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 Фамилия Имя Отчество

(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 ресторанный бизнес становится все более динамичным и требовательным к эффективному управлению. Поэтому возникает необходимость в разработке инновационных инструментов, способных оптимизировать процессы управления и обслуживания. В рамках данного проекта будет разработано web-приложение «Система управления сетью пиццерий», которое позволит значительно упростить и улучшить управление ресторанами, курьерами и пользователями.

Основной целью курсового проекта является создание комплексного и интуитивно понятного веб-приложения, которое поможет повысить эффективность управления сетью пиццерий, оптимизировать процессы заказа и доставки, а также улучшить обслуживание клиентов.

Требования к проекту:

* приложение должно быть направлено на упрощение процесса заказа пиццы для конечного пользователя, предоставляя удобный интерфейс для выбора товаров из меню, оформления заказа;
* приложение должно обеспечивать гибкое управление меню ресторана, его редактирование и администрирование, что позволит быстро реагировать на изменения в ассортименте и спросе;
* система должна предоставлять механизмы для распределения заказов между ресторанами и курьерами с учетом их географического расположения и загрузки, что позволит минимизировать время доставки и оптимизировать процессы внутри сети пиццерий;
* приложение должно предоставлять возможность уведомления пользователей о статусе их заказа, что поспособствует повышению удовлетворенности клиентов и созданию положительного опыта пользования сервисом.

Техническая реализация проекта основана на асинхронном программировании с использованием языка Typescript и платформы Node.js. Программное средство взаимодействует с базой данных и может быть запущено на различных платформах.

1 Постановка задачи

* 1. Анализ прототипов

Курсовой проект представляет из себя веб-приложение для сети пиццерий. В бизнес-сфере, в частности в сфере общественного питания, уже существуют некоторые аналогичные решения. В этом разделе будут приведены интернет-ресурсы, а также веб-приложения существующих пиццерий.

* + 1. Интернет-ресурс «Додо пицца»

Одним из альтернативных решений приложения пиццерии является интернет-ресурс «dodopizza.by». На веб-сайте можно найти широкий ассортимент блюд, напитков, десертов. Каталог товаров на сайте поделен на несколько категорий: пицца, закуски, напитки, коктейли, десерты и другие товары.

Интерфейс интернет-ресурса «dodopizza.by» представлен на рисунке 1.1.

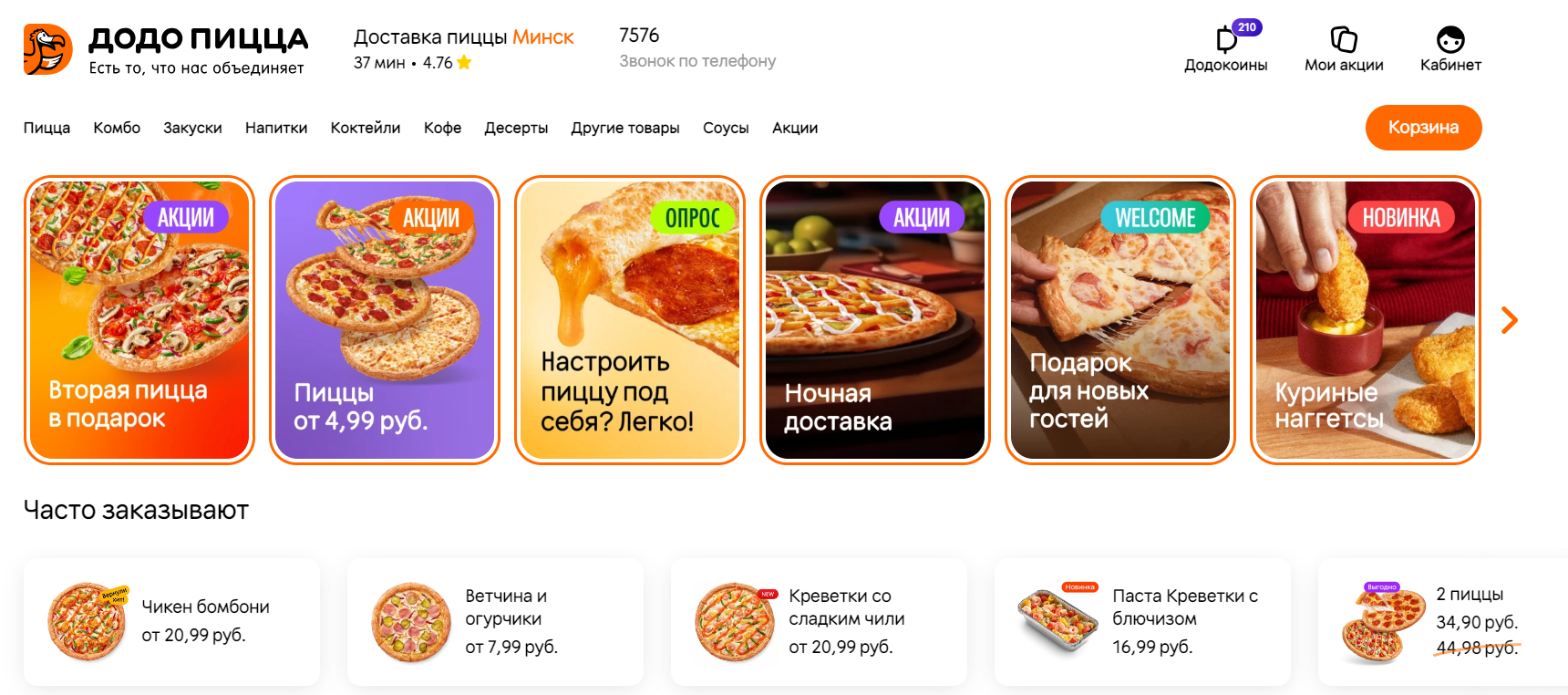


Рисунок 1.1 – Интерфейс «dodopizza.by»

Преимуществами веб-сайта являются возможность совершения заказа с выбором опции доставки, отображение карты зоны доставки, автоматическое определение местоположения пользователя, возможность изменения размеров пиццы.

К недостаткам можно отнести отсутствие поиска по меню, а также отсутствие механизма уведомлений о статусе заказа.

* + 1. Интернет-ресурс «Пицца Лисицца»

Веб-сайт «Пицца Лисицца» предлагает удобный интерфейс для заказа онлайн, что упрощает процесс выбора пиццы и других блюд, а также отображает информацию о меню, о расположении ресторанов и условиях доставки, что помогает клиентам принимать информированные решения.

Интерфейс интернет-ресурса «Пицца Лисицца» представлен на рисунке 1.2.

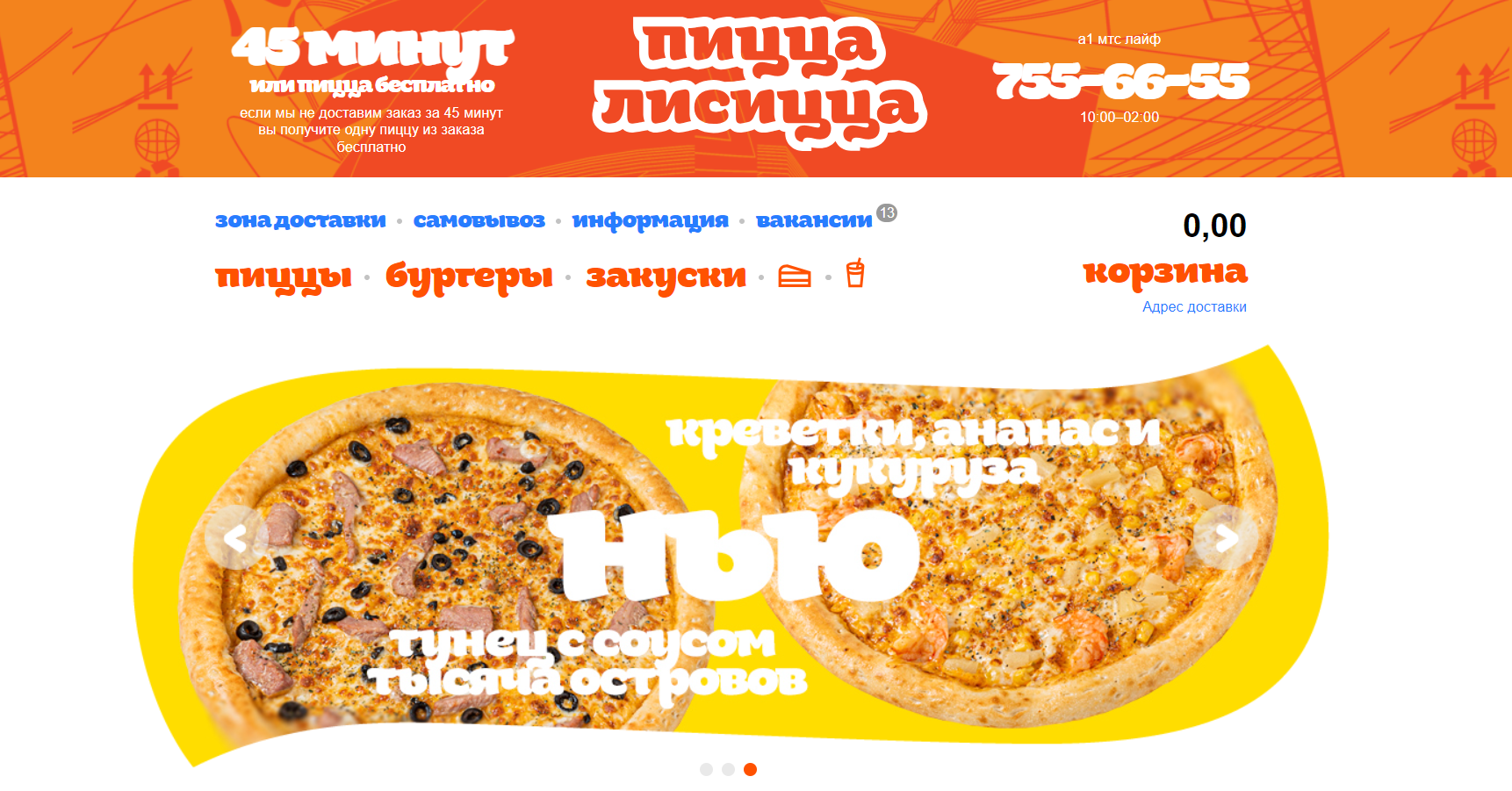


Рисунок 1.2 – Интерфейс «dominos.by»

К преимуществам веб-сайта можно отнести возможность регистрации пользователя, возможность работы в личном кабинете, управление корзиной, просмотр совершенных заказов.

Недостатками веб-сайта является отсутствие функции изменения ингредиентов пиццы и отсутствие поиска по наименованию товара.

* 1. Постановка задачи

На сегодняшний день актуальность разработки программного средства для управления сетью пиццерий велика, так как многие компании стремятся улучшить свои процессы производства, управления заказами и доставки, а также повысить удовлетворенность клиентов.

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:

* управление корзиной товаров;
* управление меню ресторана;
* совершение заказа;
* управление ресторанами;
* распределение заказов по ресторанам;
* поиск по наименованию товара;
* уведомление пользователя о статусе заказа;
* удаление ингредиентов пиццы;
* нахождение ближайшего ресторана для пользователя;
* управление курьерами.

Диаграмма вариантов использования представлена в приложении А.

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

2 Проектирование web-приложения

2.1 Архитектура приложения

Серверная часть веб-приложения реализована с использованием фреймворка Express.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, обычно 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.

Для хранения данных используется база данных Postgres. База данных Postgres с расширением PostGIS представляет собой мощный инструмент для хранения географических данных и их анализа. PostGIS добавляет в Postgres поддержку географических объектов, геометрических операций и индексации для эффективного выполнения пространственных запросов.

Преимуществом использование данной СУБД является возможность интеграции геоданных в приложения. Postgres с PostGIS обеспечивает удобные средства для работы с геоданными как на серверной, так и на клиентской стороне. Это позволяет разработчикам создавать приложения, которые могут анализировать и визуализировать геоданные, что особенно полезно в геоинформационных системах (ГИС), местных сервисах и приложениях для навигации.

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 ресторана. Основные сущности, выделенные в концептуальной модели, включают в себя:

* пользователи: представляет информацию о зарегистрированных пользователях системы;
* роли: определяет перечень допустимый пользовательских ролей;
* курьеры: содержит информацию о курьерах, работающих в сети ресторанов;
* меню: содержит информацию о блюдах, предлагаемых рестораном;
* ингредиенты: хранит информацию об ингредиентах блюд;
* заказы: хранит данные о заказах, сделанных пользователями;
* корзина: хранит данные о товарах, которые находятся в корзине пользователя;
* рестораны: представляет информацию о ресторанах сети.

Диаграмма базы данных представлена в приложении В.

По итогу были сформированы следующие таблицы: APP\_USER, USER\_ROLE, COURIER, PERSONAL\_DATA, MENU, RESTAURANT, SIZE\_CATEGORY, MENU\_ITEM\_INFO, USER\_ORDER, ORDER\_ITEM, CART, CART\_ITEM, INGREDIENT, INGREDIENT\_MENU. Далее представлено описание каждой из них.

Таблица APP\_USER используется для хранения данных о пользователях, таких как хешированный пароль, роль пользователя, персональные данные. Ее структура представлена в таблице 2.1.

Таблица 2.1 – Структура таблицы APP\_USER

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| password\_hash | text | Not null |
| user\_role | int | Not null, foreign key |
| personal\_data | int | Not null, foreign key |

Таблица USER\_ROLE хранит информацию о ролях пользователей. Ее структура представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы USER\_ROLE

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| role\_name | text | Not null |

Таблица COURIER используется для хранения информации о курьерах, а именно персональные данные, зарплата, место работы (ресторан), статус активности, готовность принятия заказа. Структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы COURIER

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| personal\_data | int | Not null, foreign key |
| salary | numeric(8, 2) | Not null |
| restaurant | int | Not null, foreign key |
| active | boolean | Not null, default |
| ready\_to\_go | boolean | Not null, default |

Таблица PERSONAL\_DATA содержит информацию о персональных данных: полное имя, адрес электронной почты, номер телефона, дата рождения. Структура представлена в таблице 2.4.

Таблица 2.4 – Структура таблицы PERSONAL\_DATA

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| full\_name | text | Not null |
| email | text | Not null, unique |
| phone\_number | text | Not null, unique |
| date\_of\_birth | date | Not null |

Таблица MENU содержит информацию блюдах ресторана: название блюда, цена, описание, путь к изображению товара на сервере. Ее структура представлена в таблице 2.5.

Таблица 2.5 – Структура таблицы MENU

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| item\_name | text | Not null |
| small\_size\_price | numeric(5, 2) | Not null |
| description | text |  |
| item\_image | text |  |

Таблица RESTAURANT содержит информацию ресторанах: адрес ресторана, местоположение, зона доставки, идентификатор администратора ресторана, время открытия, время закрытия, время начала работы доставки, время окончания работы доставки. Ее структура представлена в таблице 2.6.

Таблица 2.6 – Структура таблицы RESTAURANT

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| address | text | Not null, unique |
| location | geometry | Not null |
| coverage\_area | geometry | Not null |
| restaurant\_admin | int | Not null, unique, foreign key |
| open\_time | timestamp | Not null |
| close\_time | timestamp | Not null |
| delivery\_start\_time | timestamp | Not null |
| delivery\_end\_time | timestamp | Not null |

Таблица SIZE\_CATEGORY содержит информацию о ценах на блюда в ресторане в соответствии с размерами блюд. Таблица имеет поля, описывающие размер блюда и ценовую надбавку на данный размер. По итогу, конечная стоимость блюда будет равна произведению базовой цены на надбавку за данный размер. Структура представлена в таблице 2.7.

Таблица 2.7 – Структура таблицы SIZE\_CATEGORY

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | smallserial | Primary key |
| item\_size | numeric(4, 2) | Not null |
| markup | numeric(4, 2) | Not null |

Таблица MENU\_ITEM\_INFO связывает таблицы MENU и SIZE\_CATEGORY. Кроме уникального идентификатора, таблица также имеет поля MENU\_ITEM\_ID и MENU\_ITEM\_SIZE, которые ссылаются на первичные ключи связываемых таблиц.

Таблица USER\_ORDER содержит информацию заказах пользователей: дата заказа, идентификатор пользователя, адрес доставки, идентификатор курьера. Ее структура представлена в таблице 2.8.

Таблица 2.8 – Структура таблицы USER\_ORDER

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| date\_of\_order | timestamp | Not null |
| user\_id | int | Not null, foreign key |
| address | text | Not null |
| status | text | Not null, check, default |
| courier\_id | int | Not null, foreign key |

Таблица ORDER\_ITEM представляет информацию о блюдах в заказе и связывает таблицы USER\_ORDER и MENU\_ITEM\_INFO. Ее структура представлена в таблице 2.9.

Таблица 2.9 – Структура таблицы ORDER\_ITEM

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| order\_id | int | Not null, foreign key |
| item\_info | int | Not null, foreign key |
| item\_quantity | int | Not null |
| item**\_**total\_price | numeric(5, 2) | Not null |

Таблица CART представляет информацию о корзине пользователя. Ее структура представлена в таблице 2.10.

Таблица 2.10 – Структура таблицы CART

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| user\_id | int | Not null, foreign key |

Таблица CART\_ITEM аналогично таблице ORDER\_ITEM связывает таблицы CART и MENU\_ITEM\_INFO, что позволяет в корзинах пользователей хранить множество различных товаров с различными характеристиками.

Таблица INGREDIENT представляет информацию об ингредиентах. Она хранит название ингредиента и атрибут removable, который показывает, сможет ли пользователь его удалить. Ее структура представлена в таблице 2.11.

Таблица 2.11 – Структура таблицы INGREDIENT

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| name | text | Not null |
| removable | boolean | Not null, default |

Таблица INGREDIENT\_MENU связывает таблицы INGREDIENT и MENU, что позволяет соотносить множество ингредиентов с множеством товаров.

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

3 Разработка web-приложения

3.1 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 представлена на рисунке 3.1.

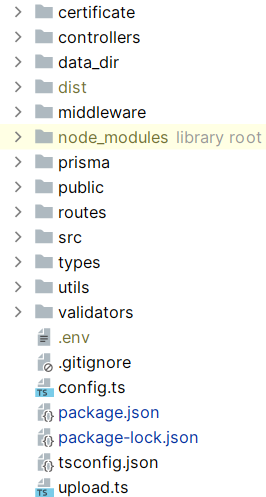


Рисунок 3.1 – Структура проекта серверного приложения

Входной точкой приложения является файл app.ts. В нем происходит инициализация express-сервера, websocket сервера, а также установка сертификатов для обеспечения защищенного соединения с сервером.

В директории prisma находится схема моделей, созданная ORM-фреймворком prisma. Модели описывают сущности базы данных и представляют интерфейс для взаимодействия с ней. Схема моделей представлена в приложении Г.

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

|  |
| --- |
| import express from "express";  import controller from "../controllers/couriersController";  import roleMiddleware from "../middleware/roleMiddleware";  import {courierDataValidator} from "../validators";  const router = express.Router();  router.get('/', roleMiddleware(['restaurant\_admin']), controller.getAll);  router.get('/:courierId', roleMiddleware(['restaurant\_admin']), controller.getOne);  router.post('/', roleMiddleware(['restaurant\_admin']), courierDataValidator, controller.createNewCourier);  router.put('/:courierId', roleMiddleware(['restaurant\_admin']), courierDataValidator, controller.changeCourier);  router.delete('/:courierId', roleMiddleware(['restaurant\_admin']), controller.deleteCourier)  export default router; |

Листинг 3.2 – Маршрутизатор для сущности «курьер»

С каждым маршрутом сопоставляется определенный метод контроллера, который непосредственно осуществляет обработку запроса. Пример метода контроллера getOne, который возвращает информацию об одном курьере представлен в листинге 3.3.

|  |
| --- |
| const getOne = async (req: Request, res: Response) => {  if (!parseInt(req.params['courierId'], 10)) {  return res.status(422).json({error: [{msg: "Неверный идентификатор курьера"}]})  }  let id: number = parseInt(req.params['courierId'], 10);  try {  let courier = await prisma.courier.findUnique({  where: {id: id},  select: {  id: true,  salary: true,  active: true,  busy: true,  restaurant\_rel: {  select: {restaurant\_admin: true}  },  personal\_data\_rel: {  select: {  full\_name: true,email: true, phone\_number: true,  date\_of\_birth: true  }  }  }  });  res.status(200).json(courier);  } catch (err) {  return res.status(422).json({error: [{msg: "Ошибка при получении курьера"}]})  }  } |

Листинг 3.3 – Метод getOne контроллера courierController

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

В приложении реализовано несколько middleware. Первый тип – это middleware для аутентификации и авторизации. Middleware userLogin необходим для того, чтобы защитить некоторые маршруты от не аутентифицированных пользователей. Код middleware представлен в листинге 3.4.

|  |
| --- |
| export default function (shouldBeLoggedIn : boolean) {  return function (req: Request, res : Response, next : NextFunction) {  if (req.method === "OPTIONS") {  next()  }  const token = req.headers.authorization ? !!req.headers.authorization.split(" ")[1] : false;  const session = req.cookies["\_es\_usr\_session"];  if (shouldBeLoggedIn && session === undefined)  return res.status(403).json({error: "Для продолжения войдите в систему"})  else if (shouldBeLoggedIn === token)  next();  else  return res.status(403).json({error: "Доступ запрещен"})  }  } |

Листинг 3.4 – Middleware userLogin

Middleware userRole предназначен для проверки прав пользователя на доступ к определенным ресурсам. Код middleware представлен в листинге 3.5.

|  |
| --- |
| export default function (roles : string[]) {  return function (req : Request, res : Response, next : NextFunction) {  if (req.method === "OPTIONS") {  next()  }  try {  const token = req.headers.authorization ? req.headers.authorization.split(" ")[1] : "";  if (token.length === 0) {  return res.status(403).json({error: "Пользователь не авторизован"})  }  const payload = jwt.verify(token, secret.secret);  let hasRole = roles.includes(payload.user\_role\_rel.role\_name);  if (!hasRole) {  return res.status(403).json({error: "У вас нет доступа"})  }  next();  } catch (e) {  console.log(e)  return res.status(403).json({error: "Пользователь не авторизован"})  }  }  }; |

Листинг 3.5 – Middleware userRole

Второй тип middleware – это middleware для валидации данных запроса. Все валидаторы находятся в директории validators. Пример валидатора данных меню представлен в листинге 3.6.

|  |
| --- |
| export const menuDataValidator = [  body("item\_name")  .trim()  .notEmpty().withMessage('Название блюда не должно быть пустым')  .isLength({min: 1, max: 255}).withMessage('Название блюда должно содержать от 1 до 255 символов'),  body("small\_size\_price")  .notEmpty().withMessage('Цена не должна быть пустой')  .isFloat({min: 0.01, max: 1000}).withMessage('Цена должна быть указана числом в пределах от 0.01 до 1000'),  body("ingredients")  .notEmpty().withMessage('Описание блюда не должно быть пустым')  ] |

Листинг 3.6 – Валидатор данных для сущности «меню»

В директории types содержаться описания типов и интерфейсов, используемых в приложении.

В директории utils находятся функции-утилиты, обеспечивающие дополнительную логику при работе с данными.

В директории certificate хранятся сертификаты https сервера.

3.2 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.

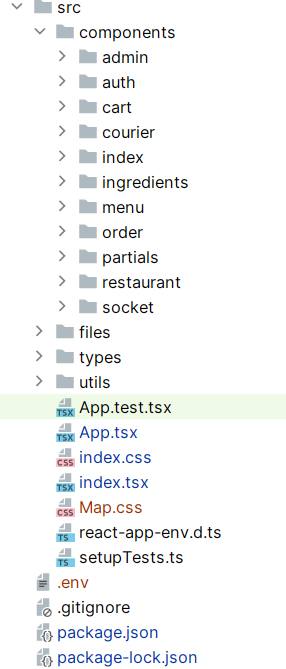


Рисунок 3.7 – Структура клиентского приложения

Директория src содержит весь исходный код приложения.

Директория components содержит все React-компоненты, организованные по группам. Каждая группа компонентов отвечает за работу с одной сущностью приложения.

Пример компонента-контейнера для элементов меню представлен в листинге 3.8.

|  |
| --- |
| const Menu = ({setTab, adminMenu}: HeadAdminPanelPropsType) => {  setTab(-1);  return (  <div>  <AdminMenuTabs currentTab={3} adminMenu={adminMenu}/>  <div className="w-full md:max-w-6xl mx-auto mt-8">  <a href="/menu/add" className="text-white bg-blue-700 hover:bg-blue-800 focus:ring-4 focus:ring-blue-300 font-medium rounded-lg text-sm  px-5 py-2.5 mr-2 mb-2 dark:bg-blue-600 dark:hover:bg-blue-700 focus:outline-none dark:focus:ring-blue-800">  Добавить товар  </a>  </div>  <DisplayMenu menuUrlAction={"update"}/>  </div>  );  }; |

Листинг 3.8 – Компонент-контейнер для элементов меню

Файл app.tsx является основным компонентом приложения. Помимо инициализации всех остальных компонентов приложения он также инициализирует websoket-клиента. Структура web-socket клиента представлена в листинге 3.9.

|  |
| --- |
| useEffect(() => {  let token = localStorage.getItem('token');  if (token !== null) setUser(jwtDecode(token));  else setUser(null);  socket.emit('setClientId', user?.id);  socket.on('orderStatusChanged', data => {  Push.create('Статус заказа изменился', {  body: `Новый статус заказа №${data.orderId}: ${data.status.toLowerCase()}`,  icon: '/favicon.png'  })  });  socket.on('newOrderAdmin', () => {  Push.create('Новый заказ!', {  body: 'Поступил новый заказ',  icon: '/favicon.png'  })  });  return () => {  socket.off('orderStatusChanged');  socket.off('newOrderAdmin');  };  }, [user?.id]); |

Листинг 3.9 – Инициализация websocket-клиента

Директория files содержит статические файлы приложения. Директория types содержит описание типов, используемых в приложении, а директория utils содержит функции с дополнительным функционалом.

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.

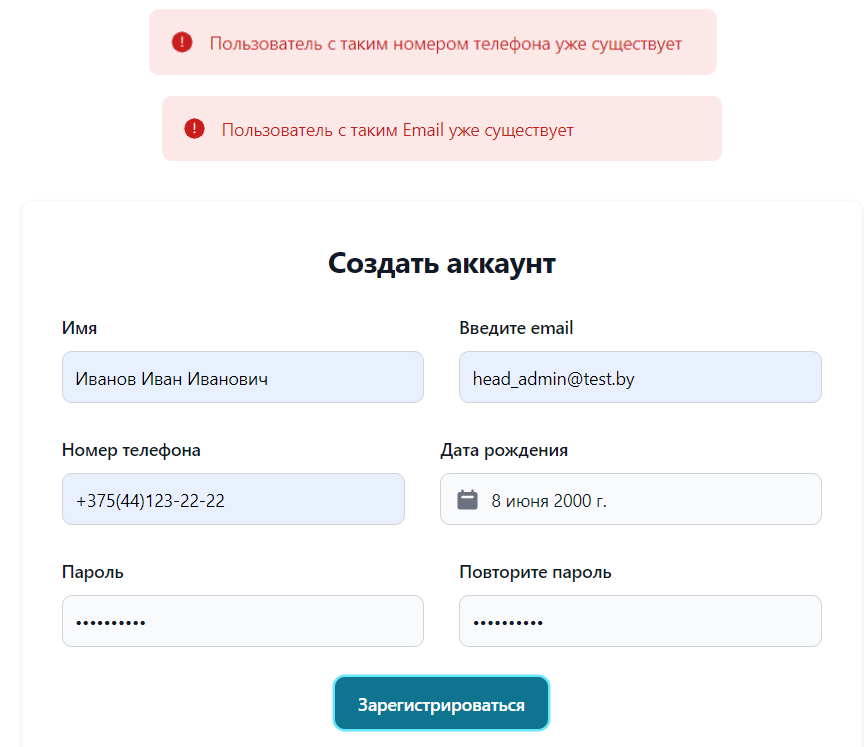


Рисунок 4.1 – Обработка некорректных данных при регистрации

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

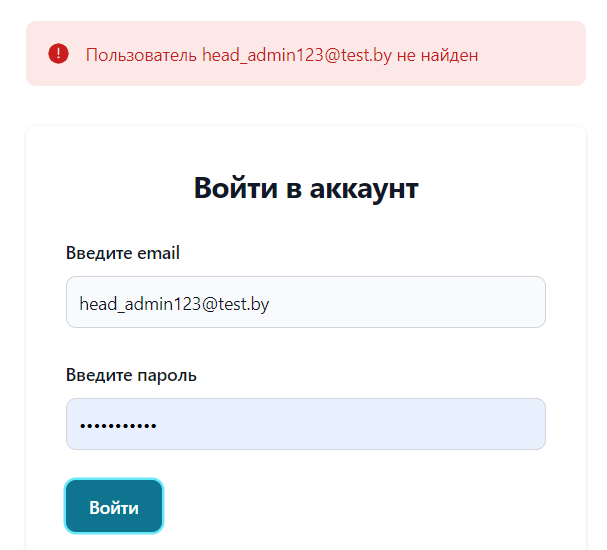


Рисунок 4.2 – Вход под несуществующим пользователем

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

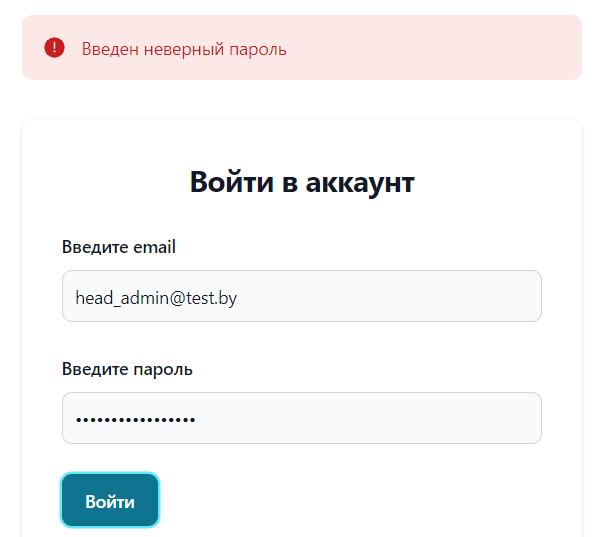


Рисунок 4.3 – Вход с неправильным паролем

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

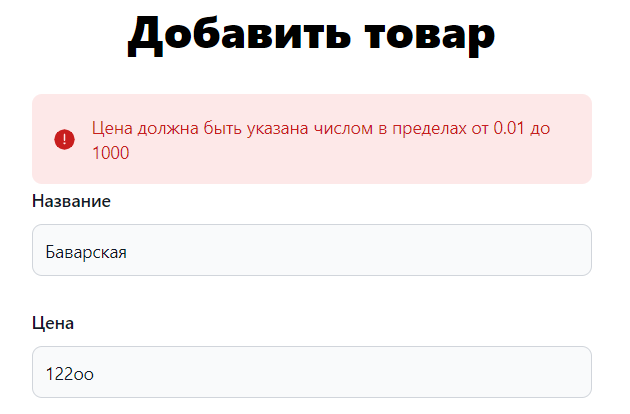


Рисунок 4.4 – Вход числовых данных неверного формата

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

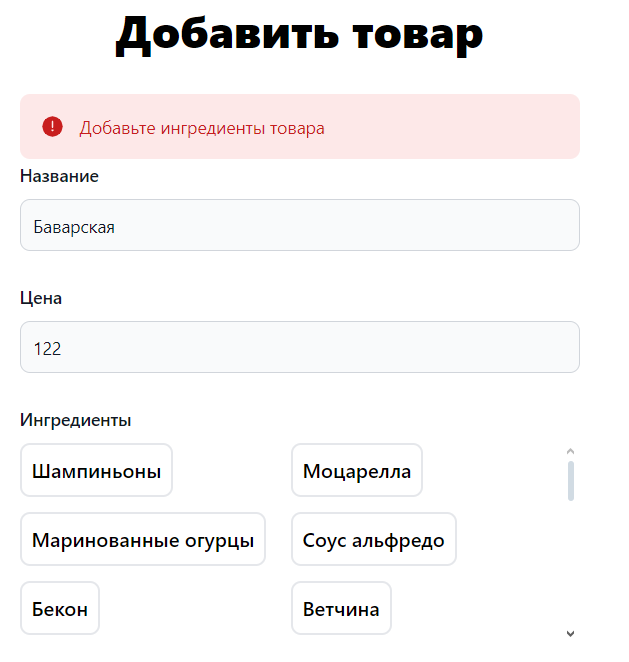


Рисунок 4.5 – Ошибка отправки пустых данных

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.

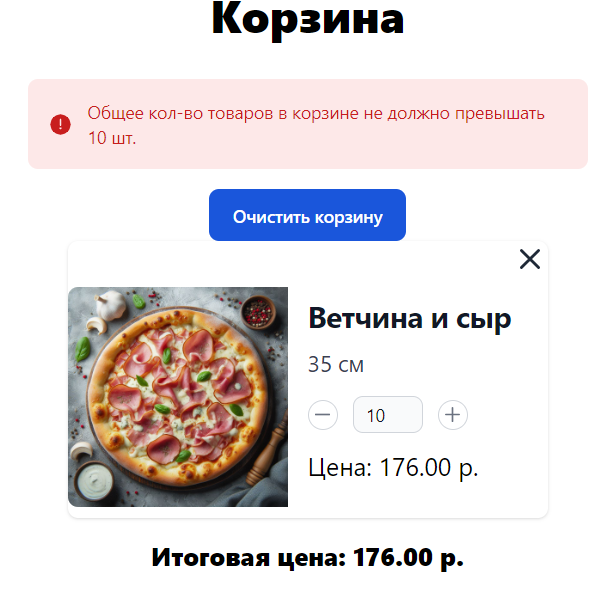


Рисунок 4.6 – Ошибка превышения количества товаров в корзине

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.

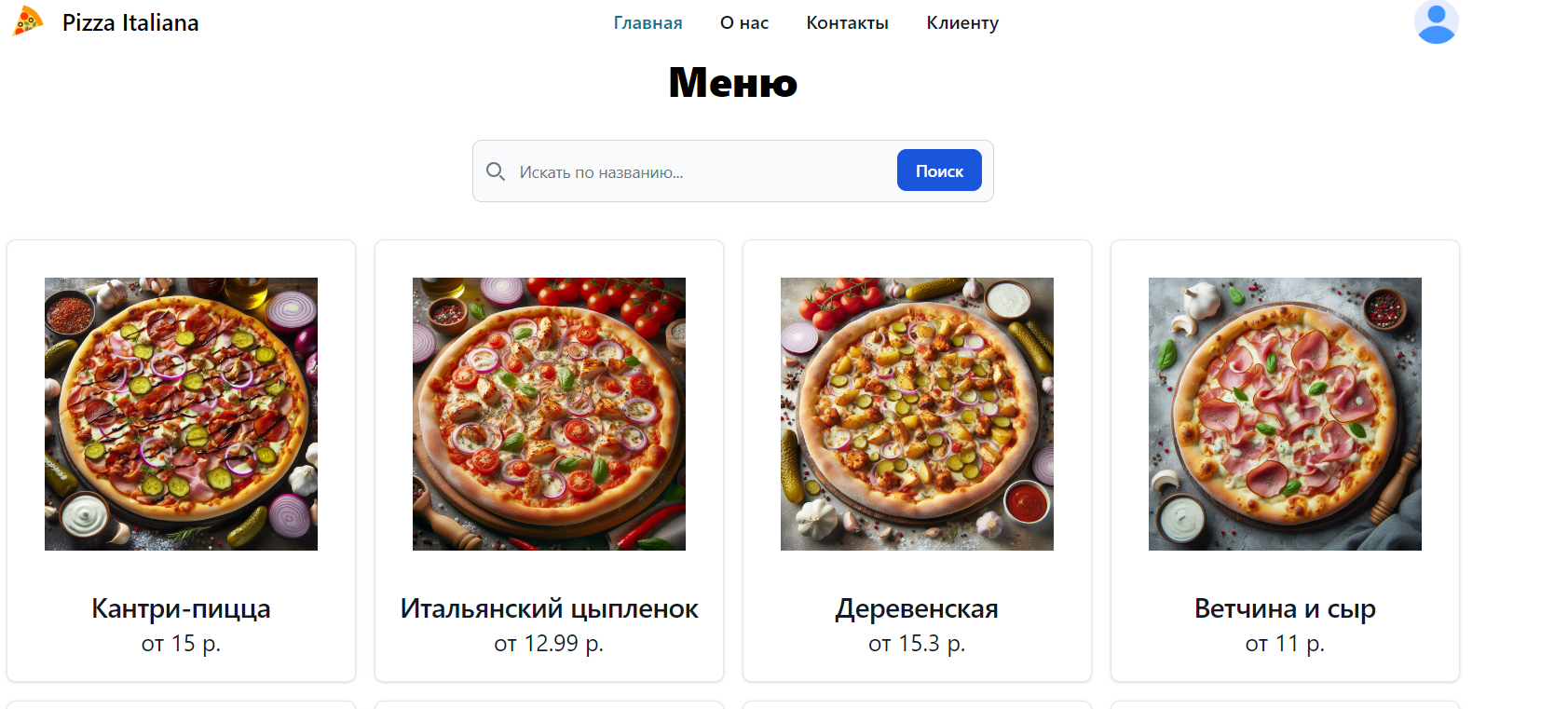


Рисунок 5.1 – Главная страница приложения

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.

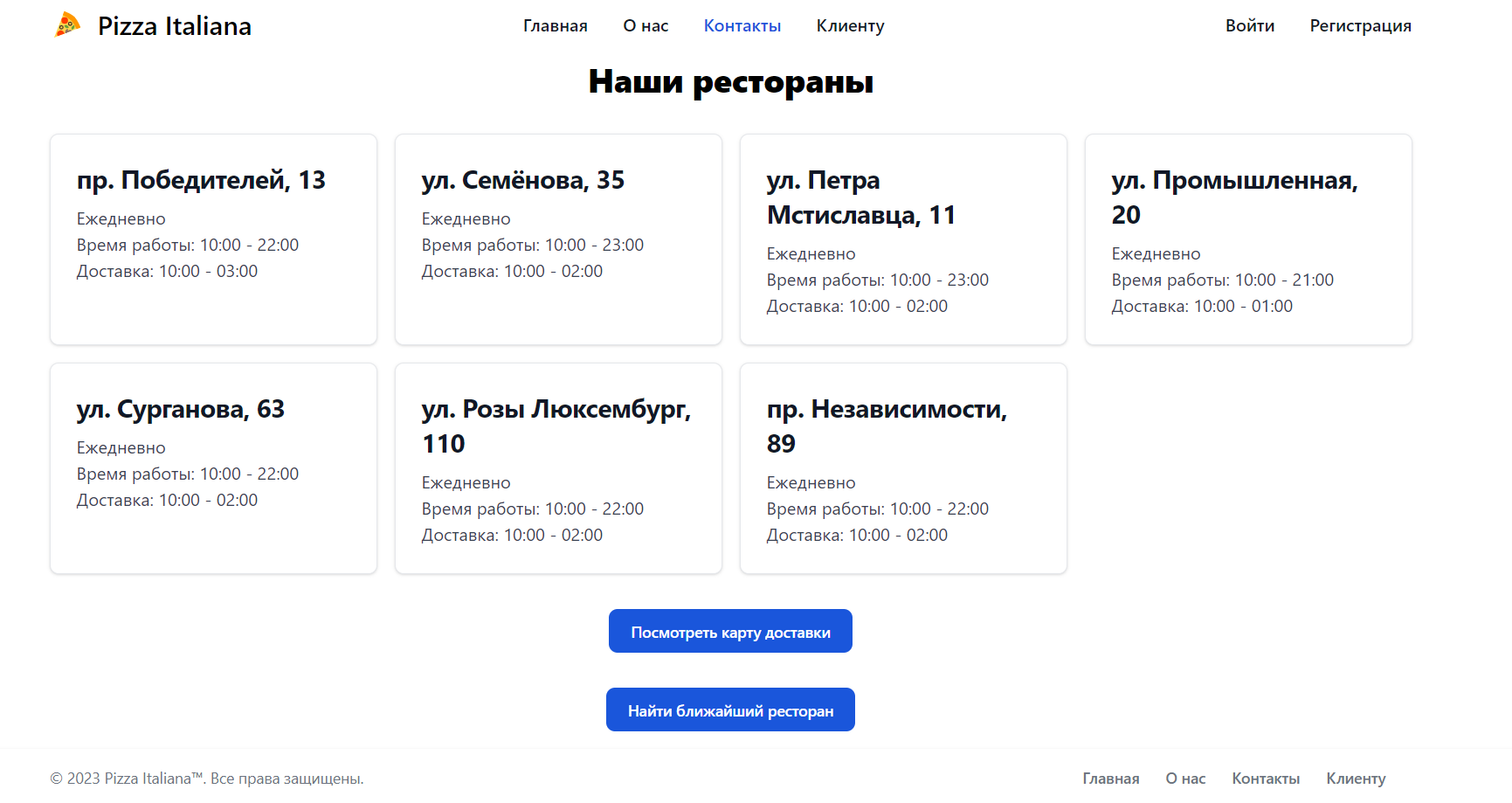


Рисунок 5.2 – Страница «Контакты»

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

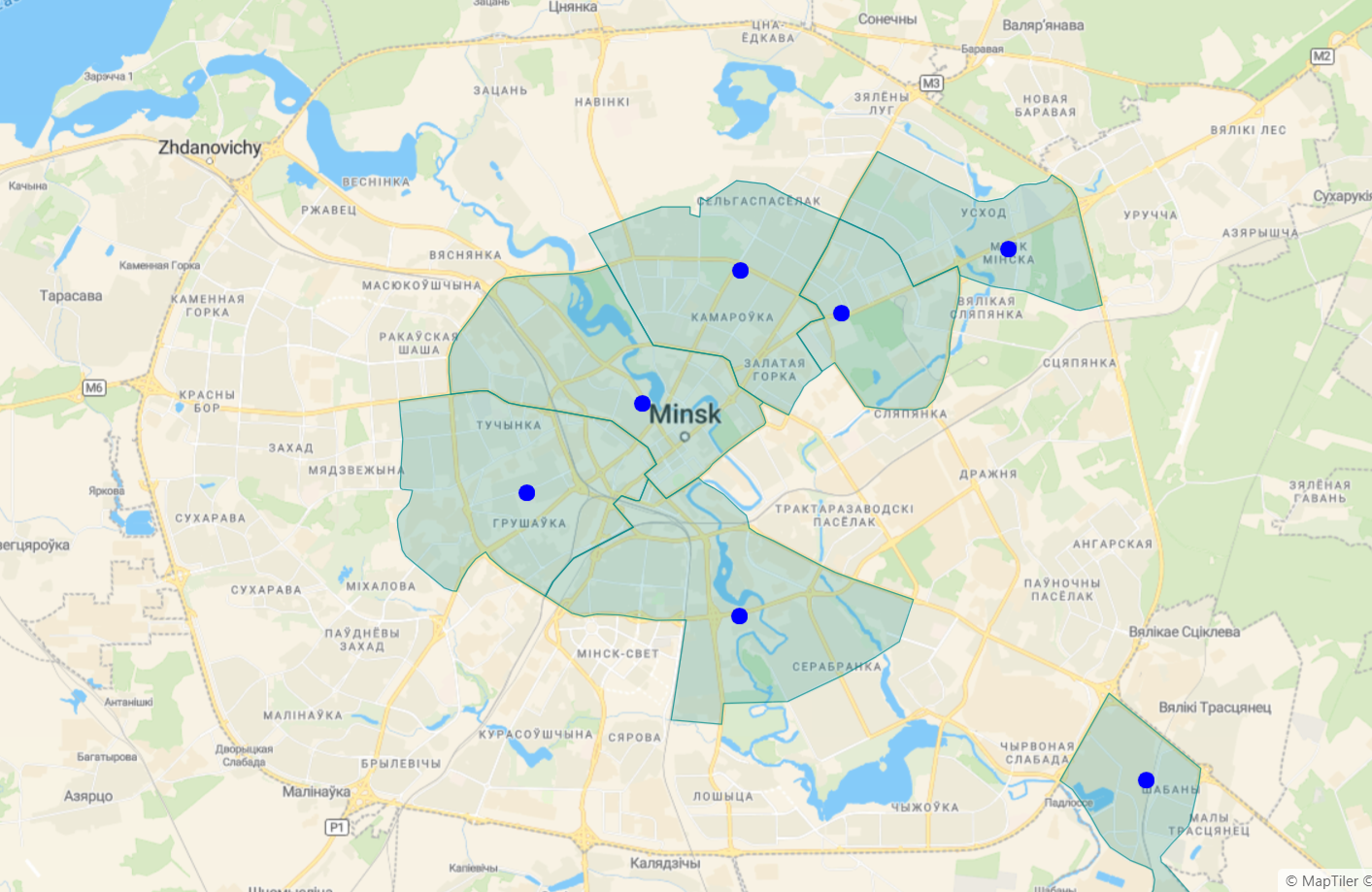


Рисунок 5.3 – Карта доставки

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.

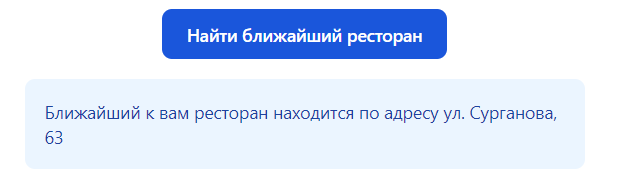


Рисунок 5.4 – Работа функции нахождения ближайшего ресторана

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.

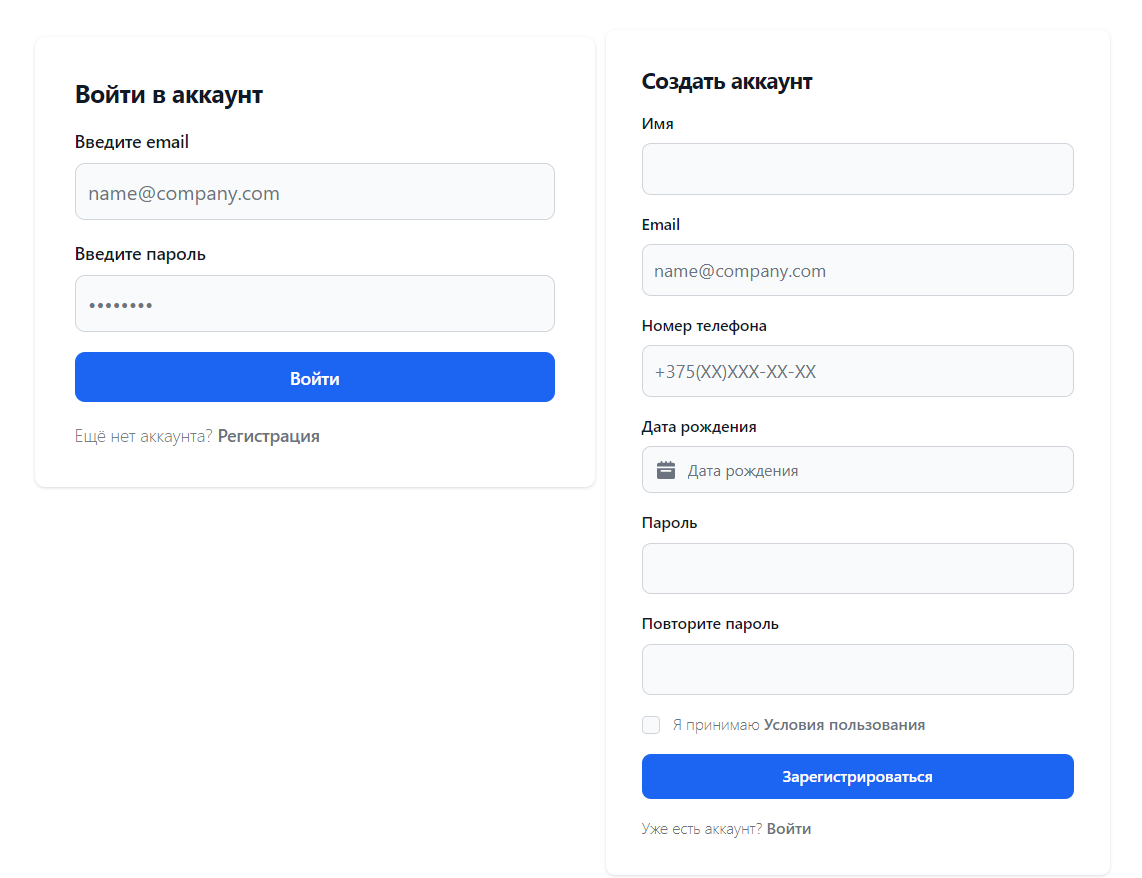


Рисунок 5.5 – Формы входа и регистрации

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.

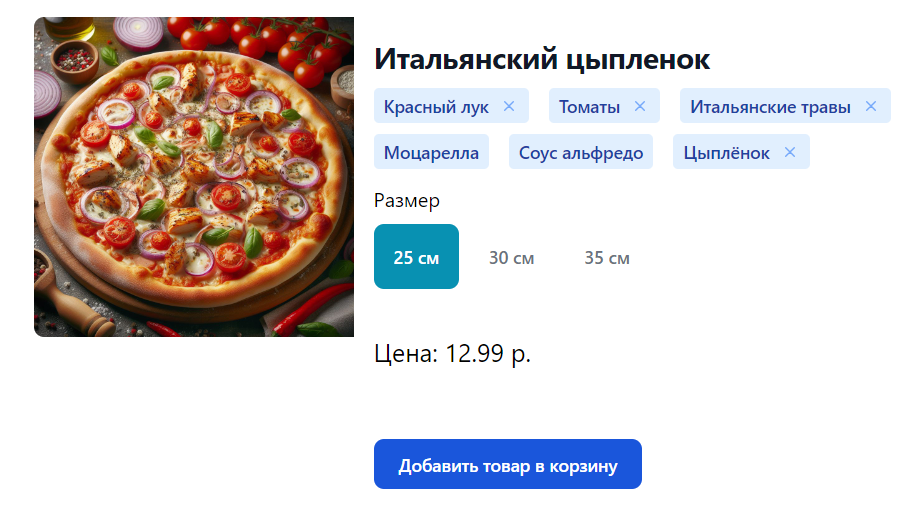


Рисунок 5.6 – Добавление товара в корзину

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

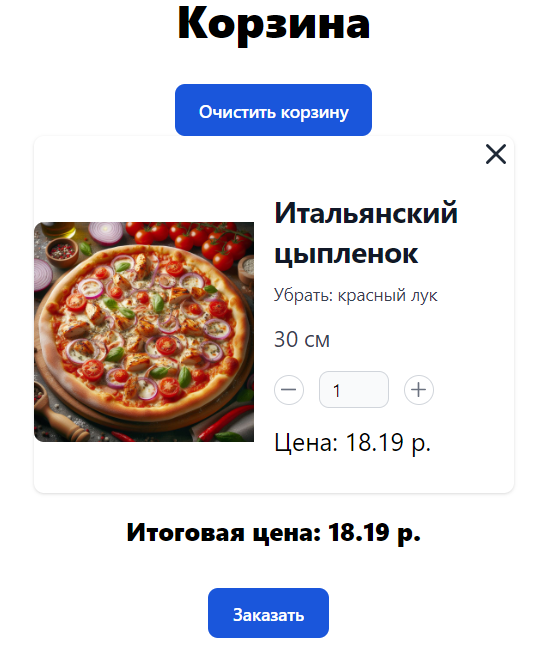


Рисунок 5.7 – Интерфейс раздела «Корзина»

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.

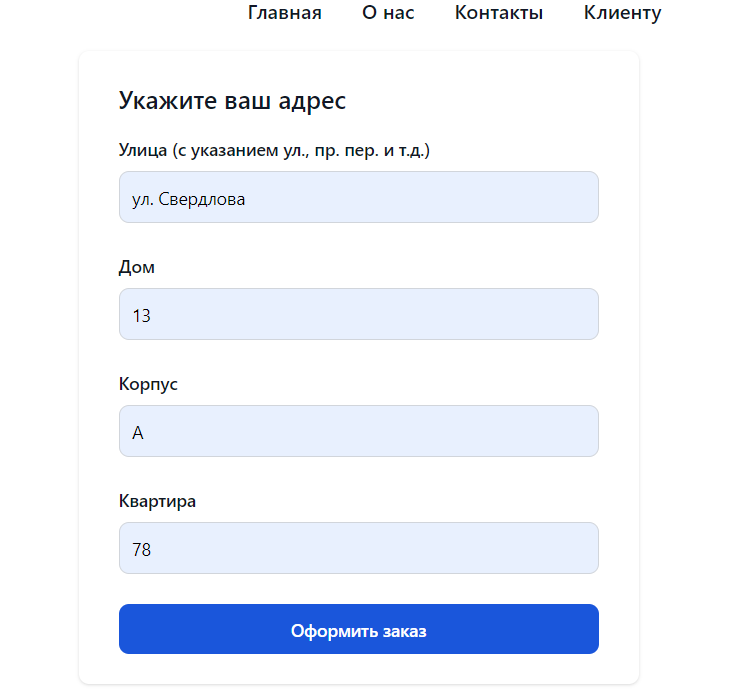


Рисунок 5.8 – Совершение заказа

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.

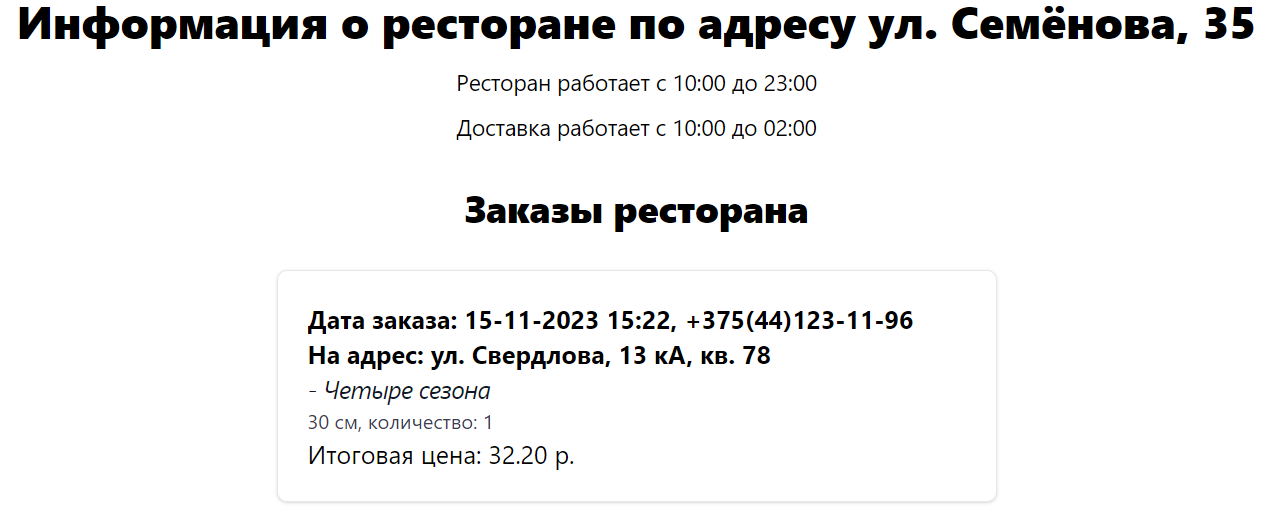


Рисунок 5.9 – Заказы в ресторане

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.

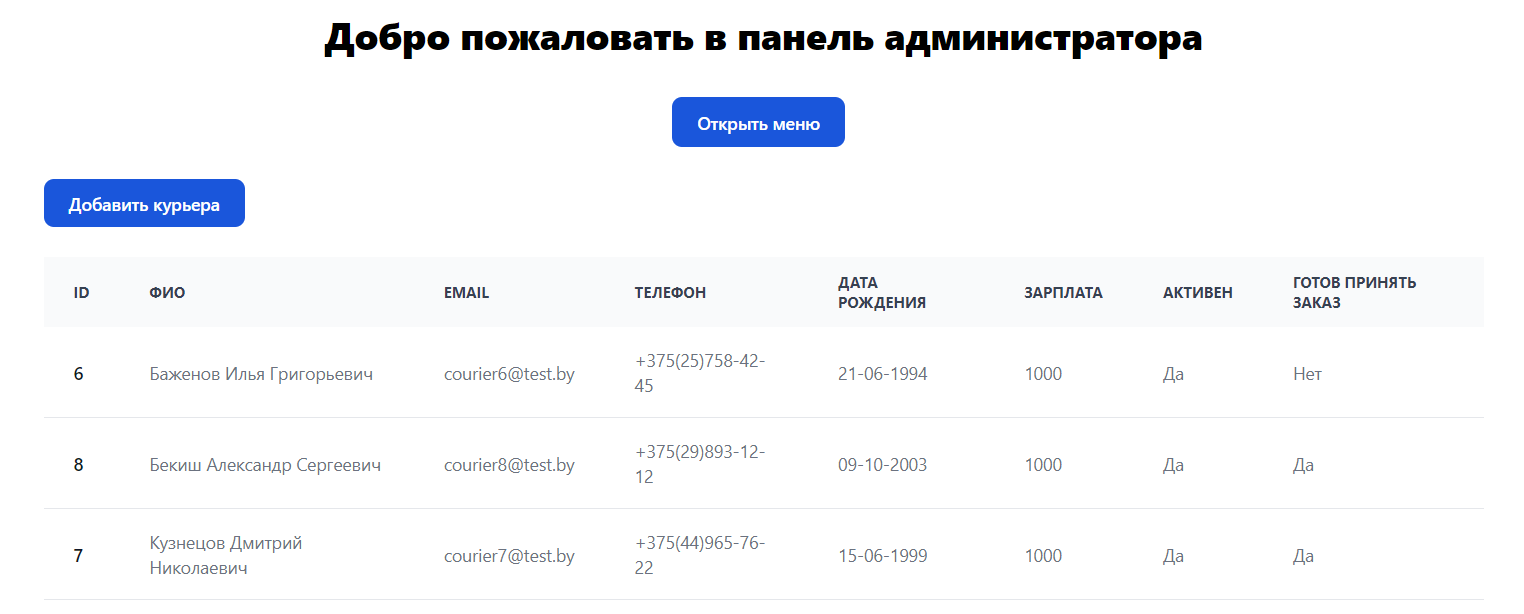


Рисунок 5.10 – Информация о курьерах на панели администратора

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.

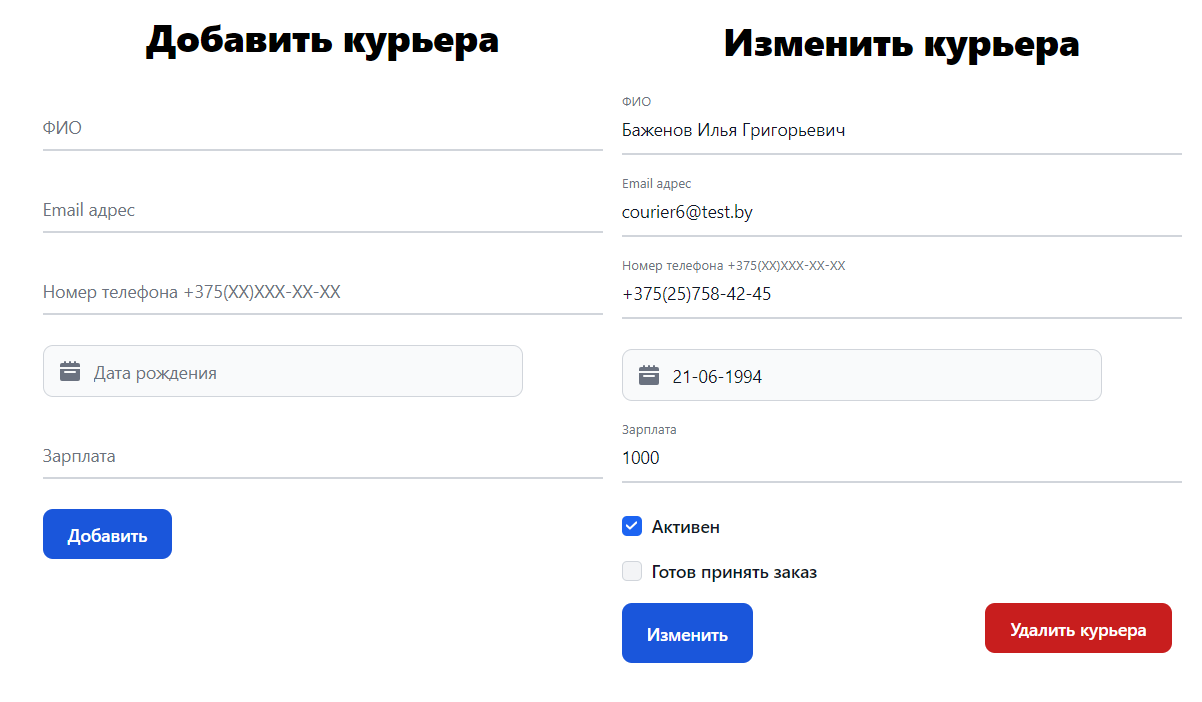


Рисунок 5.11 – Формы добавления и изменения данных курьеров

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.

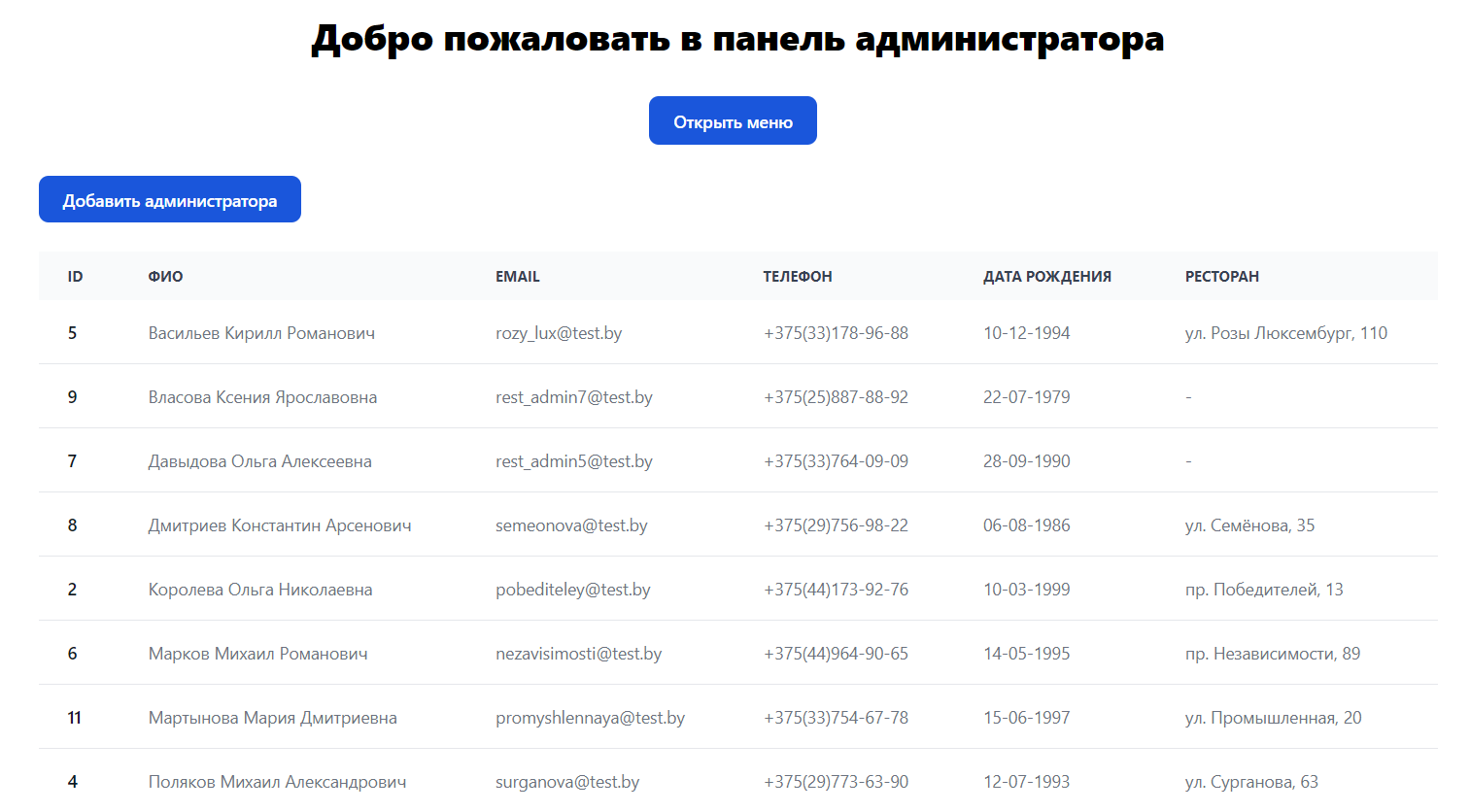


Рисунок 5.12 – Панель главного администратора

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.

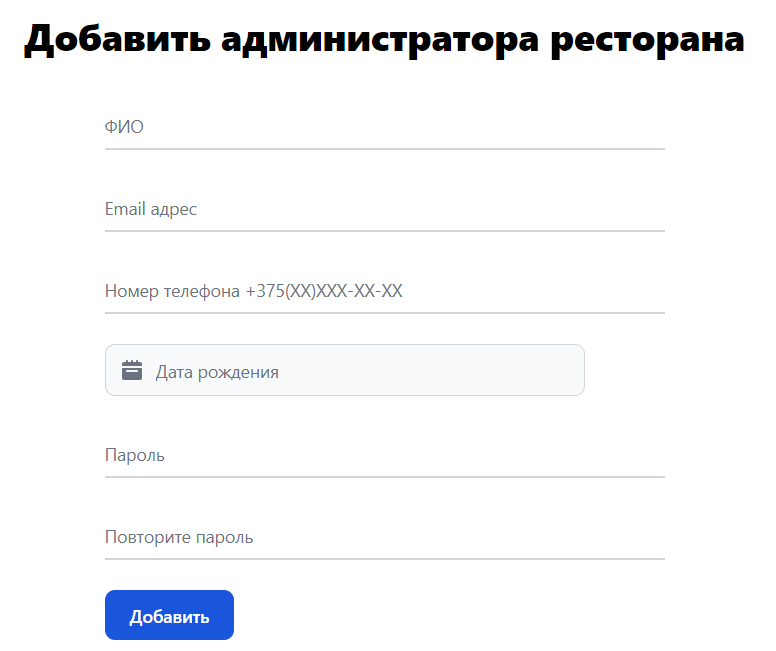


Рисунок 5.13 – Форма регистрации администратора ресторана

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

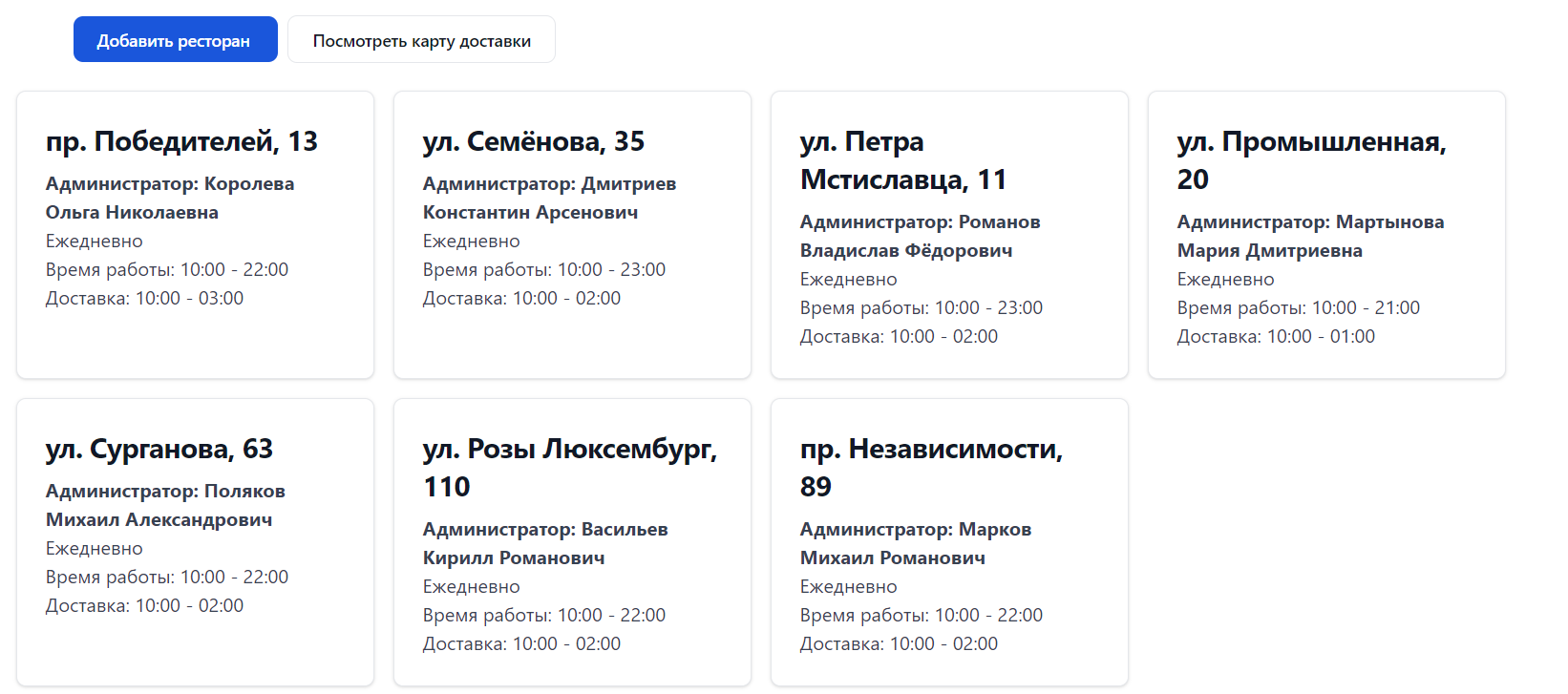


Рисунок 5.14 – Интерфейс для работы с ресторанами

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.

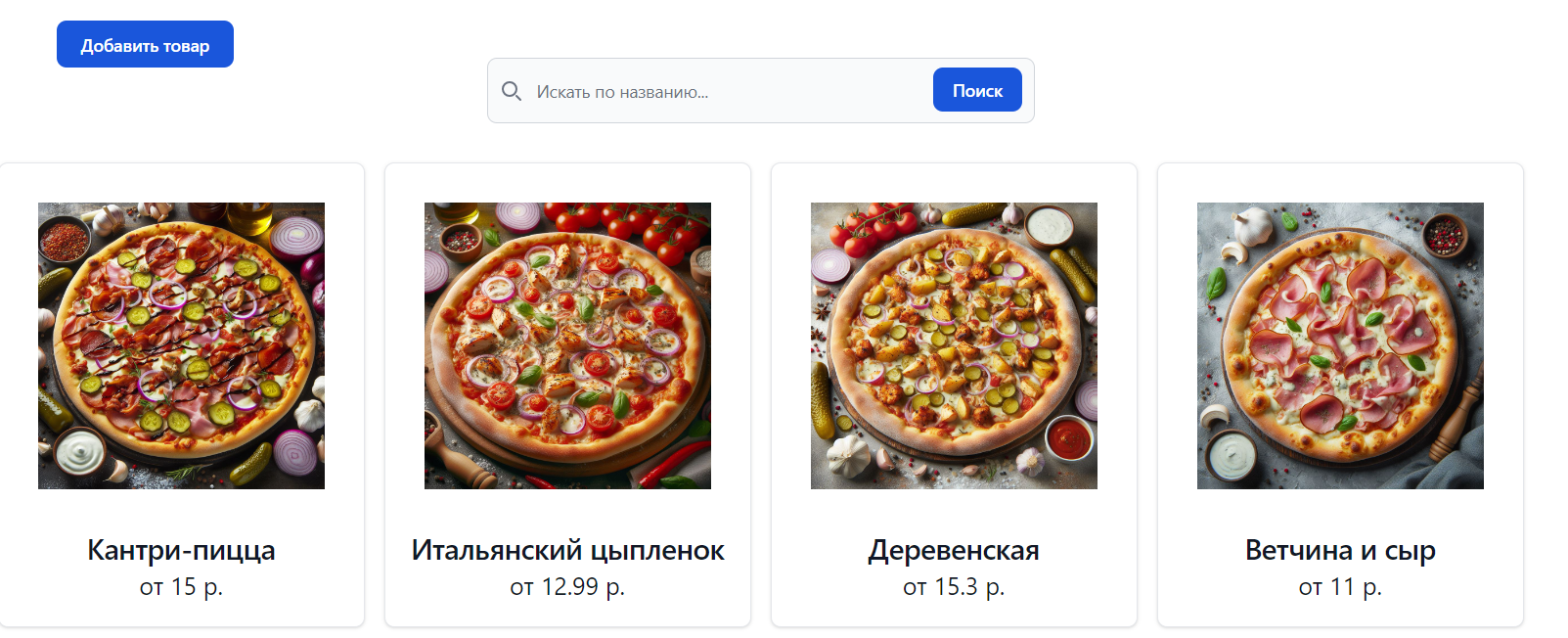


Рисунок 5.15 – Интерфейс для работы с меню

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

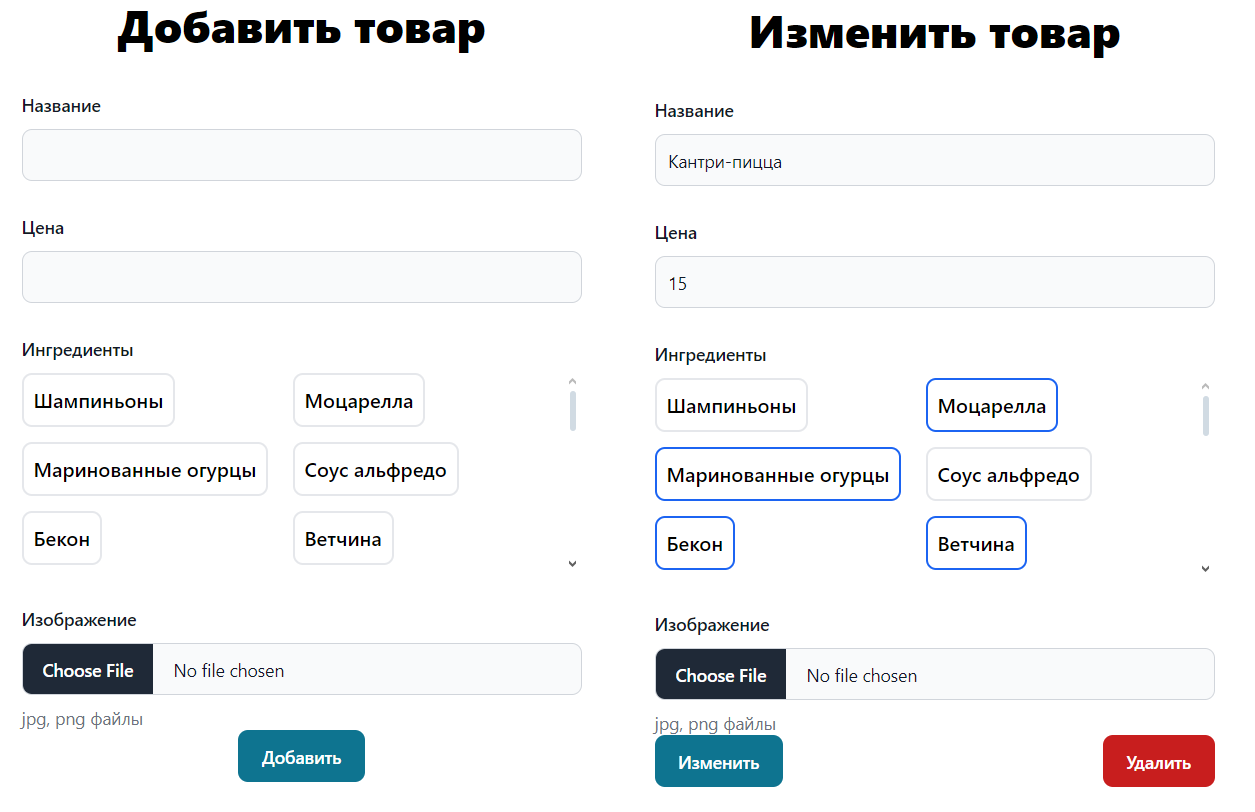


Рисунок 5.16 – Формы добавления/изменения товара

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

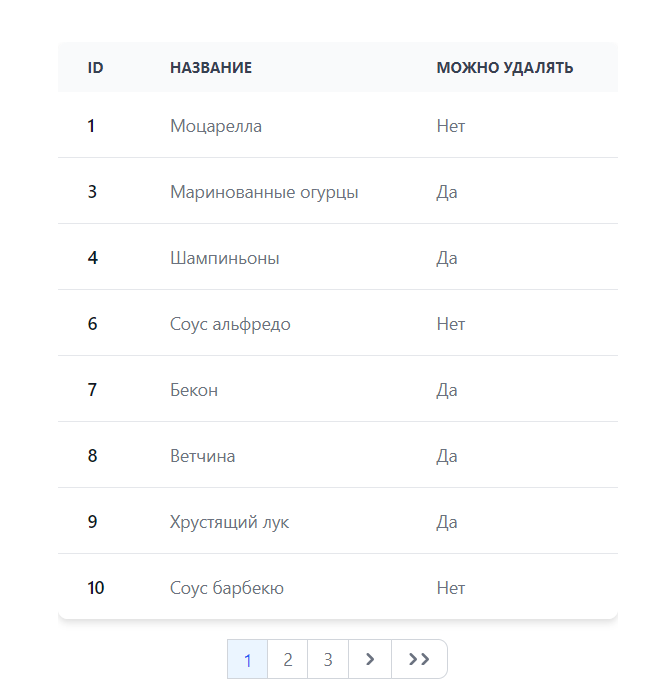


Рисунок 5.17 – Интерфейс для работы с ингредиентами блюд

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.

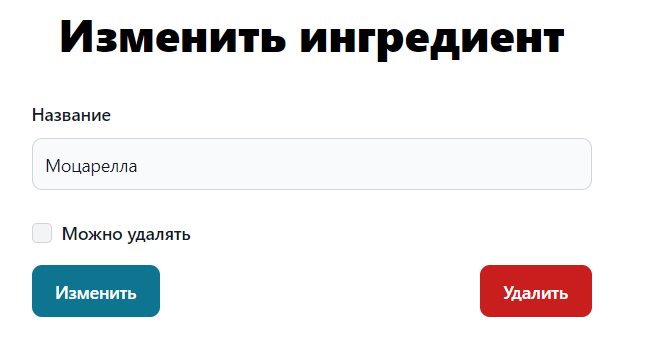


Рисунок 5.18 – Форма изменения ингредиента

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

Заключение

Результатом выполнения курсового проекта стало web-приложение, созданное с использованием серверного фреймворка Express.js, библиотеки React.js и базы данных PostgreSQL.

Перед началом разработки был произведен аналитический обзор прототипов приложений подобной тематики, а также определение функциональных возможностей разрабатываемого приложения.

Был разработан удобный и понятный интерфейс, в котором могут разобраться даже неопытные пользователи.

В процессе разработки серверной части было реализовано API для работы со всеми необходимыми сущностями, также была реализована стратегия JWT для обеспечения безопасности.

Были реализованы следующие требования:

* управление корзиной товаров;
* управление меню ресторана;
* совершение заказа;
* управление ресторанами;
* распределение заказов по ресторанам;
* поиск по наименованию товара;
* уведомление пользователя о статусе заказа;
* удаление ингредиентов пиццы;
* нахождение ближайшего ресторана для пользователя;
* управление курьерами.

Для обеспечения безопасного обмена информации между клиентом и сервером был сгенерирован ssl-сертификат.

Важным этапом разработки было тестирование программного продукта, которое позволило выявить и исправить ошибки и недостатки в работе приложения. Тестирование позволило убедиться, что программа работает верно и соответствует требованиям.

В целом, разработанное веб-приложение представляет собой важный инструмент для современного ресторанного бизнеса, способный значительно улучшить процессы управления и обслуживания, а также повысить удовлетворенность клиентов.

Список используемых источников

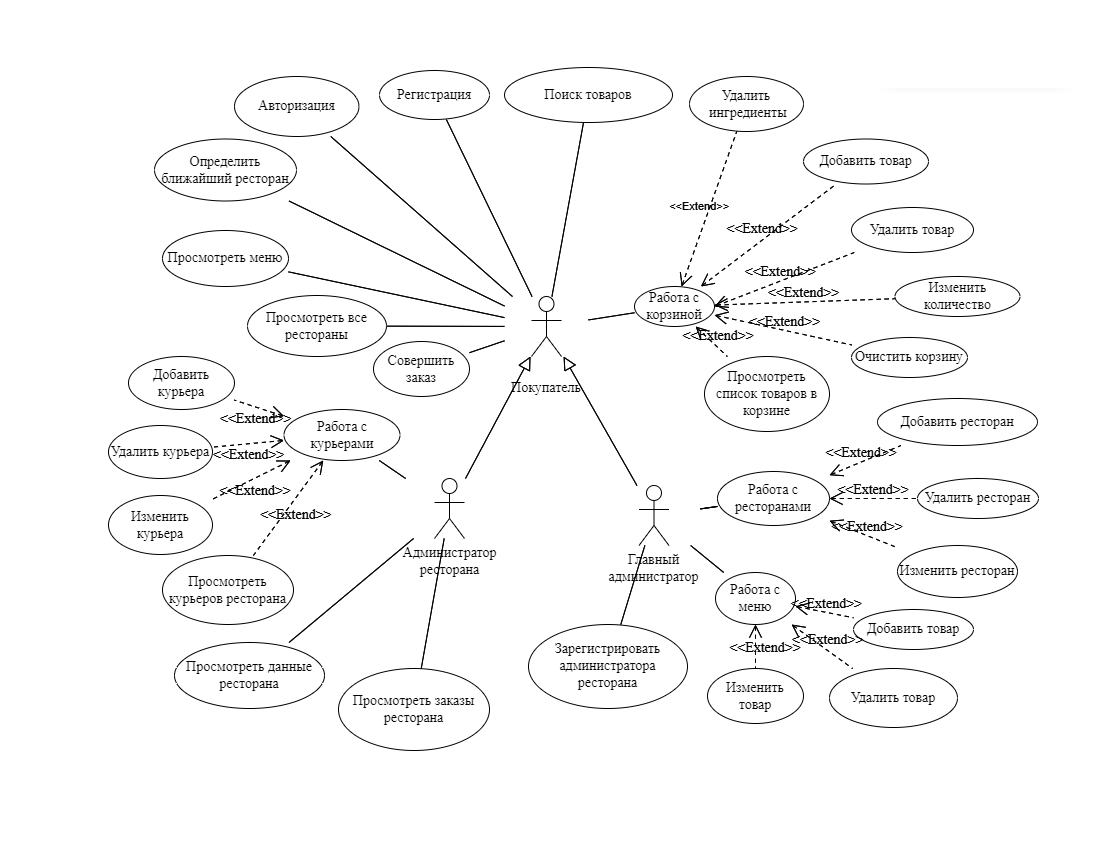
1. Express 4.x API [Электронный ресурс] / StrongLoop, Inc. – Режим доступа: https://expressjs.com/en/4x/api.html. – Дата доступа: 19.03.2024.

2. React.js документация [Электронный ресурс] / Режим доступа: https://react.dev/learn. – Дата доступа: 22.03.2024.

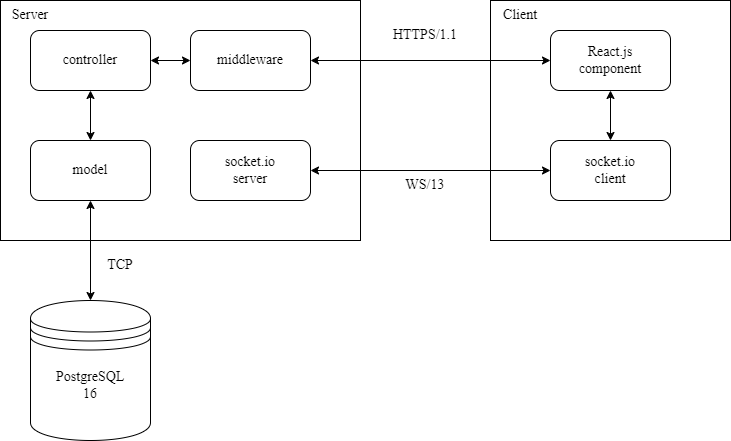
3. Socket.io документация [Электронный ресурс] / Режим доступа: https://socket.io/docs/v4/. – Дата доступа: 25.03.2024.

4. PostgreSQL документация [Электронный ресурс] / Режим доступа: https://www.postgresql.org/docs/. – Дата доступа: 15.03.2024.

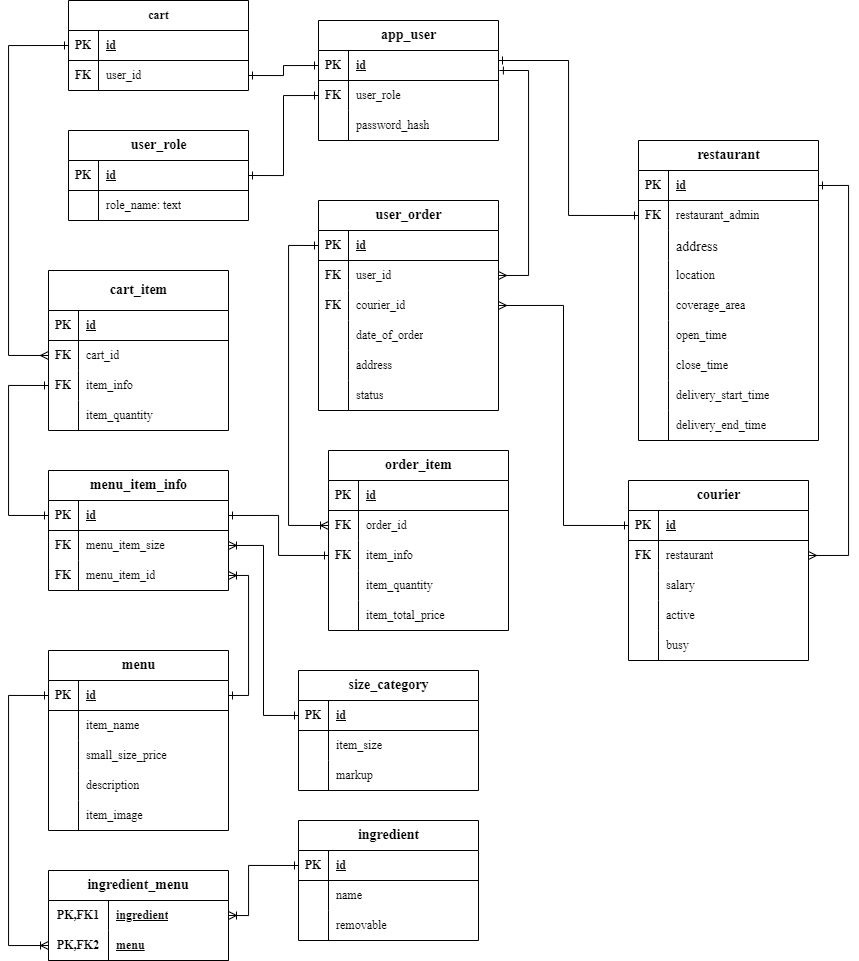
ПРИЛОЖЕНИЕ А



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")  } |