Далее необходимо создать authoring и baker скрипт. Они имеют определенную структуру и служат для запекания значений из Unity компонентов в ECS компоненты. Создание класса SpawnerAuthoring, являющегося наследником MonoBehaviour, позволяет задать значение для числа кубов-сущностей и вероятности неизменяемости их цвета. Эти значения затем запекает в Entity класс SpawnerBaker в методе Bake. В этом методе создается Entity, к которой добавляются все компоненты из SpawnerAuthoring и их значения. Далее можно использовать и обрабатывать эти значения компонент сущностей в системах.

Были созданы все нужные сущности в коде. Теперь в саб-сцене создается пустой GameObject с названием Spawner. К нему добавляем компонент SpawnerAuthoring и в инспекторе устанавливаем значения числа префабов, шанса неизменяемости, префаб куба и начальную позицию для создания первого куба.

С пакетом Entities было добавлено окно Entities Hierarchy. В нем можно включить несколько режимов просмотра entity: authoring, runtime и смешанный. Два последних режима пока-

[1]Префаб — это шаблон для объекта в игровом движке Unity, аналогичен классу в ООП.

зывают сгенерированные из всех authoring GameObject бейкерами сущности. Пример вывода этого режима показан на рис. 1.

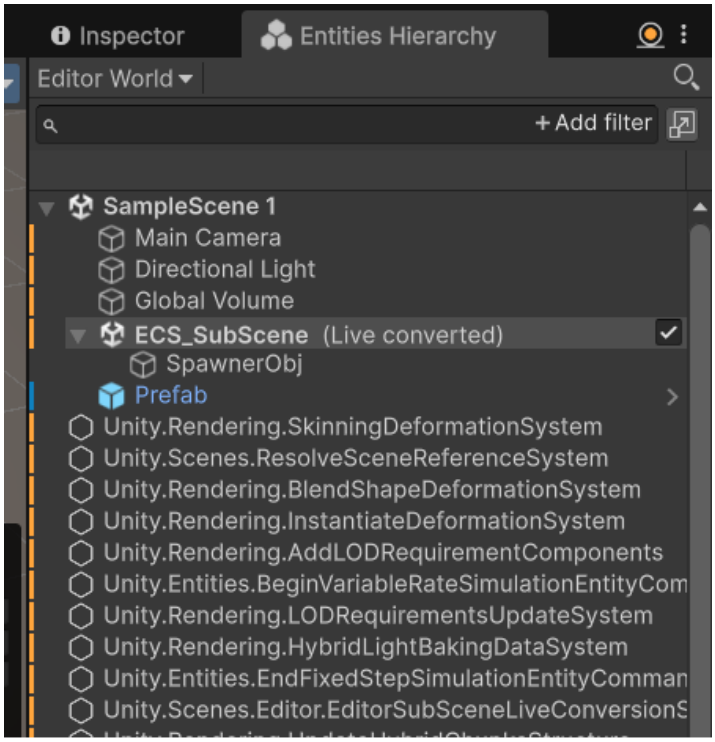


Рис. 1. Окно Entities Hierarchy

В этом окне показываются и сущности, и все GameObject. Если выбрать определенную сущность (все сущности отмечены шестиугольником), то можно просмотреть все компоненты данной сущности и их значения, установленные бейкером или системой, в окне Inspector.

Для свойств кубов-сущностей были заведены unmanaged компонента ColorComp, хранящая значения цвета меша куба, и компонент-флаг IsVisible, определяющий видно ли изменение цвета у куба-сущности.

В Unity DOTS (Data-Oriented Technology Stack) архетипы, чанки и точки синка играют ключевую роль в эффективности работы с данными сущностей. Каждый архетип представляет собой объединенную группу компонентов. Архетипы позволяют эффективно управлять группами сущностей с одинаковыми компонентами. Архетип объекта определяет, где в ECS хранятся компоненты этого объекта. ECS распределяет память в виде «чанков» или блоков, каждый из которых представлен объектом ArchetypeChunk. Блок всегда содержит объекты одного архетипа. Когда блок памяти заполняется, ECS выделяет новый блок памяти для любых новых сущностей, созданных с тем же архетипом. Блоки помогают использовать процессор и его память на максимум, они могут быть закэшированы в памяти. Если добавить или удалить компонент в уже существующем архетипе, ECS перемещает компоненты для этого объекта в другой архетип и другой блок и этот момент называется точкой синка. Таких точек пытаются избегать и минимизировать их количество, так как они требуют времени на обработку и могут быть выполнены только в главном потоке. [5]

1.3. Создание систем для обработки компонентов сущностей

Далее для обработки компонент сущностей необходимо создать системы. Именно с их помощью можно будет создавать кубы и соответствующие им сущности, менять цвет этих кубов.

Нужно реализовать систему создания на сцене кубов — OptimizedSpawnerSystem. Она реализует интерфейс ISystem. Важной для этой системы будет функция OnUpdate(ref SystemState state). В ней создается EntityCommandBuffer и определяется, и планируется параллельный запуск работы ProcessSpawnerJob. Работу и функцию OnUpdate можно ускорить, добавив к ним атрибут [BurstCompile].

EntityCommandBuffer позволяет управлять командами по созданию, обновлению и удалению сущностей. С его помощью можно планировать команды и выполнять их позднее, когда основные системы уже завершили свой расчет. Создать экземпляр этого класса можно, вызвав метод CreateCommandBuffer из синглтона BeginSimulationEntityCommandBufferSystem. Это система в Unity, которая управляет жизненным циклом буферов команд.

Работа ProcessSpawnerJob имеет один метод Execute. Здесь реализована вся логика, которую нужно распределить по потокам. В параметрах этого метода можно указывать компоненты, у которых понадобится изменять или читать значения. Работа обрабатывает сущности целыми чанками и таким образом оптимизирует процесс доступа к компонентам и их обработку. Для создания и размещения на сцене сущностей кубов была реализована следующая логика:

1. Если количество сущностей к созданию меньше или равно 0, то ничего не будет создано. Работа будет закончена.

2. Если есть сущности для создания, т. е. их количество > 0, то они будут созданы на карте. Работа продолжает свое выполнение.

3. В очередь EntityCommandBuffer записывается действие Instantiate по созданию в мире сущности с префабом, который был указан в SpawnerAuthoring и запечен в компонент Spawner с помощью бейкера SpawnerBaker.

4. Далее в очередь добавляется распоряжение об изменении уже существующего компонента LocalTransform с помощью метода SetComponent. После этого изменения префаб сущности будет перемещен в указанную точку.

5. Далее в EntityCommandBuffer добавляются новые компоненты с помощью AddComponent: ColorComp и DebugChunkColorProperty(об этом компоненте будет рассказано подробнее ниже), компонент флаг IsVisible.

6. Уменьшается количество сущностей к созданию и размещению и обновляется позиция для размещения следующей сущности.

Стоит отметить, что начальные значения для цвета меша сущности задаются случайным образом, как и вероятность, что этот меш будет менять цвет. Эти значения генерируются с помощью методов из класса UnityEngine.Random. Эти методы (в данном случае конкретно метод Range) нельзя вызывать в параллельном коде, а значит нельзя и вызывать внутри кода работы. Поэтому на моменте инициализации работы ей передаются все параметры, которые не получится получить внутри метода работы Execute, который будет впоследствии оптимизирован и распараллелен.

Система смены цвета реализована в классе ChangeColorsSystem, наследующем класс SystemBase. В его методе OnUpdate запланирован запрос всех сущностей, у которых есть компоненты DebugChunkColorProperty, ColorComp и IsVisible, и их обработка. Это реализовано с помощью метода Entities.ForEach, который позволяет итерировать по всем сущностям с компонентами, переданными как параметры этой функции. В данном примере происходит следующая обработка данных компонентов:

1. В значения компонента DebugChunkColorProperty записываются текущие значения компонента ColorComp

2. Обновляются значения компоненты ColorComp. Они получают плюс 5 к своим текущим значениям. Если значение становится больше 255, то оно обнуляется

Для рендера объектов на сцене Unity DOTS просит выбирать пайплайны, которые совместимы с созданием экземпляров DOTS. Для стандартных Universal Render Pipeline (URP) и

High Definition Render Pipeline (HDRP) это условие соблюдено. Поэтому для смены цвета меша кубов необходимо создать свой шейдер, тоже совместимый с DOTS, и назначить его шейдером для куба, который подключается в материале меша куба. Для создания необходимо в меню Create выбрать пункт ShaderGraph (если его нет, то нужно подключить в Package Manager пакет ShaderGraph). Далее в меню нужно выбрать подходящую версию шейдера в зависимости от выбранного графического пайплана. В данном примере это URP -> Lit Shader Graph. В этом шейдере нужно завести собственный параметр-вектор с произвольным именем и подключить его к полю Base Color ноды Fragment. [6]

После этого нужно реализовать компонент DebugChunkColorProperty. Это обычный компонент, у которого есть одно поле — вектор. Это поле связывается с параметром из шейдера с помощью атрибута класса [MaterialProperty("имя атрибута")]. Таким образом можно работать еще и со значениями для BaseColor, BumpScale, Cutoff, EmissionColor, Metallic, OcclusionStrength, Smoothness, SpecColor. [7]

Теперь любое изменение в компоненте DebugChunkColorProperty будет передаваться параметру из шейдера и применяться на мешах кубов.

3. Сравнение с обычным подходом к разработке в Unity

Также в рамках работы был реализован тот же самый мини-приложение, но с использованием стандартного подхода с компонентами и скриптами MonoBehaviour. Для этого был создан отдельный уровень без саб-сцены для Unity ECS, скрипт MonoBehaviour для смены цвета и создания и размещения кубов. Вся логика в этих скриптах аналогична той, которая реализована в системах мини-проекта выше. Результат работы этой реализации мини-проекта представлен на рис. 2.

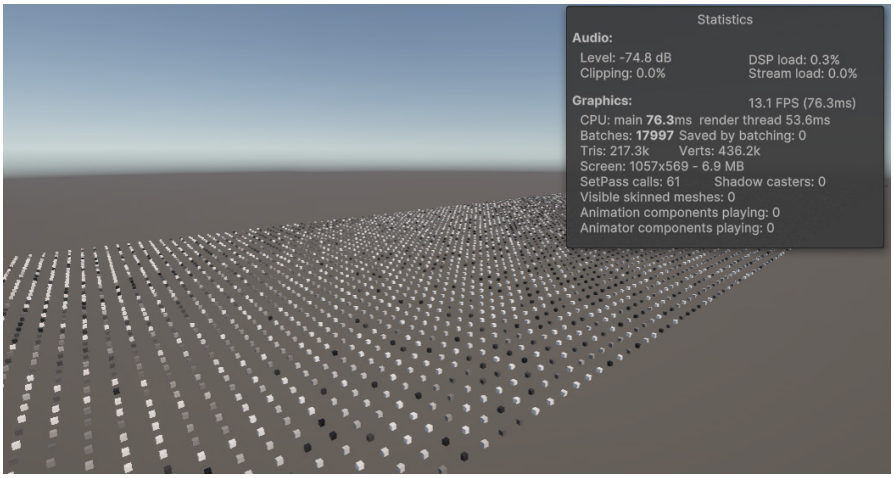


Рис. 2. Результат работы мини-примера Unity MonoBehaviour

Результат работы мини-проекта с использованием Unity DOTS приведен на рис.3. При количестве кубов на сцене равном 10_000 и шансе, что куб не будет менять своего цвета, равном 20 % обработка рендер и основного потока занимают меньше 7 миллисекунд. Количество кадров в секунду мини-примера колеблется от 120 до 150. Реализация этого же мини-примера на MonoBehaviour скриптах с такими же параметрами генерации имеет производительность в 10 раз меньше — только 11–15 кадров в секунду.

В табл. 1 показана производительность двух реализаций мини-примера.

По этим результатам видно, что подход Unity DOTS не только не отстает, но и в разы превосходит производительность подхода Unity MonoBehaviour.

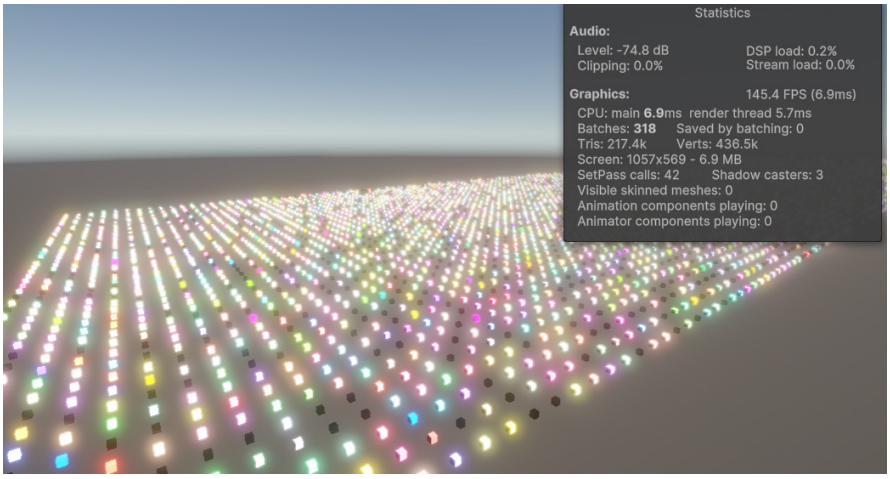


Рис. 3. Результат работы мини-примера Unity DOTS

Таблица 1

Результаты сравнения работы двух подходов

	Unity DOTS	Unity MonoBehaviour
Время в main потоке	6.9 ms	76.3 ms
Время в render потоке	5.7 ms	53.6 ms
Батчи	318	17997
Количество кадров в секунду	145.4	13.1

3.1. Когда стоит использовать Unity DOTS

Наибольшую пользу от этого стека получат проекты, в которых:

1. Есть большое количество объектов: DOTS особенно полезен, когда нужно обрабатывать огромное количество объектов одновременно, например, в играх «песочницах», стратегиях в реальном времени или симуляторах, где количество взаимодействующих объектов может достигать десятков тысяч или даже миллионов.

2. Нужны интенсивные вычисления: если игра требует интенсивных вычислений и может воспользоваться преимуществами параллельной обработки, DOTS предоставляет средства для легкой реализации многопоточности.

3. Требуется высокая производительность

3.2. Преимущества и недостатки Unity DOTS

К преимуществам DOTS можно отнести:

1. Производительность: DOTS разрабатывался с акцентом на оптимизацию работы CPU и GPU. Он позволяет обрабатывать большие объемы данных и объектов без значительного ухудшения производительности.

2. Многопоточность: DOTS обеспечивает простой и удобный способ работы с многопоточностью, позволяя распределять задачи по доступным ядрам процессора.

3. Масштабируемость: можно легко масштабировать приложения и игры, добавляя или удаляя объекты, не переживая о снижении производительности.

Главными минусами являются:

1. Кривая обучения: понимание основных принципов работы с DOTS может потребовать времени и усилий.

2. Мало обучающих материалов и краткая и не детальная документация.

Заключение

В статье были рассмотрены основные составляющие технологического стека Unity DOTS, который предлагает Unity для разработки оптимизированных и быстрых игр. Было показано, что Unity DOTS сложный в освоении стек, но он помогает улучшить производительность в разы. Это решение, которое гордо может противостоять тому, что предлагают остальные игровые движки, и может использоваться для работы с требовательными приложениями на слабых устройствах.

Литература

1. Как Unity двигает индустрию разработки в правильное направление // DTF: [сайт]. – URL: https://dtf.ru/gamedev/2195245-kak-unity-dvigaet-industriyu-razrabotki-v-pravilnoe-napravlenie (дата обращения: 10.10.2024).

2. Entities project setup // Unity3d: [сайт]. – URL: https://docs.unity3d.com/Packages/com.unity.entities@1.1/manual/getting-started-installation.html (дата обращения: 17.10.2024).

3. Further DOTS and ECS packages // Unity3d: [сайт]. – URL: https://docs.unity3d.com/Pahttps://docs.unity3d.com/Packages/com.unity.entities@1.1/manual/ecs-packages.htmlckages/com.unity.entities@1.1/manual/getting-started-installation.html (дата обращения: 17.10.2024).

4. Create the subscene for the spawner example // Unity3d: [сайт]. – URL: https://docs.unity3d.com/Packages/com.unity.entities@1.1/manual/ecs-workflow-scene.html (дата обращения: 02.11.2024).

5. ECS concepts // Unity3d: [сайт]. – URL: https://docs.unity3d.com/Packages/com.unity.entities@0.50/manual/ecs_core.html#memory-chunks (дата обращения: 05.11.2024).

6. About Shader Graph // Unity3d: [сайт]. – URL: https://docs.unity3d.com/Packages/com.unity.shadergraph@17.0/manual/index.html (дата обращения: 10.11.2024).

7. Material overrides using C# // Unity3d: [сайт]. – URL: https://docs.unity3d.com/Packages/com.unity.entities.graphics@1.0/manual/material-overrides-code.html (дата обращения: 10.11.2024).

УДК 004-8

ОБЗОР И СРАВНЕНИЕ ОСНОВНЫХ ПОДХОДОВ К ПОСТРОЕНИЮ РЕКОМЕНДАТЕЛЬНЫХ СИСТЕМ

И. Ю. Шабунин, Н. А. Каплиева

Воронежский государственный университет

Аннотация. В статье представлен сравнительный обзор трех основных моделей рекомендательных систем: контентно-ориентированных, основанных на коллаборативной фильтрации и гибридных. Рассмотрены их принципы работы, достоинства и недостатки, а также возможные применения в различных областях. Особое внимание уделено проблемам «холодного старта», «информационного пузыря» и зависимости от данных. Полученные результаты могут быть использованы для выбора подхода при проектировании рекомендательных систем, а также для их адаптации к конкретным задачам и данным.
Ключевые слова: рекомендательные системы, пользователи, объекты, рейтинг, персонализация, content-based подход, коллаборативная фильтрация, гибридная модель.

Введение

Рекомендательные системы являются неотъемлемой частью современной информационной среды.

Они помогают пользователям обнаруживать интересующий их контент — будь то товары, фильмы, музыка или статьи — анализируя их предыдущие действия и предпочтения.

Основная задача таких систем заключается в предсказании потенциальных интересов пользователей и своевременном предоставлении им релевантных рекомендаций. Это позволяет улучшить их опыт взаимодействия с платформой и повысить вовлеченность.

В условиях стремительного увеличения объема доступной информации рекомендательные системы приобретают особую значимость, упрощая процесс фильтрации данных и предлагая пользователям наиболее подходящие варианты.

1. Формализация задачи рекомендательных систем

Введем ряд обозначений.

Пусть есть множество пользователей U и множество объектов I. Для каждого пользователя $u \in U$ есть множество объектов $I_u \subset I$, с которыми он взаимодействовал и которым поставил рейтинги $R_u = (r_{ui})_{i \in I_u}$.

Рейтинг (фидбек) — это некоторая характеристика взаимодействия пользователя с объектом. В качестве примера фидбека можно привести факт добавления товара в корзину или оценку, поставленную пользователем фильму.

Таким образом, задачу рекомендательных систем можно переформулировать в следующем виде: для каждого пользователя $u \in U$ необходимо оценить значение r_{ui} для $i \in I \setminus I_u$ и выбрать несколько товаров с наибольшим оценочным рейтингом $\hat{r}_{ui}$. Иными словами, надо научиться среди непоказанных пользователю товаров находить те, которые заинтересовали бы его больше всего.

2. Различные модели рекомендательных систем

Существует множество подходов к построению рекомендательных систем, которые различаются по сложности, точности и области применения. Выбор подхода зависит от особенностей

данных, задач бизнеса и пользовательских предпочтений. В данной статье рассматриваются три ключевых подхода к построению рекомендательных систем: ***content-based*** (*контентно-ориентированный*), ***collaborative filtering*** (*коллаборативная фильтрация*) и ***hybrid*** (*гибридная модель*). В статье приводится обзор их основных характеристик, преимуществ и недостатков, а также проводится их сравнительный анализ.

3. Content-based рекомендации

Контентно-ориентированные рекомендательные системы основываются на анализе характеристик объектов и их сопоставлении с предпочтениями пользователя. Content-based рекомендации предполагают, что пользователи будут заинтересованы в элементах, схожих с теми, которые они уже оценили положительно. Основная идея заключается в том, что, если пользователю понравился определенный объект, система предложит ему похожие объекты, основываясь на их характеристиках. Например, если пользователь часто смотрит фильмы определенного жанра или с определенными актерами, система будет рекомендовать ему похожие фильмы.

Пусть каждый объект i описывается набором характеристик, которые могут быть представлены в виде векторного пространства $e_i \in R_n$. Эти векторы, называемые эмбеддингами, представляют объекты в числовом виде. Профиль пользователя строится на основе объектов, которым пользователь уже выставил оценку.

Рейтинг $\hat{r}_{ui}$ для объекта i, рекомендованного пользователю u, рассчитывается по следующей формуле:

$$\hat{r}_{ui} = \max_{j \in I_u, r_{uj} > \alpha} \rho(e_i, e_j) r_{uj},$$

где (e_i, e_j) — косинусное расстояние или скалярное произведение между эмбеддингами e_i и e_j, определяющее степень схожести объектов i и j;

I_u — множество объектов, которым пользователь u выставил оценки;

r_{uj} — рейтинг объекта j, установленный пользователем u;

α — гиперпараметр, определяющий пороговое значение оценки r_{uj}.

Таким образом, система выбирает объекты i, которые наиболее похожи на ранее оцененные объекты j с учетом их значений рейтинга r_{uj}.

Content-based рекомендательные системы обладают рядом преимуществ, которые делают их популярным выбором для персонализированных рекомендаций.

Во-первых, такие системы обеспечивают высокий уровень персонализации, так как их рекомендации строго соответствуют интересам конкретного пользователя, основываясь исключительно на его взаимодействиях с объектами. Контентно-ориентированные подходы не зависят от данных других пользователей, что делает их независимыми и автономными.

Во-вторых, эти алгоритмы относительно просты в реализации и интуитивно понятны, что снижает сложность их внедрения и обслуживания.

Однако content-based системы имеют и свои ограничения.

Одним из наиболее значимых недостатков является ограниченность разнообразия рекомендаций. Система предлагает пользователю только те объекты, которые схожи с уже оцененными, что может привести к однообразию рекомендаций и созданию так называемого «информационного пузыря».

Также существует проблема новых объектов: если пользователь еще не взаимодействовал с системой или оценил недостаточное количество объектов, она не сможет сформировать профиль предпочтений, что затрудняет выдачу качественных рекомендаций.

Кроме того, для эффективной работы таких систем требуется подробное описание объектов. Это означает, что необходимо собрать и обработать большое количество данных о харак-

теристиках товаров или контента, что иногда может быть трудозатратно и требовать значительных ресурсов.

4. Коллаборативная фильтрация

Модель коллаборативной фильтрации основывается на анализе пользовательских предпочтений и поведения. Существует два основных типа коллаборативной фильтрации:

1. User-based.

Рекомендации делаются на основе схожести между пользователями. Например, если пользователь A и пользователь B имеют схожие оценки для ряда фильмов, то фильмы, которые понравились пользователю A, могут быть рекомендованы пользователю B. Этот метод предполагает, что пользователи с похожими вкусами будут оценивать элементы аналогичным образом.

Введем меру похожести двух пользователей $s(u,v)$, которая тем больше, чем выше сходство между u и v. Для пользователя u рассмотрим множество похожих на него пользователей $N(u) = \{v \in U \setminus \{u\} \mid s(u,v) > \alpha\}$, где α — настраиваемый гиперпараметр.

Допустим, необходимо теперь оценить рейтинг r_{ui}, который пользователь u поставил бы объекту i. Сделаем это, опираясь на рейтинги, которые ставили похожие на u пользователи. Например, можно взять взвешенное среднее:

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} s(u,v) r_{vi}}{\sum_{v \in N(u)} | s(u,v) |}.$$

Модуль добавляется для того, чтобы корректно обработать непохожих пользователей, то есть с пары с отрицательной похожестью, которая может возникнуть, если при построении $N(u)$ взять достаточно маленькое α.

Можно пойти дальше и усовершенствовать метод оценивания. Для разных пользователей оценка «нормально» (и соответственно, оценки «хорошо» и «плохо») могут соответствовать разным значениям рейтинга. Для устранения этой проблемы, можно брать не сырой рейтинг пользователя r_{vi}, а его отклонение от среднего всех оценок пользователя: $r_{vi} - \overline{r}_v$. Таким образом, учитывается только разброс вокруг среднего и итоговая оценка будет выглядеть следующим образом:

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum_{v \in N(u)} s(u,v)(r_{vi} - \overline{r}_v)}{\sum_{v \in N(u)} | s(u,v) |}.$$

Для оценки схожести пользователей могут применяться также мера Жакара, скалярное произведение общих методов, корреляция Пирсона и др.

2. Item-based.

Рекомендации делаются на основе схожести между элементами. Например, если фильм X и фильм Y часто оцениваются одинаково, то пользователю, которому понравился фильм X, может быть рекомендован фильм Y. Этот метод предполагает, что элементы, которые часто оцениваются одинаково, имеют схожие характеристики.

Введем меру похожести объектов $s(i,j)$. Если необходимо оценить рейтинг, который пользователь u поставил бы еще не виденному им объекту i, то можно рассмотреть множество $N(i)$ близких к i объектов и оценить $\hat{r}_{ui}$ аналогично user-based подходу:

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} s(i,j) r_{uj}}{\sum_{j \in N(i)} | s(i,j) |}.$$

Рекомендательные системы, работающие на основе коллаборативной фильтрации, используют исключительно матрицу оценок R_{ui}, предполагая, что такой подход способен выявить важные взаимосвязи между пользователями и товарами.

Однако отсутствие дополнительной информации о новых пользователях или объектах создает препятствия для точных рекомендаций, так как истории взаимодействий попросту нет или она недостаточно обширна.

Данные системы склонны ограничиваться только известными интересами пользователя, что может привести к формированию «информационного пузыря», где пользователь получает исключительно однотипные рекомендации, не способствующие открытию новых предпочтений.

5. Гибридная модель

Гибридные рекомендательные системы объединяют различные подходы к генерации рекомендаций, чтобы преодолеть недостатки отдельных методов и улучшить качество рекомендаций. Наиболее распространенным вариантом является комбинация контентно-ориентированного подхода и методов коллаборативной фильтрации. Основная идея гибридных систем заключается в том, чтобы объединить преимущества обеих моделей и компенсировать их недостатки.

Основные стратегии гибридизации:

1. Комбинирование предсказания (*Weighted Hybrid*).

В этом подходе рекомендации из разных систем комбинируются на основе весов. Например, если контентно-ориентированная модель и модель коллаборативной фильтрации дают разные оценки для объекта, итоговая оценка может быть вычислена как взвешенная сумма их предсказаний:

$$\hat{r}_{ui} = \alpha \cdot \hat{r}_{ui}^{CB} + (1-\alpha) \cdot \hat{r}_{ui}^{CF}.$$

Здесь $\hat{r}_{ui}^{CB}$ и $\hat{r}_{ui}^{CF}$ — оценки от контентно-ориентированной и коллаборативной моделей соответственно, а α — гиперпараметр, задающий баланс между подходами.

Предположим, есть система рекомендаций фильмов. Гибридная модель может учитывать как характеристики фильмов (жанр, актеры, режиссеры), так и взаимодействие пользователей (оценки, история просмотров):

1) контентно-ориентированная часть анализирует, какие жанры предпочитает пользователь;

2) коллаборативная часть определяет, какие фильмы понравились пользователям со схожими предпочтениями;

3) гибридная модель объединяет результаты, комбинируя их с учетом весов или ранжируя объекты.

2. Каскадный подход (*Cascading Hybrid).*

В этом методе результаты одной системы используются как входные данные для другой. Например, сначала контентно-ориентированная модель выбирает список релевантных объектов, а затем коллаборативная фильтрация ранжирует их. Этот подход позволяет сузить область поиска и улучшить персонализацию.

3. Альтернативное использование (*Switching Hybrid*).

Система выбирает метод на основе характеристик данных или ситуации. Например, для новых пользователей, у которых нет истории взаимодействия (проблема «холодного старта»), используется контентно-ориентированный подход, а для пользователей с достаточным количеством данных — коллаборативная фильтрация.

4. Смешивание характеристик (*Feature Combination*).

Этот подход объединяет характеристики из разных методов в единый входной вектор для модели машинного обучения. Например, признаки, полученные из контентного анализа, комбинируются с данными о взаимодействиях пользователей, чтобы обучить одну универсальную модель.

5. Моделирование на метауровне (*Meta-level Hybrid*).

Результаты одной модели используются для обучения другой. Например, профили пользователей, построенные контентно-ориентированным методом, могут служить дополнительными признаками для коллаборативной фильтрации.

Гибридные модели сочетают преимущества контентно-ориентированных и коллаборативных методов, нивелируя их недостатки. Например, они уменьшают проблему «холодного старта» для новых объектов и пользователей, которая характерна для коллаборативной фильтрации, за счет использования информации о характеристиках объектов.

Объединение нескольких методов позволяет повысить точность рекомендаций, так как разные подходы могут дополнять друг друга. Например, если контентно-ориентированная часть модели не может предложить подходящий объект, коллаборативная часть все равно может сформировать рекомендацию.

Гибридные системы предоставляют более широкий спектр рекомендаций, уменьшая эффект «информационного пузыря», так как используют данные не только о похожих объектах, но и о предпочтениях других пользователей.

Гибридные модели легко адаптируются под конкретные задачи или данные. Можно изменить веса отдельных компонентов или интегрировать дополнительные методы, такие как контекстная фильтрация или временные параметры, для более точного формирования рекомендаций.

Однако интеграция различных методов требует дополнительных ресурсов на разработку, настройку и тестирование. Гибридные модели сложнее в реализации, чем использование одного подхода, так как необходимо учитывать множество параметров, таких как веса методов и способы их комбинирования.

Поскольку гибридные модели объединяют несколько подходов, они требуют больше вычислительных ресурсов для обработки данных, генерации эмбеддингов, поиска схожести и формирования рекомендаций. Это может усложнить их внедрение в системах с большими объемами данных.

Для успешного функционирования гибридных моделей требуется наличие детальной информации как о характеристиках объектов (для контентно-ориентированных рекомендаций), так и о взаимодействиях пользователей с объектами (для коллаборативной фильтрации). Если данные ограничены или некачественны, эффективность модели может снизиться.

Использование нескольких методов может привести к дублированию рекомендаций или предоставлению слишком большого количества рекомендаций, что усложняет восприятие для конечного пользователя. Необходима продуманная стратегия отбора и ранжирования результатов.

6. Сравнение представленных видов

Результаты сравнительного обзора трех основных подходов к построению рекомендательных систем приведены в табл. 1.

Заключение

Рекомендательные системы играют ключевую роль в современном цифровом мире, помогая пользователям эффективно находить релевантный контент среди огромного объема данных. В статье рассмотрены три подхода к построению рекомендательных систем: контентно-ориентированный, коллаборативная фильтрация и гибридный. Каждый из них имеет свои сильные и слабые стороны, а их выбор зависит от особенностей задач и данных.

Контентно-ориентированные системы обеспечивают высокий уровень персонализации, однако ограничиваются разнообразием рекомендаций. Коллаборативная фильтрация успеш-

Таблица 1

	Content-based	Коллаборативная фильтрация	Гибридная модель
Основной принцип	Анализ характеристик объектов и их сопоставление с предпочтениями пользователя.	Анализ схожести пользователей или объектов на основе матрицы оценок.	Комбинация подходов Content-Based и Collaborative Filtering.
Достоинства	– высокая персонализация; – не зависит от данных других пользователей; – простота реализации.	– улавливает латентные связи между пользователями и объектами; – эффективен при большом объеме данных.	– объединяет преимущества обеих моделей; – уменьшает проблемы «холодного старта»; – повышает разнообразие.
Недостатки	– ограниченность разнообразия рекомендаций; – проблема «холодного старта»; – требует описания объектов.	– не применима для новых пользователей и объектов; – склонна к информационным пузырям.	– более сложная реализация; – требует значительных вычислительных ресурсов.
Пример применения	Рекомендации фильмов на основе их жанра, актеров, режиссеров.	Рекомендации товаров на основе схожести вкусов пользователей.	Рекомендации фильмов с учетом жанров и предпочтений схожих пользователей.
Проблема «холодного старта»	Часто возникает, так как нет данных о пользователе.	Невозможно рекомендовать объекты, если нет данных о пользователях или новых элементах.	Меньше выражена за счет использования преимуществ обеих подходов.
Зависимость от данных	Требует данных о характеристиках объектов.	Зависит только от данных о взаимодействиях пользователей.	Использует оба типа данных.

но улавливает латентные связи между пользователями и объектами, но сталкивается с проблемой «холодного старта». Гибридные системы объединяют преимущества обеих моделей, улучшая качество рекомендаций и нивелируя их недостатки, но требуют значительных вычислительных ресурсов и более сложной реализации.

Эффективное использование рекомендательных систем требует баланса между персонализацией, вычислительными затратами и доступностью данных.

Впоследствии результаты данного сравнительного обзора будут использованы при реализации рекомендательной системы фильмов. Это позволит учесть преимущества и недостатки различных подходов и выбрать наиболее подходящую модель.

Литература

1. *Фальк К.* Рекомендательные системы на практике / Ким Фальк; пер. с англ. Д. М. Павлова. – Москва : ДМК Пресс, 2020. – 448 с.

2. *Aggarwal C.* Recommender Systems: The Textbook / Charu C. Aggarwal. – Springer, 2016. – 519 с.

3. *Schrage M.* Recommender Engines / Michael Schrage. – Cambridge : The MIT Press, 2020. – 296 с.

4. Введение в рекомендательные системы // Учебник по машинному обучению от Яндекса: [сайт]. – URL: https://education.yandex.ru/handbook/ml/article/intro-recsys (дата обращения: 05.11.2024).

5. Кино по щелчку: как работают рекомендательные алгоритмы в стриминге // РБК Тренды: [сайт]. – URL: https://trends.rbc.ru/trends/industry/cmrm/62baadf79a79477d0aafa5ae (дата обращения: 05.11.2024).

6. Лекция К. В. Воронцова «Машинное обучение. Рекомендательные системы»: видео. – URL: https://www.youtube.com/watch?v=J-QueLndVI8&t=121s (дата обращения: 07.11.2024).

7. RecSys Cookbook: строим рекомендательную систему на Python: видео / компания Evrone. – URL: https://www.youtube.com/watch?v=cxJ0KxzH_n0&t=5s (дата обращения: 07.11.2024).

УДК 004.65:004.43(075.8)

УНИФИКАЦИЯ ДОСТУПА К ДАННЫМ. ИССЛЕДОВАНИЕ МЕТОДОВ ОБЪЕДИНЕНИЯ РАЗНОРОДНЫХ БАЗ ДАННЫХ

И. М. Шаров, Р. С. Толмасов

МИРЭА – Российский технологический университет

Аннотация. В данной статье рассматриваются современные подходы к унификации доступа к разнородным базам данных. Приводится сравнительный анализ методов интеграции данных, включая их особенности, преимущества и ограничения. Предложены новые способы объединения разнородных баз данных.
Ключевые слова: разнородные базы данных, онтология данных, методы объединения данных.

Системы управления базами данных остаются одной из самых динамично развивающихся ИТ-отраслей [1]. СУБД играют важную роль в современных информационных системах, обеспечивая эффективное хранение, доступ и обработку данных. В условиях быстрого развития информационных технологий и роста объемов информации, выбор подходящей СУБД становится одним из ключевых аспектов для организаций [2]. При условии, что каждая СУБД предназначена для решения конкретных проблем, с которыми она справляется лучше других аналогов, появляется повод для создания вычислительных системы, которые состоят из различных типов аппаратного и/или программного обеспечения, объединенных для совместного выполнения задач. Гетерогенные системы обычно возникают в тех случаях, когда узлы, уже эксплуатирующие свои собственные системы с базами данных, со временем интегрируются в распределенную систему. В приведенном случае, конечный потребитель нуждается не столько в самих данных, сколько в факте объединения или агрегировании их [3].

Для воплощения взаимодействия разных подсистем требуются большие усилия. Усложняют задачу различия в архитектурных подходах к построению систем и их зависимость от используемых технологий. БД имеют разные модели хранения информации и, соответственно, разный язык запросов. Затраты на реализацию возможностей информационного обмена между подсистемами в этом случае велики и быстро начнут преобладать над затратами по реализации функциональности самих подсистем [3]. В связи с этим возникает проблема интеграции данных из различных систем [4]. Интеграция данных — это стратегический процесс, который объединяет данные из нескольких источников, чтобы предоставить организациям единое представление для более глубокого анализа, принятия обоснованных решений и целостного понимания своих бизнес-операций. На сегодняшний день можно выделить два типа методов интеграции данных: физический и логический. Основными методами интеграции данных для каждого из видов являются ETL (Extract, Transform, Load) для физической интеграции данных и виртуализация данных для логической интеграции. Выбор критериев для сравнения методов интеграции данных (ETL и виртуализации) основан на их значимости для ключевых характеристик систем, таких как производительность, актуальность данных, гибкость. Эти аспекты широко рассматриваются в современных исследованиях [4] и имеют критическое значение для оценки интеграционных подходов в условиях больших данных и высоких требований к оперативной аналитике, а также стабильной работы информационных систем. Активные разработки в этой области подчеркивают важность анализа данных как стратегического ресурса для бизнеса и технологий.

По приведенным результатам (табл. 1) можно сделать вывод, что каждый из представленных методов подходит исключительно при соблюдении ряда условии в конкретном случае для каждой организации.

Таблица 1

Анализ методов интеграции данных

Название метода / Критерий	ETL	Виртуализация данных
Доступ к данным	Предварительные запросы и обработка	Доступ в реальном времени
Производительность	Зависит от стабильности сети	Зависит от стабильности сети
Актуальность данных	Зависит от расписания загрузки	Всегда актуальные
Сложность реализации	Высокая сложность настройки и поддержки	Сложности с производительностью
Гибкость в запросах	Ограничена структурой хранилища	Высокая за счет динамической обработки

На сегодняшний день на практике тяжело встретить универсальные и закрепившиеся на рынке ПО решения, которые бы могли позволить пользователям воплотить идею объединения двух видов интеграции данных и решить недостатки часто использующихся методов, переняв положительные качества. Одним из решений в контексте данной проблемы может выступать программное обеспечение на подобие универсального шлюза (далее адаптер), который имеет свои правила приема информации и ее выдачи.

Идея адаптера отталкивается от того, что нет необходимости придумывать новые способы хранения данных, можно наладить работу с существующими. В основе такой системы, как и в методе виртуализации данных, лежит множественное подключение и параллельная работа с несколькими источниками, что даст возможность не иметь строгих ограничений на подключение разных типов баз данных. При подключении адаптер должен считывать методанные по структурам хранения с возможностью редактирования для преобразования в цельную карту маршрутов. Данный подход имеет сходство с ETL методом, однако в случае адаптера, пользователю нет необходимости знать, какое подключение использовать до СУБД и какую структуру оно имеет. Все вычисления проводятся на аппаратном уровне и по окончанию могут требовать лишь незначительного вмешательства для настройки от администратора системы. Для взаимодействия с адаптером необходимо разработать универсальный язык запросов, который позволит обращаться и связывать по нужным атрибутам данные из разных источников, которые в свою очередь уже имеют внутренние связи между хранимыми объектами. При этом язык не должен синтаксически сильно отличаться от уже существующих решений, чтобы подключение нового ПО не вызвало материальные потери из-за навязанных ситуацией компетенции сотрудников существующих организаций.

Адаптер позволяет минимизировать затраты на реорганизацию существующих систем хранения данных, эффективно используя уже доступные ресурсы. Его внедрение обеспечивает интеграцию различных источников данных без необходимости глубокого вмешательства в их текущую архитектуру. Такая модель работы открывает новые горизонты для систем с высокой степенью разнородности данных, значительно упрощая их подключение и синхронизацию.

Представленная идея может иметь перспективы в использовании за счет ситуации, связанной с импортозамещением в РФ, а также повлечь за собой новые идеи для разработки на основе карты маршрутов. Такой инструмент может стать основой для построения корпоративных информационных систем нового поколения, адаптированных под высокую нагрузку и сложные запросы.

Литература

1. Проблемы современных СУБД: Нужен ли морю данных океан железа // TADVISER: [сайт]. — URL: https://www.tadviser.ru/index.php/Статья:Проблемы_современных_СУБД:_Нужен_ли_морю_данных_океан_железа (дата обращения: 20.11.2024).

2. *Худайбердиев С. А.* Анализ самых актуальных серверных систем управления базами данных / С. А. Худайбердиев // International journal of theoretical adn practical research. – 2023. – № 5. – С. 36–46.

3. *Волушкова В. Л.* Интеграция разнородных данных в корпоративных информационных системах / В. Л. Волушкова // Программные системы и вычислительные методы. – 2019. – № 1. – С. 81–89.

4. *Шибанов С. В., Яровая М. В., Шашков Б. Д., Кочегаров И. И., Трусов В. А., Гришко А. К.* Обзор современных методов интеграции данных в информационных системах // НиКа. – 2010. – URL: https://cyberleninka.ru/article/n/obzor-sovremennyh-metodov-integratsii-dannyh-v-informatsionnyh-sistemah (дата обращения: 20.11.2024)

УДК 004.62

СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА (SOA). ПРИНЦИПЫ И ПРИМЕНЕНИЕ

С. А. Шебуняева

Воронежский государственный университет

Аннотация. Сервис-ориентированная архитектура (SOA) является основой для построения гибких, масштабируемых и независимых от технологий информационных систем. Данная статья рассматривает ключевые аспекты SOA, её эволюцию, основные паттерны и их применение в современном программировании. Представлены достоинства и недостатки подхода, а также особенности его реализации в различных технологических контекстах.

Ключевые слова: SOA, микросервисы, веб-сервисы, CORBA, ESB, очереди сообщений, распределённые системы, архитектурные паттерны.

Введение

Сервис-ориентированная архитектура (SOA) сформировалась как ответ на потребность бизнеса и технологий в интеграции разнородных систем. С момента своего появления концепция SOA претерпела значительные изменения, а её применение стало возможным в самых разных отраслях. Основной принцип SOA заключается в разделении систем на независимые сервисы, которые могут взаимодействовать между собой через стандартизированные интерфейсы. Это обеспечивает гибкость разработки, независимость от технологий и возможность масштабирования систем.

1. Основные принципы SOA

Сервис-ориентированная архитектура — модульный подход к разработке программного обеспечение, набор архитектурных принципов, независящих от технологий и продуктов.

SOA строится на ряде ключевых принципов:

1. Сочетаемость приложений: пользовательские приложения легко интегрируются друг с другом благодаря унифицированным интерфейсам.

2. Многократное использование бизнес-сервисов: сервисы проектируются для повторного использования в различных приложениях, что сокращает время и затраты на разработку.

3. Независимость от технологий: сервисы могут быть реализованы на любом языке программирования и развернуты на любой платформе.

4. Автономность сервисов: каждый сервис развивается, масштабируется и развёртывается независимо от других. Это позволяет обновлять и тестировать отдельные компоненты без влияния на всю систему.

2. Эволюция архитектуры

Эволюция SOA началась с таких технологий, как CORBA и DCOM, которые были разработаны в 1980–1990-х годах для взаимодействия приложений на разных платформах. Эти решения оказались недостаточно гибкими и сложными в реализации.

Веб-сервисы стали следующим этапом развития, обеспечив простоту и стандартизацию. Использование HTTP и XML/JSON форматов данных значительно упростило процесс интеграции.

Современные микросервисы, как одна из реализаций SOA, фокусируются на высокой автономности сервисов, обеспечивая гибкость, масштабируемость и отказоустойчивость.

3. Паттерны SOA

3.1. Общая архитектура брокера объектных запросов (CORBA)

CORBA (Common Object Request Broker Architecture) была разработана как независимая от платформы технология для взаимодействия объектов в распределённых системах. Стандарт CORBA обеспечивает вызовы удалённых процедур, не зависящие от платформы, транзакции, безопасность, независимость от особенностей передачи данных.

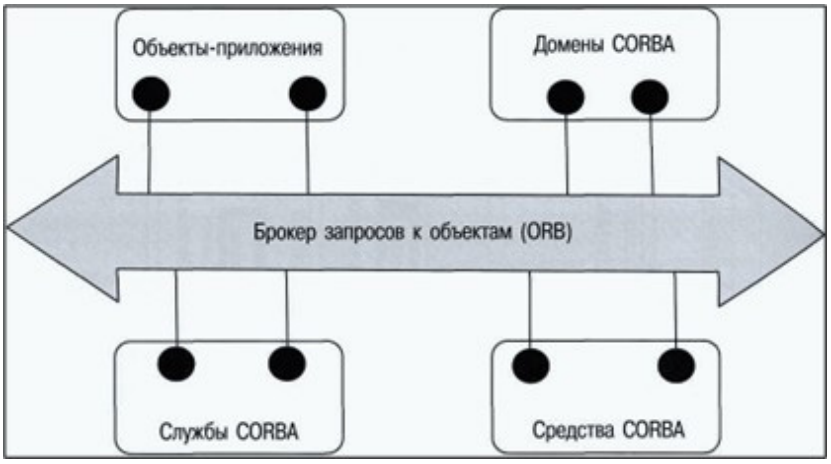


Рис. 1. Архитектура системы CORBA

Достоинства:

• обеспечивает высокую степень независимости от языков программирования и операционных систем;

• позволяет создавать сложные распределённые системы с минимальными изменениями в коде;

• поддерживает трансляцию данных между разными форматами.

Недостатки:

• высокая сложность спецификации затрудняет внедрение и поддержку;

• слабая производительность, особенно при большом объёме данных;

• ограниченная масштабируемость в сравнении с современными подходами.

3.2. Веб-сервисы

Веб-сервисы стали следующей ступенью эволюции SOA. Они базируются на использовании протоколов HTTP и стандартизированных форматов данных, таких как XML и JSON. SOAP (Simple Object Access Protocol) и REST (Representational State Transfer) являются основными подходами.

Достоинства:

• простота реализации благодаря поддержке протокола HTTP;

• универсальность: REST широко применяется для интеграции веб-приложений и мобильных устройств;

• широкая поддержка инструментов и библиотек для реализации.

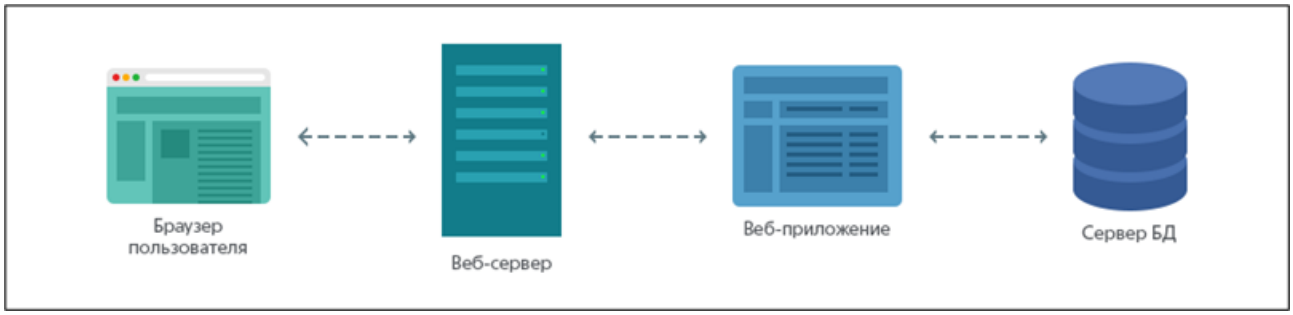


Рис. 2. Архитектура веб-сервиса

Недостатки:

• SOAP-ориентированные сервисы могут быть излишне сложными из-за жёстких требований к структуре сообщений;

• REST-сервисы недостаточно стандартизированы, что может вызывать трудности в интероперабельности (способности к взаимодействию).

3.3. Очереди сообщений

Очереди сообщений, такие как RabbitMQ, Apache Kafka и ActiveMQ, позволяют реализовать асинхронное взаимодействие между сервисами, что особенно полезно для высоконагруженных систем. Очередь сообщений улучшает масштабируемость и усиливает изолированность приложений, однако все эти приложения должны использовать один формат данных при обмене сообщениями. Такой паттерн использует в качестве компонента инфраструктуры программный брокер сообщений для реализации связей между приложениями (таких как запрос/ответ, публикация/подписка и др.).

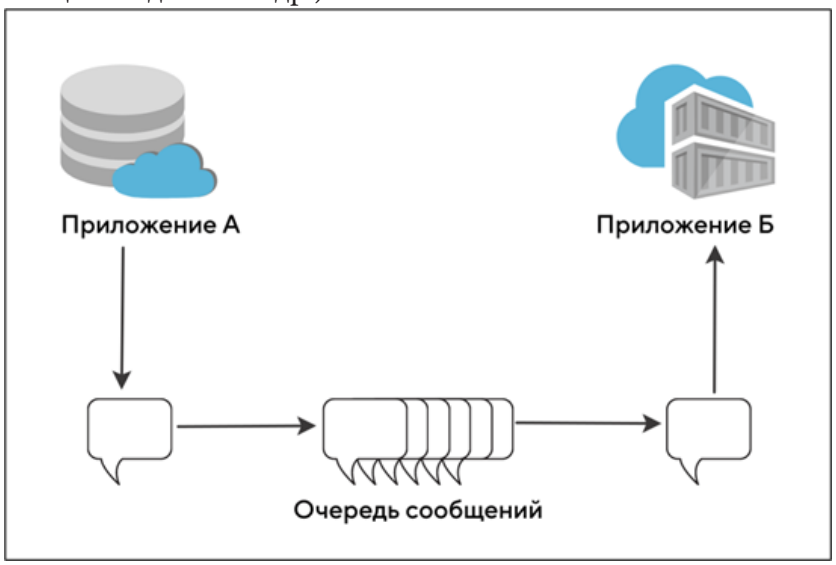


Рис. 3. Архитектура очереди сообщений

Достоинства:

• высокая масштабируемость благодаря возможности горизонтального расширения;

• повышение надёжности за счёт обработки сообщений в фоновом режиме;

• уменьшение времени отклика приложений.

Недостатки:

• сложность настройки и мониторинга;

• дополнительные затраты на инфраструктуру для хранения и маршрутизации сообщений.

3.4. Сервисная шина предприятия (ESB)

ESB (Enterprise Service Bus) — это центральный узел, который управляет маршрутизацией, преобразованием и безопасностью данных в системе. Шина выступает в качестве посредника, доставляющего сообщения. Такое ПО обеспечивает преобразование запросов, сообщений и параметров систем в нужный формат, соответствующий протоколу взаимодействующих систем, а также делает возможным интеграцию, сценарную маршрутизацию, транзакционный контроль, безопасность обмена данными, и равномерное распределение нагрузки на сервисы за счет концентрации событий и данных через единую точку.

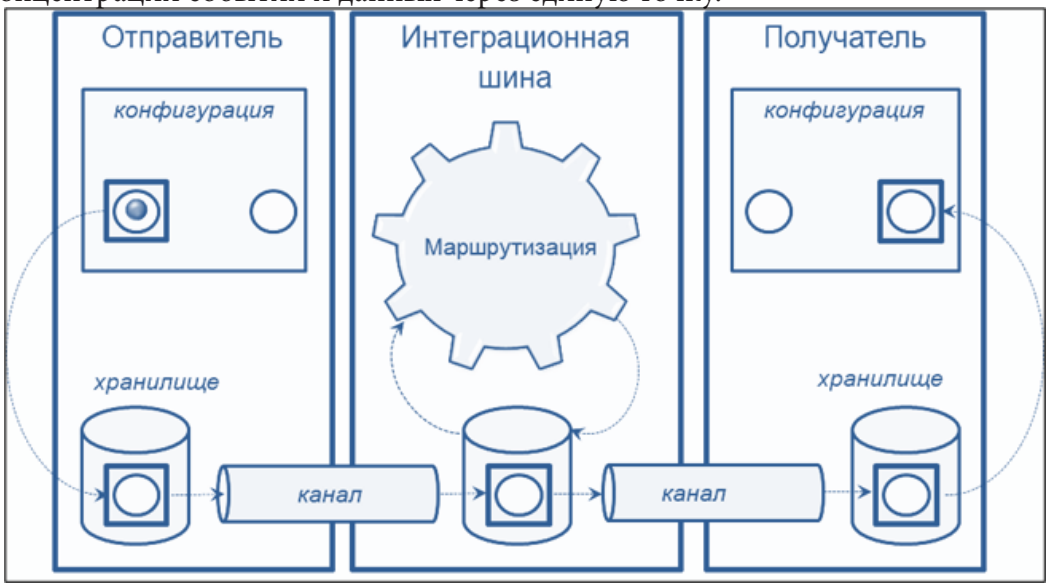


Рис. 4. Архитектура интеграционной шины

Достоинства:

- упрощение интеграции между различными сервисами;
- централизованное управление потоками данных и бизнес-логикой;
- возможность добавления новых сервисов без изменения существующих.

Недостатки:

- снижение производительности из-за узкого места — центральной шины;
- высокие затраты на внедрение и сопровождение;
- усложнение системы при добавлении новых компонентов.

3.5. Микросервисы

Микросервисы представляют собой современную реализацию SOA, где каждая бизнес-функция реализована как независимый сервис, взаимодействующий через API.

Достоинства:

- высокая степень гибкости: каждый микросервис может использовать свою базу данных и стек технологий;
- отказоустойчивость системы: сбой одного сервиса не влияет на работу других;
- ускорение разработки благодаря возможности параллельной работы команд.

Недостатки:

- увеличение сложности управления системой из-за большого числа сервисов;
- требуются дополнительные усилия для обеспечения безопасности и мониторинга;
- затраты на организацию DevOps-инфраструктуры для автоматического развертывания.

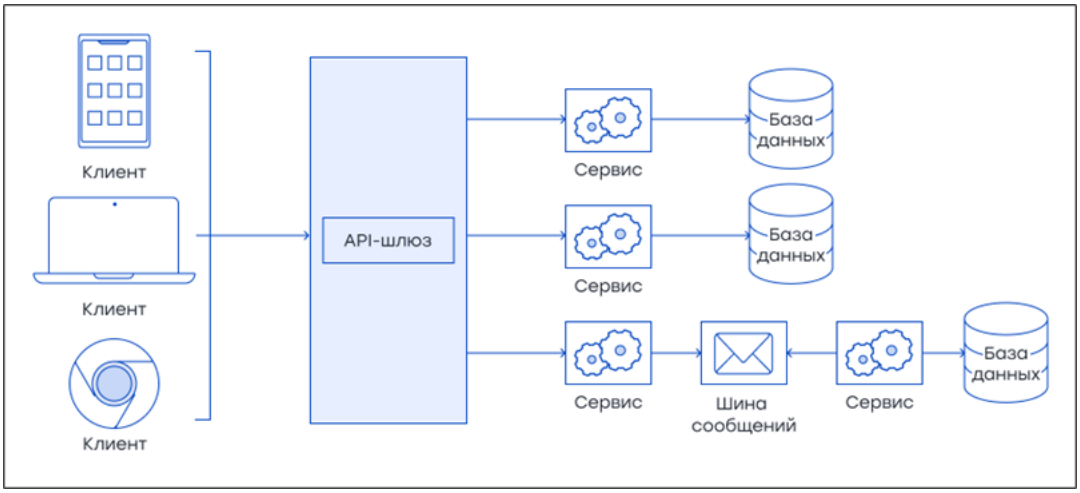


Рис. 5. Архитектура микросервисов

Заключение

Сервис-ориентированная архитектура остаётся актуальной благодаря своей универсальности и способности адаптироваться к современным требованиям бизнеса. Основные её принципы — гибкость, независимость и масштабируемость — позволяют строить системы, устойчивые к изменениям и росту нагрузки.

Однако для успешного внедрения SOA требуется тщательное проектирование, выбор подходящих паттернов и инструментов. Своевременное решение вопросов интеграции, производительности и безопасности обеспечит эффективное использование SOA в различных технологических контекстах.

Литература

1. *Спинеллис Д.* Идеальная архитектура / Д. Спинеллис, Г. Гусиос, пер. с англ Е. Матвеева. – Санкт-Петербург : Символ-Плюс, 2010. — 528 с.

2. *Биберштейн Н.* Компас в мире сервис-ориентированной архитектуры (SOA): ценность для бизнеса, планирование и план развития / Н. Биберштейн, С. Боуз, К. Джонс, М. Фиаммант, Р. Ша. – КУДИЦ, 2007. – 256 с.

3. *Бородаенко Ю. В.* Оценка качества интеграционных решений и методика их сравнитель ного анализа // Доклады БГУИР, 2019. – С. 120–125.

4. *Fielding R.* Architectural Styles and the Design of Network-based Software Architectures / R. Fielding. – University of California, Irvine., 2000.

5. *Gregor Hohpe* Enterprise Integration Patterns : Designing, Building and Deploying Messaging Solutions / Gregor Hohpe. – Pearson Education Inc., 2019. – 683 p.