**Оглавление**

[Введение 4](#_Toc161677744)

[1 Постановка задачи 5](#_Toc161677745)

[1.1 Аналитический обзор аналогов 5](#_Toc161677746)

[1.2 Средства разработки 8](#_Toc161677747)

[1.3 Разработка функциональных требований 9](#_Toc161677748)

[2 Проектирование архитектуры приложения 11](#_Toc161677749)

[2.1 Обобщенная архитектура 11](#_Toc161677750)

[2.2 Диаграмма UML и блок-схема алгоритма 12](#_Toc161677751)

[2.3 Модель данных 15](#_Toc161677752)

[2.4 Таблицы базы данных 15](#_Toc161677753)

[2.5 Таблицы базы данных 20](#_Toc161677754)

[3 Разработка web-приложения 20](#_Toc161677755)

[3.1 Микросервисы 20](#_Toc161677756)

[3.2 Создание пользователе й, ролей, и таблиц базы данных 20](#_Toc161677757)

[3.3 Пользовательские процедуры 20](#_Toc161677758)

[3.4 Индексы базы данных 20](#_Toc161677759)

[3.5 Последовательности 20](#_Toc161677760)

[3.6 Триггеры 20](#_Toc161677761)

[3.7 Директории 20](#_Toc161677762)

[4 Описание процедур импорта и экспорта 20](#_Toc161677763)

[4.1 Экспорт в JSON 20](#_Toc161677764)

[4.2 Импорт из JSON 20](#_Toc161677765)

[5 Тестирование производительности 20](#_Toc161677766)

[6 Описание технологии и ее применения в базе данных 20](#_Toc161677767)

[6.1 Шифрование данных в Oracle 20](#_Toc161677768)

[6.2 Аутентификация 20](#_Toc161677769)

[6.3 Аудит SYS операций 20](#_Toc161677770)

[6.4 Защита слушателя 20](#_Toc161677771)

[7 Краткое описание приложения для демонстрации 20](#_Toc161677772)

[8 Руководство пользователя 20](#_Toc161677773)

[8.1 Пример работы приложения от лица пользователя 20](#_Toc161677774)

[8.2 Пример работы приложения от лица администратора 20](#_Toc161677775)

[Заключение 20](#_Toc161677776)

[Список используемых источников 20](#_Toc161677777)

[Приложение А 20](#_Toc161677778)

[Приложение Б 20](#_Toc161677779)

[Приложение В 20](#_Toc161677780)

[Приложение Г 20](#_Toc161677781)

[Приложение Д 20](#_Toc161677782)

Введение

В современном мире информационных технологий и цифровизации все больше людей полагаются на персонализированные рекомендации для обнаружения нового контента. От социальных сетей до новостных агрегаторов, персонализированные рекомендации стали ключевым элементом пользовательского опыта. В то же время, платформы для публикации контента предоставляют пользователям возможность делиться своими мыслями, идеями и творчеством с миром.

В рамках данного курсового проекта будет разработано веб-приложение, которое сочетает в себе сервис персональных рекомендаций и издательскую платформу. Цель – создать интуитивно понятное и удобное приложение, которое позволит пользователям публиковать свой контент в виде текста или изображений, комментировать посты других пользователей, кастомизировать свой профиль и получать персонализированный контент, основанный на их интересах и взаимодействиях. Для достижения этой цели необходимо выполнить следующие задачи:

* провести анализ предметной области и определить основные сущности и атрибуты системы;
* разработать архитектуру приложения, используя подход микросервисов;
* реализовать каждый микросервис, включая сервисы авторизации и регистрации, уведомлений, контента, комментариев, профилей, рекомендаций и безопасности;
* разработать API Gateway для маршрутизации запросов к соответствующим микросервисам;
* создать клиентскую часть приложения на React для взаимодействия с пользователем;
* использовать Docker Compose для управления сервисами и обеспечения их совместной работы;
* наполнить базу данных тестовыми данными;
* разработать функции и методы для реализации бизнес-логики каждого микросервиса;

разработать запросы для получения необходимой информации из базы данных.

1. Постановка задачи
   1. Аналитический обзор аналогов

В современном мире издательские платформы играют ключевую роль в распространении информации. Они предоставляют обычным пользователям публиковать свои мысли, статьи, работы, делая их доступными для широкой аудитории. Интерфейс веб-приложения представлен на рисунке 1.1.

[Хабр – это одна из самых популярных платформ для IT-специалистов, где пользователи могут публиковать статьи, обмениваться знаниями и обсуждать различные темы, связанные с информационными технологиями](https://habr.com/ru/feed/).

Достоинства:

* [широкий спектр тем: Хабр покрывает множество тем, связанных с IT, включая программирование, разработку веб-сайтов, информационную безопасность, аналитику данных и многое другое](https://habr.com/ru/feed/);
* [сообщество экспертов: Хабр имеет активное сообщество пользователей, которые являются экспертами в своих областях](https://habr.com/ru/feed/);
* [персонализированный контент: Пользователи могут настроить свою ленту новостей, чтобы видеть контент, который их интересует](https://habr.com/ru/feed/).

Недостатки:

* [сложность навигации: Для новых пользователей может быть сложно найти нужную информацию из-за большого количества контента](https://habr.com/ru/feed/);
* [неравномерное качество контента: Качество публикаций может сильно варьироваться, поскольку они создаются разными пользователями](https://habr.com/ru/feed/).

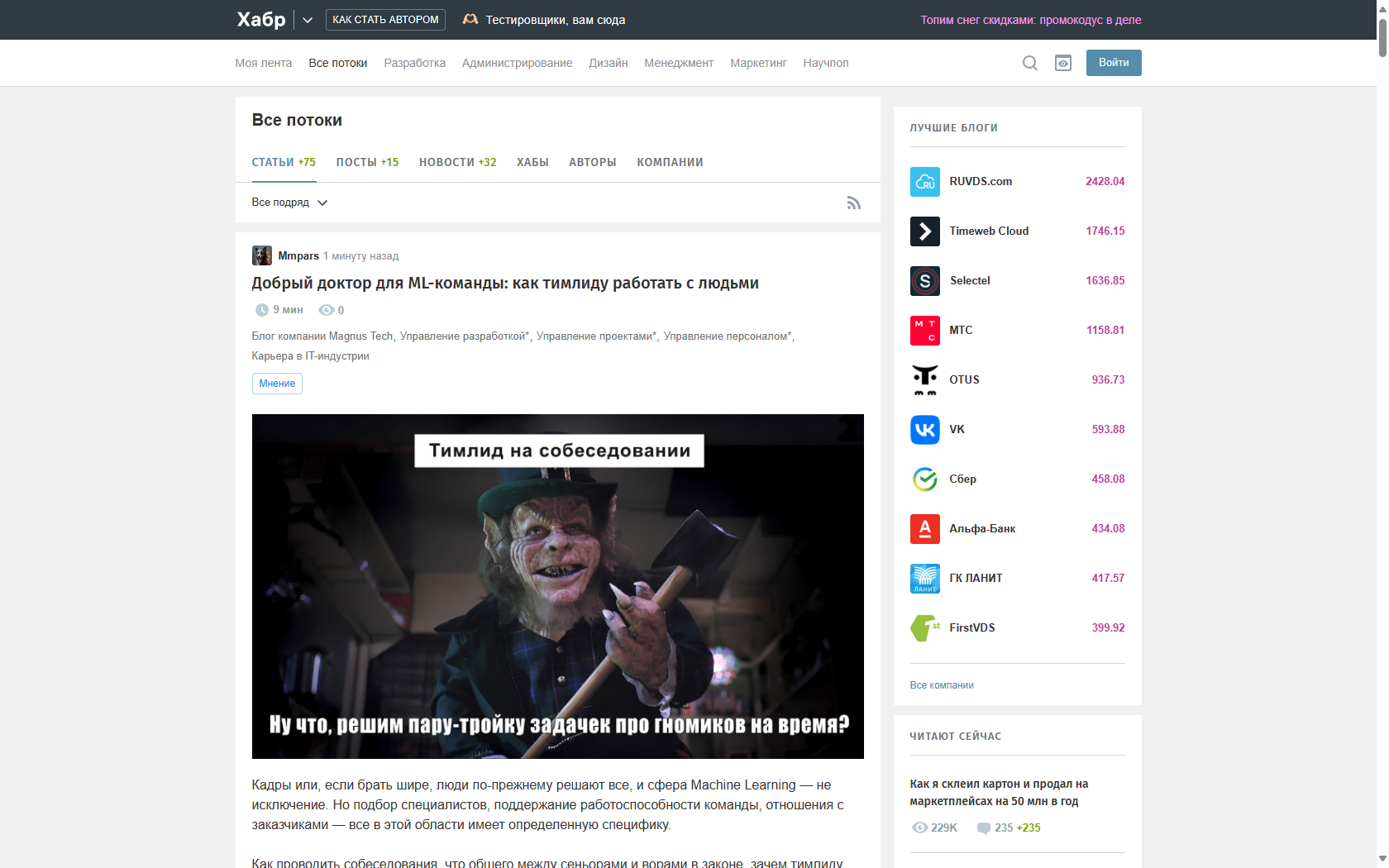


Рисунок 1.1 – Интерфейс [веб-приложения](http://unitoria.ru/blog/kadry/top-5-programm-dlya-kadrovogo-ucheta-v-2021-godu) Хабр

[Teletype — это удобная платформа для публикации статей, которая тесно интегрирована с Telegram и позволяет легко делиться контентом через телеграм-каналы](https://teletype.in/@davdeyonok/howtoteletype). [Пользователи могут создавать статьи с картинками, ссылками и другими элементами, делая информацию доступной и привлекательной](https://teletype.in/@davdeyonok/howtoteletype). Интерфейс веб-приложения представлен на рисунке 1.2.

Достоинства:

* [простота использования: Teletype предлагает интуитивно понятный интерфейс с простым редактором для создания статей](https://teletype.in/@alfiia/VVADFxVyE);
* [визуальное оформление: Платформа предоставляет разнообразные инструменты для оформления текста, включая заголовки, курсив, жирный шрифт и вставку изображений;](https://teletype.in/@davdeyonok/howtoteletype)
* [интеграция с Telegram: Teletype удобен для пользователей Telegram благодаря возможности быстрого перехода к статьям прямо из сообщений и каналов](https://teletype.in/@davdeyonok/howtoteletype).

Недостатки:

* ограниченная функциональность: В отличие от более крупных платформ, Teletype может не предлагать некоторые продвинутые функции для публикации и управления контентом;
* зависимость от Telegram: Для пользователей, которые не используют Telegram, платформа может быть менее привлекательной из-за её интеграции с этим мессенджером.

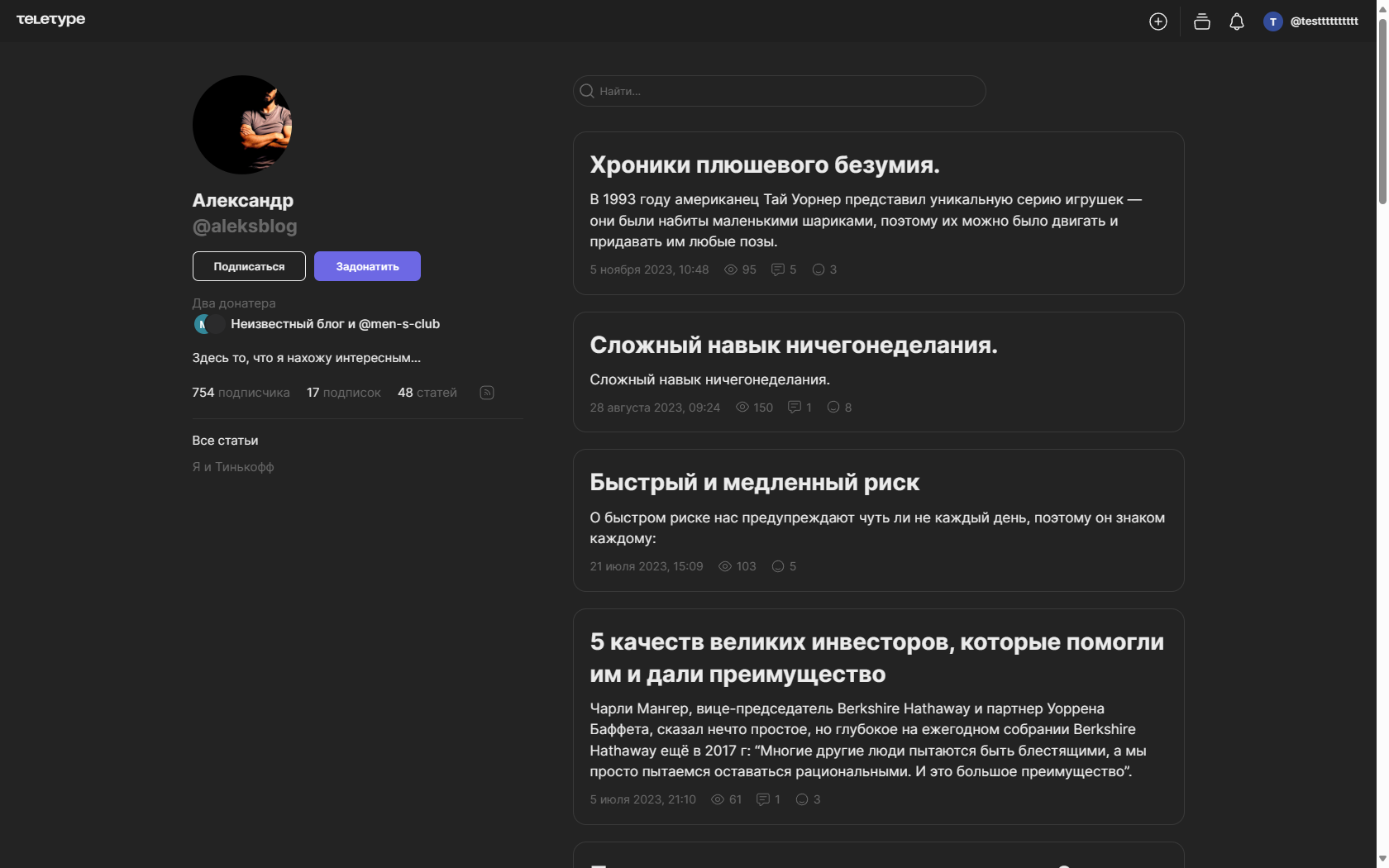


Рисунок 1.2 – Интерфейс [веб-приложения](http://unitoria.ru/blog/kadry/top-5-programm-dlya-kadrovogo-ucheta-v-2021-godu) Teletype

Сайт [Hashtap](https://www.hashtap.com/@support" \t "_blank)[представляет собой платформу, которая предлагает сервис персонализированных рекомендаций](https://www.hashtap.com/@support). [Он также предлагает платформу email-маркетинга, которая автоматически отправляет персонализированные и триггерные email-письма](https://www.hashtap.com/@support). Интерфейс веб-приложения представлен на рисунке 1.3.

Достоинства:

* [персонализированные рекомендации: Hashtap предлагает персонализированные рекомендации, что может помочь пользователям обнаружить новый контент](https://www.hashtap.com/@support);
* [email-маркетинг: Hashtap предлагает платформу email-маркетинга, которая автоматически отправляет персонализированные и триггерные email-письма](https://www.hashtap.com/@support);
* [поддержка: Hashtap имеет команду поддержки, которая может помочь пользователям с вопросами и предложениями](https://www.hashtap.com/@support).

Недостатки:

* [недостаток информации: На основе доступной информации, сложно определить недостатки Hashtap](https://www.hashtap.com/@support). Для более точной оценки могут потребоваться дополнительные данные, такие как отзывы пользователей или личный опыт использования платформы;
* [отсутствие активности на GitHub: Согласно информации на GitHub, Hashtap не имеет значительной активности в последнее время](https://www.hashtap.com/@support). Это может указывать на отсутствие обновлений или улучшений.

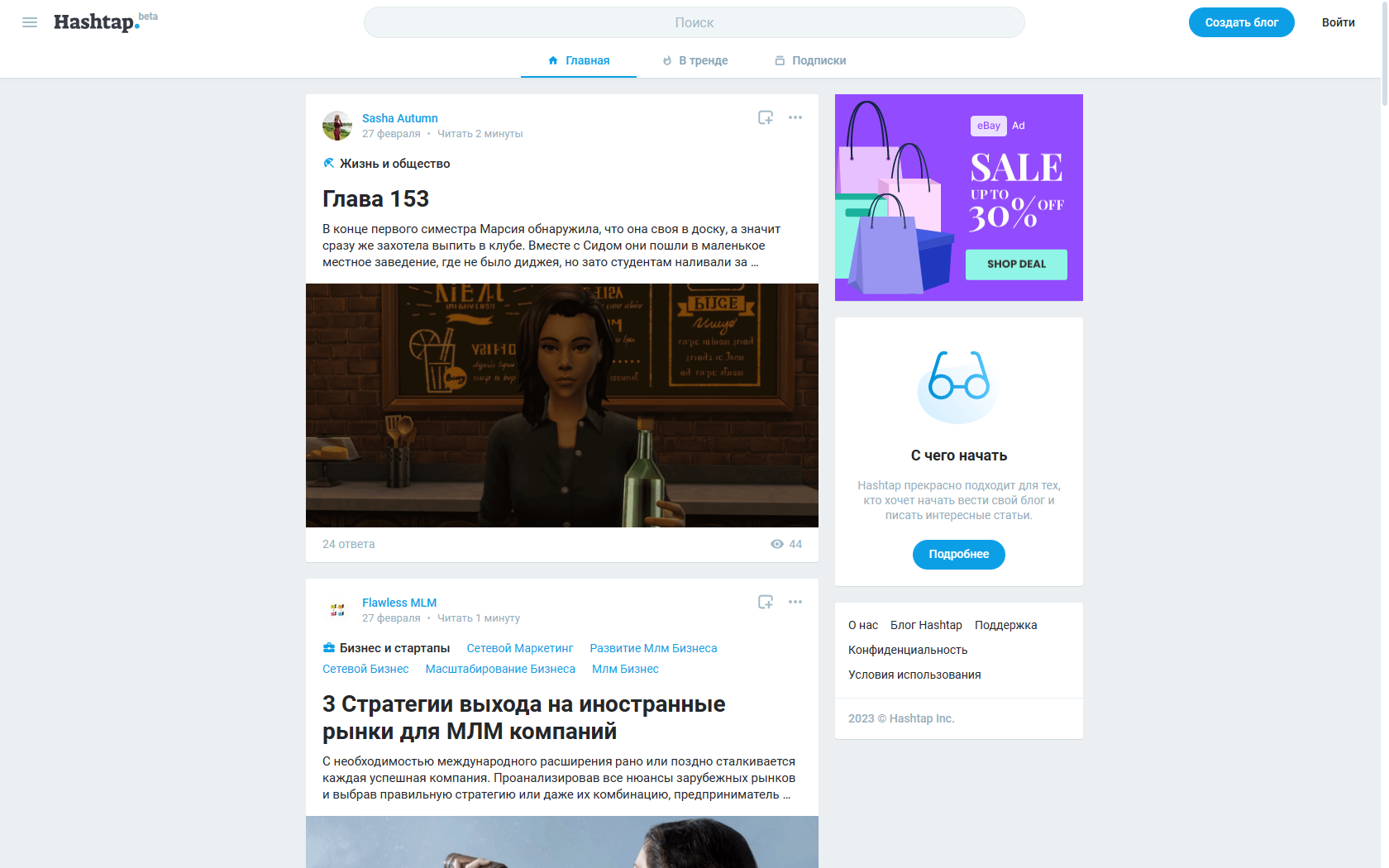


Рисунок 1.3 – Интерфейс [веб-приложения](http://unitoria.ru/blog/kadry/top-5-programm-dlya-kadrovogo-ucheta-v-2021-godu) Hashtap

[Dev.to - это популярное сообщество разработчиков, где они могут обмениваться знаниями, учиться и обсуждать различные темы, связанные с веб-разработкой](https://dev.to/t/webdev). Интерфейс веб-приложения представлен на рисунке 1.4.

Достоинства:

* [активное сообщество: Dev.to имеет большое и активное сообщество разработчиков, которые регулярно публикуют и обсуждают новые идеи и концепции](https://dev.to/t/webdev);
* [широкий спектр тем: Dev.to покрывает множество тем, связанных с веб-разработкой, включая HTML, CSS, JavaScript, React, Node.js и многое другое](https://dev.to/t/webdev);
* [обучающие материалы: Dev.to предлагает множество обучающих материалов и руководств, которые могут помочь разработчикам улучшить свои навыки](https://dev.to/t/webdev).

Недостатки:

* [неравномерное качество контента: Качество публикаций может сильно варьироваться, поскольку они создаются разными пользователями](https://dev.to/t/webdev);
* [отсутствие модерации: Некоторые пользователи могут заметить, что на сайте отсутствует строгая модерация, что может привести к появлению нерелевантного или низкокачественного контента](https://dev.to/t/webdev).

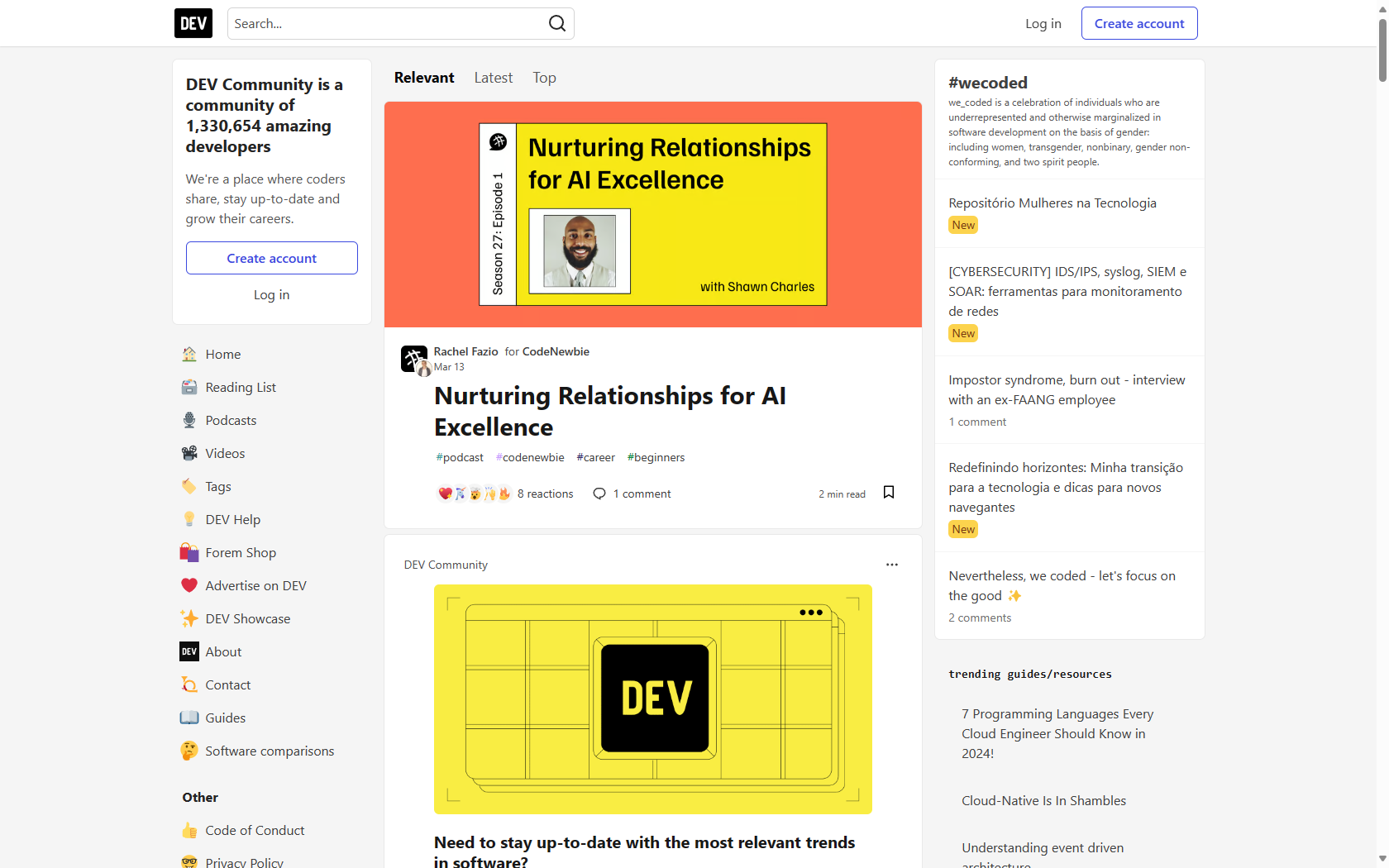


Рисунок 1.4 – Интерфейс [веб-приложения](http://unitoria.ru/blog/kadry/top-5-programm-dlya-kadrovogo-ucheta-v-2021-godu) [Dev.to](https://dev.to/t/webdev" \t "_blank)

Таким образом, можно сделать вывод, что существующие аналоги издательских платформ имеют свои преимущества и недостатки, и не полностью удовлетворяют потребностям современных пользователей.

* 1. Средства разработки

При разработке приложения будут использованы

* интегрированная среда разработки JetBrains WebStorm;
* язык программирования Javascript;
* язык гипертекстовой разметки HTML совместно с CSS;
* технологии GRPC, RabbitMQ;
* библиотеки React, Koa;
* Базы данных MS SQL Server, Redis

Использование данных технологий имеет несколько преимуществ при разработке приложения для управления образовательным процессом.

Во-первых, интегрированная среда разработки JetBrains WebStorm обеспечивает высокую производительность и удобство при разработке приложений.

Во-вторых, язык программирования Javascript — это высокопроизводительный и эффективный язык, который позволяет быстро и легко разрабатывать веб-приложения.

В-третьих, использование расширяемого языка разметки XAML позволяет создавать богатые пользовательские интерфейсы.

Четвертое преимущество заключается в том, что технологии GRPC и RabbitMQ а также библиотеки React и Koa позволяют создавать расширяемое и масштабируемое приложение, которое может обслуживать большое количество пользователей.

Базы данных MS SQL Server и Redis позволяют создавать надежные и быстрые хранилища данных, которые используются для хранения и обработки информации о пользователях, постах, лайках и других сущностях, необходимых для управления издательской платформой.

Исходя из описанных выше технологий, можно сделать вывод о том, что для реализации данного проекта были использованы современные инструменты и технологии, что позволило повысить эффективность работы. В целом, использование таких технологий позволило реализовать проект на высоком уровне и обеспечить его успешное выполнение.

* 1. Разработка функциональных требований

В целом, раздел, Разработка функциональных требований является ключевым для успешной разработки продукта или системы, так как он определяет основные функции и возможности продукта и помогает согласовать ожидания разработчиков и пользователей. Диаграмма вариантов использования представлена на рисунке 1.5.

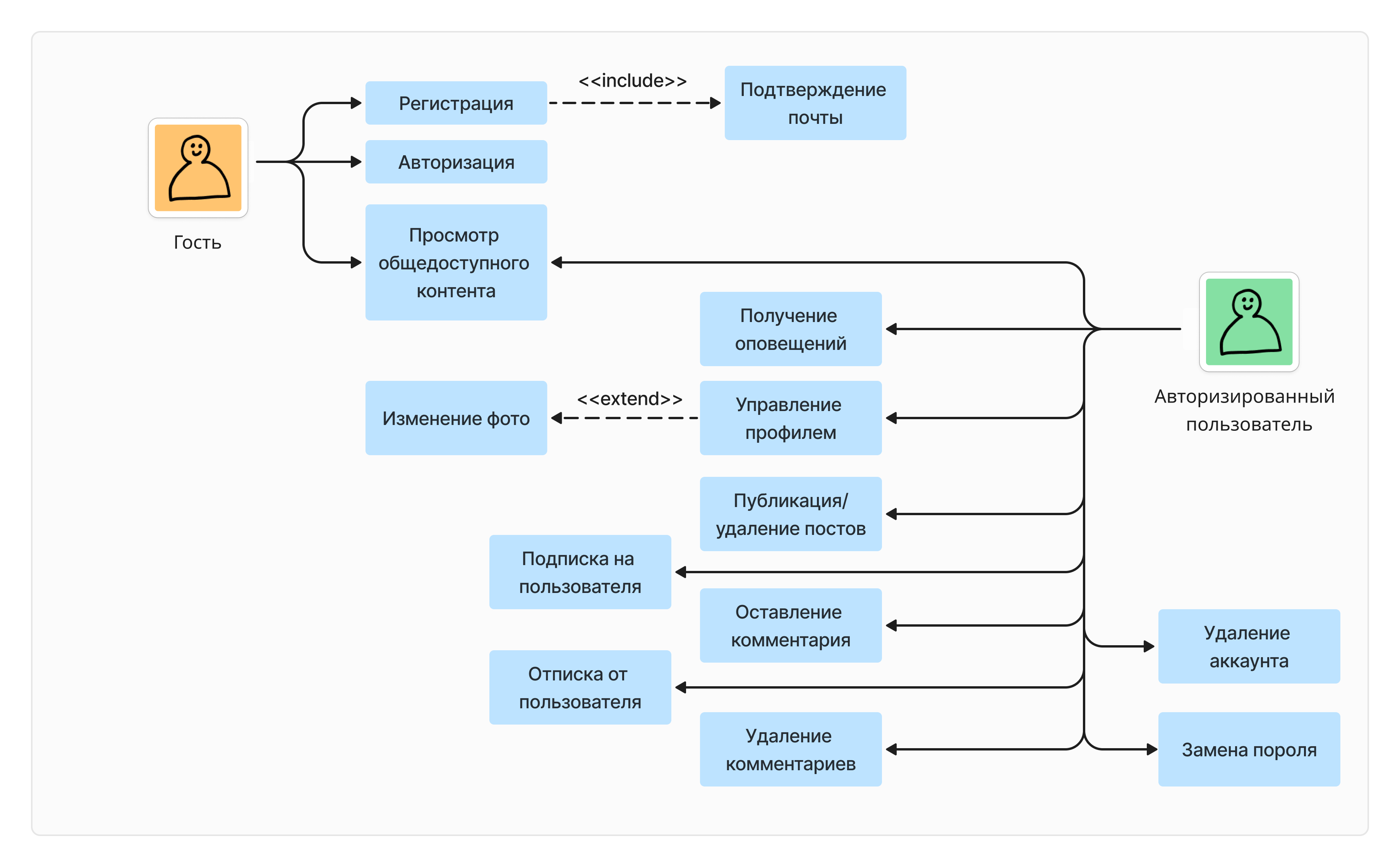


Рисунок 1.5 – Диаграмма вариантов использования

К основным функциональным требованиям относятся:

* регистрация и авторизация пользователей: Приложение должно предоставлять возможность пользователям регистрироваться и входить в систему. Это может включать в себя проверку подлинности по электронной почте и/или двухфакторную аутентификацию;
* публикация контента: Пользователи должны иметь возможность публиковать свой контент в виде текста или изображений. Это может включать в себя функции редактирования и форматирования текста;
* комментирование: Пользователи должны иметь возможность комментировать публикации других пользователей. Это может включать в себя функции ответа на комментарии и оценки комментариев;
* профили пользователей: Пользователи должны иметь возможность создавать и редактировать свои профили. Это может включать в себя функции загрузки аватара, редактирования информации о себе и настройки приватности;
* персонализированные рекомендации: Приложение должно предоставлять персонализированные рекомендации контента на основе интересов и взаимодействий пользователя.
* безопасность: Приложение должно обеспечивать безопасность пользовательских данных. Это может включать в себя шифрование паролей, защиту от атак и соблюдение нормативов по защите данных;
* API Gateway: Приложение должно иметь API Gateway для маршрутизации запросов к соответствующим микросервисам;
* Docker Compose: Все сервисы приложения должны быть управляемы через Docker Compose для обеспечения их совместной работы.

1. Проектирование архитектуры приложения
   1. Обобщенная архитектура

Большое внимание уделяется качеству организации архитектуры приложения, в связи с разработкой схожей системы, было принято решение об использовании микро сервисной архитектурой, а также использование архитектурных паттернов:Gateway, MVC. Взаимодействие сервисов осуществляется благодаря брокерам сообщений: RabbitMQ, GRPC. Данный механизмы позволят легко расширять и масштабировать сервисы, предоставляемые клиенту, не изменяя состояния других частей платформ, что обеспечивает безопасное внедрение новых продуктов на рисунке 2.1.

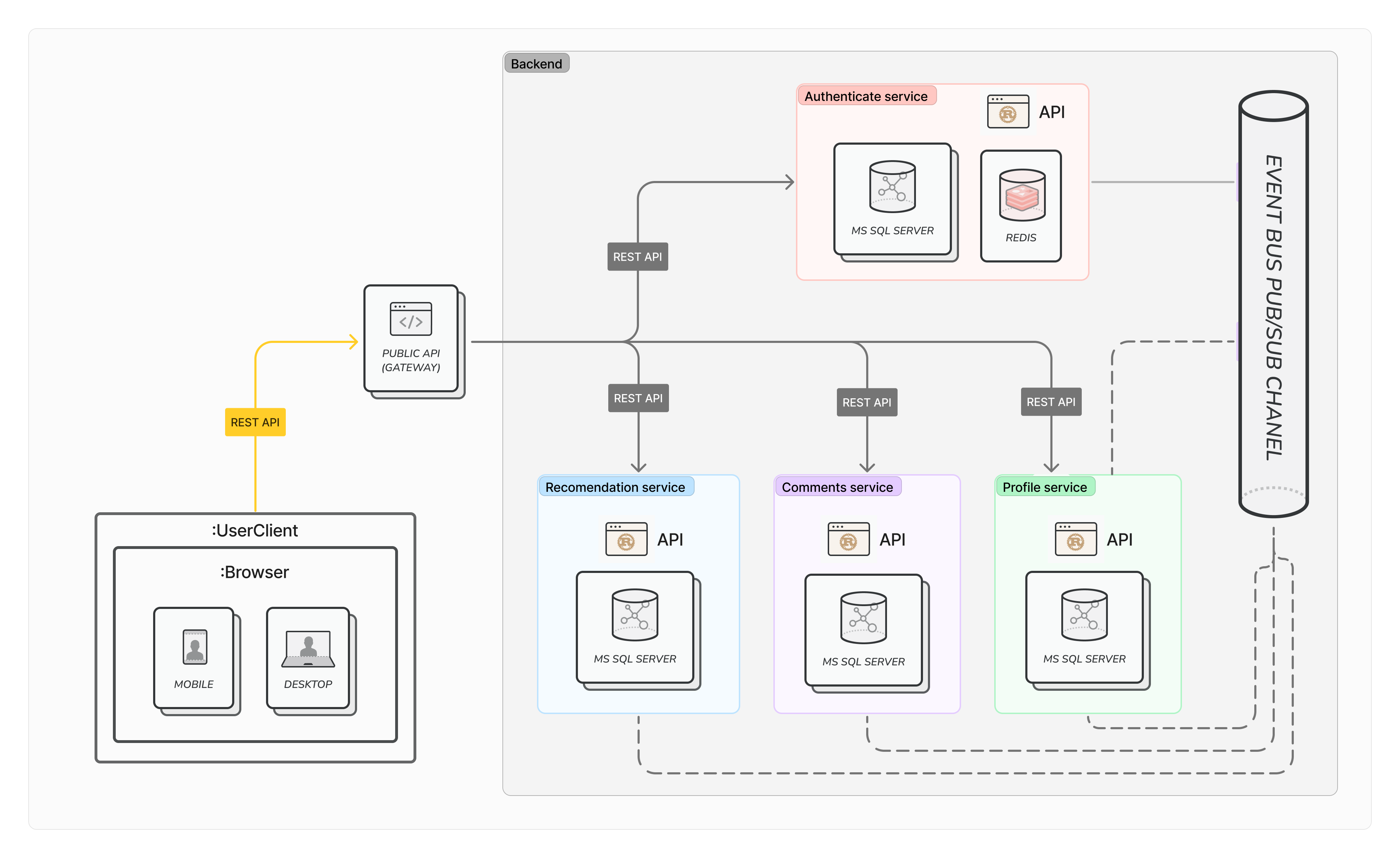


Рисунок 2.1 – Диаграмма размещения

Программное средство состоит из несколько проектов, которые взаимодействуют с друг другом, по средствам инфраструктурного слоя, что позволяет легко и быстро изменить часть приложения, тем самым, не изменять другую часть. Также это позволит в любой момент изменить реализацию одного из модуля и это никак не повлияет на дальнейшую работу системы.

* 1. Диаграмма UML и блок-схема алгоритма

Диаграмма UML представляет собой графическое изображение структуры системы и взаимосвязей между ее элементами. Для веб-приложения необходимо построить диаграмму, которая будет отражать все основные компоненты системы и связи между ними. В данном случае система состоит из нескольких микросервисов: клиент на React, сервисов по авторизации и регистрации, сервис по управлению постами, сервис по работе с профилем, сервис по работе с рекомендациями и сервис по работе с комментариями.

Для создания диаграммы UML можно использовать различные виды диаграмм, такие как диаграмма вариантов использования (Use Case), диаграмма классов (Class Diagram), диаграмма последовательности (Sequence Diagram), диаграмма компонентов (Component Diagram) и другие.

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы последовательности.

Диаграмма описывает взаимодействие пользователя с системой для получения доступа к главной странице.

Сценарий начинается с того, что пользователь пытается получить доступ к главной странице системы. Система проверяет наличие токена, переданного в запросе, и выполняет проверку его валидности. Если токен не валиден, система перенаправляет пользователя на страницу авторизации.

На странице авторизации пользователь вводит свой email и пароль. Система выполняет проверку этих данных на соответствие данным, хранящимся в базе данных системы. Если введенные данные соответствуют данным в базе данных, система создает новый токен, который далее будет использоваться для аутентификации пользователя. Если же введенные данные не соответствуют данным в базе данных, система отображает сообщение об ошибке и пользователь остается на странице авторизации. Диаграмма последовательности проекта представлена на рисунке 2.2.

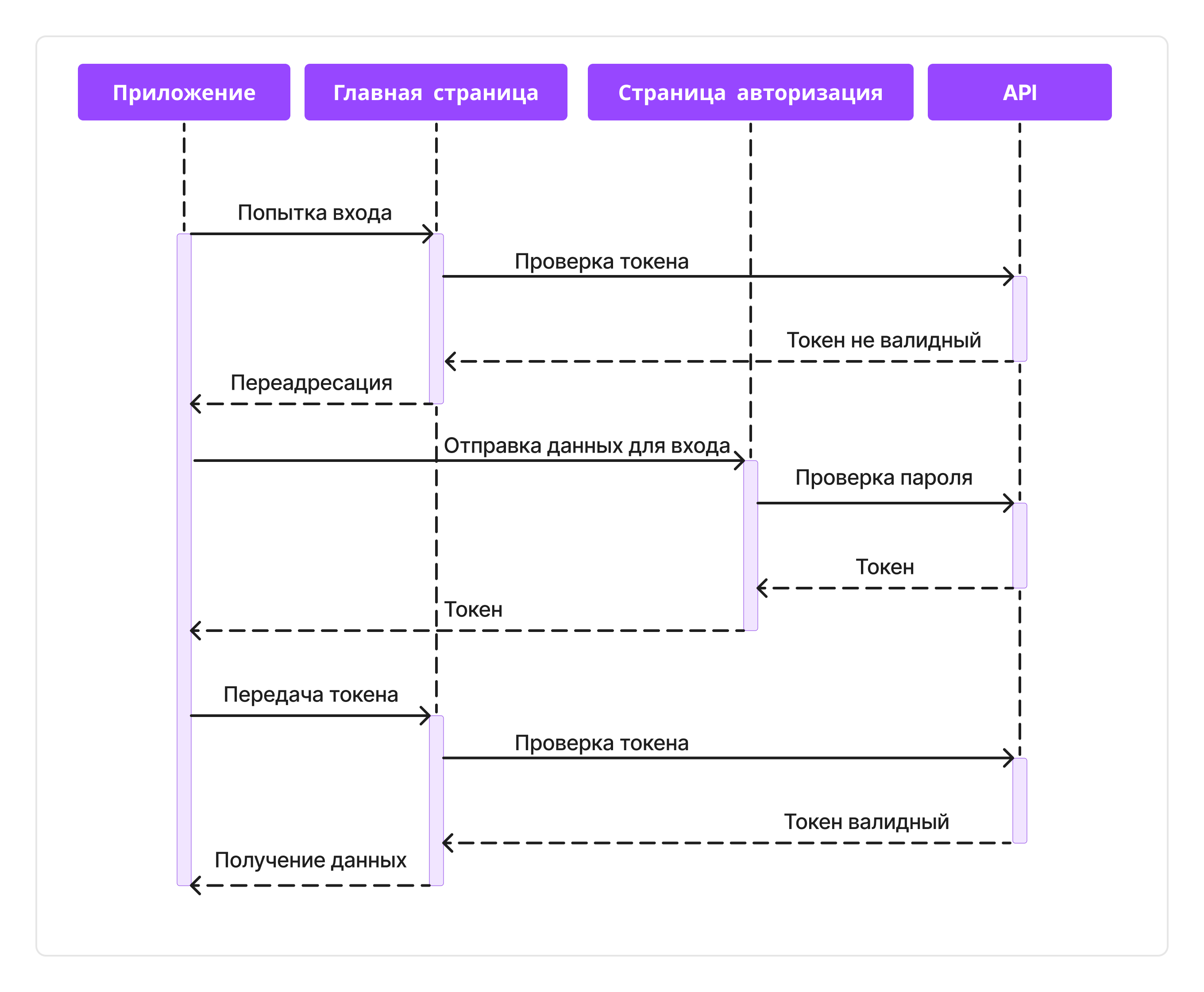


Рисунок 2.2 – Диаграмма последовательности

После успешной авторизации система перенаправляет пользователя обратно на главную страницу, и токен передается в запросе на получение страницы.

Система проверяет валидность токена и, если он валиден, открывает доступ к главной странице.

Блок-схема по регистрации пользователя начинается с заполнения пользователем формы регистрации, в которой он вводит необходимые данные, такие как имя, фамилию, email и пароль. После этого данные проходят процесс валидации, где проверяется их корректность и наличие всех необходимых полей. Если какие-либо данные не проходят валидацию, то процесс регистрации прерывается и пользователю возвращается соответствующий код ошибки, указывающий на то, какое поле заполнено неверно.

Если все данные валидные, то формируется запрос на сервер для проверки наличия такого пользователя в базе данных. Если пользователь с таким email уже существует, то регистрация прерывается и возвращается код ошибки, указывающий на то, что пользователь с таким email уже зарегистрирован. Если же такого пользователя нет, то на основании введенных данных формируется токен, который будет отправлен на указанный email. Блок-схема алгоритма авторизации на рисунке 2.3.

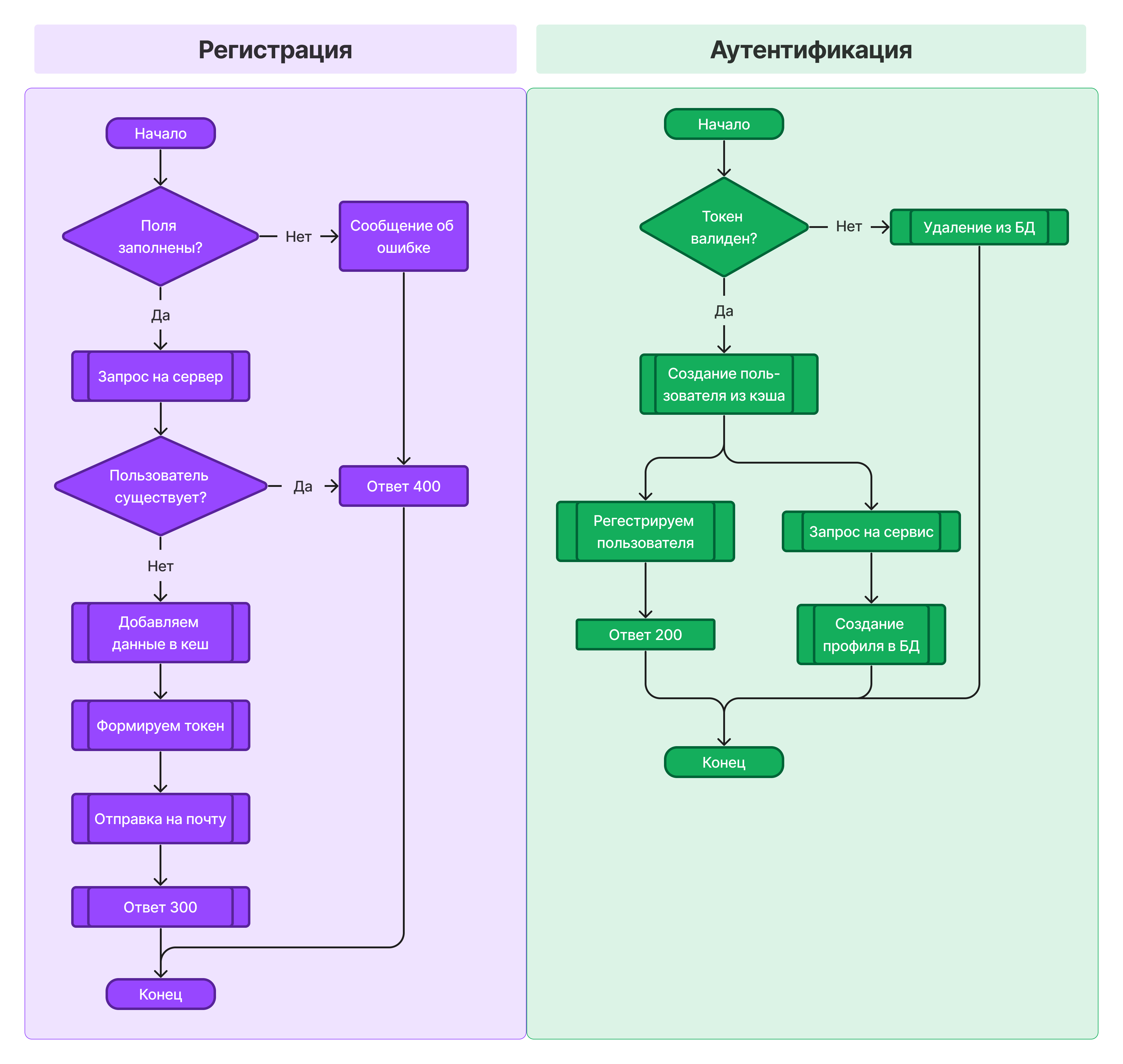


Рисунок 2.3 – Блок-схема авторизации

Чтобы продолжить регистрацию, пользователь должен зайти на свою почту и перейти по ссылке, которая расшифрует токен и получит данные из него. Данные, содержащиеся в токене, отправляются на другой микросервис для создания профиля пользователя.

После успешного создания профиля пользователю возвращается успешный статус код, указывающий на то, что процесс регистрации завершен. Пользователь может войти на сайт, используя свои учетные данные, и получить доступ к функционалу, предоставляемому системой.

* 1. Модель данных

В курсовом проекте используется MS SQL и Redis, которые являются популярными инструментами для хранения и обработки данных.

MS SQL используется как основная база данных для хранения информации, пользователей, профилей, постов и других данных, которые требуют структурированного хранения и связей между таблицами. Он обеспечивает надежность и целостность данных, обеспечивает масштабируемость и поддерживает широкий спектр функциональных возможностей для работы с данными.

Redis используется для хранения данных, которые требуют быстрого доступа, например, кеша данных, информации о токенов пользователей, временных данных, хранения промежуточных результатов вычислений и т.д. Redis является быстрым и эффективным инструментом для хранения данных в оперативной памяти, что позволяет обеспечить быстрый доступ к этим данным.

В целом, использование MS SQL и Redis совместно позволяет создать эффективное и масштабируемое приложение, которое быстро обрабатывает данные и обеспечивает быстрый доступ к ним, при этом обеспечивая надежность и целостность данных.

* 1. Таблицы базы данных

Каждая база данных и каждая таблица будет описана в далее в разделе. Скрипт создания таблиц представлен в приложении Б.

База данных AUTHENTICATE\_SERVICE для хранения идентифицирующих данных представлена на рисунке 2.4.

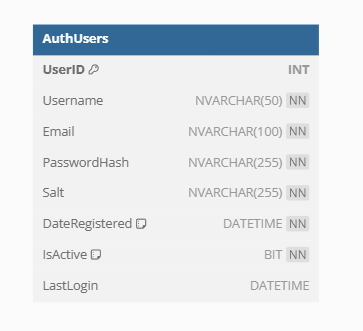


Рисунок 2.4 – Структура базы данных “ AUTHENTICATE\_SERVICE”

Таблица AuthUsers предназначена для хранения основной информации о пользователях, необходимой для процессов аутентификации и идентификации в веб-приложении издательской платформы. Она включает в себя следующие поля:

* UserID: Уникальный идентификатор пользователя в системе. Это поле является первичным ключом и автоматически инкрементируется с каждым новым пользователем;
* Username: Имя пользователя, используемое для входа в систему. Это поле должно быть уникальным среди всех пользователей;
* Email: Адрес электронной почты пользователя, который также используется для входа и восстановления доступа к аккаунту. Адрес почты должен быть уникальным;
* PasswordHash: Хеш пароля пользователя. В это поле записывается хешированный пароль для обеспечения безопасности данных;
* Salt: Случайно сгенерированная строка, используемая в сочетании с паролем пользователя для создания хеша пароля. Это повышает безопасность хранения паролей;
* DateRegistered: Дата и время регистрации пользователя в системе. Значение по умолчанию устанавливается в момент создания записи;
* IsActive: Логическое поле, указывающее, активен ли аккаунт пользователя. Значение 1 означает активный аккаунт, а 0 — деактивированный;
* LastLogin: Дата и время последнего входа пользователя в систему. Это поле может использоваться для отслеживания активности пользователя.

База данных `PROFILE\_SERVICE` предназначена для управления профилями пользователей и их взаимодействием с контентом на издательской платформе. Она включает в себя следующие таблицы:

1. Таблица Users:

* UserID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор пользователя;
* Username (NVARCHAR(50), NOT NULL): Имя пользователя, используемое для идентификации в системе;
* Email (NVARCHAR(100), NOT NULL): Электронная почта пользователя;
* PasswordHash (NVARCHAR(255), NOT NULL): Хеш пароля пользователя;
* DateCreated (DATETIME, NOT NULL): Дата и время создания профиля пользователя;
* ProfilePictureUrl (NVARCHAR(255)): URL изображения профиля пользователя;
* Bio (NVARCHAR(500)): Биографическая информация о пользователе.

1. Таблица Posts:

* PostID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор поста;
* UserID (INT, NOT NULL): Идентификатор пользователя, создавшего пост;
* Title (NVARCHAR(255), NOT NULL): Заголовок поста;
* Content (NVARCHAR(MAX), NOT NULL): Содержимое поста;
* DatePosted (DATETIME, NOT NULL): Дата и время публикации поста;
* DateUpdated (DATETIME): Дата и время последнего обновления поста;
* IsPublished (BIT, NOT NULL): Статус публикации поста.

1. Таблица Likes:

* LikeID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор лайка;
* PostID (INT, NOT NULL): Идентификатор поста, к которому относится лайк;
* UserID (INT, NOT NULL): Идентификатор пользователя, поставившего лайк;
* DateLiked (DATETIME, NOT NULL): Дата и время, когда был поставлен лайк.

1. Таблица Comments:

* CommentID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор комментария;
* PostID (INT, NOT NULL): Идентификатор поста, к которому относится комментарий;
* UserID (INT, NOT NULL): Идентификатор пользователя, написавшего комментарий;
* ParentCommentID (INT): Идентификатор родительского комментария для вложенных комментариев;
* Content (NVARCHAR(1000), NOT NULL): Текст комментария;
* DateCommented (DATETIME, NOT NULL): Дата и время написания комментария.

1. Таблица Subscriptions:

* SubscriptionID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор подписки;
* SubscriberUserID (INT, NOT NULL): Идентификатор пользователя, который подписывается;
* SubscribedToUserID (INT, NOT NULL): Идентификатор пользователя, на которого подписываются;
* DateSubscribed (DATETIME, NOT NULL): Дата и время оформления подписки.

1. Таблица Tags:

* TagID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор тега;
* Name (NVARCHAR(50), NOT NULL): Название тега.

1. Таблица PostTags:

* PostID (INT, NOT NULL): Идентификатор поста;
* TagID (INT, NOT NULL): Идентификатор тега.

Объединение PostID и TagID служит в качестве составного первичного ключа.

Каждая таблица содержит ряд полей, обеспечивающих хранение информации, необходимой для функционирования сервиса профилей. Внешние ключи используются для поддержания целостности данных между таблицами. Уникальные ограничения предотвращают дублирование информации, такое как повторные лайки или подписки. База данных PROFILE\_SERVICE представлена на рисунке 2.5.

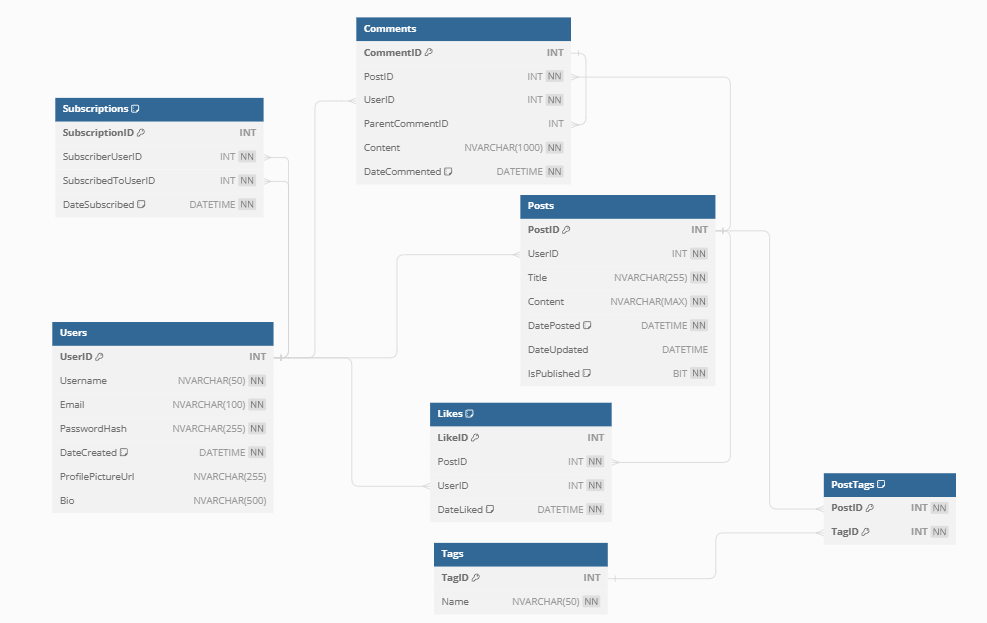


Рисунок 2.5 – Структура базы данных “PROFILE\_SERVICE”

База данных сервиса рекомендаций предназначена для хранения и обработки информации, необходимой для предоставления персонализированных рекомендаций пользователям веб-приложения издательской платформы. Она содержит следующие ключевые элементы:

1. Таблица UserPreferences:

* PreferenceID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор предпочтения пользователя;
* UserID (INT, NOT NULL): Идентификатор пользователя, которому принадлежит предпочтение. Внешний ключ, связанный с таблицей пользователей;
* TagID (INT): Идентификатор тега, который предпочитает пользователь. Внешний ключ, связанный с таблицей тегов;
* Category (NVARCHAR(100)): Категория контента, которую предпочитает пользователь;
* DateUpdated (DATETIME, NOT NULL): Дата и время последнего обновления предпочтения.

1. Таблица News:

* NewsID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор новостной статьи;
* Title (NVARCHAR(255), NOT NULL): Заголовок новостной статьи;
* Content (NVARCHAR(MAX), NOT NULL): Содержимое новостной статьи;
* DatePublished (DATETIME, NOT NULL): Дата и время публикации статьи;
* PublisherID (INT): Идентификатор издателя статьи. Внешний ключ, связанный с таблицей пользователей.

1. Таблица UserViews:

* ViewID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор просмотра;
* UserID (INT, NOT NULL): Идентификатор пользователя, совершившего просмотр;
* ContentID (INT, NOT NULL): Идентификатор просмотренного контента (может быть PostID или NewsID);
* ContentType (NVARCHAR(50), NOT NULL): Тип просмотренного контента (например, 'Post' или 'News') ;
* DateViewed (DATETIME, NOT NULL): Дата и время просмотра.

1. Таблица UserInteractions:

* InteractionID (INT, PRIMARY KEY, IDENTITY): Уникальный идентификатор взаимодействия пользователя с контентом;
* UserID (INT, NOT NULL): Идентификатор пользователя, совершившего взаимодействие;
* ContentID (INT, NOT NULL): Идентификатор контента, с которым было совершено взаимодействие;
* ContentType (NVARCHAR(50), NOT NULL): Тип контента, с которым взаимодействовал пользователь;
* InteractionType (NVARCHAR(50), NOT NULL): Тип взаимодействия (например, 'Like', 'Bookmark');
* DateInteracted (DATETIME, NOT NULL): Дата и время взаимодействия.

Эти таблицы позволяют системе рекомендаций отслеживать интересы и взаимодействия пользователей с различными типами контента, что в свою очередь позволяет формировать персонализированные рекомендации на основе их предпочтений и истории просмотров. База данных RECOMMENDATION\_SERVICE представлена на рисунке 2.6.

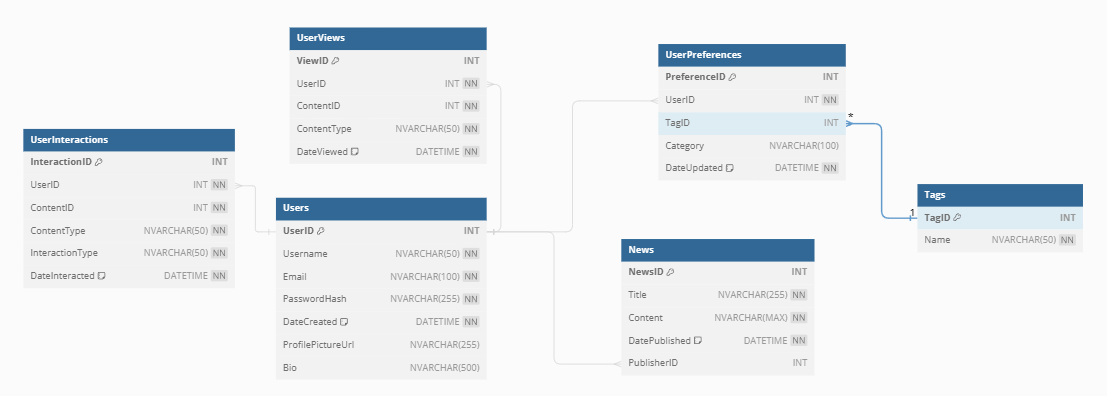


Рисунок 2.6 – Структура базы данных “RECOMMENDATION SERVICE”

* 1. Таблицы базы данных

Каждая таблица базы данных будет описана в далее в таблицах. Скрипт создания таблиц представлен в приложении Б.

Таблица 2.1 – EMPLOYEES

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| emp\_id | NUMBER(6) | Идентификатор сотрудника |
| first\_name | VARCHAR2(20) | Имя сотрудника |
| last\_name | VARCHAR2(25) NOT NULL | Фамилия сотрудника |
| email | VARCHAR2(25) NOT NULL | Электронная почта сотрудника |
| phone\_number | VARCHAR2(25) NOT NULL | Номер телефона сотрудника |
| job\_id | DATE NOT NULL | Идентификатор должности сотрудника |
| salary | VARCHAR2(10) NOT NULL | Зарплата сотрудника |
| commission\_pct | NUMBER(8,2) | Процент комиссии сотрудника |
| manager\_id | NUMBER(6) | Идентификатор менеджера сотрудника |
| department\_id | NUMBER(4) | Идентификатор отдела сотрудника |
| password\_hash | VARCHAR2(128) | Пароль |
| salt | VARCHAR2(128) | Соль для пароля |
| IsAuthentic | NUMBER(1) NOT NULL | Активирован ли аккаунт |

Эта таблица хранит информацию о сотрудниках.

Таблица 2.2 – JOBS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| job\_id | VARCHAR2(10)PRIMARY KEY | Идентификатор должности |
| job\_title | VARCHAR2(35) NOT NULL | Название должности |
| min\_salary | NUMBER(6) | Минимальная зарплата |
| max\_salary | NUMBER(6) | Максимальная зарплата |

Эта таблица хранит информацию о должностях.

Таблица 2.3 – DEPARTMENTS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| department\_id | NUMBER(4)PRIMARY KEY | Идентификатор отдела |
| department\_name | VARCHAR2(30) NOT NULL | Название отдела |
| manager\_id | NUMBER(6) | Идентификатор менеджера отдела |
| location\_id | NUMBER(4) | Идентификатор местоположения отдела |

Эта таблица хранит информацию об отделах.

Таблица 2.4 – LOCATIONS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| location\_id | NUMBER(4) PRIMARY KEY | Идентификатор местоположения |
| street\_address | VARCHAR2(40) | Улица местоположения |
| postal\_code | VARCHAR2(12) | Почтовый индекс |
| City | VARCHAR2(30) NOT NULL | Город местоположения |
| state\_province | VARCHAR2(25) | Область местоположения |
| country\_id | CHAR(2) NOT NULL | Идентификатор страны |

Эта таблица хранит информацию местоположений отдел.

Таблица 2.5 – COUNTRIES

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| country\_id | CHAR(2) PRIMARY KEY | Идентификатор страны |
| country\_name | VARCHAR2(40) NOT NULL | Название страны |

Эта таблица хранит информацию о странах.

Таблица 2.6 – PROJECTS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| project\_id | NUMBER(4) PRIMARY KEY | Идентификатор проекта |
| project\_name | VARCHAR2(50) NOT NULL | Название проекта |
| end\_date | DATE | Дата окончания проекта (может быть null) |
| start\_date | DATE NOT NULL | Дата начала проекта |

Продолжение таблицы 2.6

|  |  |  |
| --- | --- | --- |
| budget | NUMBER(12,2) | Бюджет проекта |
| department\_id | NUMBER(4) NOT NULL | Идентификатор отдела, ответственного за проект |

Эта таблица хранит информацию о проектах.

Таблица 2.7 – PARTICIPATION

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| emp\_email | VARCHAR2(25) | Электронная почта сотрудника |
| project\_id | NUMBER(4) NOT NULL | Идентификатор проекта, в котором участвует сотрудник |
| role | VARCHAR2(50) | Роль сотрудника в проекте |
| Hours | NUMBER(4) | Количество часов, затраченных сотрудником на проект |
| PRIMARY KEY (emp\_id, project\_id) |  | Составной первичный ключ из emp\_id и project\_id |

Эта таблица хранит информацию об участии сотрудников в проектах.

Таблица 2.8 – TASKS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| task\_id | NUMBER(6) PRIMARY KEY | Идентификатор задачи |
| task\_name | VARCHAR2(100) NOT NULL | Название задачи |
| Description | VARCHAR2(500) | Описание задачи |
| Duration | NUMBER(4) NOT NULL | Продолжительность задачи в часах |
| project\_id | NUMBER(4) NOT NULL | Идентификатор проекта, к которому относится задача |

Эта таблица хранит информацию о задачах сотрудников.

Таблица 2.9 – VACATIONS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| vacation\_id | NUMBER(6)PRIMARY KEY | Идентификатор отпуска |
| Reason | VARCHAR2(500) | Причина отпуска |
| emp\_email | VARCHAR2(25) | Электронная почта сотрудника |
| start\_date | DATE NOT NULL | Дата начала отпуска |
| end\_date | DATE NOT NULL | Дата окончания отпуска |

Эта таблица хранит информацию о отпусках сотрудников.

1. Разработка web-приложения
   1. Микросервисы

Приложение разделено на множество сервисов, что позволило сделать проект модульным. Исходя из этого многие реализация являются независимыми, что позволяет легко менять либо подставлять другую реализацию

* 1. Создание пользователе й, ролей, и таблиц базы данных

Для работы с базой данных необходимо выделить несколько пользователей с определёнными привилегиями. Начало разработки базы данных начинается с созданий её объектов. Для этого создадим пользователя и администратора. Администратору будут выданы привилегии создания сессии, создание и изменение основных объектов базы данных: таблицы, последовательностей и т.д. Пользователю будут выданы права на вызов получение данных из определённых таблиц. С выданными правами пользователь сможет использовать основную функциональность базы данных. Скрипт создания представлен на листинге 3.1.

|  |
| --- |
| *-- Создание профиля для пользователя* CREATE PROFILE user\_profile LIMIT  FAILED\_LOGIN\_ATTEMPTS 5  SESSIONS\_PER\_USER 10  CPU\_PER\_SESSION UNLIMITED  CPU\_PER\_CALL 3000  CONNECT\_TIME 45  IDLE\_TIME 15  LOGICAL\_READS\_PER\_SESSION DEFAULT  LOGICAL\_READS\_PER\_CALL DEFAULT  PRIVATE\_SGA 15K  COMPOSITE\_LIMIT 5000000;  *-- Создание профиля для администратора* CREATE PROFILE admin\_profile LIMIT  SESSIONS\_PER\_USER UNLIMITED  CPU\_PER\_SESSION UNLIMITED  CPU\_PER\_CALL UNLIMITED  CONNECT\_TIME UNLIMITED  IDLE\_TIME UNLIMITED  LOGICAL\_READS\_PER\_SESSION DEFAULT  LOGICAL\_READS\_PER\_CALL DEFAULT  PRIVATE\_SGA UNLIMITED  COMPOSITE\_LIMIT UNLIMITED;  CREATE USER admin\_user IDENTIFIED BY admin\_password  DEFAULT TABLESPACE USERS\_TBS  TEMPORARY TABLESPACE TEMP\_TBS  PROFILE admin\_profile;  CREATE USER user\_user IDENTIFIED BY user\_password  DEFAULT TABLESPACE USERS\_TBS  TEMPORARY TABLESPACE TEMP\_TBS  PROFILE user\_profile; |

Листинг 3.1 – Создание пользователя для менеджера

* 1. Пользовательские процедуры

Следующим этапом будет разработка основных процедур, для осуществления работы с базой данных. После этого станет ясно какие дополнительные объекты базы данных будет необходимо создать для оптимизации выполнения процедур.

На листинге 3.2 представлена процедура для повышения сотрудника в должности. Процедура promote\_employee обновляет должность сотрудника в таблице employees. Затем процедура выполняет операцию UPDATE, чтобы изменить должность сотрудника на новую, и, если обновление проходит успешно, происходит COMMIT, который сохраняет изменения в базе данных.

|  |
| --- |
| create PROCEDURE promote\_employee ( p\_emp\_id IN employees.emp\_id%TYPE,  p\_new\_job\_id IN employees.job\_id%TYPE) IS BEGIN  UPDATE employees SET job\_id = p\_new\_job\_id WHERE emp\_id = p\_emp\_id; COMMIT; EXCEPTION WHEN OTHERS THEN  ROLLBACK; RAISE; END promote\_employee; |

Листинг 3.2 – Процедура для повышения сотрудника в должности

На листинге 3.3 представлена процедура для найма сотрудника. Процедура hire\_employee вставляет новую запись в таблицу employees. Значение password\_hash шифруется функцией ENCRYPT перед вставкой. Если вставка проходит успешно, происходит COMMIT, который сохраняет изменения в базе данных.

|  |
| --- |
| CREATE PROCEDURE hire\_employee (p\_first\_name VARCHAR2,p\_last\_name VARCHAR2,p\_email VARCHAR2,p\_phone\_number VARCHAR2,p\_job\_id VARCHAR2,p\_salary NUMBER,p\_commission\_pct NUMBER,p\_manager\_id NUMBER,p\_department\_id NUMBER,p\_password\_hash VARCHAR2 ) IS v\_emp\_id employees.emp\_id%TYPE; v\_salt employees.salt%TYPE; BEGIN  INSERT INTO employees (  emp\_id, first\_name, last\_name, email, phone\_number, job\_id, salary, commission\_pct, manager\_id, department\_id, password\_hash, salt, ISAUTHENTIC)  VALUES (v\_emp\_id, p\_first\_name, p\_last\_name, p\_email, p\_phone\_number, p\_job\_id, p\_salary, p\_commission\_pct, p\_manager\_id, p\_department\_id, ENCRYPT(p\_password\_hash, v\_salt), v\_salt, 0);  COMMIT; EXCEPTION WHEN OTHERS THEN ROLLBACK; RAISE; END hire\_employee; |

Листинг 3.3 – Процедура для найма сотрудника

В листинге 3.4 представлена процедура для увольнения сотрудника. Процедура fire\_employee удаляет сотрудника из таблицы employees.

|  |
| --- |
| create PROCEDURE fire\_employee (p\_emp\_id NUMBER) IS BEGIN  -- Удалить данные о сотруднике из таблицы employees  DELETE FROM employees WHERE emp\_id = p\_emp\_id;  -- Вывести сообщение об успешном увольнении сотрудника  DBMS\_OUTPUT.PUT\_LINE('Employee ' || p\_emp\_id || ' has been fired.');  COMMIT; EXCEPTION  WHEN OTHERS THEN  ROLLBACK; RAISE; END fire\_employee;/ |

Листинг 3.4 – Процедура для увольнения сотрудника

Так же, для всех таблиц были реализованы процедуры, которые обеспечивают базовые операции управления данными в базе данных, такие как вставка, обновление и удаление данных. Некоторые вспомогательные процедуры представлены ниже в таблице 3.1.

Таблица 3.1 – Вспомогательные процедуры

|  |  |
| --- | --- |
| Название процедуры | Описание процедуры |
| encrypt(p\_plain\_text VARCHAR2) | шифрует обычный текст. |
| decrypt(p\_encrypted\_text RAW) | расшифровывает зашифрованный текст |
| generate\_salt(p\_email VARCHAR2) | генерирует соль на основе электронной почты сотрудника |
| export\_json() | экспортирует данные из таблицы employees в файл JSON |
| import\_json() | импортирует данные из файла JSON в таблицу employees |

* 1. Индексы базы данных

Индексы — это структуры данных, построенные на основе одного или нескольких столбцов таблицы. Индекс предоставляет быстрый доступ к данным таблицы и позволяет ускорить выполнение операций SELECT, UPDATE и DELETE, так как он содержит отсортированные значения из одного или нескольких столбцов и указатели на строки таблицы.

Применение индексов представляет собой компромисс между ускорением получения результатов запросов и замедлением обновлений и вставок данных. Первая часть этого компромисса – ускорение запросов – довольно очевидна: если поиск выполняется по отсортированному индексу вместо полного сканирования всей таблиц, то запрос проходит намного быстрее. Но всякий раз, когда вы обновляете, вставляете или удаляете строку таблицы с индексами, индексы также должны быть обновлены соответствующим образом. То есть такие операции на таблицах с индексами обходятся дороже.

На основании проделанных тестов производительности было принято решение о создании индексов для основных таблиц.

* 1. Последовательности

Последовательности в SQL используются для генерации уникальных числовых идентификаторов. Они часто используются для автоматического создания первичных ключей. Создание последовательностей представлено в листинге 3.5.

Каждая последовательность начинается с 1 и увеличивается на 1 при каждом вызове. Это обеспечивает уникальность каждого идентификатора и предотвращает возможные конфликты при вставке новых записей в таблицы.

|  |
| --- |
| *-- Создание последовательностей* CREATE SEQUENCE department\_id\_seq START WITH 1 INCREMENT BY 1;CREATE SEQUENCE location\_id\_seq START WITH 1 INCREMENT BY 1; CREATE SEQUENCE project\_id\_seq START WITH 1 INCREMENT BY 1;  CREATE SEQUENCE task\_id\_seq START WITH 1 INCREMENT BY 1; CREATE SEQUENCE emp\_id\_seq START WITH 1 INCREMENT BY 1; CREATE SEQUENCE vacation\_id\_seq START WITH 1 INCREMENT BY 1; CREATE SEQUENCE participation\_id\_seq START WITH 1 INCREMENT BY 1; |

Листинг 3.5 – Создание последовательностей

Создание этих последовательностей позволяет нам автоматизировать процесс присвоения идентификаторов, что упрощает вставку новых данных и уменьшает вероятность ошибок.

* 1. Триггеры

Триггеры в SQL - это специальные процедуры, которые автоматически запускаются при определенных событиях в базе данных, таких как вставка, обновление или удаление записей. Они играют важную роль в поддержании целостности данных и могут быть использованы для автоматизации определенных операций.

В данной работе триггеры обеспечивают автоматическое присвоение уникальных идентификаторов при вставке новых записей в таблицы. Это упрощает процесс вставки данных и уменьшает вероятность ошибок, связанных с дублированием идентификаторов. Таким образом, триггеры помогают обеспечить целостность данных и улучшить эффективность работы с базой данных. Листинг создания триггеров представлен в приложении В.

* 1. Директории

Для реализации экспорта и импорта в JSON создадим необходимую директорию. Реализация представлена дальше в листинге 3.6

|  |
| --- |
| *CREATE DIRECTORY MY\_DIRECTORY AS 'C:\Humanix\Files\JSON';* |

Листинг 3.6 – Создание директории для хранения результатов импорта

В ней будут создаваться файлы JSON экспорта и импорта. Объект Directory является логической ссылкой в базе данных на каталог файловой системы сервера, где установлена БД Oracle. Владельцем всех объектов Directory в базе данных является пользователь SYS, даже если объект Directory создан другим пользователем. Имена объектов Directory уникальны внутри всей БД. Все объекты Directory хранятся в табличном пространстве SYS.

1. Описание процедур импорта и экспорта
   1. Экспорт в JSON

Процедура export\_json предназначена для экспорта данных из таблицы employees в формате JSON. Процедура открывает файл employees\_export.json для записи и затем проходит по всем записям в таблице employees. Каждая запись преобразуется в JSON-объект с помощью функции JSON\_OBJECT, где каждому полю записи соответствует ключ JSON-объекта. Полученный JSON-объект записывается в файл. После обработки всех записей файл закрывается. Если в процессе работы процедуры происходит ошибка, файл также закрывается, и ошибка передается вызывающей стороне. Это обеспечивает корректное завершение работы с файлом даже в случае возникновения ошибок. Этот файл можно использовать для обмена данными между различными системами или для резервного копирования данных. Скрипт создания процедуры export\_json представлен в приложении Г.

* 1. Импорт из JSON

Процедура import\_json предназначена для импорта данных в формате JSON в таблицу employees. Скрипт создания процедуры import\_json представлен в приложении Д. Процедура открывает файл employees\_import.json для чтения и считывает все строки в переменную v\_data. Затем эти данные преобразуются в массив JSON-объектов. Процедура проходит по каждому JSON-объекту в массиве, извлекает данные и вставляет их в таблицу employees. Если в процессе работы процедуры происходит ошибка, все изменения откатываются, и ошибка передается вызывающей стороне. Это обеспечивает корректное завершение работы с файлом и таблицей даже в случае возникновения ошибок.

1. Тестирование производительности

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса. Для тестирования производительности мы добавляем в таблицу Employees 100000 строк. Скрипт вставки представлен на листинге 5.1.

|  |
| --- |
| DECLARE v\_counter NUMBER := 0; BEGIN  WHILE v\_counter <= 100000 LOOP  INSERT INTO employees (first\_name, last\_name, email, phone\_number, job\_id, salary, commission\_pct, manager\_id, department\_id, ISAUTHENTIC) VALUES (  DBMS\_RANDOM.STRING('A', 10),  DBMS\_RANDOM.STRING('A', 10),  'email' || TO\_CHAR(v\_counter) || '@example.com',  '1234567890',  'AD\_PRES',  DBMS\_RANDOM.VALUE(2000, 8000),  DBMS\_RANDOM.VALUE(0, 0.2),  null,  10,  0  );  v\_counter := v\_counter + 1;  END LOOP; END; / |

Листинг 5.1 – Вставка 100000 строк в таблицу

Для получения плана выполнения SQL-запроса в Oracle можно воспользоваться инструментом EXPLAIN PLAN используя функцию “Explain Plan”. [Результат будет отображаться в специальной вкладке “Plan” в формате, смешивающем древовидную структуру и таблицу.](https://www.jetbrains.com/help/datagrip/query-execution-plan.html)

Каждая строка плана выполнения представляет операцию, которую база данных выполнит, а также ее стоимость, учитывая различные факторы, такие как использование индексов, объем данных и другие. Каждая операция оценивается по ожидаемому количеству возвращаемых строк. В данном случае план выполнения состоит из двух строк:

SELECT STATEMENT: это верхний уровень плана выполнения, который описывает операцию SELECT в целом.

TABLE ACCESS FULL: это операция, которая читает все строки таблицы (без использования индексов), и возвращает их в качестве результата запроса. Стоимость этого плана выполнения равна 377 (без указания процента использования ресурсов ЦПУ) и означает, что база данных ожидает, что выполнение этого запроса займет примерно 377 единицы измерения работы (например, время CPU или количество операций ввода-вывода). Результат представлен на рисунке 5.1.

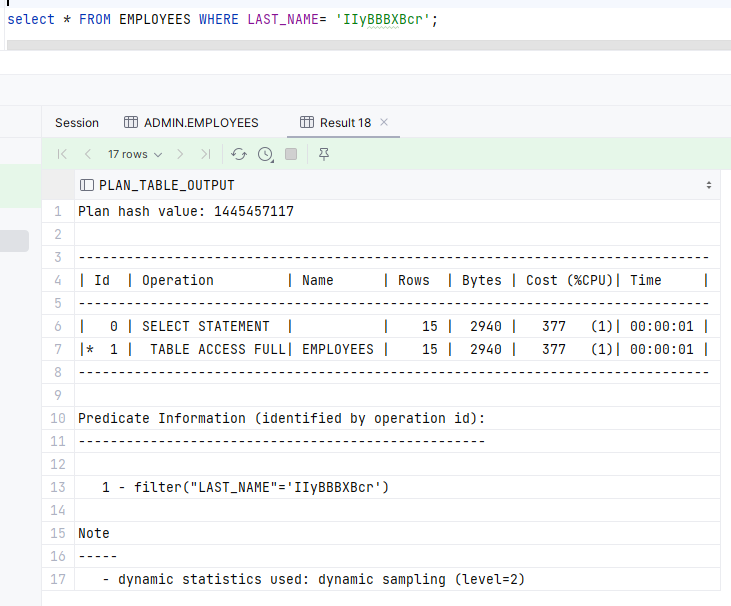


Рисунок 5.1 – Получения плана выполнения запроса

Проведенный анализ производительности базы данных с таблицей EMPLOYEES, содержащей большое количество строк, позволяет сделать следующие выводы. Первоначально, до создания индексов, запрос к таблице осуществлялся полным сканированием, что может привести к замедлению при работе с большим объемом данных.

Создание индексов на соответствующих столбцах таблицы существенно улучшило время выполнения запроса. Индексы позволяют эффективнее организовывать доступ к данным, снижая стоимость выполнения операций. Таким образом, база данных оказывается более подготовленной к обработке больших объемов данных, что является важным аспектом в условиях активной работы системы с множеством запросов и операций.

1. Описание технологии и ее применения в базе данных
   1. Шифрование данных в Oracle

Для шифрования данных в Oracle существует пакет DBMS\_CRYPTO позволяющий использовать алгоритмы симметричного и асимметричного шифрования, хэширования и цифровой подписи. DBMS\_CRYPTO поддерживает алгоритмы шифрования, такие как AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard), RSA (Rivest-Shamir-Adleman) и другие. В данном курсовом проекте мы будем использовать алгоритм AES (Advanced Encryption Standard) для шифрования и дешифрования данных в базе данных Oracle. 256 AES является одним из наиболее распространенных и надежных алгоритмов симметричного шифрования, который широко применяется в различных сферах, включая информационную безопасность, защиту данных и конфиденциальность. AES основан на блочном шифровании, где данные разбиваются на фиксированные блоки и каждый блок шифруется независимо. Алгоритм AES поддерживает различные длины ключей, включая 128, 192 и 256 бит, что позволяет выбирать уровень безопасности и производительности в соответствии с требованиями проекта. Функция шифрования данных, используется, если в таблице нужно зашифровать конфиденциальные данные. Функция представлена на листинге 6.1

|  |
| --- |
| CREATE OR REPLACE FUNCTION encrypt(  p\_plain\_text VARCHAR2,  p\_salt VARCHAR2)  RETURN RAW IS  encryption\_key RAW(256) := HEXTORAW('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF');  encrypted\_raw RAW(2048); BEGIN  encrypted\_raw := DBMS\_CRYPTO.ENCRYPT(  src => UTL\_I18N.STRING\_TO\_RAW(p\_plain\_text || p\_salt, 'AL32UTF8'),  typ => DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  key => encryption\_key  );  RETURN encrypted\_raw; END encrypt;  / |

Листинг 6.1 – Функция для шифрования данных

Реализация функции дешифрования представлены в листинге 6.2.

|  |
| --- |
| CREATE OR REPLACE FUNCTION decrypt(p\_encrypted\_text RAW, p\_salt VARCHAR2) RETURN VARCHAR2 IS  encryption\_key RAW(256) := HEXTORAW('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF');  decrypted\_raw RAW(2048); BEGIN  decrypted\_raw := DBMS\_CRYPTO.DECRYPT(  src => p\_encrypted\_text,  typ => DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  key => encryption\_key);  RETURN separate\_string(UTL\_I18N.RAW\_TO\_CHAR(decrypted\_raw, 'AL32UTF8'), p\_salt); END decrypt; |

Листинг 6.2 – Функция для дешифрования данных

* 1. Аутентификация

Аутентификация пользователей является важной частью любой системы управления базами данных. Это процесс проверки идентификации пользователя, который пытается получить доступ к системе. Один из распространенных методов аутентификации - это аутентификация посредством электронной почты.

Аутентификация посредством электронной почты – это процесс, при котором пользователь вводит свой адрес электронной почты и пароль для входа в систему. Этот метод обеспечивает уровень безопасности, поскольку только пользователь, знающий правильный адрес электронной почты и пароль, сможет получить доступ.

После найма сотрудника на работу и создания его аккаунта администратором, пользователь обязан активироваться в системе посредством ввода аутентификационного кода, полученного по электронной почте.

* 1. Аудит SYS операций

По умолчанию в базе данных Oracle не активирован аудит исполняемых команд SQL, выполняемый в привилегированном режиме от учетной записи SYS, а также при подключении пользователей с привилегиями (учетными записями) SYSDBA или SYSOPER. Так если база данных будет взломана, эти учетные записи будут использованы хакерами в первую очередь. Дабы этого избежать необходимо включить аудит SQL-команд из этих привилегированных учетных записей следующей командой, представленной на листинге 6.3.

|  |
| --- |
| alter system set audit\_sys\_operations=true scope=spfile; |

Листинг 6.3 – Включение аудита команд

* 1. Защита слушателя

Для слушателя всегда следует применять пароль, чтобы воспрепятствовать подключению неправомочных пользователей к базе данных Oracle. Как только пароль для слушателя установлен, привилегированные действия, такие как остановка или запуск слушателя, не смогут выполняться без ввода соответствующего пароля. Можно также запретить пользователю применять команду SET для вмешательства в функции слушателя. Для этого потребуется добавить строку, представленную на листинге 6.4, в файл конфигурации listener.ora.

|  |
| --- |
| ADMIN\_RESTRICTIONS=ON |

Листинг 6.4 – Параметр ADMIN\_RESTRICTIONS

По умолчанию этот параметр установлен в значение false. Следует также избегать удаленного управления службой слушателя, поскольку ее пароль не шифруется при передаче по сети. Пароль слушателя хранится в файле listener.ora, поэтому нужно защитить этот файл.

1. Краткое описание приложения для демонстрации

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс — Windows Presentation Foundation (WPF), позволяющий создавать красивые и производительные приложения с разнообразными элементами управления и реагировать на различные действия пользователя.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом — С#, разработанный для создания приложений на платформе Microsoft .NET Framework.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). В разрабатываемом приложении используется архитектурный шаблон Model-View-ViewModel (MVVM).

Шаблон MVVM имеет три основных слоя: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления.

1. Руководство пользователя
   1. Пример работы приложения от лица пользователя

После открытия приложения пользователя встречает форма для входа, которая представлена на рисунке 8.1. Здесь пользователь может ввести свой логин и пароль для авторизации в системе. После успешной авторизации система автоматически определяет роль пользователя и предоставляет ему соответствующие возможности в зависимости от роли, которую он имеет.

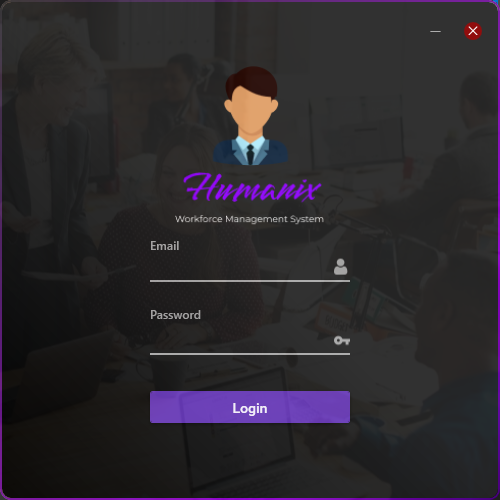


Рисунок 8.1 – Авторизация в приложении

После успешной авторизации в приложении открывается окно с профилем сотрудника. В данном окне пользователь имеет возможность отредактировать некоторую персональную информацию. Страница профиля представлена на рисунке 8.2.

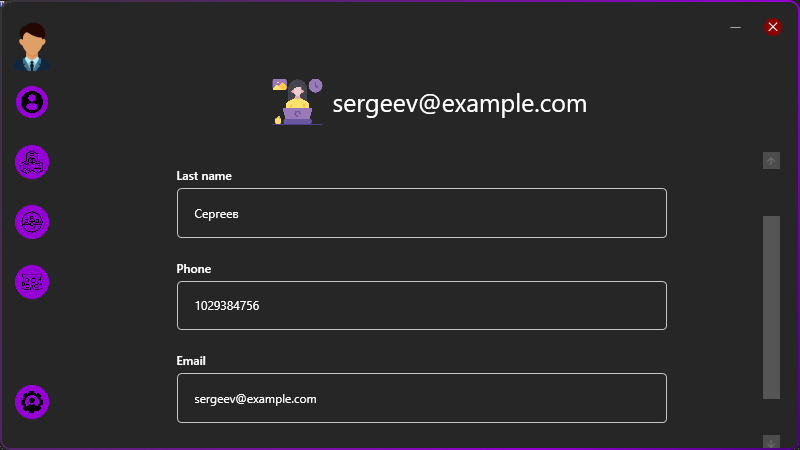


Рисунок 8.2 – Страница профиля

При помощи панели, представленной в левой части окна, пользователь имеет возможность открыть страницы со списком сотрудников в своем отделе, списком проектов, в которых он принимает участие, и списком отпусков.

* 1. Пример работы приложения от лица администратора

Если пользователь авторизован с правами администратора, то в панели, представленной в левой части окна, появляется дополнительный пункт меню, при нажатии на который открывается панель администратора. В ней, пользователь с права администратора имеет возможность просматривать содержимое всех таблиц, представленных в базе данных, а также, в случае таблицы сотрудников, увольнять, нанимать и повышать в должности. Панель администратора представлена на рисунке 8.3.

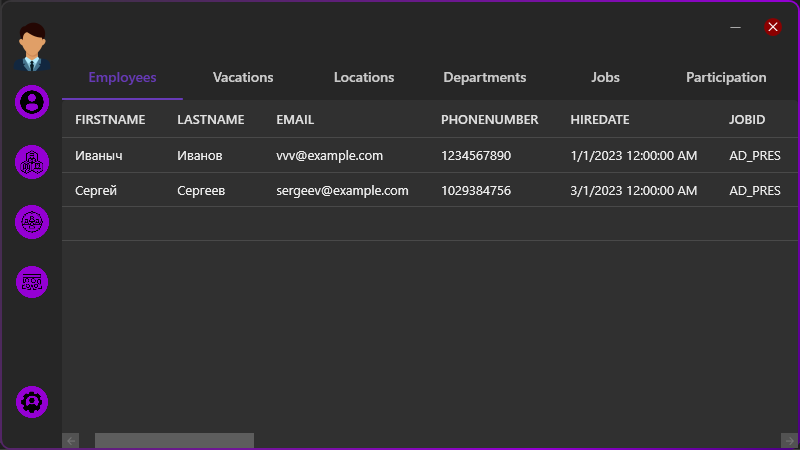


Рисунок 8.3 – Панель администратора

В случае авторизации без прав администратора, пользователь имеет доступ только к общим данным, таким как список сотрудников в его отделе отпуска и т.д. Пользователь не может влиять на состав и статус сотрудников, а также на другие аспекты работы системы.

Заключение

В ходе выполнения курсового проекта была успешно разработана реляционная база данных для учета кадровых данных с использованием системы управления базами данных Oracle 21c.

Был проведен анализ предметной области, в результате которого были определены основные сущности и атрибуты базы данных. На основе этого анализа была разработана концептуальная модель базы данных в нотации ER-диаграмм, которая затем была преобразована в логическую модель в нотации реляционной алгебры.

Эта логическая модель была реализована в физической модели с использованием Oracle. База данных была наполнена тестовыми данными для проверки ее работоспособности и эффективности.

Для реализации бизнес-логики базы данных были разработаны хранимые процедуры, функции и триггеры. Эти триггеры обеспечивают автоматическое присвоение уникальных идентификаторов при вставке новых записей в таблицы, что упрощает процесс вставки данных и уменьшает вероятность ошибок, связанных с дублированием идентификаторов.

Были разработаны запросы для получения необходимой информации из базы данных, что позволяет быстро и эффективно получать данные для анализа и принятия решений.

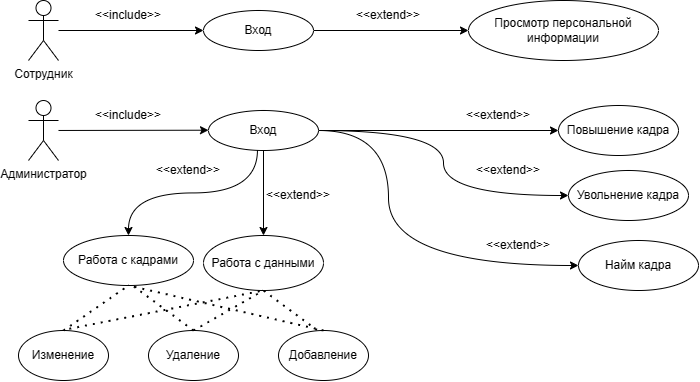
Таким образом, цель данной работы была успешно достигнута. Разработанная база данных обеспечивает эффективное управление и хранение информации о персонале, обеспечивает быстрый доступ к хранимой информации и упрощает процесс учета кадровых данных.

Список используемых источников

1. Фейерштуйн С., Прибыл Б. Ф36 Oracle PL/SQL. Для профессионалов. 6-е издание — СПб.: Питер, 2015. — 1024 с.: ил. — (Серия «Бестселлеры O’Reilly»). Стр. 486 – 752 . – Дата доступа: 15.09.2023.
2. Нанда А., Фейерштейн С. Oracle PL/SQL для администраторов баз данных. СПб: Символ-Плюс, 2008. Стр. 114 – 148. – Дата доступа: 27.09.2023.
3. Работа с данными формата Xml в Oracle [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS\_XMLGEN.html. – Дата доступа: 28.09.2023.
4. Oracle Advanced Analytics: анализ «больших данных» [Электронный ресурс]. – Режим доступа: https://bytemag.ru/oracle-advanced-analytics-analiiz-bolshiih-dannie-8198/. – Дата доступа: 30.09.2023.

Приложение А

Диаграмма вариантов использования



Приложение Б

Листинг создания таблиц

|  |
| --- |
| -- Создаем таблицу сотрудников с первичным ключом emp\_id CREATE TABLE employees (  emp\_id NUMBER(6) PRIMARY KEY , -- Идентификатор сотрудника  first\_name VARCHAR2(20), -- Имя сотрудника  last\_name VARCHAR2(25) NOT NULL, -- Фамилия сотрудника  email VARCHAR2(25) , -- Электронная почта сотрудника  phone\_number VARCHAR2(15), -- Номер телефона сотрудника  hire\_date DATE NOT NULL, -- Дата найма сотрудника  job\_id VARCHAR2(10) NOT NULL, -- Идентификатор должности сотрудника  salary NUMBER(8,2), -- Зарплата сотрудника  commission\_pct NUMBER(2,2), -- Процент комиссии сотрудника  manager\_id NUMBER(6), -- Идентификатор менеджера сотрудника  department\_id NUMBER(4), -- Идентификатор отдела сотрудника  password\_hash VARCHAR2(128)  salt VARCHAR2(128), -- Соль для пароля сотрудника  IsAuthentic NUMBER(1) NOT NULL ) TABLESPACE USERS\_TBS;  -- Создаем таблицу должностей с первичным ключом job\_id CREATE TABLE jobs (  job\_id VARCHAR2(10) PRIMARY KEY, -- Идентификатор должности  job\_title VARCHAR2(35) NOT NULL, -- Название должности  min\_salary NUMBER(6), -- Минимальная зарплата для должности  max\_salary NUMBER(6) -- Максимальная зарплата для должности ) TABLESPACE USERS\_TBS;  -- Создаем таблицу отделов с первичным ключом department\_id CREATE TABLE departments (  department\_id NUMBER(4) PRIMARY KEY, -- Идентификатор отдела  department\_name VARCHAR2(30) NOT NULL, -- Название отдела  manager\_id NUMBER(6), -- Идентификатор менеджера отдела  location\_id NUMBER(4) -- Идентификатор местоположения отдела ) TABLESPACE USERS\_TBS;  -- Создаем таблицу местоположений с первичным ключом location\_id CREATE TABLE locations (  location\_id NUMBER(4) PRIMARY KEY, -- Идентификатор местоположения  street\_address VARCHAR2(40), -- Улица местоположения  postal\_code VARCHAR2(12), -- Почтовый индекс местоположения  city VARCHAR2(30) NOT NULL, -- Город местоположения  state\_province VARCHAR2(25), -- Область местоположения  country\_id CHAR(2) NOT NULL -- Идентификатор страны местоположения ) TABLESPACE USERS\_TBS;  -- Создаем таблицу стран с первичным ключом country\_id CREATE TABLE countries (  country\_id CHAR(2) PRIMARY KEY, -- Идентификатор страны  country\_name VARCHAR2(40) NOT NULL -- Название страны ) TABLESPACE USERS\_TBS;  -- Создаем таблицу проектов с первичным ключом project\_id CREATE TABLE projects (  project\_id NUMBER(4) PRIMARY KEY, -- Идентификатор проекта  project\_name VARCHAR2(50) NOT NULL, -- Название проекта  start\_date DATE NOT NULL, -- Дата начала проекта  end\_date DATE, -- Дата окончания проекта (может быть null)  budget NUMBER(12,2), -- Бюджет проекта  department\_id NUMBER(4) NOT NULL -- Идентификатор отдела, ответственного за проект ) TABLESPACE USERS\_TBS;  -- Создаем таблицу задач с первичным ключом task\_id CREATE TABLE tasks (  task\_id NUMBER(6) PRIMARY KEY, -- Идентификатор задачи  task\_name VARCHAR2(100) NOT NULL, -- Название задачи  description VARCHAR2(500), -- Описание задачи  duration NUMBER(4) NOT NULL, -- Продолжительность задачи в часах  project\_id NUMBER(4) NOT NULL -- Идентификатор проекта, к которому относится задача ) TABLESPACE USERS\_TBS;  -- Создаем таблицу участия сотрудников в проектах с составным первичным ключом из emp\_id и project\_id CREATE TABLE participation (  emp\_id NUMBER(6) NOT NULL, -- Идентификатор сотрудника, участвующего в проекте  project\_id NUMBER(4) NOT NULL, -- Идентификатор проекта, в котором участвует сотрудник  role VARCHAR2(50), -- Роль сотрудника в проекте  hours NUMBER(4), -- Количество часов, затраченных сотрудником на проект  PRIMARY KEY (emp\_id, project\_id) -- Составной первичный ключ из emp\_id и project\_id ) TABLESPACE USERS\_TBS;   CREATE TABLE vacations (  vacation\_id NUMBER(6) PRIMARY KEY, -- Идентификатор отпуска  emp\_id NUMBER(6) NOT NULL, -- Идентификатор сотрудника  start\_date DATE NOT NULL, -- Дата начала отпуска  end\_date DATE NOT NULL, -- Дата окончания отпуска  reason VARCHAR2(500) -- Причина отпуска ) TABLESPACE USERS\_TBS; -- Создаем внешние ключи для связи между таблицами  -- Связываем таблицу employees с таблицей jobs по атрибуту job\_id ALTER TABLE employees ADD CONSTRAINT fk\_employees\_jobs FOREIGN KEY (job\_id) REFERENCES jobs (job\_id);  -- Связываем таблицу employees с самой собой по атрибуту manager\_id ALTER TABLE employees ADD CONSTRAINT fk\_employees\_managers FOREIGN KEY (manager\_id) REFERENCES employees (emp\_id);  -- Связываем таблицу employees с таблицей departments по атрибуту department\_id ALTER TABLE employees ADD CONSTRAINT fk\_employees\_departments FOREIGN KEY (department\_id) REFERENCES departments (department\_id);  -- Связываем таблицу departments с таблицей locations по атрибуту location\_id ALTER TABLE departments ADD CONSTRAINT fk\_departments\_locations FOREIGN KEY (location\_id) REFERENCES locations (location\_id);  -- Связываем таблицу locations с таблицей countries по атрибуту country\_id ALTER TABLE locations ADD CONSTRAINT fk\_locations\_countries FOREIGN KEY (country\_id) REFERENCES countries (country\_id);  -- Связываем таблицу projects с таблицей departments по атрибуту department\_id ALTER TABLE projects ADD CONSTRAINT fk\_projects\_departments FOREIGN KEY (department\_id) REFERENCES departments (department\_id);  -- Связываем таблицу tasks с таблицей projects по атрибуту project\_id ALTER TABLE tasks ADD CONSTRAINT fk\_tasks\_projects FOREIGN KEY (project\_id) REFERENCES projects (project\_id);  -- Связываем таблицу participation с таблицей employees по атрибуту emp\_id ALTER TABLE participation ADD CONSTRAINT fk\_participation\_employees FOREIGN KEY (emp\_id) REFERENCES employees (emp\_id);  -- Связываем таблицу participation с таблицей projects по атрибуту project\_id ALTER TABLE participation ADD CONSTRAINT fk\_participation\_projects FOREIGN KEY (project\_id) REFERENCES projects (project\_id);  -- Связываем таблицу vacations с таблицей employees по атрибуту emp\_id ALTER TABLE vacations ADD CONSTRAINT fk\_vacations\_emp\_id FOREIGN KEY (emp\_id) REFERENCES employees(emp\_id); |

Приложение В

Листинг создания триггеров

|  |
| --- |
| CREATE OR REPLACE TRIGGER emp\_id\_trg  BEFORE INSERT ON employees FOR EACH ROW  BEGIN  :NEW.salt := generate\_salt(:NEW.email);  SELECT emp\_id\_seq.NEXTVAL INTO :new.emp\_id FROM dual;  END;/  CREATE OR REPLACE TRIGGER department\_id\_trg  BEFORE INSERT ON departments FOR EACH ROW  BEGIN  SELECT department\_id\_seq.NEXTVAL INTO :new.department\_id FROM dual;  END;/  CREATE OR REPLACE TRIGGER location\_id\_trg  BEFORE INSERT ON locations FOR EACH ROW  BEGIN  SELECT location\_id\_seq.NEXTVAL INTO :new.location\_id FROM dual;  END;/  CREATE OR REPLACE TRIGGER project\_id\_trg  BEFORE INSERT ON projects FOR EACH ROW  BEGIN  SELECT project\_id\_seq.NEXTVAL INTO :new.project\_id FROM dual;  END;/  CREATE OR REPLACE TRIGGER task\_id\_trg  BEFORE INSERT ON tasks FOR EACH ROW  BEGIN  SELECT task\_id\_seq.NEXTVAL INTO :new.task\_id FROM dual;  END;/  CREATE OR REPLACE TRIGGER vacation\_id\_trg  BEFORE INSERT ON vacations FOR EACH ROW  BEGIN  SELECT vacation\_id\_seq.NEXTVAL INTO :new.vacation\_id FROM dual;  END;/ |

Приложение Г

Скрипт создания процедуры export\_json

|  |
| --- |
| CREATE OR REPLACE PROCEDURE export\_json AS  v\_file UTL\_FILE.FILE\_TYPE;  v\_data employees%ROWTYPE;  v\_json VARCHAR2(32767);  v\_count NUMBER := 0;  v\_total NUMBER := 0; BEGIN  -- Получаем общее количество записей  SELECT COUNT(\*) INTO v\_total FROM employees;   -- Открываем файл для записи  v\_file := UTL\_FILE.FOPEN('MY\_DIRECTORY', 'employees.json', 'w', 32767);   -- Записываем открывающую квадратную скобку  UTL\_FILE.PUT\_LINE(v\_file, '[');   -- Начинаем цикл по всем записям в таблице employees  FOR v\_data IN (SELECT \* FROM employees) LOOP  -- Преобразуем каждую запись в JSON  v\_json := JSON\_OBJECT(  'emp\_id' VALUE v\_data.emp\_id,  'first\_name' VALUE v\_data.first\_name,  'last\_name' VALUE v\_data.last\_name,  'email' VALUE v\_data.email,  'phone\_number' VALUE v\_data.phone\_number,  'job\_id' VALUE v\_data.job\_id,  'salary' VALUE v\_data.salary,  'commission\_pct' VALUE v\_data.commission\_pct,  'manager\_id' VALUE v\_data.manager\_id,  'department\_id' VALUE v\_data.department\_id,  'password\_hash' VALUE v\_data.PASSWORD\_HASH,  'isauthentic' value v\_data.ISAUTHENTIC  );   -- Увеличиваем счетчик  v\_count := v\_count + 1;   -- Записываем JSON в файл  IF v\_count < v\_total THEN  UTL\_FILE.PUT\_LINE(v\_file, v\_json || ',');  ELSE  UTL\_FILE.PUT\_LINE(v\_file, v\_json);  END IF;   -- Записываем JSON в файл -- UTL\_FILE.PUT\_LINE(v\_file, v\_json);  END LOOP;    -- Записываем закрывающую квадратную скобку  UTL\_FILE.PUT\_LINE(v\_file, ']');  -- Закрываем файл  UTL\_FILE.FCLOSE(v\_file); EXCEPTION  WHEN OTHERS THEN  -- Если произошла ошибка, закрываем файл  IF UTL\_FILE.IS\_OPEN(v\_file) THEN  UTL\_FILE.FCLOSE(v\_file);  END IF;  RAISE; END export\_json; / |

Приложение Д

Скрипт создания процедуры import\_json

|  |
| --- |
| CREATE OR REPLACE PROCEDURE import\_json IS  v\_file UTL\_FILE.FILE\_TYPE;  v\_data CLOB;  v\_line VARCHAR2(32767);  v\_json JSON\_ARRAY\_T;  v\_json\_obj JSON\_OBJECT\_T;  v\_emp\_rec employees%ROWTYPE; BEGIN  -- Открываем файл для чтения  v\_file := UTL\_FILE.FOPEN('MY\_DIRECTORY', 'employees.json', 'r');   -- Читаем данные из файла  BEGIN  LOOP  UTL\_FILE.GET\_LINE(v\_file, v\_line);  v\_data := v\_data || v\_line;  END LOOP;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  NULL; -- Игнорируем исключение, так как это означает, что мы достигли конца файла  END;   -- Закрываем файл  UTL\_FILE.FCLOSE(v\_file);   -- Парсим JSON  v\_json := JSON\_ARRAY\_T.parse(v\_data);   -- Извлекаем данные из JSON и вставляем их в таблицу  FOR i IN 0 .. v\_json.get\_size() - 1 LOOP  v\_json\_obj := JSON\_OBJECT\_T(v\_json.get(i));  v\_emp\_rec.emp\_id := v\_json\_obj.get\_number('emp\_id');  v\_emp\_rec.first\_name := v\_json\_obj.get\_string('first\_name');  v\_emp\_rec.last\_name := v\_json\_obj.get\_string('last\_name');  v\_emp\_rec.email := v\_json\_obj.get\_string('email');  v\_emp\_rec.phone\_number := v\_json\_obj.get\_string('phone\_number');  v\_emp\_rec.job\_id := v\_json\_obj.get\_string('job\_id');  v\_emp\_rec.salary := v\_json\_obj.get\_number('salary');  v\_emp\_rec.commission\_pct := v\_json\_obj.get\_number('commission\_pct');  v\_emp\_rec.manager\_id := v\_json\_obj.get\_number('manager\_id');  v\_emp\_rec.department\_id := v\_json\_obj.get\_number('department\_id');  v\_emp\_rec.password\_hash := v\_json\_obj.get\_string('password\_hash');  v\_emp\_rec.ISAUTHENTIC := v\_json\_obj.get\_string('isauthentic');   INSERT INTO employees VALUES v\_emp\_rec;  END LOOP;   COMMIT; EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE; END import\_json; / |