МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

Web-приложение «Система управления сетью пиццерий»

Выполнил студент Пузиков Алексей Алексеевич

(Ф.И.О.)

Руководитель работы ассист. Нистюк Ольга Александровна (учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2024

Содержание

[Введение 3](#_Toc165296317)

[1 Постановка задачи 4](#_Toc165296318)

[1.1 Анализ прототипов 4](#_Toc165296319)

[1.1.1 Интернет-ресурс «Додо пицца» 4](#_Toc165296320)

[1.1.2 Интернет-ресурс «Пицца Лисицца» 4](#_Toc165296321)

[1.2 Постановка задачи 5](#_Toc165296322)

[2 Проектирование web-приложения 6](#_Toc165296323)

[2.1 Архитектура приложения 6](#_Toc165296324)

[2.2 Проектирование структуры базы данных 7](#_Toc165296325)

[3 Разработка web-приложения 11](#_Toc165296326)

[3.1 Разработка серверной части приложения 11](#_Toc165296327)

[3.2 Разработка клиентской части приложения 14](#_Toc165296328)

[4 Тестирование web-приложения 17](#_Toc165296329)

[5 Руководство пользователя 20](#_Toc165296330)

[Заключение 29](#_Toc165296331)

[Список используемых источников 30](#_Toc165296332)

[ПРИЛОЖЕНИЕ А 31](#_Toc165296333)

[ПРИЛОЖЕНИЕ Б 32](#_Toc165296334)

[ПРИЛОЖЕНИЕ В 33](#_Toc165296335)

[ПРИЛОЖЕНИЕ Г 34](#_Toc165296336)

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. В этом разделе будут приведены интернет-ресурсы, а также веб-приложения существующих издательский платформ.

* + 1. Интернет-ресурс «Хабр»

[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/).

Интерфейс интернет-ресурса «Хабр» представлен на рисунке 1.1.

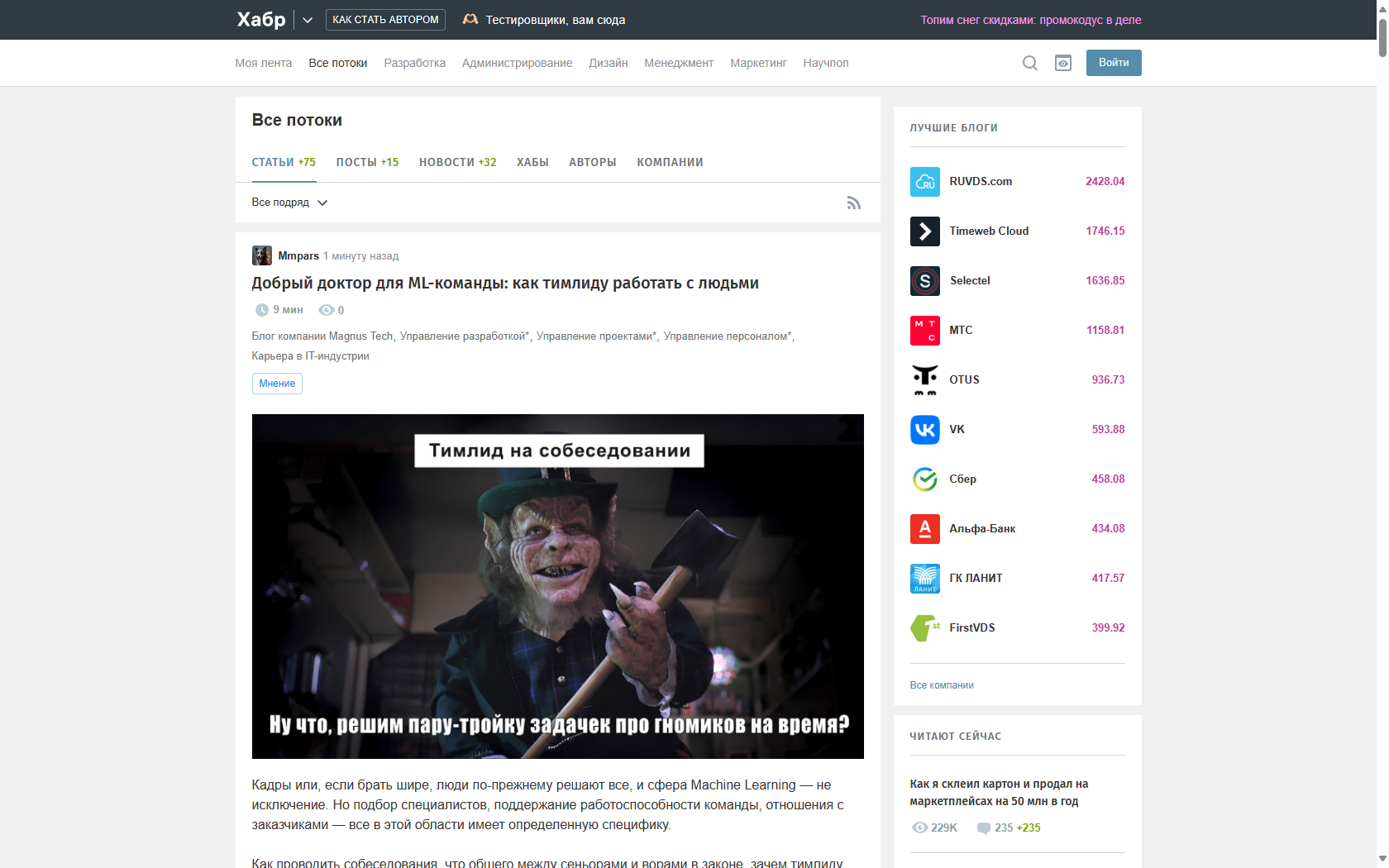


Рисунок 1.1 – Интерфейс интернет-ресурса «Хабр»

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/).
  + 1. Интернет-ресурс «Teletype»

[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 интернет-ресурса «Teletype» представлен на рисунке 1.2.

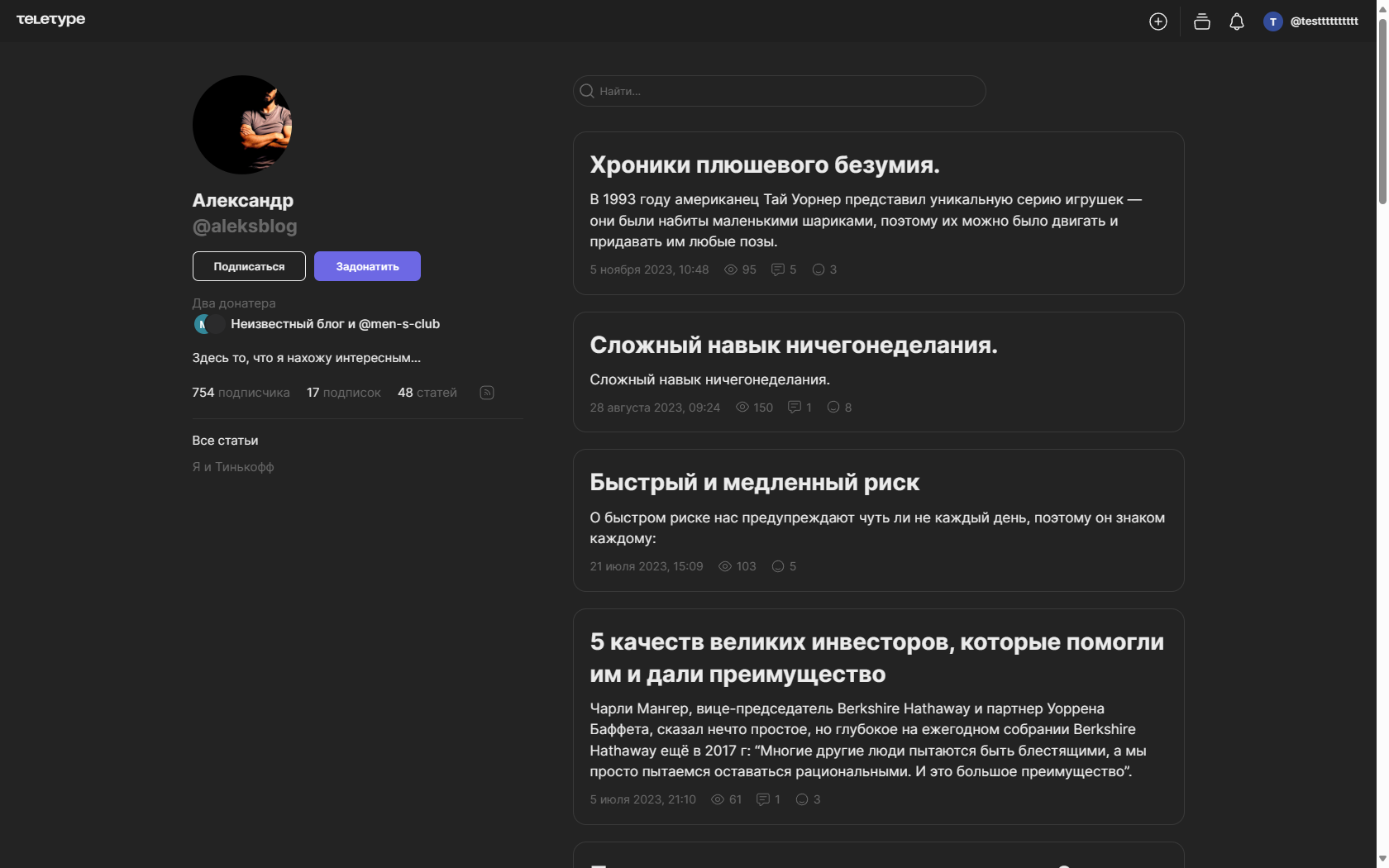


Рисунок 1.2 – Интерфейс интернет-ресурса «Teletype»

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.
  + 1. Интернет ресурс «Hashtap»

Сайт [Hashtap 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 интернет-ресурса «Hashtap» представлен на рисунке 1.3.

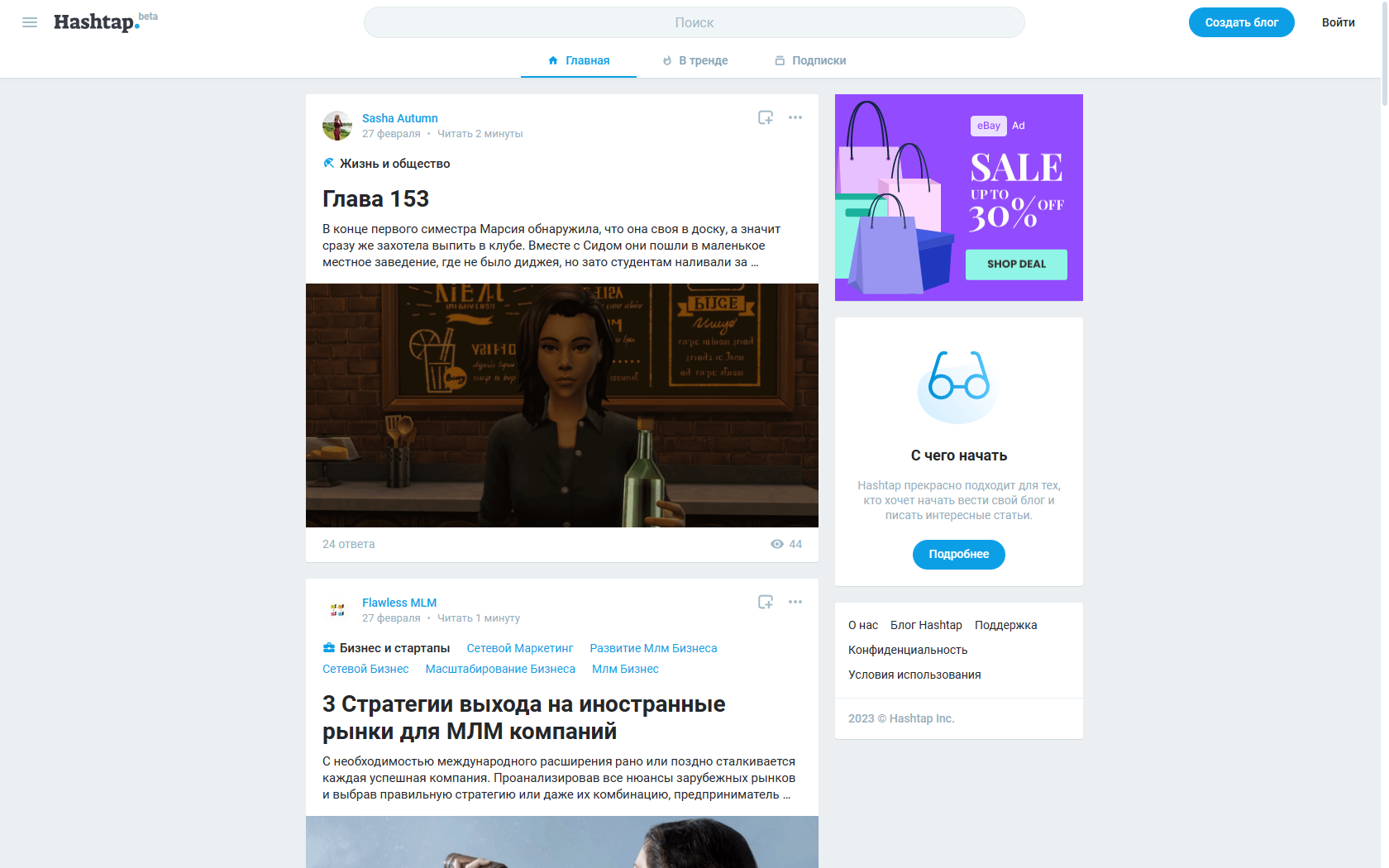


Рисунок 1.3 – Интерфейс интернет-ресурса «Hashtap»

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.
  + 1. Интернет ресурс «Dev.to»

[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 интернет-ресурса «Dev.to» представлен на рисунке 1.4.

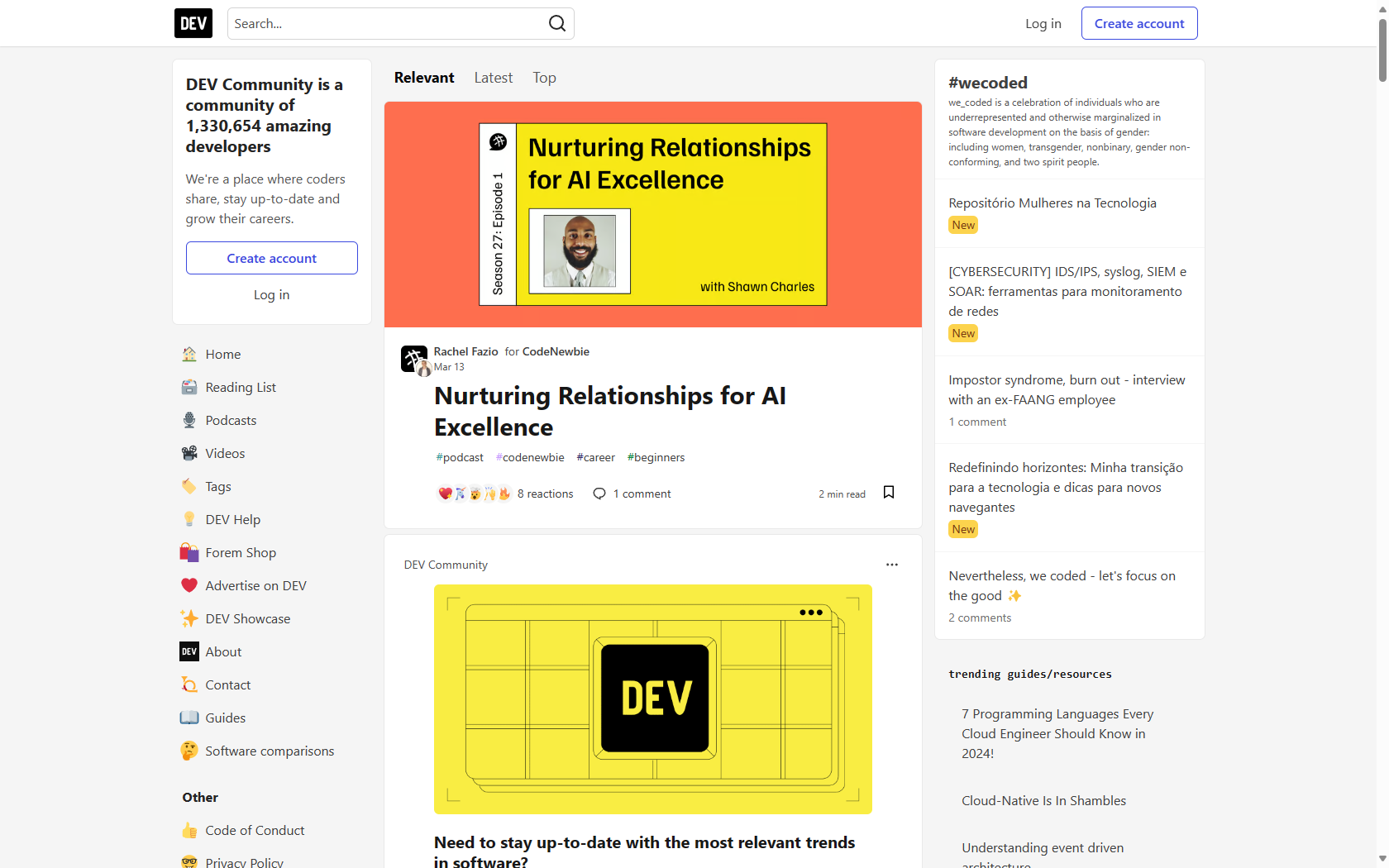


Рисунок 1.2 – Интерфейс интернет-ресурса «Dev.to»

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).
  1. Постановка задачи

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.

Обзор аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и сформулировать список требований, предъявляемых к разрабатываемому в данном курсовом проекте программному средству. Программное средство должно реализовывать следующие функции:

* обеспечивать возможность регистрации и авторизации;
* отправлять уведомления о новом контенте, который может быть интересен пользователю;
* иметь интуитивно понятный и удобный пользовательский интерфейс;
* предоставлять возможность пользователям публиковать свой контент в виде текста или изображений;
* предоставлять возможность пользователям оставлять комментарии к постам;
* предоставлять пользователю возможность кастомизировать свой профиль;
* предоставлять пользователю возможность редактировать и удалять собственные посты;
* предоставлять персонализированный контент, основанный на интересах и взаимодействиях пользователя;
* обеспечивать защиту всех пользовательских данных.

Диаграмма вариантов использования представлена в приложении А.

В данной главе были сформулированы основные требования к приложению на основе преимуществ и недостатков некоторых аналогов.

* 1. Средства разработки

При разработке приложения будут использованы:

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

Использование данных технологий имеет несколько преимуществ при разработке веб-приложения для публикации контента.

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 CSS-фреймворка Bootstrap позволяет быстро и легко 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.

2 Проектирование web-приложения

2.1 Архитектура приложения

Большое внимание уделяется качеству организации архитектуры приложения, в связи с разработкой схожей системы, было принято решение об использовании микросервисной архитектуры, 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 компонентов и изменение существующих. Диаграмма размещения представлена на рисунке 2.1.

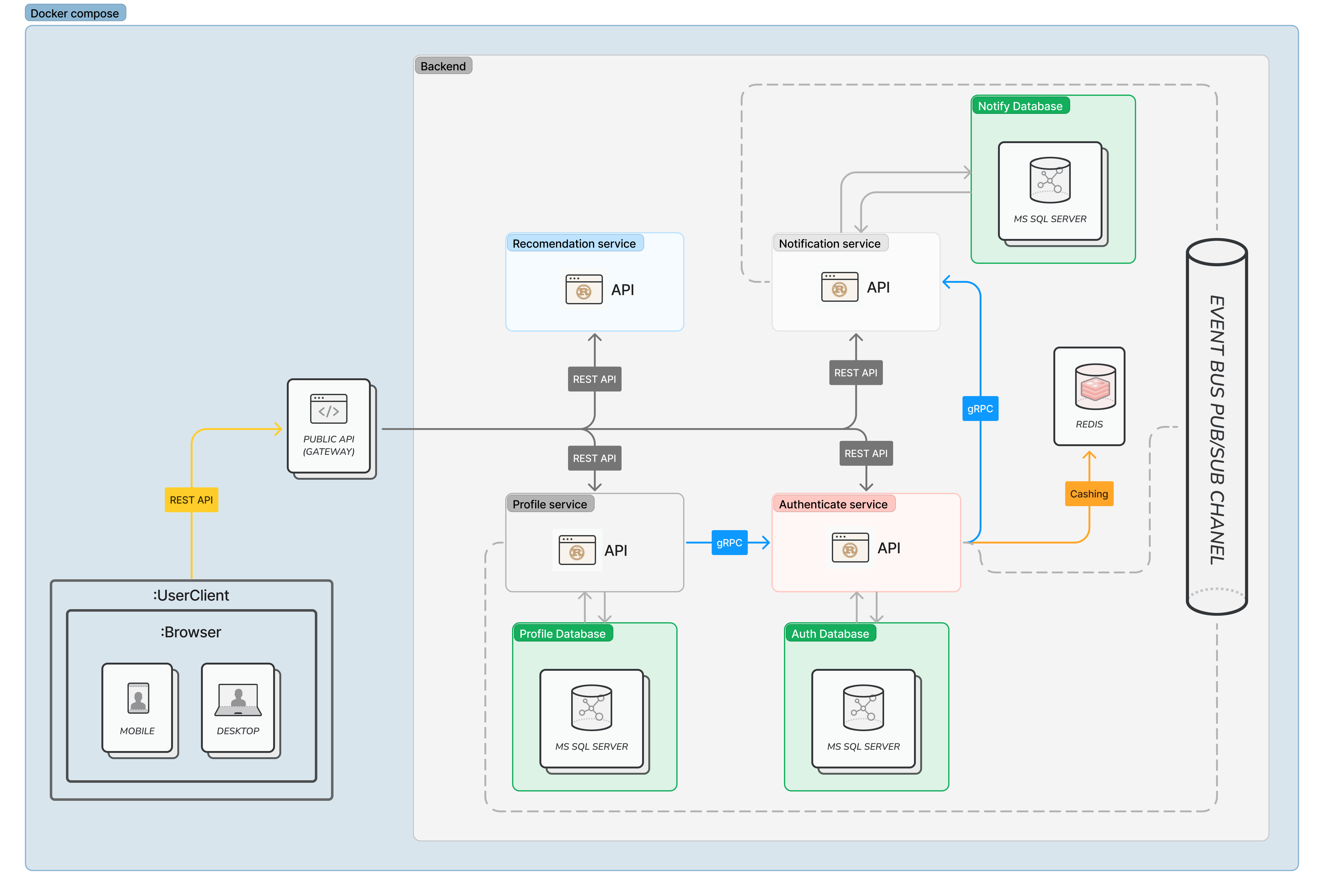


Рисунок 2.1 – Диаграмма размещения

Серверная часть веб-приложения реализована с использованием фреймворка Koa.js, который применяет архитектуру MVC (Model-View-Controller) для организации компонентов. Эта архитектура разделяет приложение на три основных уровня: модели (models), представления (views) и контроллеры (controllers).

Контроллеры (controllers) являются центральным звеном в архитектуре MVC. Они отвечают за обработку входящих запросов от клиентов и формирование соответствующих ответов. Контроллеры определяют конечные точки (endpoints) API и управляют логикой обработки запросов. Они принимают запросы от клиентов, вызывают соответствующие методы моделей для получения или изменения данных и передают данные в представления для отображения.

Модели (models) представляют собой структуры данных и логику работы с этими данными. Они отвечают за доступ к базе данных, выполнение операций CRUD (Create, Read, Update, Delete) и валидацию данных. Модели обычно содержат методы для работы с данными, такие как сохранение, поиск и удаление записей, а также бизнес-логику, связанную с этими данными.

Представления (views) отвечают за отображение данных пользователю. Они представляют собой HTML-шаблоны, которые могут быть заполнены данными из контроллеров или моделей. Представления определяют, как данные будут отображаться пользователю в браузере или другом клиентском приложении. Однако в данном проекте роль представлений была делегирована библиотеке React.js.

Архитектура MVC в Express.js помогает разделить логику приложения на отдельные компоненты, улучшая читаемость и поддерживаемость кода. Эта структура помогает упростить разработку, отладку и поддержку приложения.

Клиентская часть проекта использует React.js, который представляет собой библиотеку JavaScript для разработки динамичных пользовательских интерфейсов. React.js позволяет создавать переиспользуемые компоненты, которые эффективно обновляются и реагируют на изменения данных. Это достигается за счет использования виртуального DOM (Virtual DOM), который позволяет React обновлять только необходимые элементы интерфейса без перерисовки всего дерева компонентов. Такой подход улучшает производительность взаимодействия с динамическими интерфейсами и значительно упрощает процесс разработки.

Взаимодействие между клиентской и серверной частями осуществляется посредством Web API. Сервер предоставляет API, определяющий доступные конечные точки и форматы данных для обмена информацией с клиентским приложением. Клиентское приложение, построенное на React.js, отправляет HTTP-запросы на сервер, используя соответствующие конечные точки, и принимает ответы в формате JSON. Это обеспечивает надежное взаимодействие между клиентом и сервером, позволяя передавать данные и обновления между ними и обеспечивая функциональность и интерактивность всего приложения.

Также в приложении используется протокол WebSockets. Использование WebSockets в web-приложении позволяет создать интерактивный и динамичный пользовательский опыт. WebSockets обеспечивают постоянное соединение между клиентом и сервером, что позволяет мгновенно передавать данные в обоих направлениях без необходимости постоянного обновления страницы.

Диаграмма развертывания представлена в приложении Б.

2.2 Проектирование структуры базы данных

В данном разделе представлена концептуальная и логическая модель базы данных для проекта. На этапе разработки концептуальной модели было проведено детальное изучение бизнес-требований и основных процессов необходимых веб-приложению издательской платформы.

Так как архитектура приложения подразумевает что у каждого сервиса будет собственная база данных, то все сущности рассредоточены между ними.

База данных сервиса аутентификации содержит единственную таблицу Users. Эта таблица содержит информацию о пользователях, которые зарегистрировались в системе. Её структура представлена в таблице 2.1.

Таблица 2.1 – Структура таблицы USERS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| ID | INT | PRIMARY KEY IDENTITY |
| GOOGLE\_ID | NVARCHAR(255) |  |
| GITHUB\_ID | NVARCHAR(255) |  |
| USERNAME | NVARCHAR(255) | NOT NULL |
| EMAIL | NVARCHAR(255) | NOT NULL |
| PASSWORD | NVARCHAR(255) |  |
| SALT | NVARCHAR(255) |  |
| IS\_EMAIL\_VERIFIED | BIT | DEFAULT 0 |
| VERIFICATION\_TOKEN | NVARCHAR(255) |  |
| RESET\_PASSWORD\_TOKEN | NVARCHAR(255) |  |
| RESET\_PASSWORD\_EXPIRES | DATETIME |  |

База данных сервиса уведомлений содержит единственную таблицу Notifications. Эта таблица содержит информацию о уведомлениях, отправленных пользователям. Её структура представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы NOTIFICATIONS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| ID | INT | PRIMARY KEY IDENTITY |
| AUTHOR | NVARCHAR(255) | NOT NULL |
| USER\_ID | INT | NOT NULL |
| CONTENT | TEXT | NOT NULL |
| IMAGE | NVARCHAR(255) | DEFAULT 'default\_profile.png' |
| PUBLISHED\_AT | DATETIME | DEFAULT CURRENT\_TIMESTAMP |

База данных сервиса профилей содержит следующие таблицы:

* Users – содержит профильную информацию о пользователях. Её структура представлена в таблице 2.3;
* Posts – содержит информацию о публикациях пользователей. Её структура представлена в таблице 2.4;
* Subscriptions – содержит информацию о подписках пользователей. Её структура представлена в таблице 2.5;
* Comments – содержит информацию о комментариях к публикациям. Её структура представлена в таблице 2.6;

Таблица 2.3 – Структура таблицы USERS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| ID | INT | PRIMARY KEY IDENTITY |
| USERNAME | NVARCHAR(255) |  |
| EMAIL | NVARCHAR(255) |  |
| FIRSTNAME | NVARCHAR(255) |  |
| LASTNAME | NVARCHAR(255) |  |
| PHONE\_NUMBER | NVARCHAR(255) |  |
| PROFILE\_PICTURE | NVARCHAR(255) | DEFAULT 'default\_profile.png' |
| SUBSCRIBERS\_COUNT | INT | DEFAULT 0 |
| SUBSCRIPTIONS\_COUNT | INT | DEFAULT 0 |
| POSTS\_COUNT | INT | DEFAULT 0 |

Таблица 2.4 – Структура таблицы POSTS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| ID | INT | PRIMARY KEY IDENTITY |
| AUTHOR | NVARCHAR(255) | NOT NULL |
| USER\_ID | INT | NOT NULL |
| CONTENT | TEXT | NOT NULL |
| IMAGE | NVARCHAR(255) | DEFAULT 'default\_profile.png' |
| PUBLISHED\_AT | DATETIME | DEFAULT CURRENT\_TIMESTAMP |

Таблица 2.5 – Структура таблицы SUBSCRIPTIONS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| ID | INT | PRIMARY KEY IDENTITY |
| AUTHOR | NVARCHAR(255) | NOT NULL |
| USER\_ID | INT | NOT NULL |
| CONTENT | TEXT | NOT NULL |
| IMAGE | NVARCHAR(255) | DEFAULT 'default\_profile.png' |
| PUBLISHED\_AT | DATETIME | DEFAULT CURRENT\_TIMESTAMP |

Таблица 2.6 – Структура таблицы COMMENTS

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| ID | INT | PRIMARY KEY IDENTITY |
| AUTHOR | NVARCHAR(255) | NOT NULL |
| USER\_ID | INT | NOT NULL |
| CONTENT | TEXT | NOT NULL |
| IMAGE | NVARCHAR(255) | DEFAULT 'default\_profile.png' |
| PUBLISHED\_AT | DATETIME | DEFAULT CURRENT\_TIMESTAMP |

В данной главе была спроектирована архитектура приложения, включая серверную и клиентскую часть, а также базы данных.

3 Разработка web-приложения

3.1 Обобщенная структура приложения

Структура приложения представлена на рисунке 3.1.

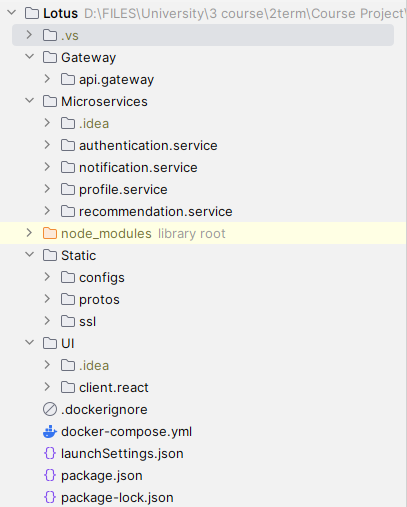


Рисунок 3.1 – Структура приложения

В папке Static хранятся статические данные, общие для всего приложения. В дочерней папке configs хранятся файлы конфигурации json, в которых содержатся данные, используемые в тех или иных файлах для конфигурации различных утилит приложения, таких как например nodemailer для отправки сообщений на почту. В дочерней папке protos содержатся proto файлы – схемы данных Protobuf для использования gRPC. В папке ssl находятся сертификаты для настройки https серверов.

В папке Microservices находятся собственно сами сервисы приложения, реализованные на koa.

В папке UI находится клиент React, который подробно описан в главе 3.3.

3.2 Разработка серверной части приложения

Серверная часть приложения обеспечивает обработку клиентских запросов, выполняет запросы к базе данных, обеспечивает аутентификацию и авторизацию, а также поддерживает websocket сервер.

Структура серверного приложения на примере сервиса профилей представлена на рисунке 3.1.

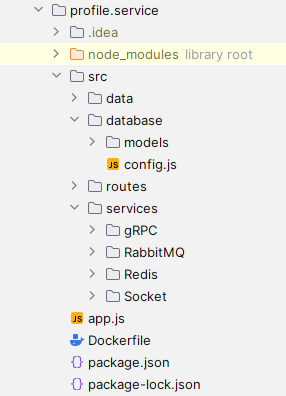


Рисунок 3.1 – Структура серверного приложения на примере сервиса профилей

Входной точкой каждого сервера является файл app.js. В нем происходит инициализация koa-сервера, websocket сервера (при наличии), а также установка сертификатов для обеспечения защищенного соединения с сервером.

В директории database находятся модели и файл конфигурации для настройки ORM-фреймворка sequelize. Модели описывают сущности базы данных и представляют интерфейс для взаимодействия с ней.

В директории routes находятся маршрутизаторы для каждой сущности. Пример маршрутизатора для сущности «User» представлен в листинге 3.2.

|  |
| --- |
| router.get('/api/user/subscriptions', koaJwt({ secret: SECRET\_KEY }), getUserSubscriptions); router.get('/api/user/subscribers', koaJwt({ secret: SECRET\_KEY }), getUserSubscribers); router.post('/api/user/subscribe', koaJwt({ secret: SECRET\_KEY }), subscribeUser); router.post('/api/user/unsubscribe', koaJwt({ secret: SECRET\_KEY }), unsubscribeUser); router.post('/api/user/support', koaJwt({ secret: SECRET\_KEY }), support); |

Листинг 3.2 – Маршрутизатор для сущности «User»

С каждым маршрутом сопоставляется определенный метод, который непосредственно осуществляет обработку запроса. Пример метода support, который обрабатывает запрос пользователя в техническую поддержкупредставлен в листинге 3.3.

|  |
| --- |
| async function support(ctx) {  const user\_id = ctx.state.user.user\_id;  const username = ctx.state.user.username;  const email = ctx.state.user.email;  const { problem\_message } = ctx.request.body;   try {  const newSupportMessage = await SUPPORT.create({  USER\_ID: user\_id,  PROBLEM\_MESSAGE: problem\_message  });   const data = {  EMAIL: email,  USERNAME: username,  PROBLEM\_MESSAGE: problem\_message  }  sendToQueue("SupportEmailNotificationQueue", data);   ctx.status = 201;  ctx.body = { message: 'Support message saved successfully', supportMessageId: newSupportMessage.ID };  } catch (error) {  console.error('Error saving support message:', error);  ctx.status = 500;  ctx.body = { message: 'Internal Server Error' };  } } |

Листинг 3.3 – Метод support

Кроме конечных точек приложения, маршрутизаторы также используют промежуточную обработку запроса за счет использования middleware.

В приложении используется middleware koaJwt, который имеют стандартную реализацию в библиотеке «koa-jwt», и используется для для аутентификации и авторизации. Этот middleware необходим для того, чтобы защитить некоторые маршруты от не аутентифицированного доступа.

3.3 Разработка клиентской части приложения

Клиентская часть приложения предоставляет интерфейс для взаимодействия с серверным приложением. Клиентская часть представляет собой набор React-компонентов, и является одновременно http и websocket-клиентом.

Структура клиентского приложения представлена на рисунке 3.7.

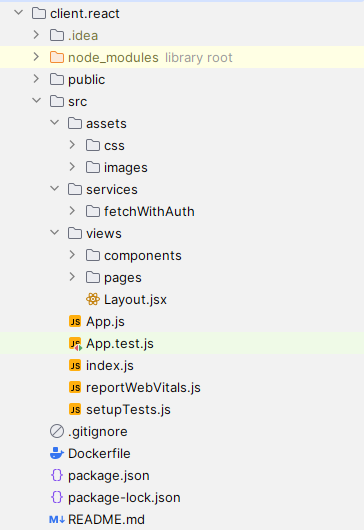


Рисунок 3.7 – Структура клиентского приложения

Директория src содержит весь исходный код приложения.

Директория pages содержит все React-страницы, к которым пользователь может получить доступ.

Директория components содержит все React-компоненты, повторно используемые в приложении.

директория services содержит функции с дополнительным функционалом.

Директория assets содержит статические данные, используемые в приложение, такие как иконки, стили CSS и т.п.

В основном файле проекта App.js прописана маршрутизация приложения, реализованная при помощи библиотеки «react-router». Содержимое файла App.js представлены на листинге 3.8.

|  |
| --- |
| function App() {  useEffect(() => AOS.init , []);   return (  <>  <Routes>  <Route path="/" element={<Layout/>}>  <Route index element={<HomePage/>} />  <Route path="home" element={<HomePage/>} />  <Route path="news" element={<NewsPage/>} />  <Route path="news/:topic" element={<NewsPage/>} />  <Route path="news/:id" element={<SelectedNews/>} />  <Route path="support" element={<SupportPage/>} />  <Route path="profile/:username" element={<ProfilePage/>} />  <Route path="profile/:username/edit" element={<ProfileEditPage/>} />  <Route path="profile/:username/change-password" element={<ChangePasswordPage/>} />  <Route path="/:username/:postid/comments" element={<CommentsPage/>} />  <Route path="/about" element={<AboutPage/>} />  <Route path="/people" element={<PeoplePage/>} />  <Route path="/saved" element={<SavedPage/>} />  <Route path="/subscriptions" element={<SubscriptionsPage/>} />  <Route path="/notifications" element={<NotificationsPage/>} />  </Route>  <Route path="login" element={<AuthenticationPage />} />  <Route path="register" element={<RegistrationPage />} />  <Route path="\*" element={<NotFoundPage />} />  </Routes>  </>  ); }   export default App; |

Листинг 3.8 – Содержимое файла App.js

Также, были использованы сторонние библиотеки и фреймворки для разработки вилузального интерфейса приложения.

Bootstrap – это библиотека, которая предоставляет готовые компоненты и классы для быстрой и удобной стилизации веб-страниц. Мы использовали Bootstrap для создания адаптивного и красивого дизайна нашего приложения. Bootstrap позволяет легко создавать сетки, стилизовать формы, кнопки, карточки и многое другое. Это значительно ускоряет процесс разработки и обеспечивает единообразие стилей.

AOS (Animate On Scroll) – это небольшая библиотека, которая позволяет анимировать элементы, когда они появляются в области просмотра. Мы использовали AOS для добавления динамических анимаций в наше приложение. Это делает интерфейс более живым и интересным для пользователя.

4 Тестирование web-приложения

В данном разделе описано тестирование приложения с использованием клиентской части.

Сначала, было проведено тестирование регистрации. На рисунке 4.1 представлена ошибка, возникающая если пользователь пытается зарегистрировать email или номер телефона, которые уже зарегистрированы.

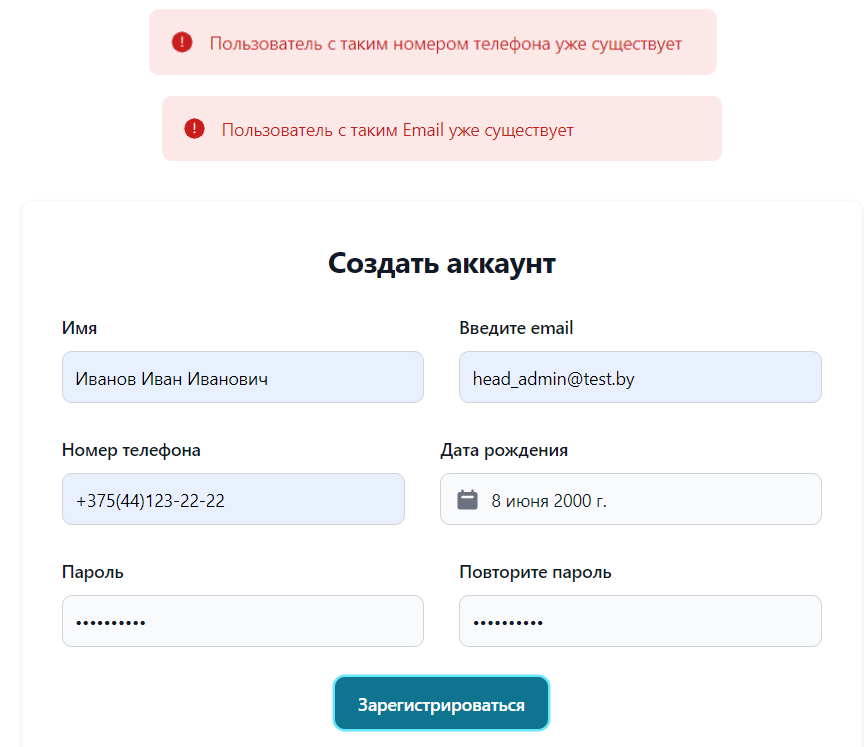


Рисунок 4.1 – Обработка некорректных данных при регистрации

При авторизации в приложении также могут возникать ошибки. Пример обработки ошибки, возникающей если пользователя не существует в системе, представлен на рисунке 4.2.

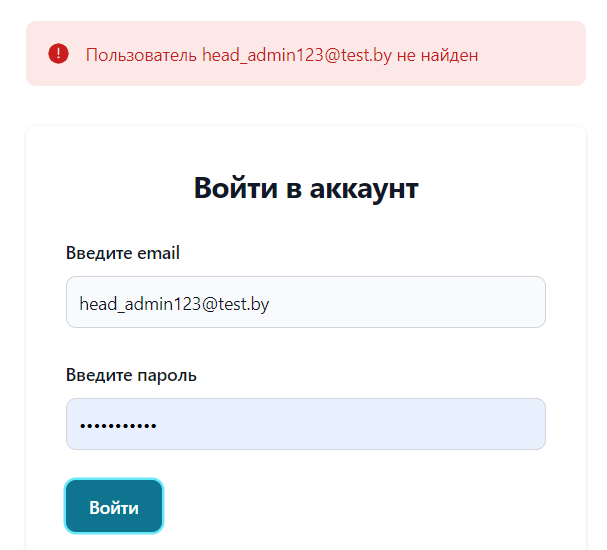


Рисунок 4.2 – Вход под несуществующим пользователем

Пример обработки ошибки, возникающей если пользователь ввел некорректный пароль, представлен на рисунке 4.3.

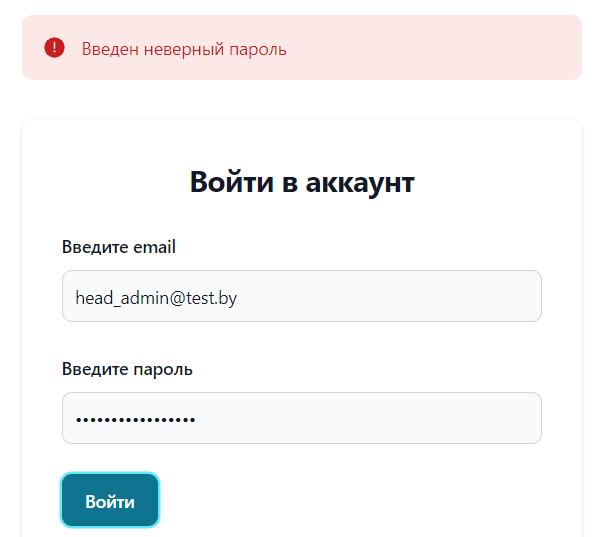


Рисунок 4.3 – Вход с неправильным паролем

Ошибки также обрабатываются при добавлении или изменении данных различных сущностей. Пример обработки ошибок числовых данных при добавлении товара представлен на рисунке 4.4.

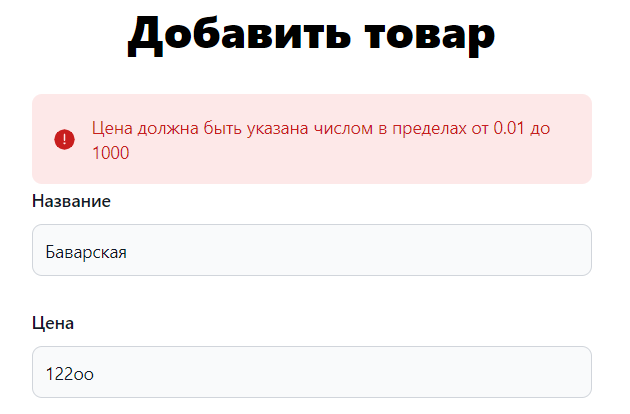


Рисунок 4.4 – Вход числовых данных неверного формата

Пример обработки ошибок при отправке пустых данных представлен на рисунке 4.5.

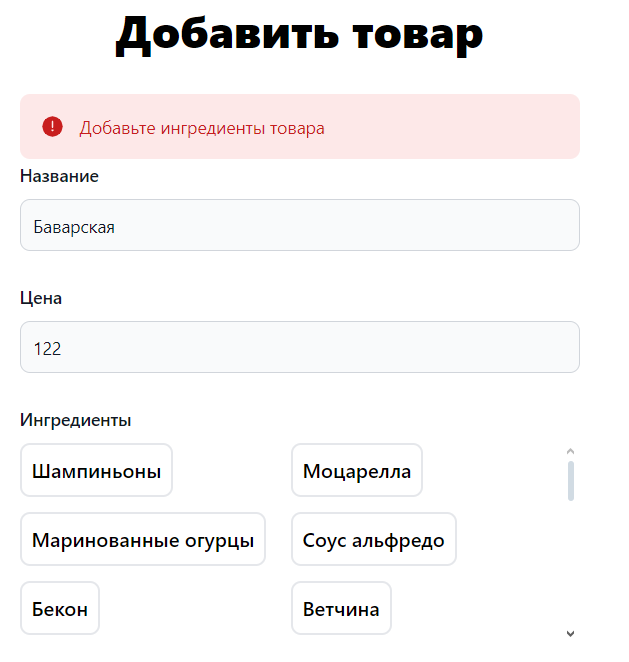


Рисунок 4.5 – Ошибка отправки пустых данных

Приложение также имеет ограничение на количество товаров в одном заказе. Обработка ошибок на превышение ограничения на количество товаров в корзине представлена на рисунке 4.6.

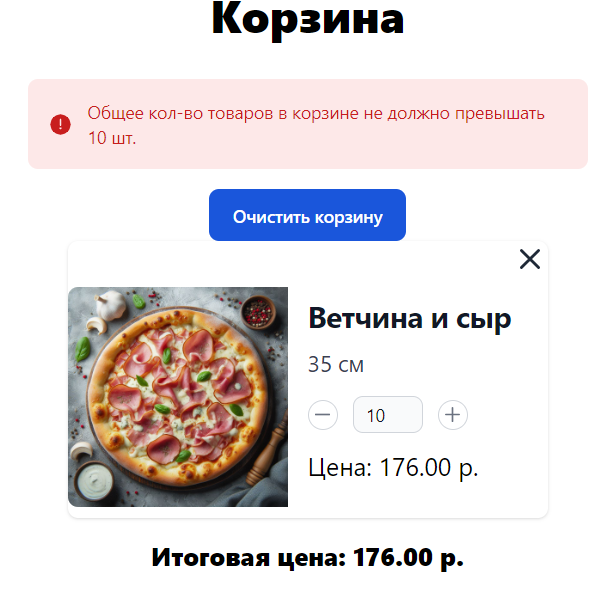


Рисунок 4.6 – Ошибка превышения количества товаров в корзине

Таким образом было проведено тестирование приложения. Были продемонстрированы типичные ситуации, вызывающие ошибки и реакция на них. Результаты тестирования позволили выявить и исправить множество ошибок и проблем, связанных с функциональностью и производительностью приложения.

5 Руководство пользователя

При запуске приложения пользователь попадает на главную страницу, интерфейс которой представлен на рисунке 5.1.

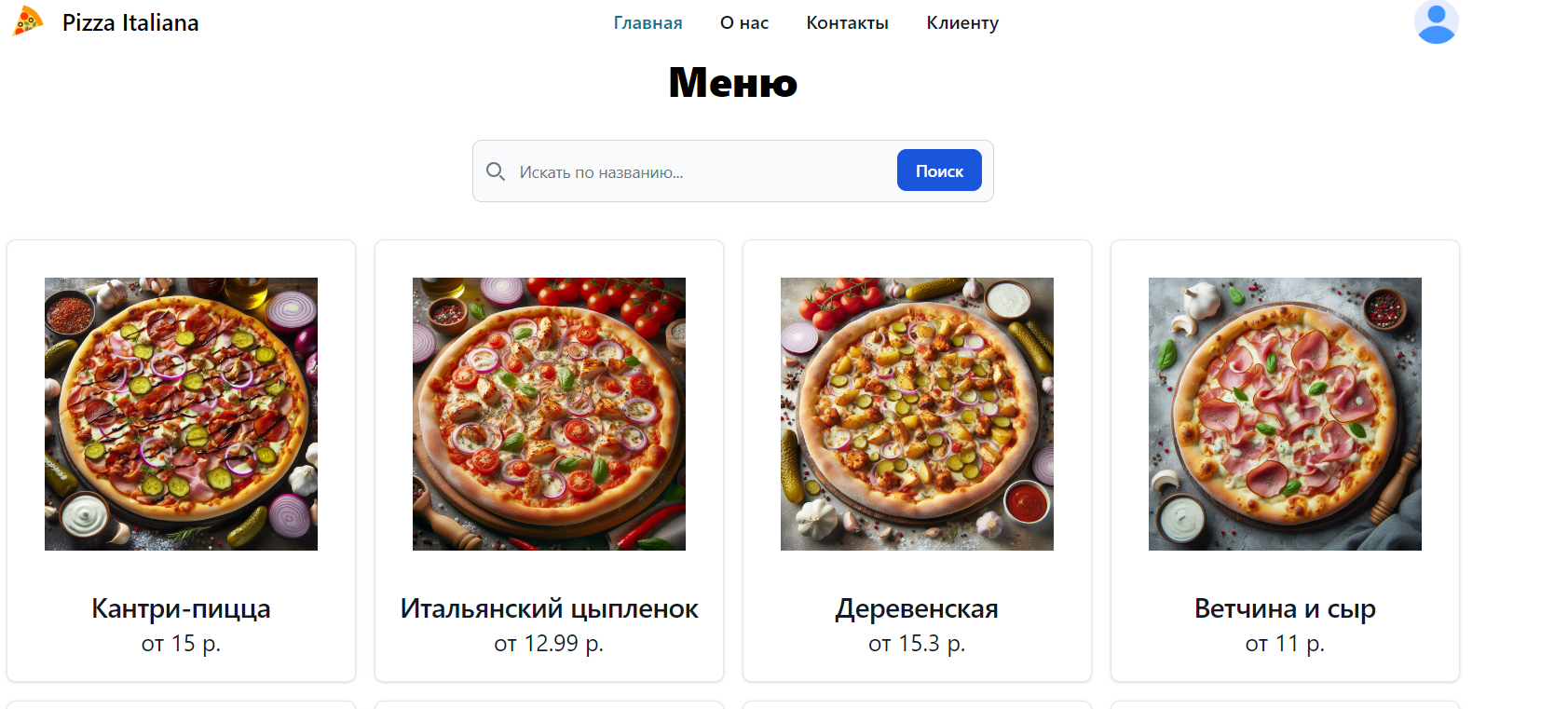


Рисунок 5.1 – Главная страница приложения

Он может просматривать весь каталог блюд, а также просмотреть детали одного блюда. Далее он может зайти на страницу «Контакты» и просмотреть данные о ресторанах. Интерфейс страницы представлен на рисунке 5.2.

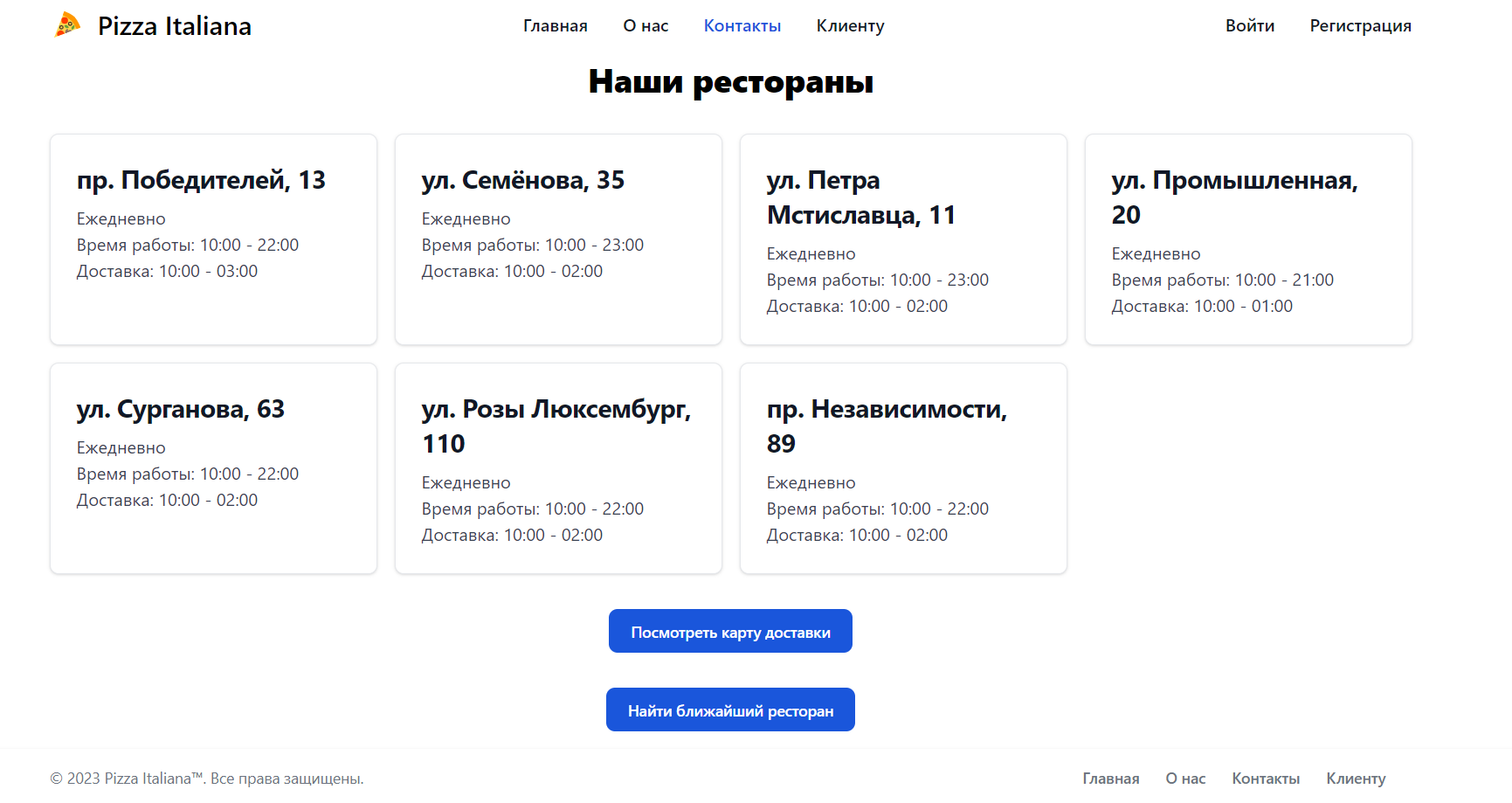


Рисунок 5.2 – Страница «Контакты»

Нажав на кнопку «Посмотреть карту доставки», пользователь может увидеть визуализацию зон доставки каждого ресторана. Изображение карты представлено на рисунке 5.3.

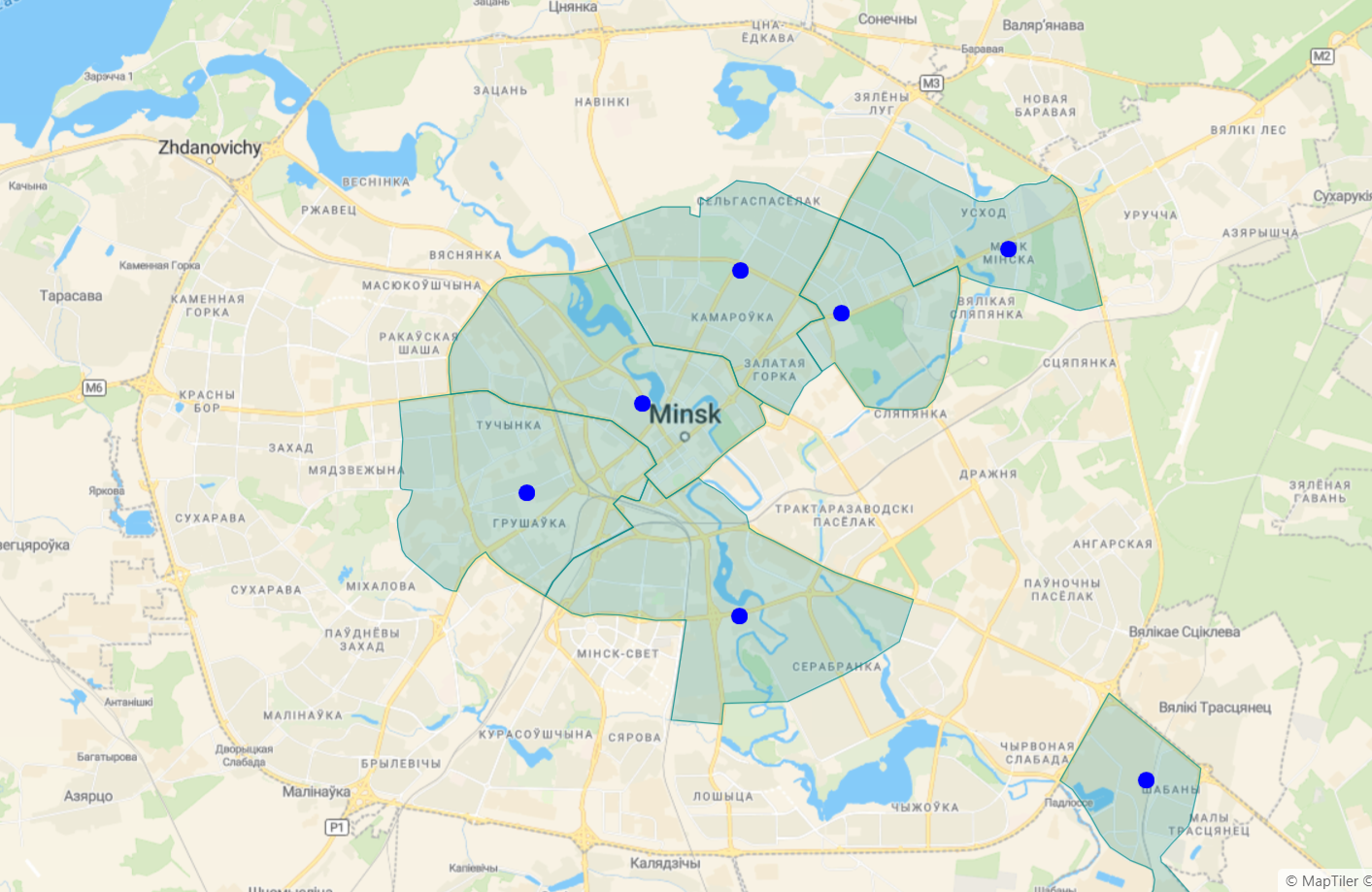


Рисунок 5.3 – Карта доставки

Нажав на кнопку «Найти ближайший ресторан», пользователь должен разрешить приложению получать местоположение, и далее система вернет пользователю адрес ближайшего к нему ресторана. Пример представлен на рисунке 5.4.

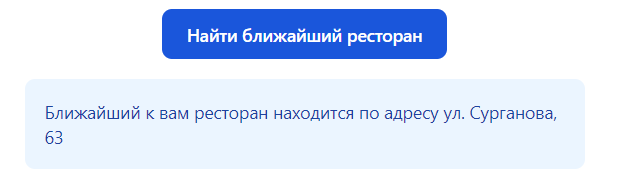


Рисунок 5.4 – Работа функции нахождения ближайшего ресторана

Стоит заметить, что точность определения местоположения пользователя зависит от встроенного в пользовательское устройство GPS.

Далее пользователь может войти в систему, нажав кнопку «Войти». Если же у него еще нет аккаунта, он может зарегистрироваться в системе нажав на кнопку «Регистрация». Формы входа и регистрации представлены на рисунке 5.5.

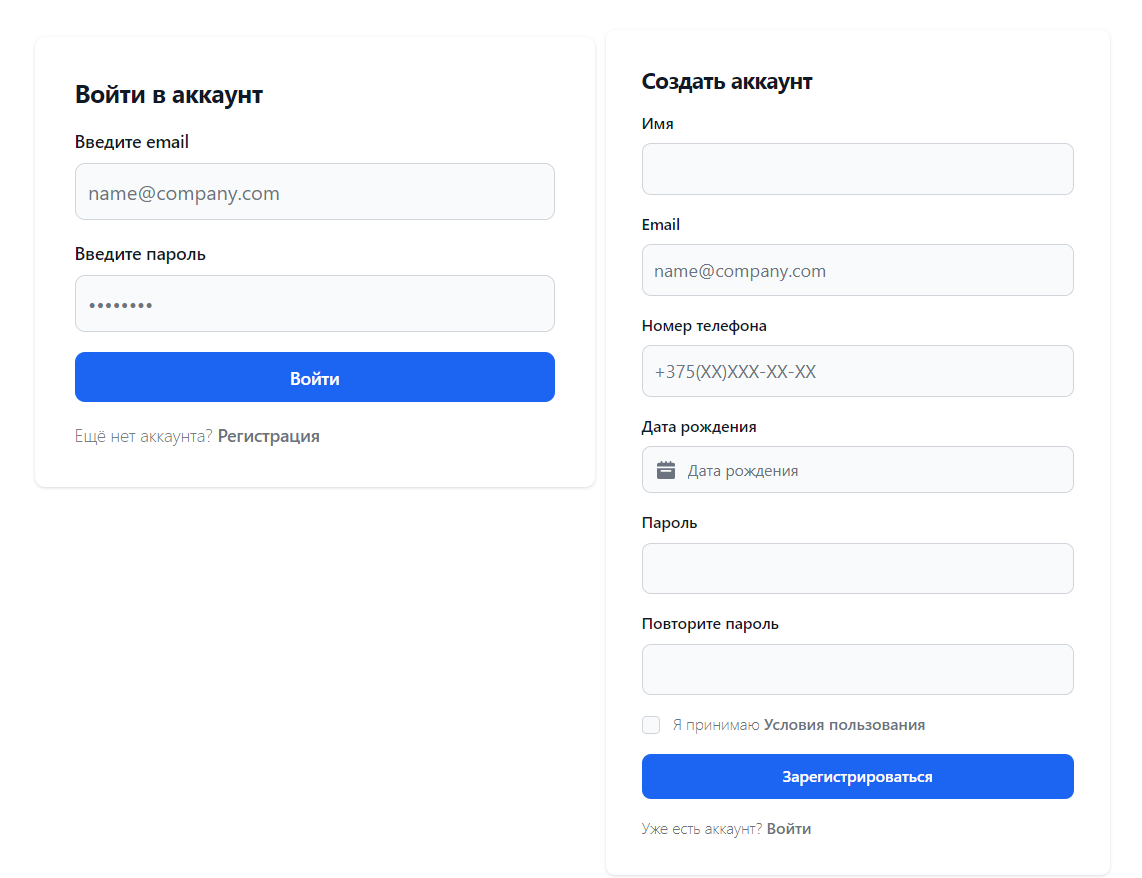


Рисунок 5.5 – Формы входа и регистрации

Если вход был выполнен успешно, функционал приложения может расширяться по-разному в зависимости от роли пользователя. По умолчанию пользователь регистрируется под ролью «пользователь». Теперь он может добавлять товары в корзину и делать заказ.

Для того, чтобы добавить товар в корзину, нужно на главной странице выбрать любое блюдо и далее выбрать его размер. Также пользователь может убрать некоторые ингредиенты блюда. Затем нужно нажать на кнопку «Добавить товар в корзину». Пример представлен на рисунке 5.6.

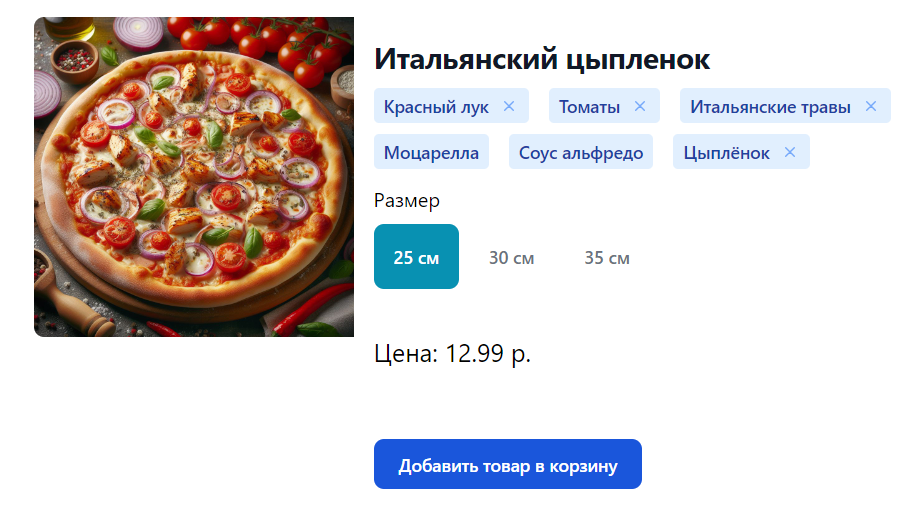


Рисунок 5.6 – Добавление товара в корзину

Далее, нажав на значок профиля в правом верхнем углу, пользователь может перейти в раздел «Корзина». Интерфейс раздела представлен на рисунке 5.7.

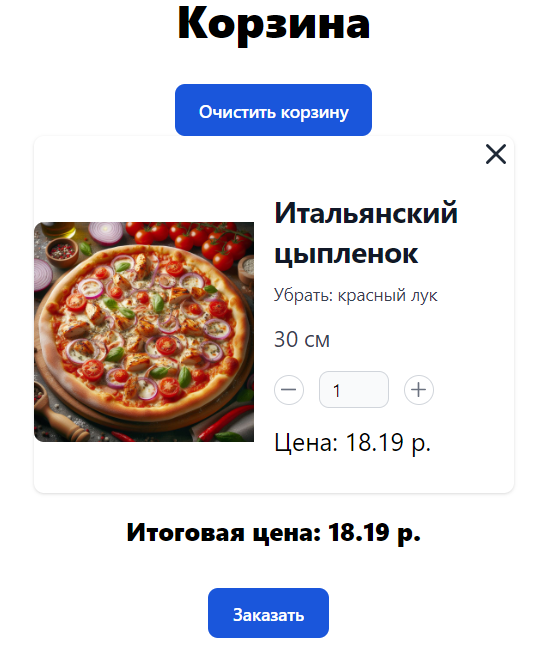


Рисунок 5.7 – Интерфейс раздела «Корзина»

В данном разделе, пользователь может изменить количество товара нажав на кнопки «-» или «+». Если количество товара достигает нуля, то товар удаляется из корзины. Максимальное количество товаров в корзине – 10 шт. При изменении количества товара также меняется цена в соответствии с количеством. Нажав на значок «крестик», пользователь может удалить товар из корзины. Кнопка «Очистить корзину» удаляет все товары из корзины, независимо от их количества. Далее пользователь может совершить заказ, нажав на кнопку «Заказать» и заполнив поля формы. Пример заполнения представлен на рисунке 5.8.

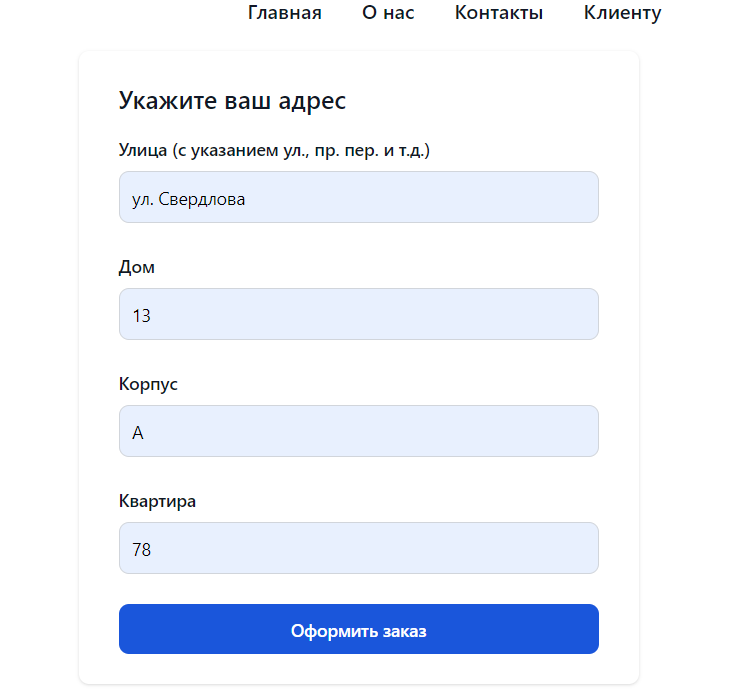


Рисунок 5.8 – Совершение заказа

Пользователь будет перенаправлен на страницу просмотра заказов, где он сможет увидеть свой заказ. Для удобства доставки заказы распределяются по ресторанам. Соответственно совершенный заказ отправится в ресторан, в зону доставки которого попадает адрес, указанный пользователем. Если зайти от пользователя, имеющего роль «администратор ресторана», то можно увидеть все заказы, поступившие в данный ресторан. Пример представлен на рисунке 5.9.

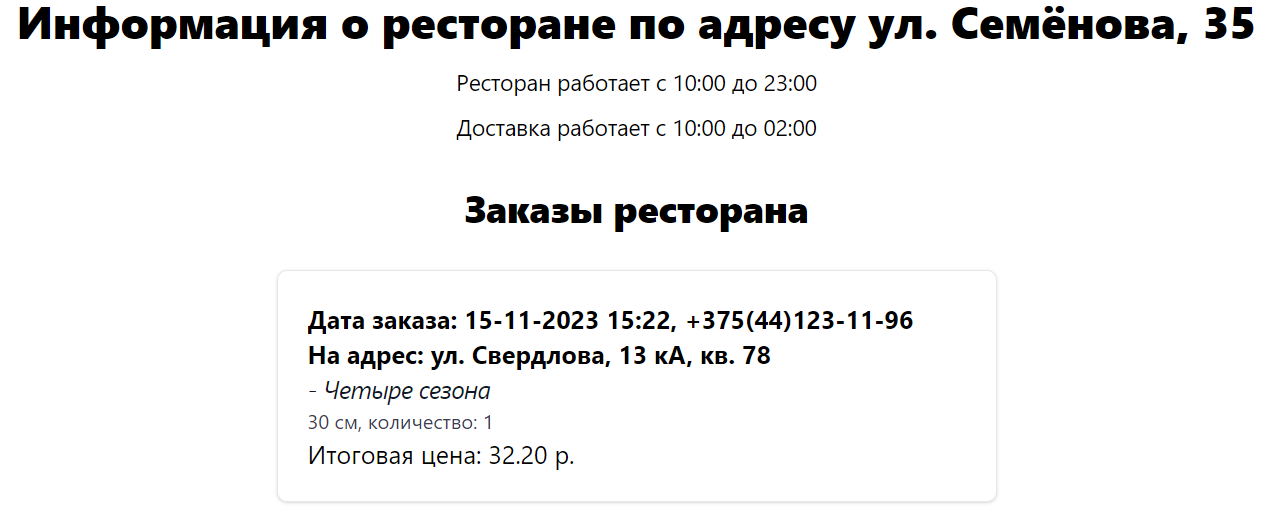


Рисунок 5.9 – Заказы в ресторане

Также заказы распределяются по курьерам. При получении заказа, курьер не может принять сразу еще один заказ, поэтому при отображении его данных на панели администратора ресторана будет показана соответствующая пометка. Интерфейс панели представлен на рисунке 5.10.

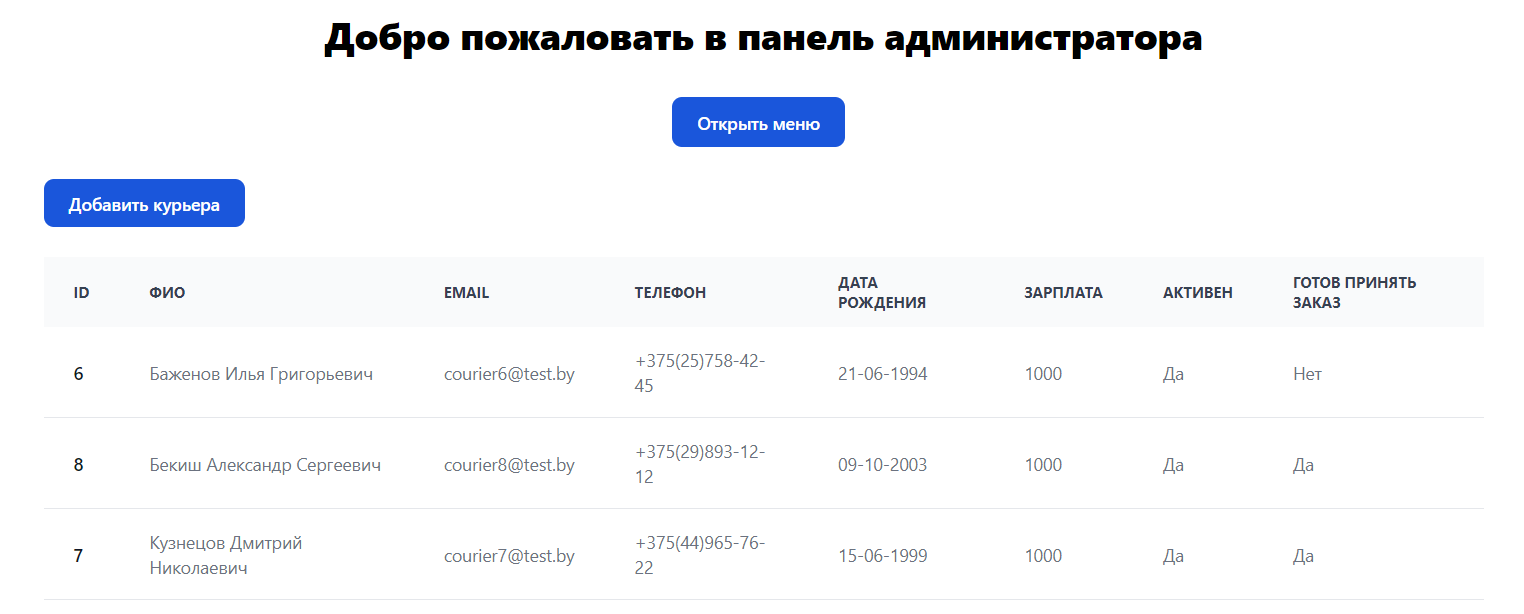


Рисунок 5.10 – Информация о курьерах на панели администратора

Кроме того, с панели администратора можно управлять данными курьеров. Например, добавлять нового курьера, изменять данные существующих, а также удалять курьеров. Примеры форм добавления и изменения данных курьеров представлены на рисунке 5.11.

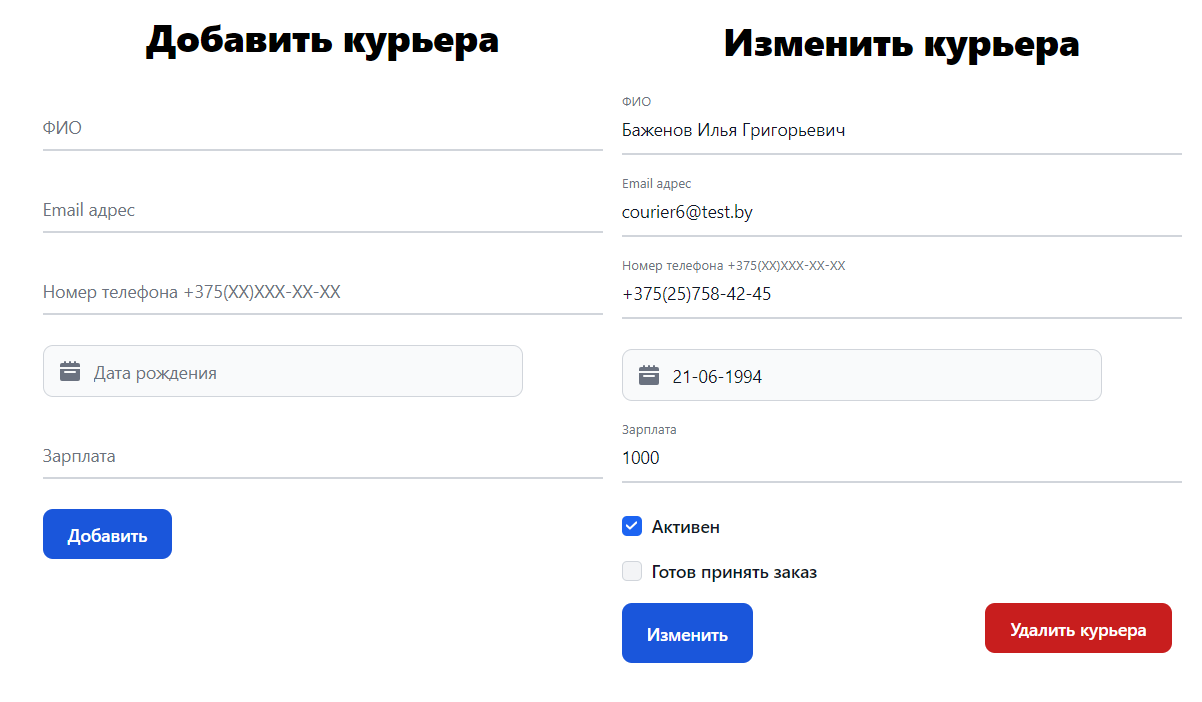


Рисунок 5.11 – Формы добавления и изменения данных курьеров

Панель администратора, имеющего роль «главный администратор» отличается от панели пользователя «администратор ресторана». Например, главный администратор имеет возможность просматривать и создавать администраторов ресторанов. Пример интерфейса панели главного администратора представлен на рисунке 5.12.

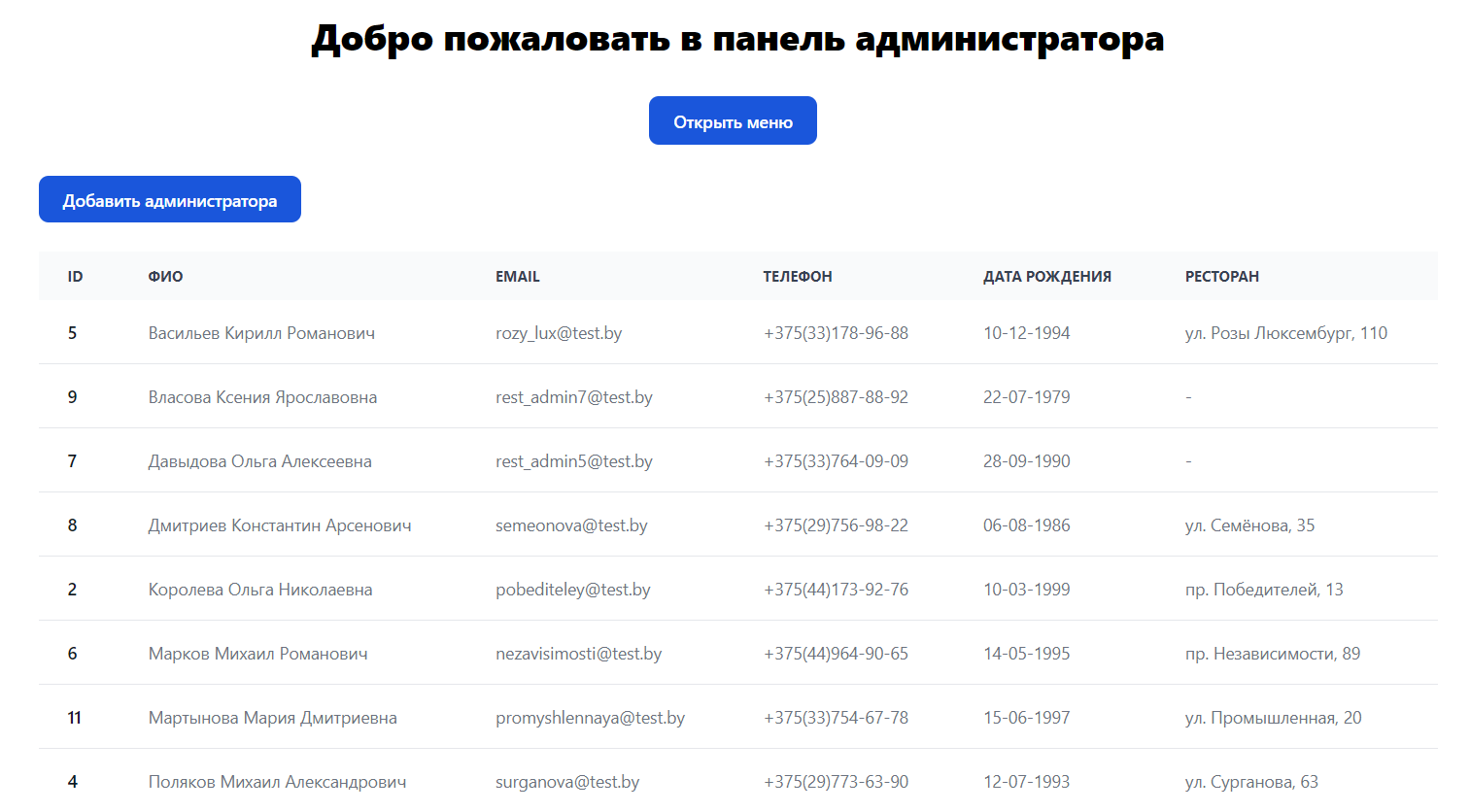


Рисунок 5.12 – Панель главного администратора

Главный администратор может зарегистрировать администратора ресторана в приложении. Для этого он должен указать личные данные нового пользователя, а также новый пароль. С этим паролем администратор ресторана сможет в дальнейшем осуществить вход в приложение. Пример формы добавление администратора представлен на рисунке 5.13.

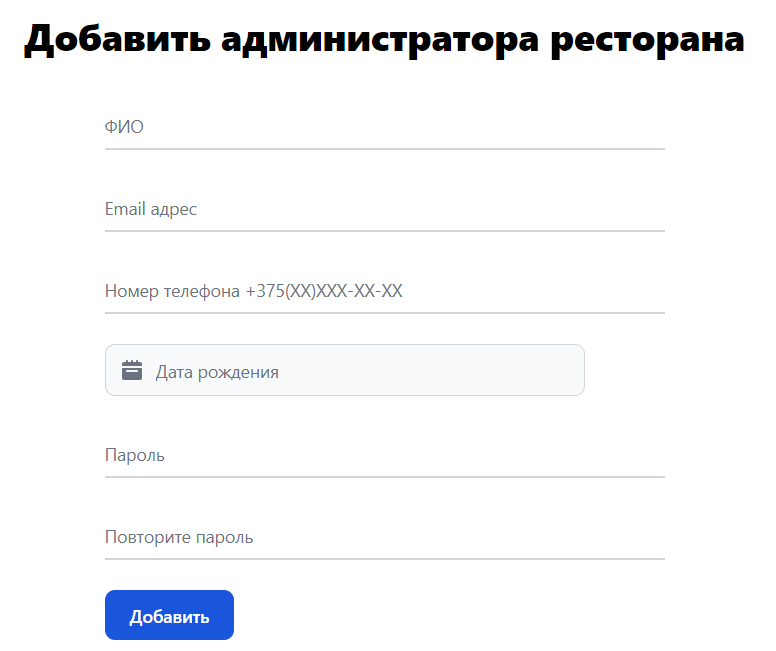


Рисунок 5.13 – Форма регистрации администратора ресторана

Далее главный администратор имеет возможность просматривать, добавлять, изменять и удалять рестораны. Пример интерфейса для работы с ресторанами представлен на рисунке 5.14.

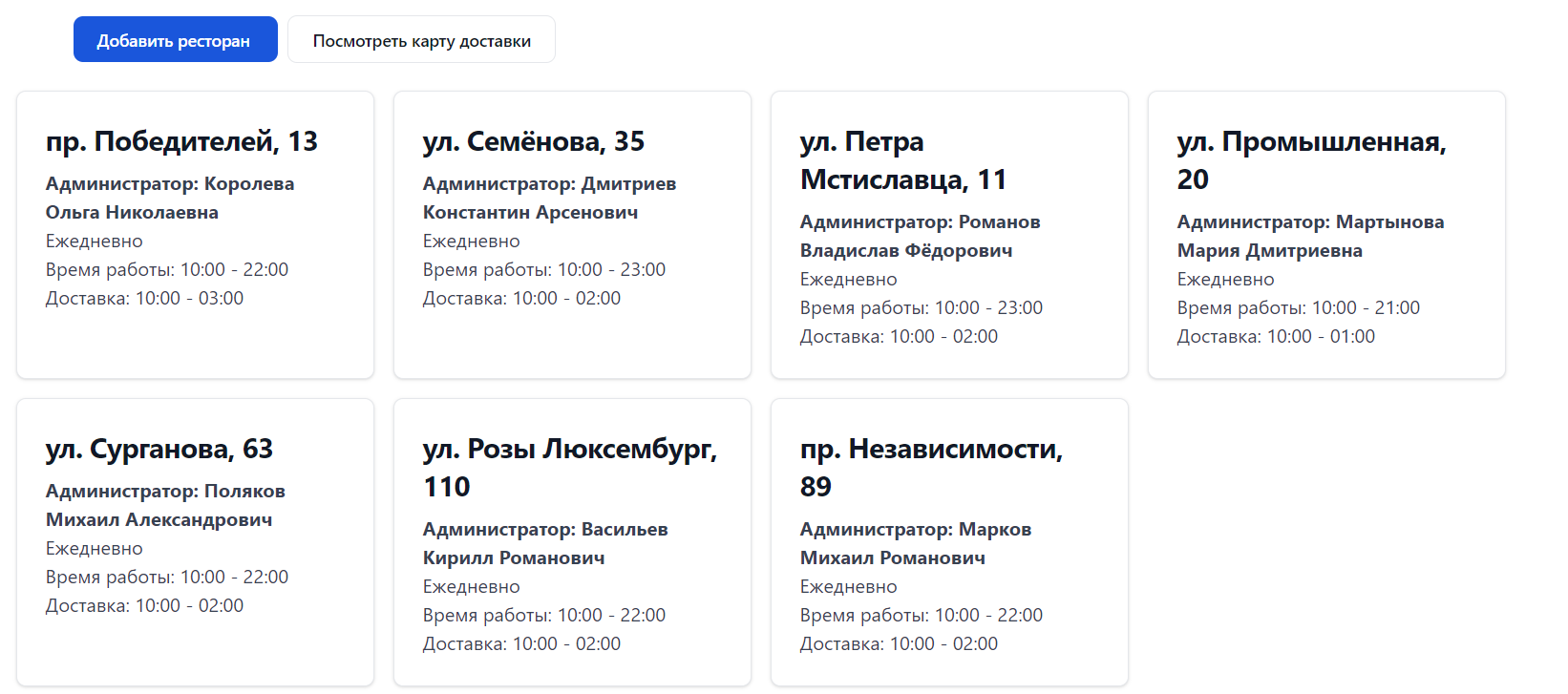


Рисунок 5.14 – Интерфейс для работы с ресторанами

При создании или изменении ресторана, главный администратор может изменить зону доставки ресторана. Для этого ему надо загрузить соответствующий файл в формате GEOJSON. Также он может выбрать администратора ресторана из предложенного списка. В списке будут находится только те администраторы, на которых еще не зарегистрирован ни один ресторан.

Кроме того, главный администратор также может управлять товарами в меню. Ему также доступны функции добавления, изменения и удаления блюд. Интерфейс для работы с меню представлен на рисунке 5.15.

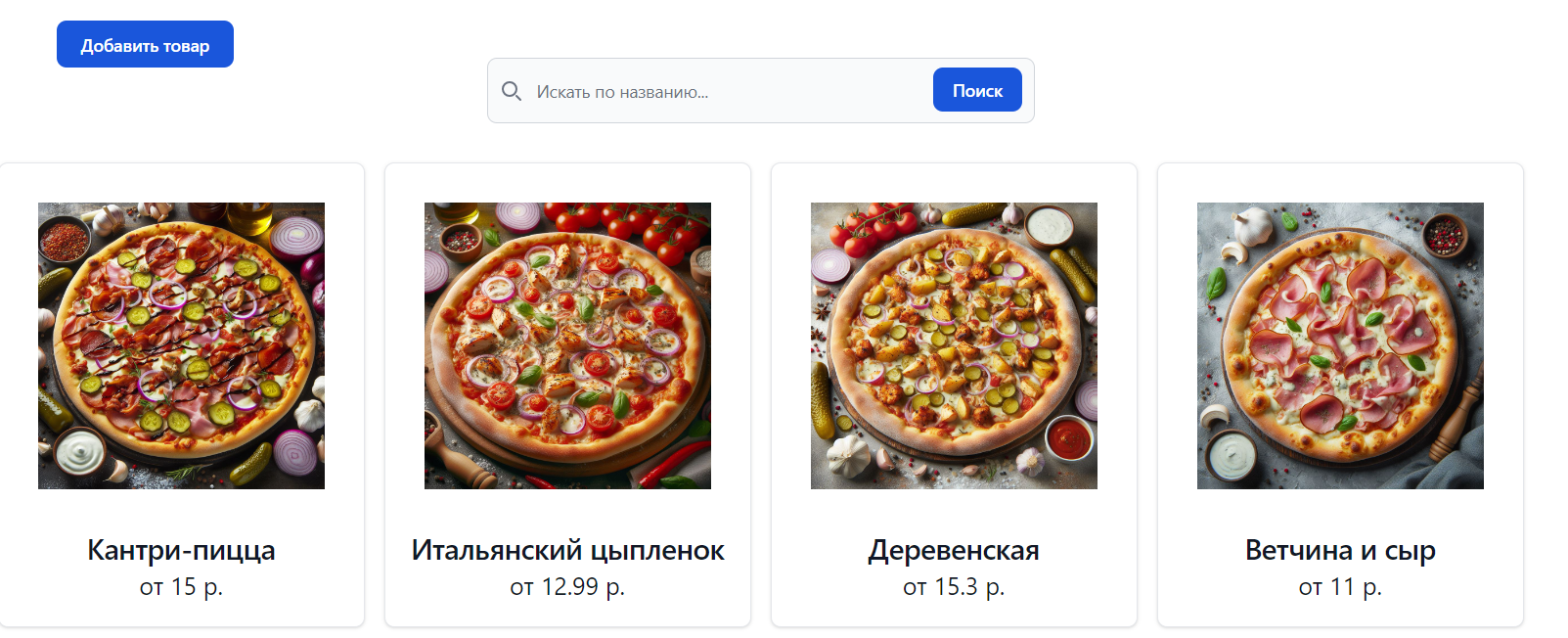


Рисунок 5.15 – Интерфейс для работы с меню

При добавлении или изменении товара администратор может загрузить изображение товара на сервер. Формы для добавления и изменения блюд представлены на рисунке 5.16.

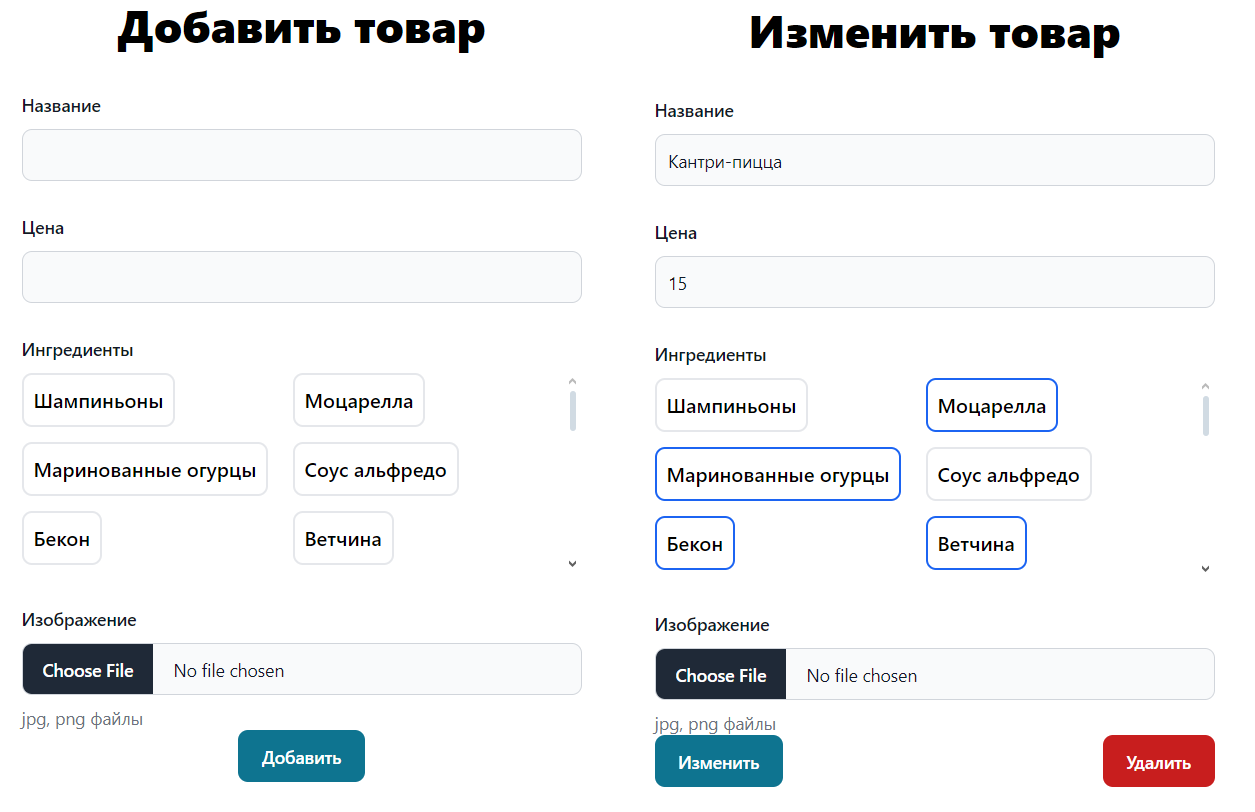


Рисунок 5.16 – Формы добавления/изменения товара

Помимо изменения самих блюд, главный администратор имеет возможность добавлять, изменять и удалять ингредиенты блюд. Интерфейс для работы с ингредиентами представлен на рисунке 5.17.

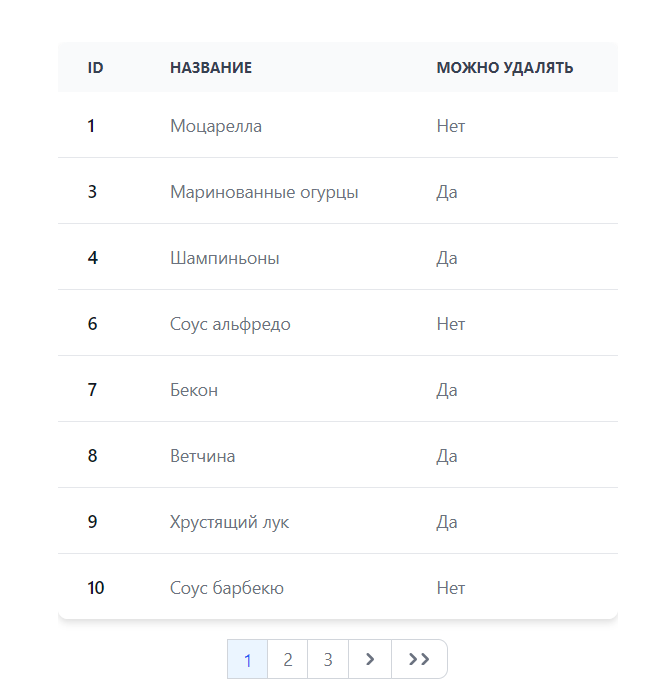


Рисунок 5.17 – Интерфейс для работы с ингредиентами блюд

При добавлении и изменении ингредиента администратор должен указать название ингредиента, а также указать, может ли пользовать убирать данный ингредиент при оформлении заказа. Это необходимо для того, чтобы покупатель не мог убрать самые базовые ингредиенты, без которых приготовление блюда не будет возможным. Пример формы для изменения ингредиента представлен на рисунке 5.18.

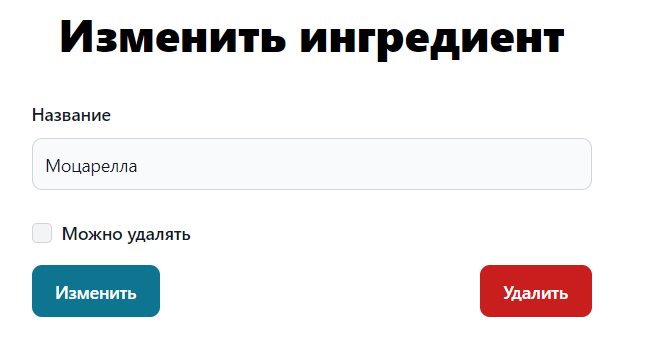


Рисунок 5.18 – Форма изменения ингредиента

Таким образом, в данной главе было разработано руководство по использованию, которое облегчает пользователю освоение функционала приложения.

Заключение

В ходе выполнения курсового проекта было разработано веб-приложение «Издательская платформа», сочетающее в себе функционал персонализированных рекомендаций и возможности издательской платформы. Проект успешно реализует поставленные задачи, предоставляя пользователям интуитивно понятный интерфейс для публикации контента, взаимодействия с другими пользователями и получения персонализированных рекомендаций.

* Реализация архитектуры: Архитектура приложения, основанная на микросервисах, обеспечивает гибкость и масштабируемость системы.
* Функциональность микросервисов: Каждый микросервис, включая авторизацию, уведомления, контент, комментарии, профили, рекомендации и безопасность, был реализован согласно требованиям задания.
* Клиентская часть и API Gateway: Клиентская часть на React и API Gateway для маршрутизации запросов были успешно разработаны и интегрированы с микросервисами.
* Тестирование и деплой: Приложение было тщательно протестировано и готово к деплою, что подтверждается наполнением базы данных тестовыми данными и стабильной работой всех сервисов.

Проект демонстрирует эффективное применение современных технологий и подходов в разработке программного обеспечения и может служить основой для дальнейшего развития и внедрения новых функций.

Таким образом, цель данной работы была успешно достигнута. Созданное веб-приложение «Издательская платформа» успешно сочетает в себе функциональность и безопасность, предоставляя пользователям мощный инструмент для обмена и открытия нового контента. Это делает его ценным ресурсом как для авторов, так и для читателей, и открывает путь для дальнейших инноваций в области цифрового издательства.

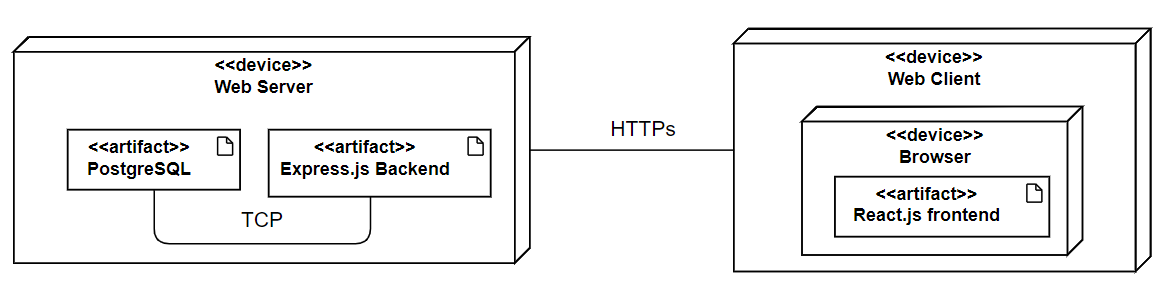
Список используемых источников

1. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com – Дата доступа: 15.03.2023.
2. Koa.js [Электронный ресурс] / Режим доступа: https://koajs.com. – Дата доступа: 19.03.2024.
3. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 23.03.2024.
4. React.js документация [Электронный ресурс] / Режим доступа: https://react.dev/learn. – Дата доступа: 19.04.2024.
5. Socket.io документация [Электронный ресурс] / Режим доступа: https://socket.io/docs/v4/. – Дата доступа: 27.04.2024.

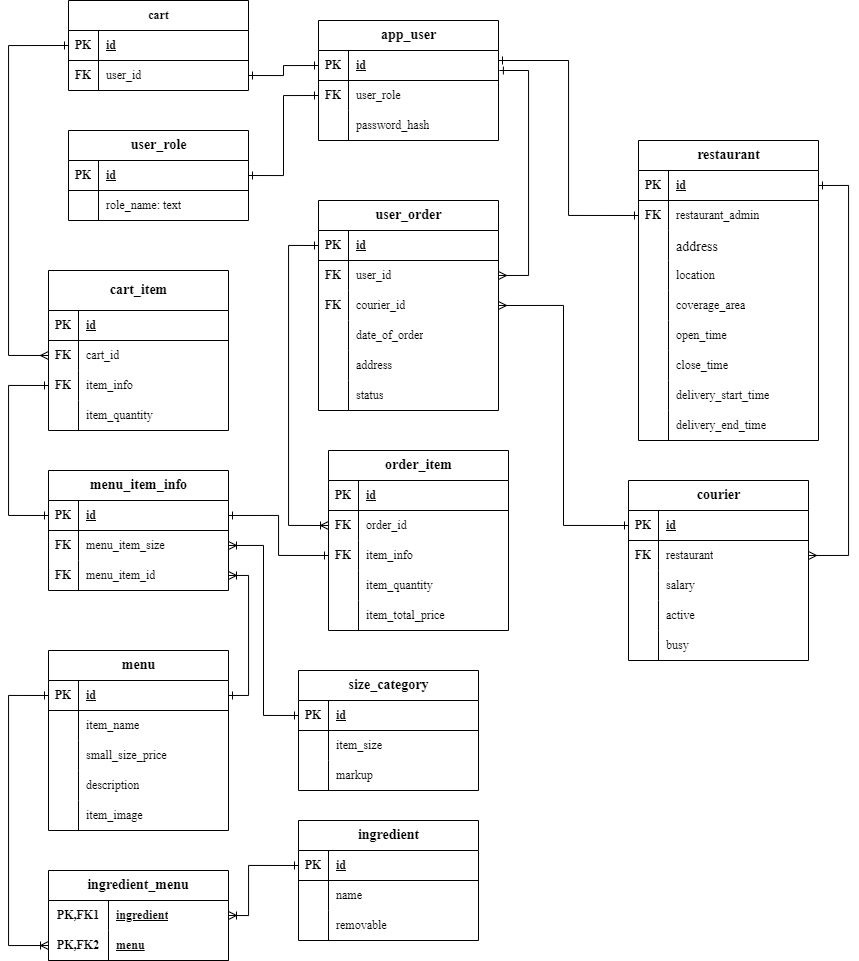
ПРИЛОЖЕНИЕ А



ПРИЛОЖЕНИЕ Б



ПРИЛОЖЕНИЕ В



ПРИЛОЖЕНИЕ Г

|  |
| --- |
| model App\_user {  id Int @id @default(autoincrement())  password\_hash String  user\_role Int  personal\_data Int  personal\_data\_rel Personal\_data @relation("app\_user\_personal\_dataTopersonal\_data", fields: [personal\_data], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "fk\_app\_user\_personal\_data")  user\_role\_rel User\_role @relation("app\_user\_user\_roleTouser\_role", fields: [user\_role], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "fk\_app\_user\_user\_role")  cart Cart?  restaurant Restaurant?  user\_orders User\_order[]  @@map("app\_user")  }  model Cart {  id Int @id @default(autoincrement())  user\_id Int @unique  app\_user App\_user @relation(fields: [user\_id], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "fk\_cart\_app\_user")  @@map("cart")  }  model Cart\_item {  id Int @id @default(autoincrement())  cart\_id Int  item\_info Int  item\_quantity Int  notes String?  menu\_item\_info Menu\_item\_info @relation(fields: [item\_info], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "fk\_cart\_menu\_item\_info")  @@index([cart\_id], map: "idx\_cart\_item\_cart\_id")  @@index([cart\_id, item\_info], map: "idx\_cart\_item\_cart\_info")  @@index([item\_info], map: "idx\_cart\_item\_item\_info")  @@map("cart\_item")  }  model Courier {  id Int @id @default(autoincrement())  personal\_data Int  salary Decimal @db.Decimal(8, 2)  restaurant Int  active Boolean @default(false)  busy Boolean @default(false)  personal\_data\_rel Personal\_data @relation("courier\_personal\_dataTopersonal\_data", fields: [personal\_data], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "fk\_courier\_personal\_data")  restaurant\_rel Restaurant @relation("courier\_restaurantTorestaurant", fields: [restaurant], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "fk\_courier\_restaurant")  user\_orders User\_order[]  @@index([active], map: "idx\_courier\_active")  @@index([active, busy], map: "idx\_courier\_active\_busy")  @@index([personal\_data], map: "idx\_courier\_personal\_data")  @@index([busy], map: "idx\_courier\_busy")  @@index([restaurant], map: "idx\_courier\_restaurant")  @@map("courier")  }  model Menu {  id Int @id @default(autoincrement())  item\_name String  small\_size\_price Decimal @db.Decimal(5, 2)  item\_image String?  ingredients Ingredient\_menu[] @relation("ingredient\_menu\_menuTomenu")  menu\_item\_infos Menu\_item\_info[]  @@index([item\_name], map: "idx\_item\_name")  @@index([small\_size\_price], map: "idx\_small\_size\_price")  @@map("menu")  }  model Menu\_item\_info {  id Int @id @default(autoincrement())  menu\_item\_id Int  menu\_item\_size Int @db.SmallInt  cart\_items Cart\_item[]  size\_category Size\_category @relation(fields: [menu\_item\_size], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "fk\_menu\_ii\_size\_category")  menu Menu @relation(fields: [menu\_item\_id], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "fk\_menu\_item\_info\_menu")  order\_items Order\_item[]  @@map("menu\_item\_info")  }  model Order\_item {  id Int @id @default(autoincrement())  order\_id Int  item\_info Int  item\_quantity Int  item\_total\_price Decimal @db.Decimal(5, 2)  notes String?  menu\_item\_info Menu\_item\_info @relation(fields: [item\_info], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "fk\_order\_item\_menu\_item\_info")  user\_order User\_order @relation(fields: [order\_id], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "fk\_order\_item\_user\_order")  @@index([order\_id, item\_info], map: "idx\_order\_item\_order\_item\_info")  @@map("order\_item")  }  model Personal\_data {  id Int @id @default(autoincrement())  full\_name String  email String @unique  phone\_number String @unique  date\_of\_birth DateTime? @db.Date  app\_users App\_user[] @relation("app\_user\_personal\_dataTopersonal\_data")  couriers Courier[] @relation("courier\_personal\_dataTopersonal\_data")  @@map("personal\_data")  }  model Restaurant {  id Int @id @default(autoincrement())  address String @unique  location String  coverage\_area String  restaurant\_admin Int @unique  open\_time DateTime @db.Timestamp(6)  close\_time DateTime @db.Timestamp(6)  delivery\_start\_time DateTime @db.Timestamp(6)  delivery\_end\_time DateTime @db.Timestamp(6)  couriers Courier[] @relation("courier\_restaurantTorestaurant")  app\_user App\_user @relation(fields: [restaurant\_admin], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "fk\_restaurant\_app\_user")  @@map("restaurant")  }  model Size\_category {  id Int @id @default(autoincrement()) @db.SmallInt  item\_size Decimal @db.Decimal(4, 2)  markup Decimal @db.Decimal(4, 2)  menu\_item\_infos Menu\_item\_info[]  @@index([item\_size, markup], map: "idx\_item\_size\_markup")  @@map("size\_category")  }  model User\_order {  id Int @id @default(autoincrement())  date\_of\_order DateTime @db.Timestamp(6)  user\_id Int  address String  courier\_id Int  status String @default("В работе")  order\_items Order\_item[]  app\_user App\_user @relation(fields: [user\_id], references: [id], onDelete: NoAction, onUpdate: NoAction, map: "fk\_user\_order\_app\_user")  courier Courier @relation(fields: [courier\_id], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "fk\_user\_order\_courier")  @@map("user\_order")  }  model User\_role {  id Int @id @default(autoincrement())  role\_name String  app\_users App\_user[] @relation("app\_user\_user\_roleTouser\_role")  @@map("user\_role")  }  model Ingredient {  id Int @id @default(autoincrement())  name String  removable Boolean @default(false)  ingredient\_menu Ingredient\_menu[] @relation("ingredient\_menu\_ingredientToingredient")  @@map("ingredient")  }  model Ingredient\_menu {  ingredient Int  menu Int  ingredient\_ingredient Ingredient @relation("ingredient\_menu\_ingredientToingredient", fields: [ingredient], references: [id], onDelete: NoAction, onUpdate: NoAction)  menu\_menu Menu @relation("ingredient\_menu\_menuTomenu", fields: [menu], references: [id], onDelete: NoAction, onUpdate: NoAction)  @@id([ingredient, menu])  @@map("ingredient\_menu")  } |