

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Информационных систем и технологий
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:**

Web-приложение «Служба доставки японской кухни»

Выполнил студент Елисеева Анастасия Денисовна
(Ф.И.О.)

Руководитель проекта асс. Нистюк О. А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Нистюк О. А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Норм контролёр асс. Нистюк О. А.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Минск 2023

Реферат

Пояснительная записка содержит 30 страниц, 22 иллюстрации, 8 источников литературы, 2 приложения.

NODEJS, EXPRESS, NEXTJS, SQL SERVER, WEBSOCKETS, CROSSPLATFORM APPLICATION, REST API

Основная цель курсового проекта: проектирование базы данных и разработка приложения службы доставки японской кухни.

Пояснительная записка состоит из введения, пяти разделов, заключения и списка используемой литературы.

В введении формулирована цель и задачи проекта.

В первом разделе проводится постановка задачи, обзор аналогичных решений.

Второй раздел посвящен проектированию базы данных и веб-приложения.

В третьем разделе описаны детали реализации серверной и клиентской частей приложения.

В четвертом разделе описаны основные шаги тестирования проекта для разных компонентов.

В пятом разделе описано руководство пользователя, позволяющее понять принцип взаимодействия с интерфейсом клиентского приложения.

В заключении представлены итоги курсового проекта и задачи, которые были решены в ходе разработки проекта.

Содержание

1 Аналитический обзор литературы	5
1.1 Аналитический обзор источников	5
1.2 Обзор аналогов.....	5
2 Проектирование web-приложения	7
2.1 Диаграмма использования	7
2.2 Диаграмма развертывания	9
2.3 Проектирование базы данных	10
2.4 Идентификация и авторизация.....	13
3 Разработка программного средства	14
3.1 Разработка модели базы данных.....	14
3.2 Разработка серверной части	15
3.3 Разработка клиентской части	18
4 Тестирование приложения.....	21
5 Руководство пользователя	24
Заключение	29
Список литературы.....	30
Приложение А.....	31
Приложение Б.....	32

Введение

Целью данной работы является разработка кроссплатформенного веб-приложения на тему «Служба доставки японской кухни». Приложение представляет из себя сервис, на котором предоставляется удобный функционал для заказа готовой еды, которую доставят курьеры. Приложение должно быть разработано с помощью языка JavaScript и программной платформы Node.js реализуя клиент-серверную архитектуру.

Node.js – программная платформа, основанная на движке Chrome V8. Node.js асинхронен и событийно-ориентирован [1]. Данная платформа позволяет JavaScript взаимодействовать с устройствами ввода-вывода через свой API, подключать другие внешние библиотеки. Node.js предназначен для построения масштабируемых сетевых приложений, преимущественно серверов. Npm – менеджер пакетов, входящий в состав Node.js, позволяющий расширить возможности приложения.

Для серверной части приложения был выбран фреймворк Express.js. Он представляет собой слой фундаментальных функций для упрощения разработки веб-приложений.

Для клиентской части приложения были выбраны React.Js и Next.js. React.Js – JavaScript библиотека с открытым исходным кодом для разработки пользовательских интерфейсов [7]. Next.js – открытый JavaScript фреймворк, созданный поверх React.js для создания веб-приложений [5].

В качестве СУБД для базы данных была выбрана Microsoft SQL Server. Для для реализации ORM в Node.js используется Prisma.

Данный курсовой проект должен выполнять следующие требования:

- обеспечивать возможность регистрации и авторизации;
- поддерживать роли администратора, курьера и пользователя;
- позволять изменять информацию о товарах;
- отправлять подтверждающие письма на электронную почту;
- позволять отправлять отзывы к товару;
- предоставлять возможность оформлять заказы;
- предоставлять возможность удаления и добавления новых товаров;
- поддерживать обработку оплаты заказа;
- предоставлять возможность поиска товара по названию.

Пояснительная записка содержит краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя, а также информацию о тестировании приложения.

1 Аналитический обзор литературы

1.1 Аналитический обзор источников

В ходе разработки приложения была изучена специальная техническая, учебно-методическая и справочная литература, статьи и материалы, опубликованные в сети интернет.

Основная информация о работе с фреймворками и библиотеками была получена из их официальных документаций.

1.2 Обзор аналогов

На сегодняшний день существует множество подобных приложений, однако функционал их шире. Далее я попытаюсь рассмотреть приложения, которые максимально полно соответствуют моим задачам.

Основными компонентами таких web-платформ являются: клиентская часть приложения – что видит пользователь при посещении сайта; серверная часть – то, что управляет бизнес-логикой всего приложения. В открытом доступе мы можем рассмотреть только клиентскую часть приложений. Серверная часть обычно скрыта от обычного пользователя.

Одним из таких сайтов является «Artfood.by». Сайт разделен на категории, что делает навигацию по сайту очень простой.

Рассмотрим в качестве примера каталог товаров данного сайта на рисунке 1.1.

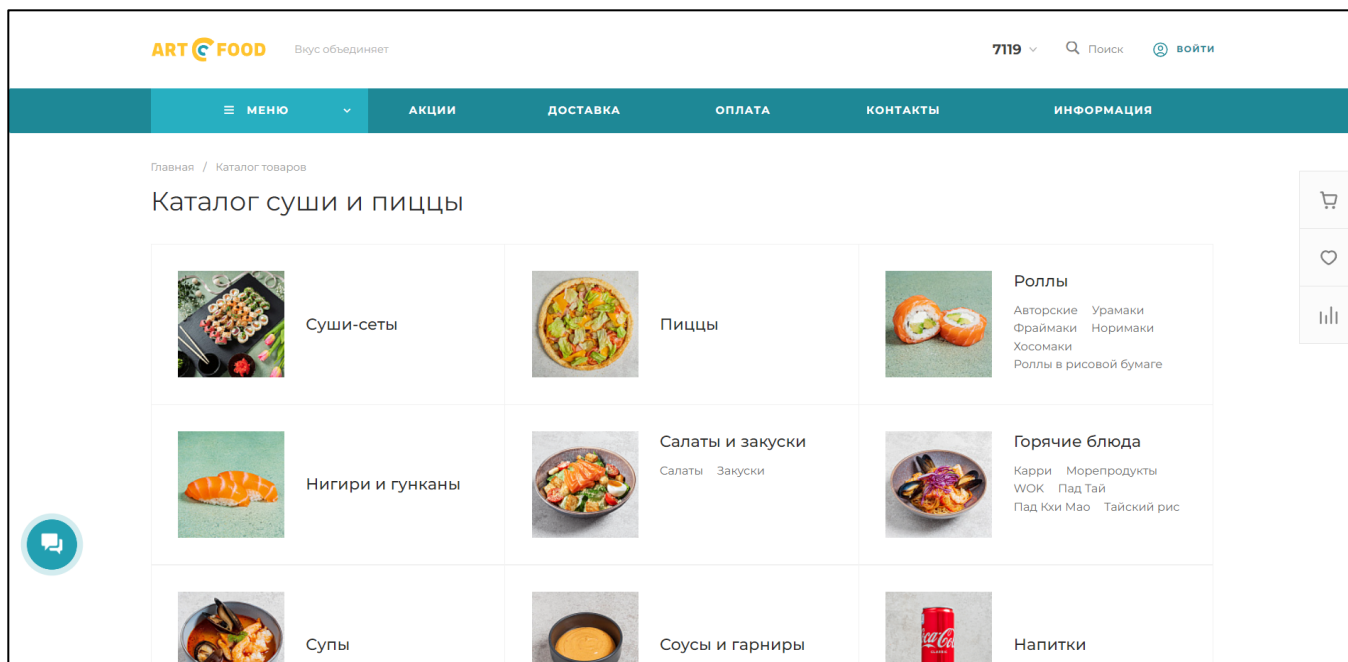


Рисунок 1.1 – Пример каталога «Artfood.by»

У каждого товара есть следующие особенности:

- Цена;
- Название;

- Рейтинг;
- Характеристики;
- Отзывы;
- Возможность добавления в корзину.

Так же рассмотрим интернет-магазин «sushihouse.by», одна из страниц из которого представлена на рисунке 1.2.

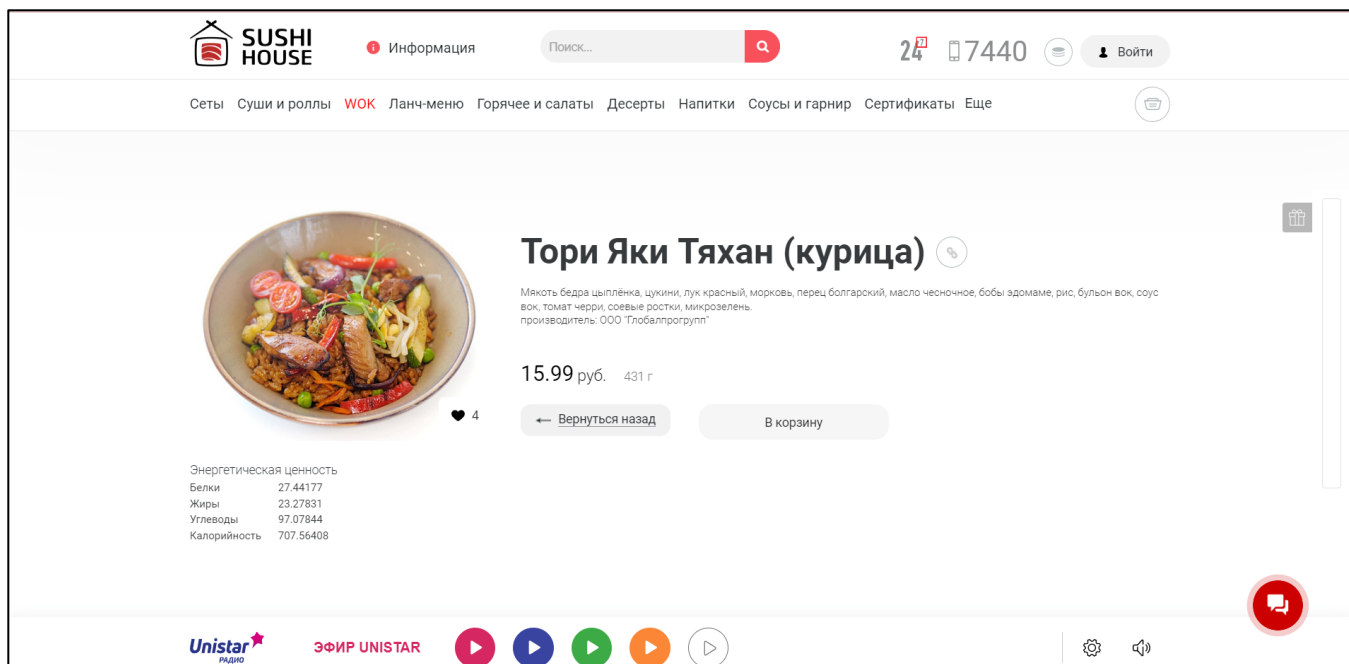


Рисунок 1.2 – Пример страницы товара «sushihouse.by»

Его основные особенности такие же, как и у «Artfood.by», кроме отсутствия отзывов на блюда.

Оба интернет-магазина предоставляют возможность оформления заказов. Кроме того, пользователь может выбрать удобный для него способ оплаты.

Проанализировав приложения на схожую тематику, были разработаны технические требования к разработке веб-приложения. Приложение должно:

- реализовывать валидацию вводимых данных;
 - иметь интуитивно понятное и простое управление;
 - обращаться к базе данных, успешно извлекать, изменять и дополнять данные, осуществлять поиск по базе и автоматическую запись некоторых полей;
 - иметь различный функционал в зависимости от учётной записи;
 - выглядеть аккуратно, не иметь лишней информации.
- Эти пункты и составляют техническое задание разрабатываемого проекта.

2 Проектирование web-приложения

В данной главе описываются принципы проектирования web-приложения. При разработке курсового проекта весь процесс был разбит на 3 составляющие: база данных, сервер, клиент.

2.1 Диаграмма использования

Приложение должно иметь разделение ролей. Для каждой роли были разработаны UML-диаграммы вариантов использования.

Диаграмма UML – это графическое представление набора элементов, изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

При открытии сайта пользователь не авторизован. Диаграмма вариантов использования неавторизованного пользователя отображена на рисунке 2.1.

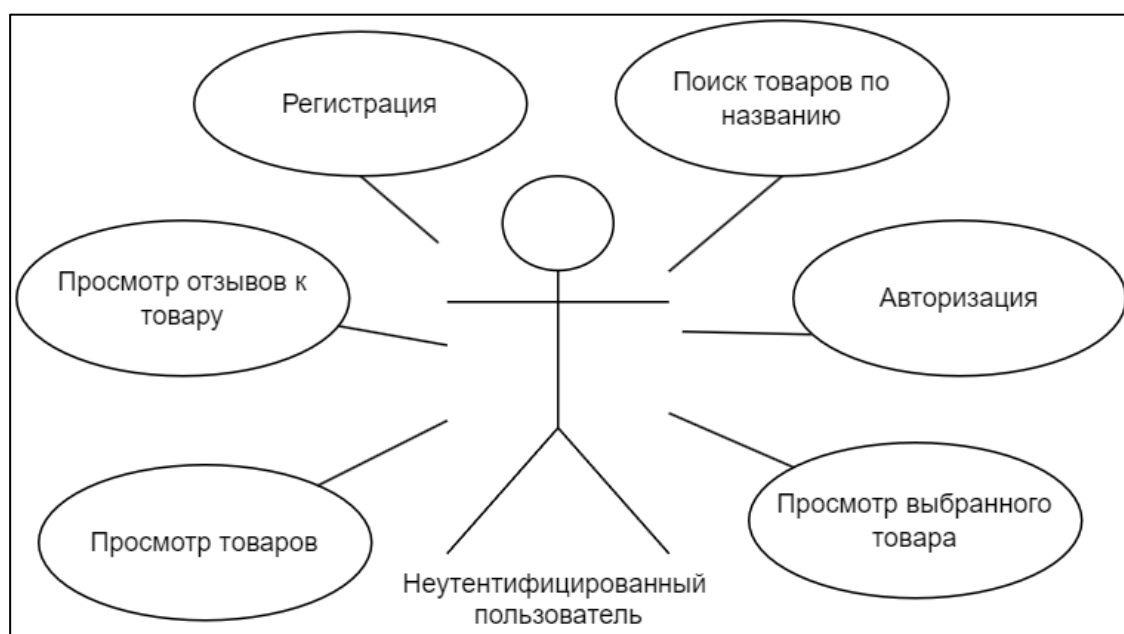


Рисунок 2.1 – Диаграмма использования неавторизованного пользователя

Как видно из диаграммы, неавторизованный пользователь может просматривать перечень всех товаров, искать товар по названию. На части сайта, которая отвечает за конкретный товар, неавторизованный пользователь может просматривать его информацию и отзывы к нему, а для дальнейших действий необходимо зарегистрироваться, либо авторизоваться. При попытке заказа товара, пользователю будет отказано в доступе.

При попытке зайти на страницу, требующую авторизации (например, путём явного ввода url), будет переводиться на страницу авторизации.

Роль каждой учётной записи определена в базе данных. Создать администратора или курьера на сайте невозможно. Итак, зарегистрировавшись и войдя в аккаунт, пользователь получает роль авторизованного пользователя.

2.2. Диаграмма вариантов использования для этой роли представлена на рисунке



Рисунок 2.2 – Диаграмма использования авторизованного пользователя

В приложении есть роль администратора. Для этой роли диаграмма вариантов использования отображена на рисунке 2.3.

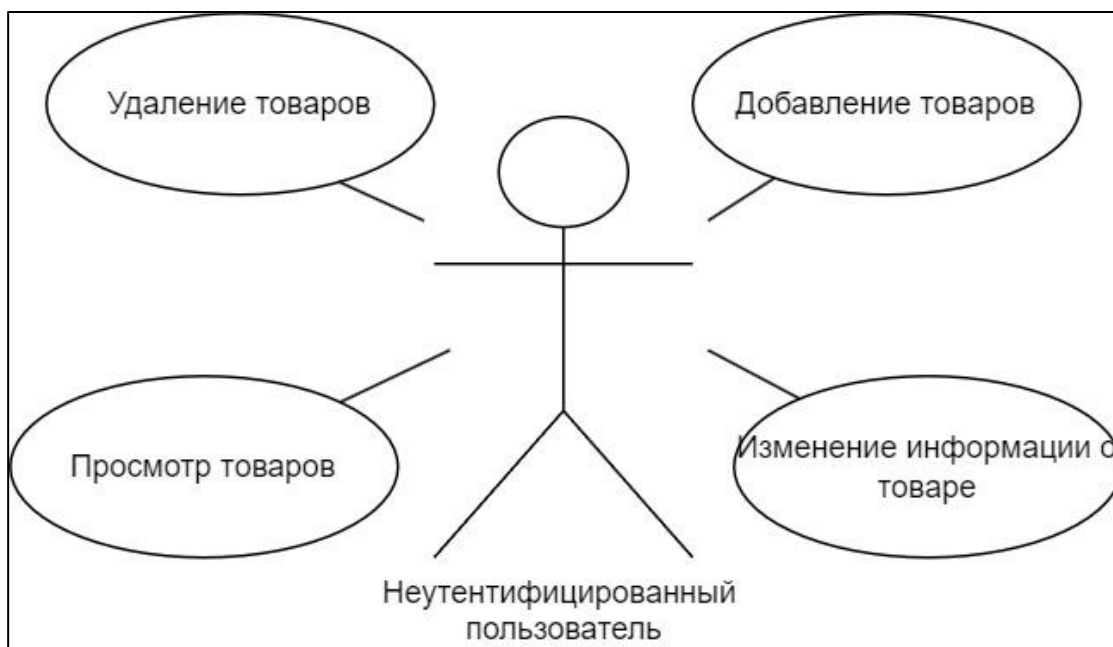


Рисунок 2.3 – Диаграмма использования для администратора

Так же в приложении есть роль курьера, который может просматривать заказы для осуществления доставки. Для удобства он может их сортировать по различным признакам. На странице отдельного товара курьер может поменять статус заказа.

На основе данных UML-диаграмм и словесного описания функциональных требований, происходит построение всей бизнес логики программы. Бизнес логика определяет, как должны выполняться определённые операции и взаимодействия внутри системы.

2.2 Диаграмма развертывания

Диаграмма развертывания (синоним – диаграмма размещения). Она применяется для представления общей конфигурации и топологии распределенной программной системы и содержит распределение компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений–маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы. Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. На рисунке 2.4 предложена схема приложения.

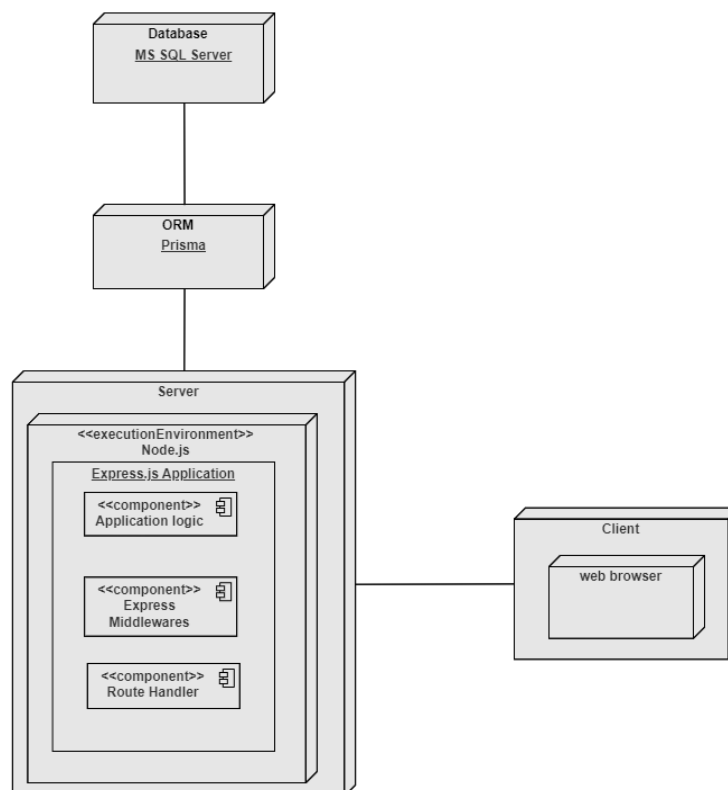


Рисунок 2.4 – Диаграмма развёртывания

При разработке сервера приложения была выбрана архитектура Web Api. Web Api представляет собой один из самых популярных подходов к построению архитектуры web-приложений с передачей данных по протоколу HTTPS.

Основными преимуществами Web API являются:

- гибкость и масштабируемость;
- универсальность;
- безопасность;
- простота разработки и тестирования;
- низкие затраты на обслуживание.

Для проектирования приложения используется платформа NodeJS, для хранения данных выбрана объектно-реляционная система управления базами данных «MS SQL Server». Для взаимодействия используется ORM, такой подход уменьшит количество работы при смене базы данных.

Выбор данного системного продукта произошел по нескольким очевидным причинам: легкая установка, дизайн оболочки IDE SQL Management Studio, бесплатна для разработки, ранее полученные знания по использованию данной базы данных.

2.3 Проектирование базы данных

Одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие – составление такой модели данных, которая будет правильно отражать то, как с этими данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Схема базы данных для разрабатываемого программного средства отображена в приложении А. Рассмотрим её подробнее.

Схема отображает таблицы базы данных и связи между ними.

Как видно из схемы, в приложении используются 8 таблиц: Products, Users, Comments, Orders, OrderDetails, Address, Basket, BasketProduct, Token.

Таблица Products содержит данные о каждом товаре. Перечень полей таблицы Products приведен в таблице 2.1

Таблица 2.1 – Описание полей таблицы Products

Поле	Типы данных
ProductID	int
ProductName	varchar(50)
ProductWeight	int
ProductShortDesc	nvarchar(200)
ProductImage	nvarchar(1000)
ProductPrice	float

В столбце ProductName хранится название продукта, ProductWeight – его вес, ProductShortDesc – краткое описание товара, ProductImage – название файла с изображением продукта, ProductPrice – цена продукта.

Таблица Comments хранит в себе информацию об оставленных отзывах. Перечень полей таблицы Comments приведен в таблице 2.2

Таблица 2.2 – Описание полей таблицы Comments

Поле	Типы данных
CommentsID	int
CommentsUserID	int
CommentsProductID	int
Comment	varchar(1000)

В столбце CommentUserID хранится информация о идентификаторе пользователя, который оставил отзыв, CommentProductID – идентификатор товара, Comment – текст комментария.

Таблица Users, хранит личные данные каждого пользователя. Перечень полей таблицы Users приведен в таблице 2.3.

Таблица 2.3 – Описание полей таблицы Users

Поле	Типы данных
UserID	int
UserEmail	varchar(100)
UserPassword	varchar(270)
Role	tinyint
isActivated	bit
ActivationLink	varchar(250)

Столбец UserEmail хранит информацию о почте зарегистрированного пользователя, UserPassword – пароль пользователя, Role – роль пользователя. В столбце isActivated – информация о том, активирован ли аккаунт пользователя. ActivationLink – ссылка для активации аккаунта.

Таблица Orders хранит информацию о заказах, сделанных пользователем. Перечень полей таблицы Orders приведен в таблице 2.4.

Таблица 2.4 – Описание полей таблицы Orders

Поле	Типы данных
OrderID	int
OrderUserID	int
OrderAddressID	int
OrderDate	datetime
OrderAmount	float
OrderStatus	tinyint

В столбце OrderUserID хранится информация о пользователе, который сделал заказ, столбец OrderAddressID – внешний ключ на таблицу Address. OrderDate хранит дату заказа, OrderAmount – общая стоимость заказа, OrderStatus – статус заказа.

Таблица OrderDetails хранит данные о том, какие товары и в каком количестве заказал пользователь. Перечень полей таблицы OrderDetails приведен в таблице 2.5.

Таблица 2.5 – Описание полей таблицы OrderDetails

Поле	Типы данных
DetailsID	int
DetailsOrderID	int
DetailsProductID	int
Quantity	int

В столбце DetailsOrderID хранится внешний ключ на таблицу Orders, в столбце DetailsProductID – идентификатор продукта, добавленного в заказ. В Quantity указывается сколько единиц данного товара заказал пользователь.

Таблица Address хранит адрес для доставки заказа. Перечень полей таблицы Address приведен в таблице 2.6.

Таблица 2.6 – Описание полей таблицы Address

Поле	Типы данных
AddressID	int
Country	varchar(20)
City	varchar(50)
Address	varchar(50)

Столбец AddressID хранит идентификатор адреса, Country – страну, City – город. В столбце Address хранится более точное местоположение (улица, дом, корпус и т.п.)

Таблица Basket хранит корзину пользователя. Перечень полей таблицы Basket приведен в таблице 2.7.

Таблица 2.7 – Описание полей таблицы Basket

Поле	Типы данных
BasketID	int
UserID	int
BasketAmount	float

Столбец BasketID хранит идентификатор корзины, UserID – внешний ключ на таблицу Users, BasketAmount – общая стоимость товаров в корзине.

Таблица BasketProduct хранит данные о том, какие товары и в каком количестве пользователь добавил в корзину. Перечень полей таблицы BasketProduct приведен в таблице 2.8.

Таблица 2.8 – Описание полей таблицы BasketProduct

Поле	Типы данных
BasketProductID	int
ProductID	int
BasketID	int
Quantity	int

В столбце BasketID хранится внешний ключ на таблицу Basket, в столбце ProductID – идентификатор продукта. В Quantity указывается сколько единиц данного товара пользователь добавил в корзину.

Таблица Token хранит информацию о jwt токенах. Перечень полей таблицы Token приведен в таблице 2.9.

Таблица 2.9 – Описание полей таблицы Token

Поле	Типы данных
TokenID	int
UserID	int
refreshToken	nvarchar(350)

В столбце TokenID хранится идентификатор токена, UserID – идентификатор пользователя, которому принадлежит этот токен, refreshToken – токен.

2.4 Идентификация и авторизация

Для выполнения задач при разработке приложения, существует необходимость регистрации, авторизации и идентификации, для этой задачи используется JWT.

JWT (JSON Web Token) - это открытый стандарт (RFC 7519) для передачи информации в формате JSON между двумя сторонами. Он используется для авторизации и аутентификации пользователей в приложениях и веб-сервисах.

JWT состоит из трех частей: заголовка (header), полезной нагрузки (payload) и подписи (signature). Заголовок содержит тип токена и используемый алгоритм шифрования, полезная нагрузка - информацию о пользователе, например, идентификатор пользователя и его роли, а подпись - защищает токен от подмены данных и обеспечивает его подлинность.

Когда пользователь входит в систему, сервер генерирует JWT и отправляет его обратно клиенту. Клиент сохраняет токен и использует его при каждом запросе к серверу, чтобы доказать свою легитимность и получить доступ к защищенным ресурсам. Сервер проверяет токен и, если он действителен, разрешает доступ к защищенным ресурсам.

JWT имеет ряд преимуществ, таких как легкость в использовании, возможность передавать информацию в безопасном формате, масштабируемость и поддержку стандартов.

3 Разработка программного средства

При разработке курсового проекта весь процесс был разбит на следующие три этапа:

- разработка модели базы данных;
- разработка серверной части приложения;
- разработка клиентской части приложения.

3.1 Разработка модели базы данных

Проектирование модели базы данных – это важный этап в разработке информационной системы. Одним из подходов к проектированию модели базы данных является объектно-реляционное отображение (ORM), который используется для связывания объектной модели приложения с реляционной моделью базы данных.

ORM позволяет разработчикам использовать объектно-ориентированный подход при работе с базой данных. Это позволяет создавать более гибкие и удобные приложения, поскольку объектно-ориентированный подход более естественен для программистов, чем работа с реляционными таблицами и SQL-запросами.

Для реализации ORM были рассмотрены различные инструменты, и выбор был сделан в пользу Prisma. Prisma – это ORM для Node.js и TypeScript, который обеспечивает быстрый и безопасный доступ к базе данных [2]. Он предоставляет мощные инструменты для создания и управления базами данных, а также облегчает разработку приложений, использующих базы данных.

Одним из преимуществ использования Prisma является его автоматическая генерация кода на основе модели базы данных. Это позволяет быстро и легко создавать новые таблицы и поля в базе данных, а также выполнять различные операции с базой данных. Сгенерированный код на основе модели базы данных называется схемой, который представлен в виде листинга в приложении Б.

Пример кода запроса prisma представлен в листинге 3.1.

```
let bsktID = basket.BasketID;
const basketExist = await conn.basketproduct.findFirst({
  where: {
    ProductID: Number.parseInt(productID),
    BasketID: Number.parseInt(bsktID),
  },
});
```

Листинг 3.1 – Пример запроса Prisma

Данный код использует библиотеку Prisma для работы с базой данных и выполняет операцию поиска в таблице basketproduct.

В первой строке кода объявляется константа PrismaClient, которая представляет собой класс из библиотеки @prisma/client и используется для установления соединения с базой данных.

Во второй строке создается экземпляр класса `PrismaClient` и сохраняется в переменную `conn`. Этот экземпляр будет использоваться для выполнения операций с базой данных.

Затем в коде определяется переменная `bsktID`, которая присваивается значению `BasketID` из объекта `basket`. Далее выполняется операция поиска в таблице `basketproduct` с использованием метода `findFirst` объекта `conn`.

Метод `findFirst` принимает объект с условиями поиска, указанными в параметре `where`. В данном случае, условия поиска указывают, что значение поля `ProductID` должно быть равно числовому значению `productID`, а значение поля `BasketID` должно быть равно числовому значению `bsktID`.

Операция поиска выполняется асинхронно с использованием ключевого слова `await`, что позволяет программе ожидать завершения операции поиска, прежде чем продолжить выполнение следующих инструкций.

Результат операции поиска сохраняется в переменной `basketExist`, которая содержит информацию о найденной записи в таблице `basketproduct`.

3.2 Разработка серверной части

В качестве основы серверной части был взят REST API. REST является архитектурным стилем, который определяет правила и принципы для создания распределенных систем. REST API позволяет взаимодействовать с сервером посредством стандартных HTTP-методов, таких как GET, POST, PUT и DELETE, для выполнения операций чтения, создания, обновления и удаления данных.

Разработка серверной части на основе REST API позволяет создавать гибкие и масштабируемые приложения, которые могут обрабатывать запросы от различных клиентов, таких как веб-браузеры, мобильные приложения и другие сервисы. Это открывает широкий спектр возможностей для разработчиков, позволяя создавать высокопроизводительные и многофункциональные приложения.

Входной точкой сервера является файл `server.js`, в ней инициализируется и запускается сервер. Данный код представлен в листинге 3.2.

```
const start = async () => {
  try {
    https
      .createServer(
        app
      )
      .listen(PORT, () => {
        console.log(`start https on ${PORT}`);
      });
  } catch (e) {
    console.log(e);
  }
};
```

Листинг 3.2 – Код запуска сервера

Сервисы представляют собой модули, которые содержат логику и функциональность, связанную с конкретной областью приложения. Они обычно отвечают за выполнение бизнес-логики, взаимодействие с базой данных и другими сторонними сервисами, а также предоставление необходимых данных контроллерам. Сервисы в Express обеспечивают модульность и отделение ответственности, что способствует поддержке кодовой базы и упрощает тестирование и сопровождение приложения.

Пример рабочего сервиса представлен в листинге 3.3.

```
class BasketService {
  async userBasket(userID) {
    const currentBasket = await conn.basket.findFirst({
      where: {
        UserID: Number.parseInt(userID),
      },
    });
    return currentBasket;
  }
  async recountPrice(userID) {
    const currentBasket = await conn.basket.findFirst({
      where: {
        UserID: Number.parseInt(userID),
      },
      include: {
        basketproduct: {
          include: {
            products: {
              select: {
                ProductPrice: true,
              },
            },
          },
        },
      },
    });
    let newPrice = 0;
    if (currentBasket.basketproduct != null) {
      let i;
      for (i = 0; i < currentBasket.basketproduct.length; i++) {
        newPrice +=
          currentBasket.basketproduct[i].products.ProductPrice *
          currentBasket.basketproduct[i].Quantity;
      }
    }
  }
}
```

Листинг 3.3 – Код сервиса для работы с корзиной

Данный сервис используется для получения содержания корзины по идентификатору пользователя и для подсчета стоимости товаров внутри неё.

Роутеры в Express являются компонентами, отвечающими за маршрутизацию входящих HTTP-запросов. Роутеры определяют, какие обработчики (handlers) должны быть вызваны для определенных маршрутов и HTTP-методов. Они предоставляют гибкую систему для организации маршрутов приложения и позволяют логически группировать и управлять различными обработчиками запросов. Роутеры также могут быть вложенными, что позволяет создавать иерархическую структуру маршрутов для удобства организации и масштабируемости приложения.

Код роутера для корзины представлен в листинге 3.4.

```
const Router = require("express");
const router = new Router();
const basketController = require("../controllers/basketController");

router.post("/", basketController.addProductToBasket);
router.get("/", basketController.getProductsInBasket);
router.delete("/", basketController.removeItemFromBasket);
router.post("/basketToOrder", basketController.CreateOrder);
router.get("/get", basketController.GetBasket);
module.exports = router;
```

Листинг 3.4 – Код роутера для работы с корзиной

Данный роутер используется для установления путей обработки запросов. Запрос обрабатывают методы контроллера. Контроллеры являются промежуточным звеном между роутерами и сервисами. Они представляют собой функции или методы, которые принимают запросы и возвращают ответы. Контроллеры обычно вызывают соответствующие сервисы для выполнения бизнес-логики и обработки данных. Они также могут быть ответственными за валидацию данных, обработку ошибок и форматирование ответов. Контроллеры в Express помогают структурировать логику обработки запросов и обеспечивают чистоту и модульность кода. Часть кода контроллера для работы с корзиной представлен в листинге 3.5.

```
class basketController {
  async addProductToBasket(req, res) {
    try {
      let { productID } = req.body;
      let userID = req.cookies.userID;
      const basket = await basketService.userBasket(userID);
      if (basket == null) {
        const newBasket = await conn.basket.create({
          data: {
            UserID: Number.parseInt(userID),
            BasketAmount: 0,
          },
        });
      }
    }
  }
}
...

```

Листинг 3.5 – Часть кода контроллера для работы с корзиной

3.3 Разработка клиентской части

Так как проект является кроссплатформенным приложением, клиентская часть была разработана для использования в браузере. Для разработки клиентской части был выбран фреймворк Next.js, который в себе имеет библиотеку React.js. Next.js – сильный и гибкий фреймворк. Основным его плюсом является наличие рендера страниц на стороне сервера. Это означает то, что при выходе приложения из стадии разработки оно собирается. Основные компоненты разработанные с использованием библиотеки React.js кэшируются и собранному приложению нужно получать только данные, которые будут использоваться в компонентах интерфейса.

React.js помогает облегчить разработку пользовательского интерфейса за счет многократного использования ранее разработанных разработчиков компонентов, которые могут быть использованы в любой части приложения. Так же библиотека React.js предоставляет такой функционал как React Hooks. React Hooks позволяют использовать ранее разработанную логику обработки данных в функциональных компонентах.

Чтобы клиент не видел задержки, при запросе у сервера необходимых данных, клиент запрашивает данные асинхронно, с помощью promise-ориентированного fetch [6]. Пример реализации fetch-запроса представлен на листинге 3.6.

```
export const getServerSideProps = async () => {
  const response = await fetch(
    `${process.env.NEXT_PUBLIC_BACK_DOMAIN}/v1/products/getPag`,
    {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        offset: 0,
        take: 20,
      }),
    }
  );
  const content = await response.json();
  return {
    props: { data: content },
  };
};
```

Листинг 3.6 – Пример реализации fetch-запроса

Стоит обратить внимание на метод `getServerSideProps`. Это один из методов, специфичных для фреймворка Next.js, который позволяет ускорить получение данных при первой загрузке страницы.

Форматирование страниц начинается с корневой функции приложения, называемой `App`.

Код функции App представлен в листинге 3.7.

```
import { AuthProvider } from "@lib/auth/AuthContext";

export default function App({ Component, pageProps }) {
  return (
    <AuthProvider>
      <Component {...pageProps} />
    </AuthProvider>
  );
}
```

Листинг 3.7 – Код функции App

Функция с именем "App" представляет собой компонент React, который принимает два параметра: "Component" и "pageProps". Компонент React – это переиспользуемая единица кода, которая отвечает за отображение пользовательского интерфейса и его поведение. Параметр "Component" представляет собой другой компонент, который будет отображаться внутри данного компонента "App". Параметр "pageProps" содержит свойства страницы, которые будут переданы в компонент "Component" для его корректного отображения.

Внутри функции "App" применяется компонент "AuthProvider". "AuthProvider" является компонентом, который предоставляет контекст аутентификации для приложения. Контекст предоставляет способ передачи данных через иерархию компонентов без явной передачи пропсов через каждый промежуточный компонент. В данном случае, компонент "AuthProvider" обеспечивает функциональность аутентификации пользователей и предоставляет доступ к данным аутентификации для всех вложенных компонентов.

Компонент "Component" исходного кода передается внутрь компонента "AuthProvider" с помощью оператора spread (`{...pageProps}`). Это позволяет передать все свойства объекта "pageProps" в компонент "Component" в виде отдельных пропсов. Таким образом, компонент "Component" будет иметь доступ к переданным свойствам страницы и сможет использовать их для своей работы.

В итоге, данная функция компонента "App" выполняет важную роль в приложении, обеспечивая контекст аутентификации и передавая свойства страницы во вложенный компонент "Component". Это позволяет создать централизованный механизм аутентификации и обеспечить доступ к данным аутентификации во всем приложении.

Компоненты – это основные строительные блоки в разработке программного обеспечения, используемые для создания пользовательского интерфейса и его поведения [3]. В контексте веб-разработки, компоненты представляют собой переиспользуемые единицы кода, которые объединяют в себе различные аспекты функциональности и внешнего вида для создания интерактивных и динамических веб-приложений.

Всего в приложении было разработано 18 функциональных компонентов. Компонент `BasketItem` отвечает за отображение элемента корзины в интернет-магазине. Код данного компонента представлен в листинге 3.8.

```
const BasketItem = (props) => {
  return (
    <div key={props.products.ProductID} className={styles.product}>
      <h3>{props.products.ProductName}</h3>
      <p>Price: {props.products.ProductPrice} BYN</p>
      <p>Quantity: {props.Quantity}</p>
      <div className="d-flex">
        <br />
        <button
          className={styles.product_button}
          onClick={() => props.removeItem(props.BasketProductID)}
        >
          <BsFillTrashFill />
        </button>
      </div>
    </div>
  );
};

export default BasketItem;
```

Листинг 3.8 – Компонент для отображения содержимого корзины

Функция компонента `BasketItem` принимает в качестве параметра объект `props`, который содержит свойства, связанные с продуктом в корзине. Каждый продукт в корзине имеет уникальный идентификатор `ProductID`, название `ProductName`, цену `ProductPrice` и количество `Quantity`.

Визуальное представление каждого элемента корзины состоит из заголовка с названием продукта, а также цены и количества продукта. Кнопка удаления продукта, представленная иконкой корзины, размещается в правой части элемента корзины. При клике на кнопку, вызывается функция `removeItem`, переданная в компонент через свойство `props`, и передает идентификатор продукта `BasketProductID` для удаления из корзины.

4 Тестирование приложения

Тестирование является неотъемлемой частью разработки любого программного продукта. Оно позволяет проверить работоспособность и соответствие приложения заданным требованиям. В данном приложении было проведено ручное тестирование.

Ручное тестирование приложения Next представляет собой процесс проверки функциональности, качества и согласованности пользовательского интерфейса на основе выполнения различных действий и ввода данных с использованием человеческого труда. Этот вид тестирования осуществляется вручную, в отличие от автоматизированного тестирования, и требует участия тестировщиков, которые выполняют действия, наблюдают за результатами и анализируют поведение приложения.

Целью ручного тестирования приложения Next является выявление ошибок, дефектов и несоответствий, которые могут возникнуть в процессе использования приложения конечными пользователями. Это включает проверку корректности отображения и работоспособности элементов пользовательского интерфейса, правильного функционирования интерактивных элементов, обработки пользовательского ввода и реакции на различные сценарии использования.

Процесс ручного тестирования приложения Next начинается с определения тестовых сценариев и требований к приложению. Это может включать проверку навигации между страницами, взаимодействие с формами, обработку ошибок и другие важные функции.

На формах входа и регистрации предусмотрены правила валидации, такие как ввод неверного логина или пароля и обязательные к заполнению поля.

При вводе электронной почты в неверном формате пользователь увидит следующее уведомление, которое представлено на рисунке 4.1.

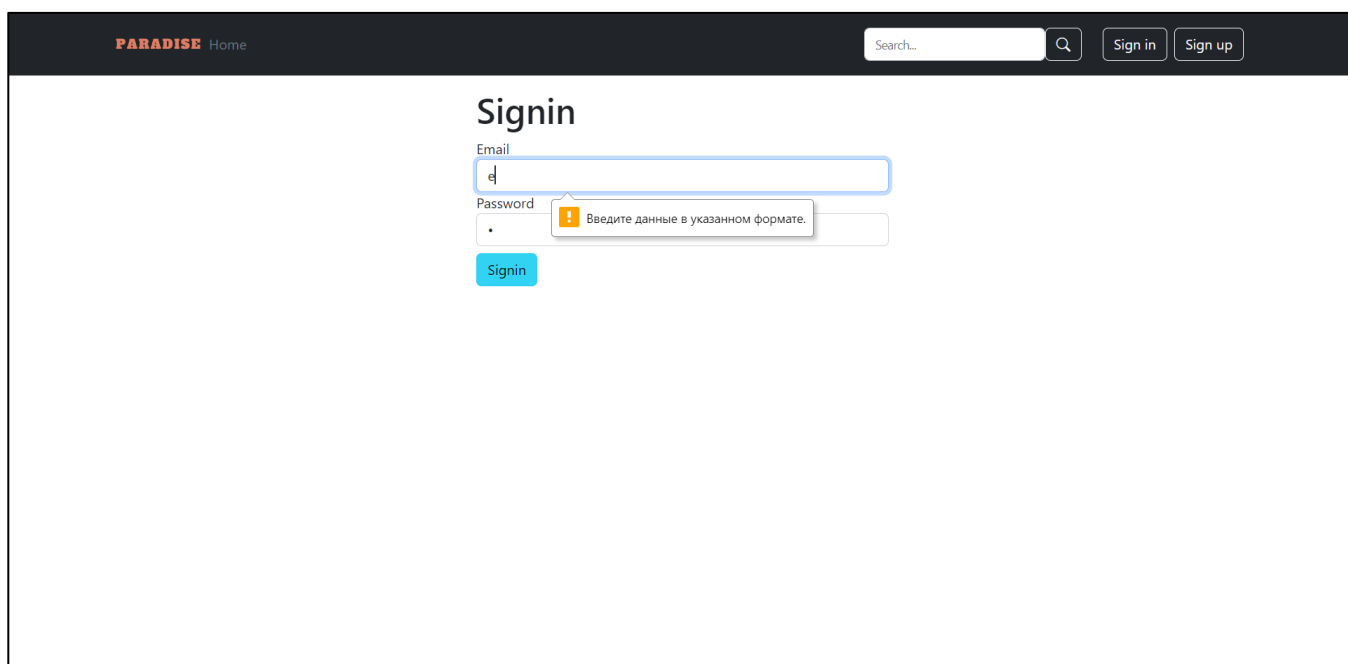
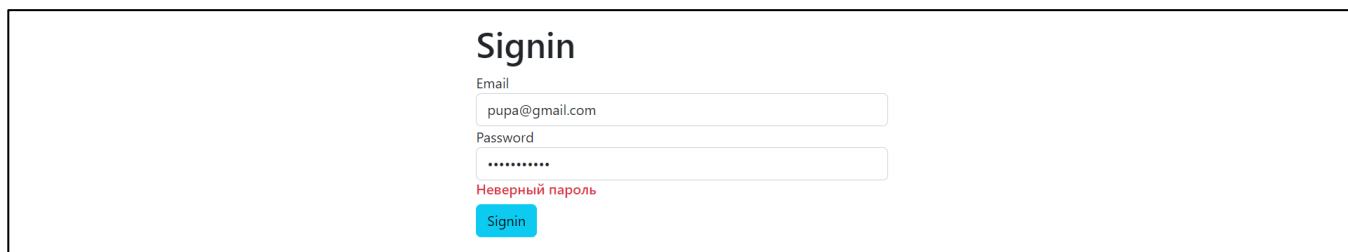


Рисунок 4.1 – Неверный формат электронной почты в форме входа

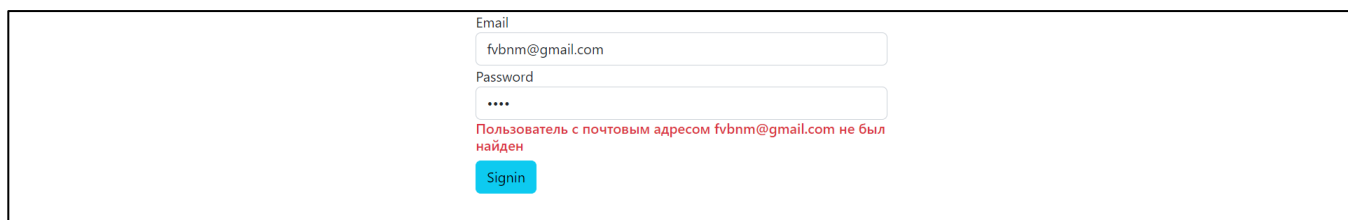
Результат попытки входа с правильной электронной почтой, но неправильным паролем представлен на рисунке 4.2.



The screenshot shows a 'Signin' form with two input fields: 'Email' containing 'pupa@gmail.com' and 'Password' containing a masked password. Below the password field, a red error message reads 'Неверный пароль' (Incorrect password). A blue 'Signin' button is located at the bottom of the form.

Рисунок 4.2 – Попытка входа с неправильным паролем

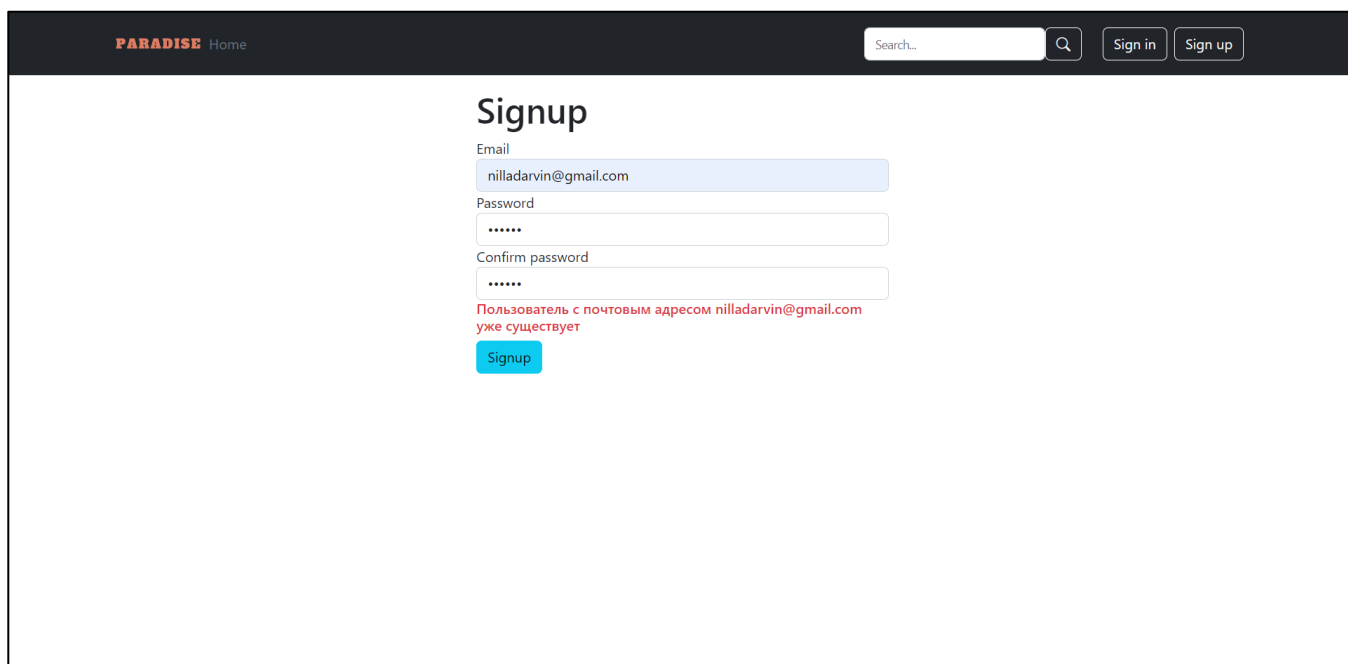
При попытке авторизоваться, используя почту, которая не зарегистрирована на сайте, пользователь увидит сообщение об ошибке. Результат этого теста представлен на рисунке 4.3.



The screenshot shows a 'Signin' form with 'Email' set to 'fvbnm@gmail.com' and 'Password' masked. A red error message below the password field states 'Пользователь с почтовым адресом fvbnm@gmail.com не был найден' (User with email address fvbnm@gmail.com was not found). A blue 'Signin' button is at the bottom.

Рисунок 4.3. – Попытка входа под незарегистрированной почтой

Также была совершена попытка зарегистрировать пользователя с уже имеющимся логином, результат этого теста представлен на рисунке 4.4.



The screenshot shows a 'Signup' form on a website with a dark header. The header includes the 'PARADISE Home' logo, a search bar, and 'Sign in' and 'Sign up' buttons. The 'Signup' form has three input fields: 'Email' (nilladarvin@gmail.com), 'Password', and 'Confirm password', all masked. A red error message below the fields says 'Пользователь с почтовым адресом nilladarvin@gmail.com уже существует' (User with email address nilladarvin@gmail.com already exists). A blue 'Signup' button is at the bottom.

Рисунок 4.4 – Попытка зарегистрировать существующего пользователя

В данном случае нарушилось правило уникальности имени пользователя и его почты, что не позволило совершить регистрацию.

Также при регистрации, пользователь должен подтвердить введенный пароль, если пароли не будут совпадать, то пользователь получит сообщение, представленное на рисунке 4.5.

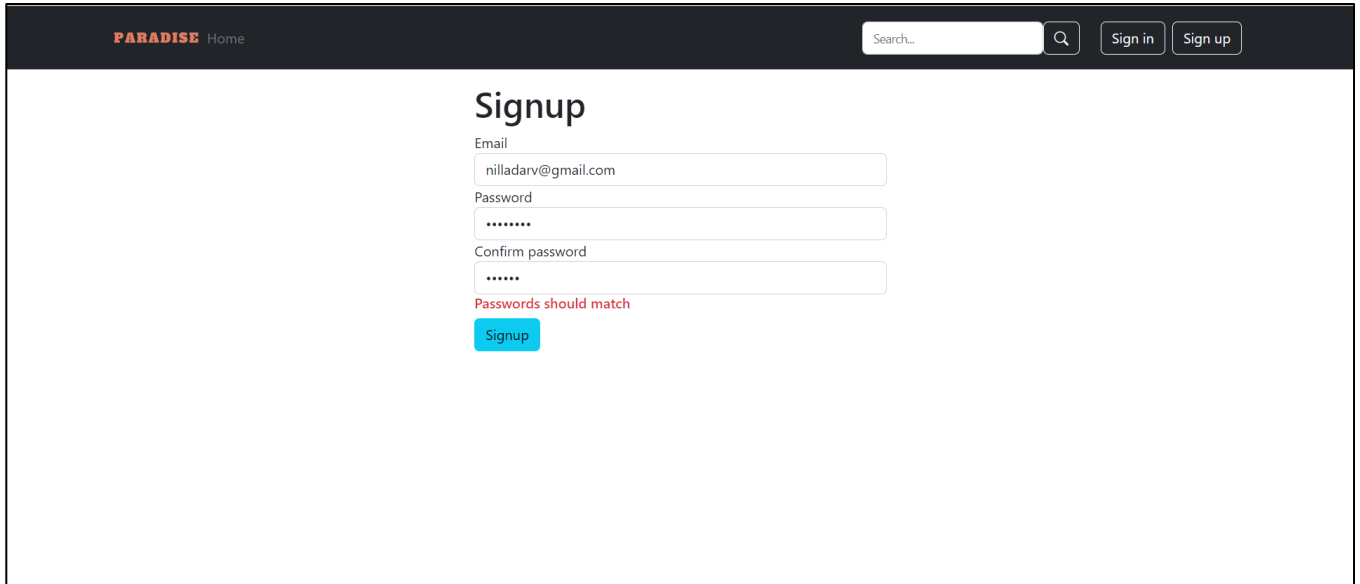


Рисунок 4.5 – Ввод разных паролей

Важной частью является проверка форм для добавления и изменения товаров. Если администратор оставит поля пустыми при отправке формы, то получит уведомление, представленное на рисунке 4.6.

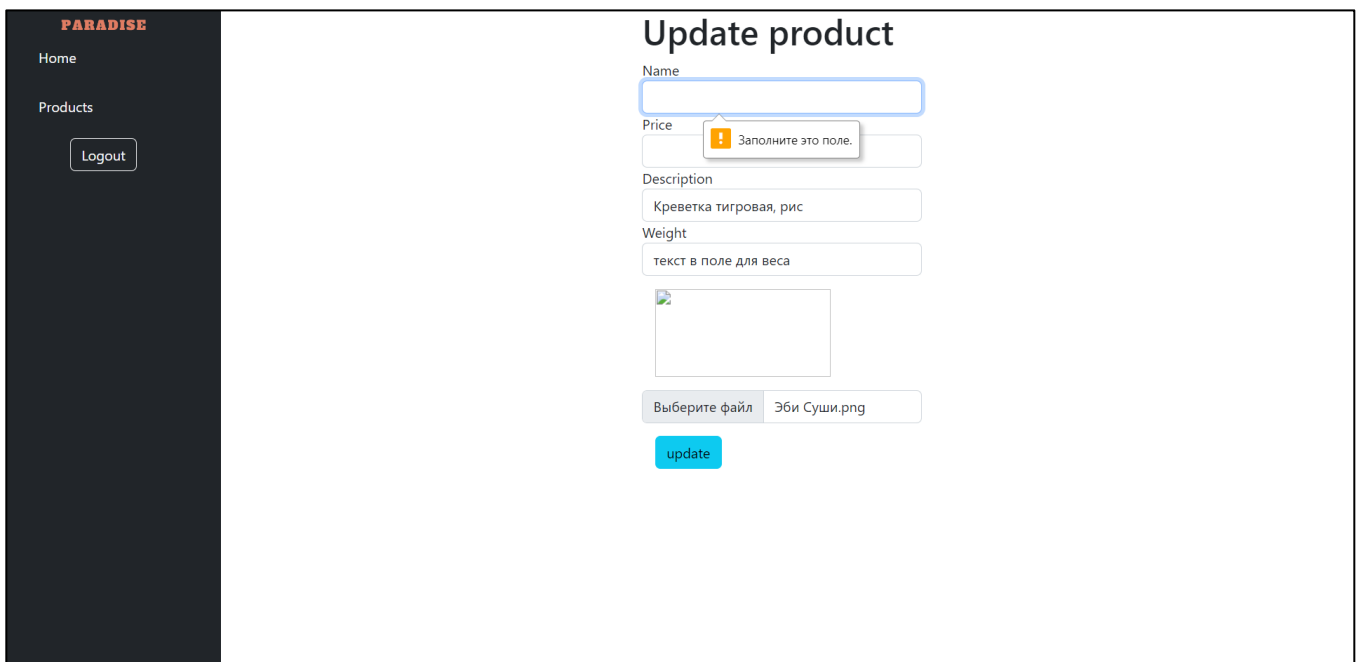


Рисунок 4.6 – Попытка оставить поля пустыми

5 Руководство пользователя

При входе пользователя на главную страницу, он может авторизоваться на сайте или зарегистрироваться. Главная страница приложения представлена на рисунке 5.1.

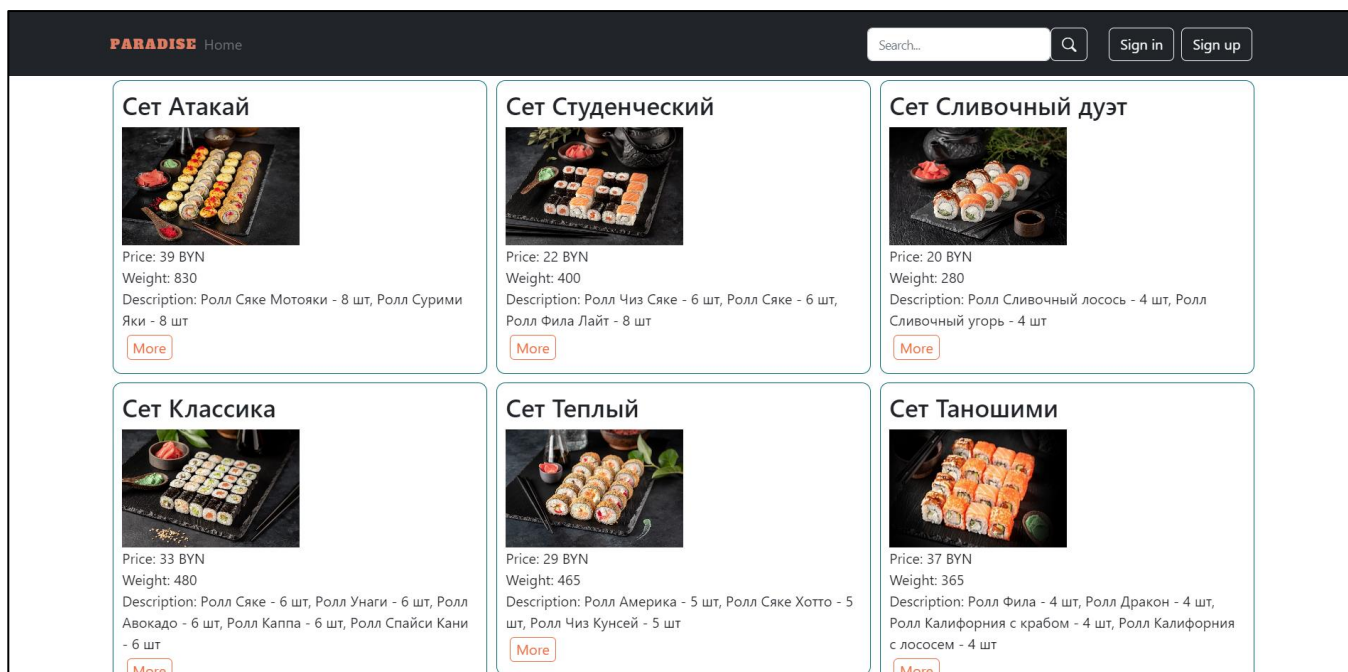


Рисунок 5.1 – Главная страница приложения

Также неавторизованному пользователю предоставляется возможность просмотра товаров и поиска по названию. Чтобы просмотреть страницу товара, показанную на рисунке 5.2, необходимо нажать на кнопку «More» под интересующим товаром.



Рисунок 5.2 – Страница товара

На этой же странице пользователь может просмотреть отзывы на продукт, которые оставили другие пользователи. Оставлять комментарии могут только авторизованные пользователи.

Чтобы перейти к формам авторизации или регистрации, необходимо нажать на соответствующие кнопки вверху экрана. Страница с регистрацией представлена на рисунке 5.3.

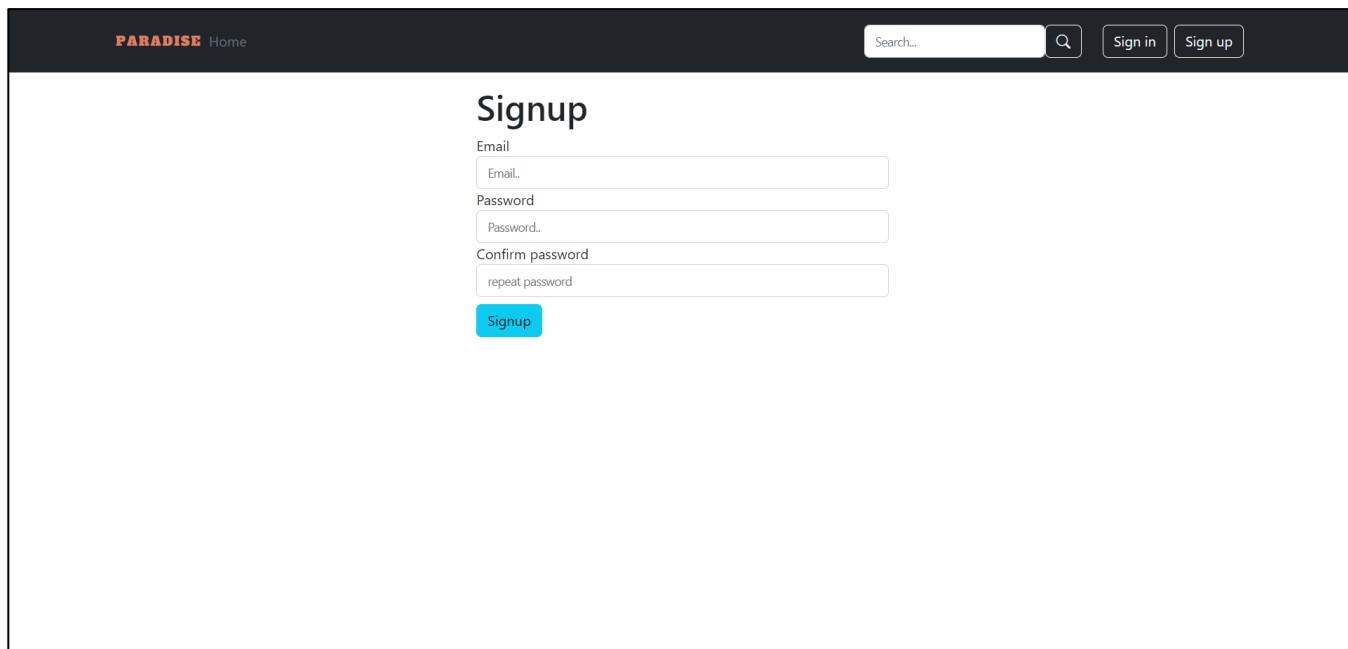


Рисунок 5.3 – Страница с формой регистрации

После ввода валидных данных пользователь будет перенаправлен на страницу с информацией о том, что ему нужно проверить почту. Информационная страница представлена на рисунке 5.4.

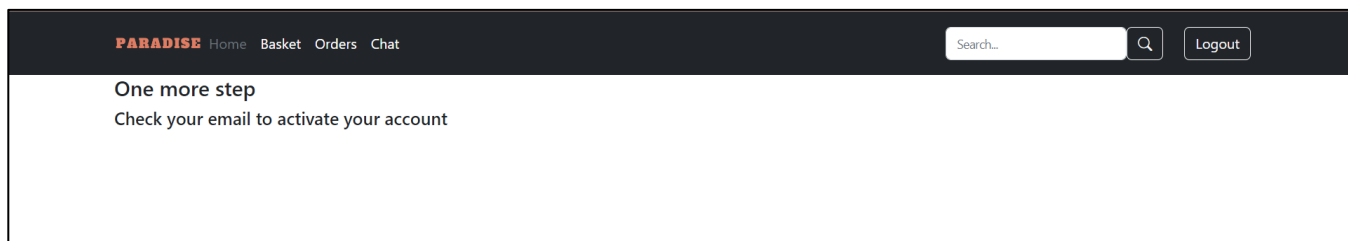


Рисунок 5.4 – Информационная страница после успешной регистрации

На почту придет письмо, содержащее ссылку, при переходе по которой аккаунт пользователя будет активирован.

В приложении есть три роли: администратор, обычный пользователь и курьер. При авторизации администратора его сразу перенаправит на главную страницу администратора, где ему будет доступен общий чат с курьером и обычными пользователями. Пример главной страницы администратора приведен на рисунке 5.5. Так же ему будет доступна страница с продуктами, где он может добавлять новые товары, изменять и удалять старые.

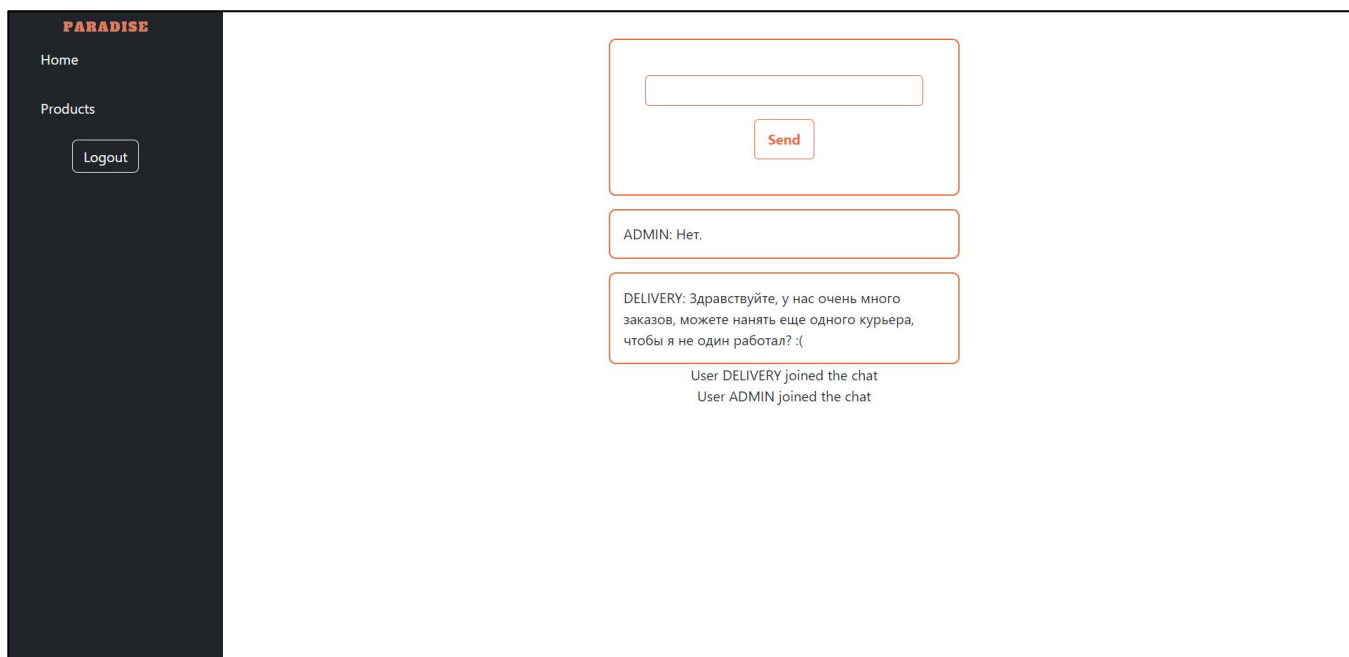


Рисунок 5.5 – Главная страница администратора с открытым чатом

На странице продуктов товары представлены в виде таблицы с двумя дополнительными кнопками, которые отвечают за удаление и изменение товара, так же ниже таблицы присутствует кнопка добавления. Страница товаров приведена на рисунке 5.6.

Id	Name	Price	Weight	Update	Delete
2	Сет Атакай	39	830		
3	Сет Студенческий	22	400		
4	Сет Сливочный дуэт	20	280		
5	Сет Классика	33	480		
7	Сет Теплый	29	465		
8	Сет Таношими	37	365		
9	Сет Нью	31	560		
10	Спринг Цезарь с курицей	19	350		
11	Спринг Цезарь с тигровой креветкой	20	280		
12	Аризона Кранч	21	235		

Рисунок 5.6 – Страница управления товарами

При авторизации курьера его перенаправит на главную страницу курьера, функционал и интерфейс которой схожи на страницу администратора. Курьер тоже имеет доступ к чату, а также к странице управления заказами, где он может поменять статус заказа. Страница изменения статуса заказа представлена на рисунке 5.7.

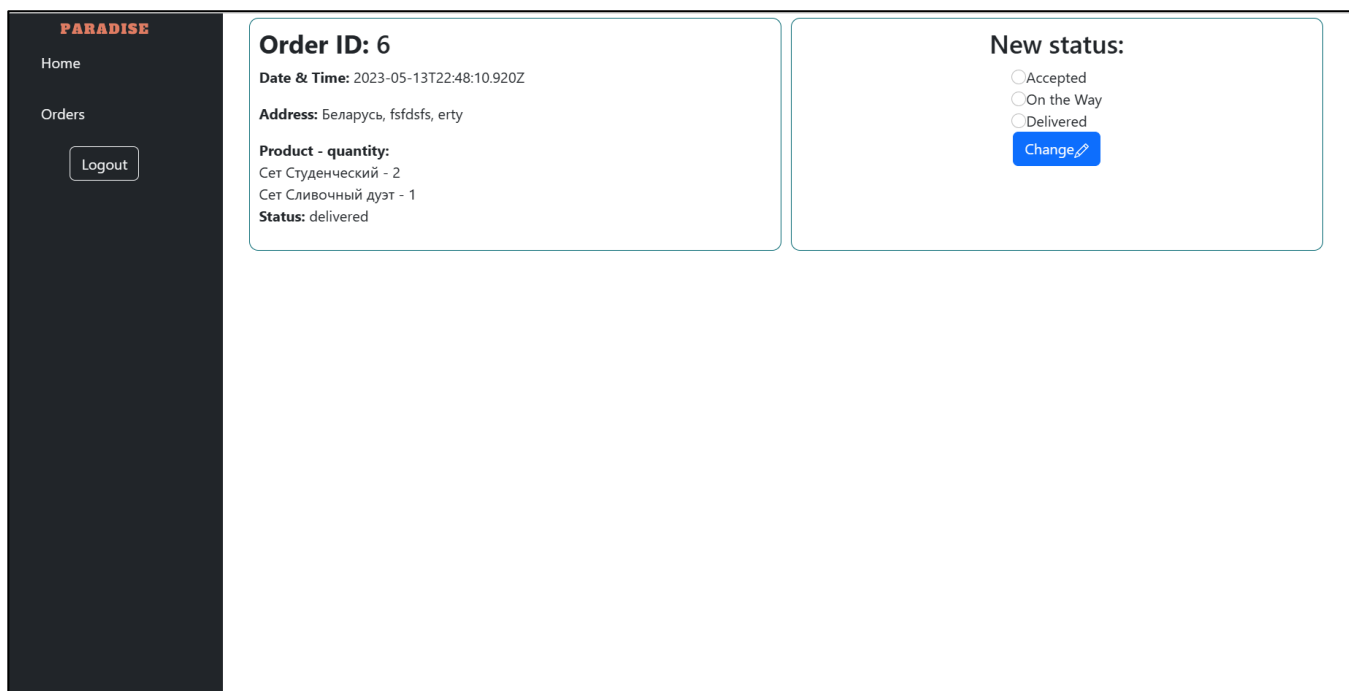


Рисунок 5.7 – Страница изменения статуса заказа

При авторизации обычный пользователь получает доступ к корзине, истории заказов, добавлению комментариев и добавлению товара к корзине. Пример добавления товара приведен на рисунке 5.8.

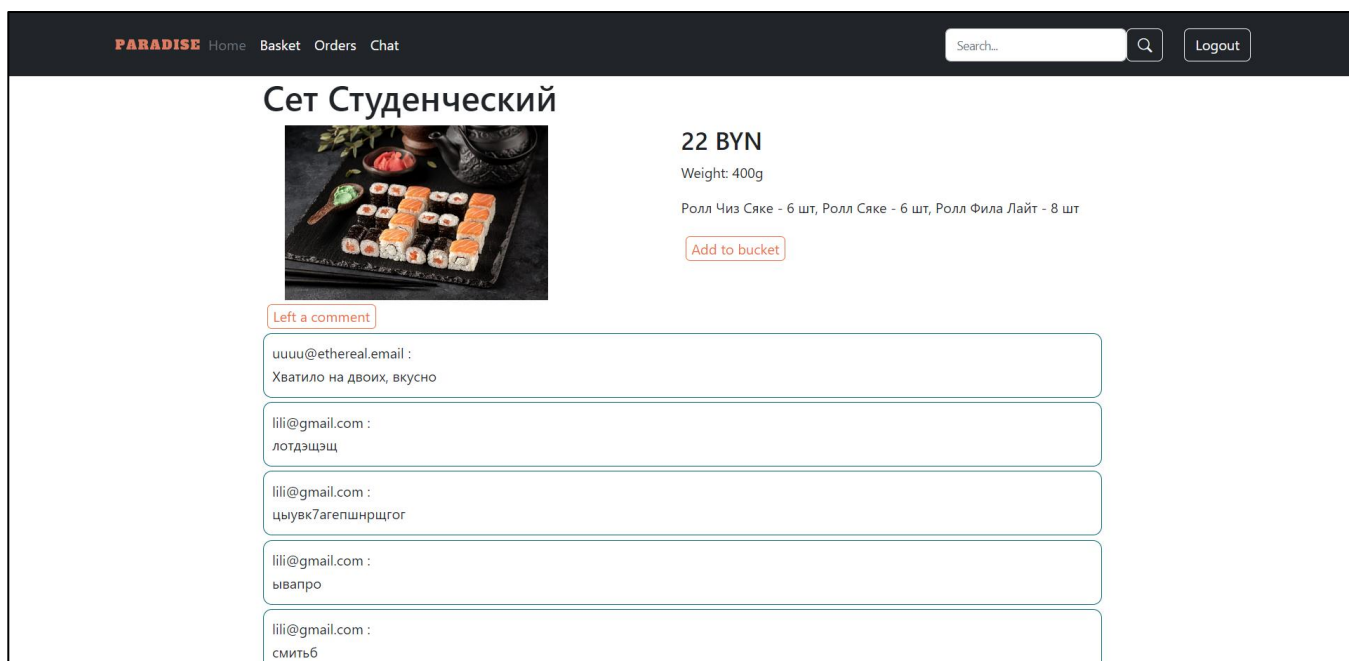


Рисунок 5.8 – Пример добавления товара в корзину.

После добавления товара в корзину пользователь может добавлять товары дальше и просмотреть их в корзине. На странице корзины пользователь может удалить товар из списка или же произвести оплату. Перед переходом на страницу

оплаты пользователю будет необходимо ввести адрес для доставки в появившейся форме. Далее пользователя перенесет на страницу оплаты через сервис Stripe. При отмене оплаты заказ не оформляется, а корзина очищается от товаров. Пример страницы оплаты приведен на рисунке 5.9.

← TEST MODE

Заплатить

72,00 BYN

Сет Студенческий	44,00 BYN
Кол-во 2	22,00 BYN шт.
Мисо с грибами	8,00 BYN
Кол-во 1	
Том Ям	20,00 BYN
Кол-во 1	

На платформе stripe | Условия | Конфиденциальность

Оплатить картой

Эл. почта

Данные карты

1234 1234 1234 1234

MM / ГГ

Код CVV/CVC

Имя и фамилия, указанные на карте

Страна или регион

Беларусь

☐ Надежно сохранить мои данные для оформления платежа одним щелчком

Выполняйте оплату быстрее на этом сайте и везде, где принимают Link.

Оплатить

Рисунок 5.9 – Пример страницы оплаты заказа

После успешной оплаты пользователя перенаправит на страницу, где он может просмотреть все свои заказы и узнать их статус. Страница истории заказов представлена на рисунке 5.10.

PARADISE Home Basket Orders Chat

Search... Logout

ID: 6

Date & Time: 2023-05-13T22:48:10.920Z

Address: Беларусь, fsdfs, erty

Product - quantity:

Сет Студенческий - 2

Сет Сливочный дуэт - 1

Status: delivered

Пользователю так же, как и администратору с курьером, доступен общий чат. Авторизованный пользователь имеет доступ к поиску товаров по названию, точно так же, как и неавторизованный.

Заключение

В процессе решения поставленной задачи была достигнута поставленная цель по созданию программного средства «Интернет магазин автозапчастей». Основой курсового проекта стало проектирование кроссплатформенного приложения, которое помогло облегчить взаимодействие с пользователем. Это было достигнуто за счёт гибкой программной платформы сервера на Node.js и клиентской части на Next.js с использованием React.js.

При разработке выполнены следующие пункты:

- авторизация и регистрация пользователя;
- создание новых учетных записей пользователя;
- отправка подтверждающих писем на электронную почту;
- панель администратора;
- отправка отзывов к товару;
- добавление товаров в корзину;
- система оплаты;

В курсовом проекте были реализованы следующие задачи:

- Создание базы данных;
- Создание сервера;
- Создание клиента с пользовательским интерфейсом;
- Реализация функций работы приложения;
- Тестирование программного продукта.

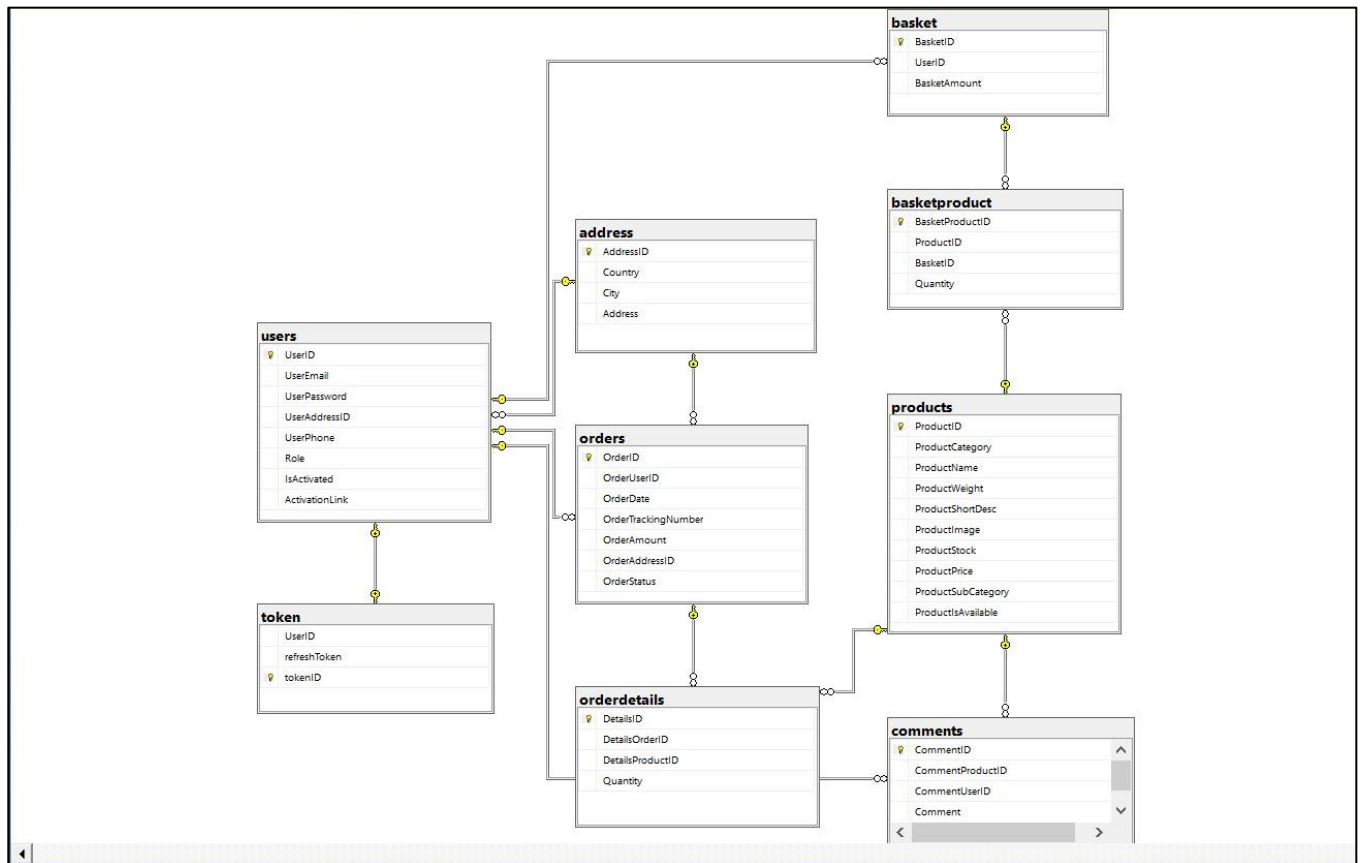
Данный проект является показательным примером программного продукта, реализующего интернет магазин, позволяющий охватить необходимые требования администрирования товаров, позволяя работать как с заказами, так и с сущностями, связанными с ними.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

Список литературы

1. About Node.js [Электронный ресурс] / OpenJS Foundation. – Режим доступа: <https://nodejs.org/en/about/>. – Дата доступа: 19.03.2023.
2. What is Object/Relational Mapping? [Электронный ресурс] / Red Hat. – Режим доступа: <https://hibernate.org/orm/what-is-an-orm/>. – Дата доступа: 3.04.2022.
3. React components [Электронный ресурс] / React. – Режим доступа: [https://www.w3schools.com/react/react_components.asp#:~:text=Components%20are%20independent%20and%20reusable,will%20concentrate%20on%20Function%20components](https://www.w3schools.com/react/react_components.asp#:~:text=Components%20are%20independent%20and%20reusable,will%20concentrate%20on%20Function%20components.). – Дата доступа: 18.04.2023.
4. Prisma Documentation [Электронный ресурс] / Prisma. – Режим доступа: <https://www.prisma.io/docs/>. – Дата доступа: 15.03.2022.
5. Stripe Documentation [Электронный ресурс] / Stripe. – Режим доступа: <https://stripe.com/docs/payments>. – Дата доступа: 26.04.2022.
6. Next.js Documentation [Электронный ресурс] / Vercel. – Режим доступа: <https://nextjs.org/docs> – Дата доступа: 22.04.2022.
7. Fetch API [Электронный ресурс] / Mozilla. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API. – Дата доступа: 28.04.2022.
8. React.js Documentation [Электронный ресурс] / React.org. – Режим доступа: <https://reactjs.org/docs/context.html>. – Дата доступа: 20.03.2022.

Приложение А



Приложение Б

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlserver"
  url      = env("DATABASE_URL")
}

model address {
  AddressID Int          @id(map: "PK_address") @unique(map:
"index_AddressID") @default(autoincrement())
  Country   String?      @db.VarChar(20)
  City      String?      @db.VarChar(50)
  Address   String?      @db.VarChar(50)
  orders    orders[]

  @@index([Address], map: "index_Address")
  @@index([City], map: "index_City")
  @@index([Country], map: "index_Country")
}

model basket {
  BasketID      Int          @id(map: "PK_basket") @unique(map:
"index_BasketID") @default(autoincrement())
  UserID        Int?
  BasketAmount  Float?
  users         users?       @relation(fields: [UserID],
references: [UserID], onDelete: NoAction, onUpdate: NoAction, map:
"FK_basketUserFK")
  basketproduct basketproduct[]

  @@index([UserID], map: "index_BasketUserID")
}

model basketproduct {
  BasketProductID Int          @id(map: "PK_basketproduct")
@unique(map: "index_BasketProductID") @default(autoincrement())
  ProductID       Int?
  BasketID        Int?
  Quantity        Int?        @default(1)
  basket          basket?     @relation(fields: [BasketID],
references: [BasketID], onDelete: NoAction, onUpdate: NoAction, map:
"FK_basketbasketFK")
  products        products? @relation(fields: [ProductID],
references: [ProductID], onDelete: Cascade, onUpdate: NoAction, map:
"FK_basketProductFK")
}
```



```

    @@index([ProductID], map: "index_BasketProductProductID")
}

model comments {
    CommentID          Int          @id(map: "PK_comments") @unique(map:
"index_CommentID") @default(autoincrement())
    CommentProductID  Int?
    CommentUserID      Int?
    Comment             String?     @db.VarChar(1000)
    products            products? @relation(fields: [CommentProductID],
references: [ProductID], onDelete: Cascade, onUpdate: NoAction, map:
"FK_CommentProductPK")
    users               users?      @relation(fields: [CommentUserID],
references: [UserID], onDelete: NoAction, onUpdate: NoAction, map:
"FK_CommentUserPK")

    @@index([Comment], map: "index_Comment")
    @@index([CommentProductID], map: "index_CommentProductID")
    @@index([CommentUserID], map: "index_CommentUserID")
    @@index([CommentProductID], map: "IX_FK_CommentProductPK")
}

model orderdetails {
    DetailsID          Int          @id(map: "PK_orderdetails")
@unique(map: "index_DetailsID") @default(autoincrement())
    DetailsOrderID     Int?
    DetailsProductID   Int?
    Quantity           Int?
    orders              orders?    @relation(fields: [DetailsOrderID],
references: [OrderID], onDelete: NoAction, onUpdate: NoAction, map:
"FK_DetailsOrderFK")
    products            products? @relation(fields: [DetailsProductID],
references: [ProductID], onDelete: Cascade, onUpdate: NoAction, map:
"FK_DetailsProductFK")

    @@index([DetailsOrderID], map: "index_DetailsOrderID")
    @@index([DetailsProductID], map: "index_DetailsProductID")
    @@index([DetailsOrderID], map: "IX_FK_DetailsOrderFK")
    @@index([DetailsProductID], map: "IX_FK_DetailsProductFK")
}

model orders {
    OrderID            Int          @id(map: "PK_orders")
@unique(map: "index_OrderID") @default(autoincrement())
    OrderUserID        Int?
    OrderDate           DateTime?   @db.DateTime
    OrderAmount         Float?
    OrderAddressID      Int?
    OrderStatus         Int?        @default(1) @db.TinyInt

```

```

    orderdetails          orderdetails[]
    address                address?          @relation(fields:
[OrderAddressID], references: [AddressID], onDelete: NoAction,
onUpdate: NoAction, map: "FK_OrdersAddressFK")
    users                  users?            @relation(fields:
[OrderUserID], references: [UserID], onDelete: NoAction, onUpdate:
NoAction, map: "FK_OrdersUsersFK")

    @@index([OrderAddressID], map: "index_OrderAddressID")
    @@index([OrderAmount], map: "index_OrderAmount")
    @@index([OrderDate], map: "index_OrderDate")
    @@index([OrderStatus], map: "index_OrderStatus")
    @@index([OrderUserID], map: "index_OrderUserID")
    @@index([OrderUserID], map: "IX_FK_OrdersUsersFK")
}

model products {
    ProductID
    Int @id(map: "PK_products") @unique(map:
"index_ProductID") @default(autoincrement())
    ProductName
    String? @db.VarChar(50)
    ProductWeight
    Int?
    ProductShortDesc
    String? @db.NVarChar(200)
    ProductImage
    String?
    ProductStock
    Int?
    ProductPrice
    Float?
    basketproduct
    basketproduct[]
    comments
    comments[]
    orderdetails
    orderdetails[]

    @@index([ProductName], map: "index_ProductName")
    @@index([ProductPrice], map: "index_ProductPrice")
    @@index([ProductShortDesc], map: "index_ProductShortDesc")
    @@index([ProductStock], map: "index_ProductStock")
    @@index([ProductWeight], map: "index_ProductWeight")
}

model sysdiagrams {
    name String @db.NVarChar(128)

```

```

principal_id Int
diagram_id   Int      @id(map: "PK__sysdiagr__C2B05B6108BE5BAA")
@default(autoincrement())
version      Int?
definition   Bytes?

@@unique([principal_id, name], map: "UK_principal_name")
}

model token {
  UserID      Int      @unique
  refreshToken String @unique @db.NVarChar(350)
  tokenID     Int      @id(map: "PK_token") @unique
@default(autoincrement())
  users       users @relation(fields: [UserID], references:
[UserID], onUpdate: NoAction, map: "FK_token_users")
}

model users {
  UserID      Int      @id(map: "PK_users") @unique(map:
"index_UserID") @default(autoincrement())
  userEmail   String   @unique(map:
"UQ__users__08638DF8F78A2DE1") @db.VarChar(100)
  UserPassword String @db.VarChar(270)
  Role        Int      @default(0, map: "DefaultRole")
@db.TinyInt
  IsActivated Boolean @default(false, map:
"DF_users_IsActivated")
  ActivationLink String @unique @db.VarChar(250)
  basket      basket[]
  comments     comments[]
  orders       orders[]
  token        token[]

  @@index([UserEmail], map: "index_UserEmail")
  @@index([UserPassword], map: "index_UserPassword")
  @@index([Role], map: "index_UserRole")
}

```