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 «Чайная лавка»

Выполнил студент Левша Марк Сергеевич

(Ф.И.О.)

Руководитель проекта ст.препод. Дубовик М.В.

(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 ст.препод. Дубовик М.В.

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

Нормоконтролер ст.препод. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 3](#_Toc135390964)

[1 Постановка задачи 5](#_Toc135390965)

[1.1 Анализ аналогов 5](#_Toc135390966)

[1.1.1 Teashop.by 5](#_Toc135390967)

[1.1.2 Teaplantation.by 7](#_Toc135390968)

[1.1.3 Результаты анализа 9](#_Toc135390969)

[2 Проектирование web-приложения 10](#_Toc135390970)

[2.1 Проектирование архитектуры базы данных 10](#_Toc135390971)

[2.2 Проектирование структуры сервера 11](#_Toc135390972)

[2.2.1 Модули 11](#_Toc135390973)

[2.2.2 Контроллеры 12](#_Toc135390974)

[2.2.3 Провайдеры 12](#_Toc135390975)

[2.2.4 Middleware 12](#_Toc135390976)

[2.2.5 Фильтры и Интерсепторы 13](#_Toc135390977)

[2.3 Диаграмма развертывания приложения 13](#_Toc135390978)

[2.4 Диаграмма вариантов использования 15](#_Toc135390979)

[3 Разработка web-приложения 16](#_Toc135390980)

[3.1 Реализация серверной части 16](#_Toc135390981)

[3.1.1 Структура проекта 16](#_Toc135390982)

[3.1.2 Реализация взаимодействия с prisma ORM 18](#_Toc135390983)

[3.1.3 Реализация слоя контроллеров 19](#_Toc135390984)

[3.1.4 Реализация слоя сервисов 20](#_Toc135390985)

[3.1.5 Реализация слоя модулей 21](#_Toc135390986)

[3.2 Реализация клиентской части 21](#_Toc135390987)

[3.2.1 Структура проекта 21](#_Toc135390988)

[3.2.2 Компоненты 22](#_Toc135390989)

[4 Тестирование web-приложения 24](#_Toc135390990)

[4.1 Тестирование серверной части 24](#_Toc135390991)

[4.1.1 Ручное тестирование 24](#_Toc135390992)

[5 Руководство пользователя 27](#_Toc135390993)

[Заключение 31](#_Toc135390994)

[Список используемых источников 32](#_Toc135390995)

[ПРИЛОЖЕНИЕ А 33](#_Toc135390996)

[ПРИЛОЖЕНИЕ Б 35](#_Toc135390997)

[ПРИЛОЖЕНИЕ В 38](#_Toc135390998)

[ПРИЛОЖЕНИЕ Г 40](#_Toc135390999)

# 

# Введение

Веб-приложения стали неотъемлемой частью нашей повседневной жизни, обеспечивая удобный и быстрый доступ к различным услугам и ресурсам. Одной из сфер, где веб-приложения проявляются наиболее эффективно, является сфера ресторанного и кулинарного бизнеса. В рамках данного курсового проекта мы сосредоточимся на создании веб-приложения для чайной лавки.

Чайная лавка – это место, где любители чая могут насладиться широким ассортиментом чая, от классических сортов до экзотических вкусовых композиций. Однако, часто посетители сталкиваются с проблемой выбора из множества предлагаемых чаев, а также испытывают затруднения в получении информации о свойствах и происхождении каждого сорта.

Целью нашего курсового проекта является разработка веб-приложения, которое позволит посетителям чайной лавки удобно и эффективно выбирать чай, получать информацию о каждом сорте, а также совершать покупки онлайн. Веб-приложение также предоставит возможность регистрации и авторизации пользователей, а также предложит дополнительные функциональные возможности, такие как сохранение списка избранных чаев, оставление отзывов и рекомендаций, а также возможность связаться с администрацией лавки.

Таким образом, целью курсового проектирования является разработка веб-приложения, которое бы смогло собирать необходимые данные с систем для их последующего анализа.

Для реализации проекта мы будем использовать современные веб-технологии, включая язык программирования JavaScript, фреймворк для разработки веб-приложений и базу данных для хранения информации о чаях и пользователях. Кроме того, мы проведем исследование существующих веб-приложений в сфере чайной индустрии, чтобы учесть передовые практики и предложить инновационные решения.

Ожидается, что создание веб-приложения для чайной лавки позволит улучшить пользовательский опыт, повысить эффективность работы лавки и привлечь новых клиентов. Кроме того, разработка данного проекта может стать примером применения современных веб-технологий в сфере ресторанного бизнеса.

Для достижения поставленной цели необходимо выполнить следующие задачи:

* провести аналитический обзор литературы, аналогов приложения;
* спроектировать архитектуру приложения;
* спроектировать структуру базы данных;
* разработать приложение;
* протестировать приложение.

**Реферат**

Пояснительная записка курсового проекта содержит 32 страниц пояснительной записки, 29 иллюстраций, 7 листингов, 4 источников литературы, 4 приложений.

NEST, JAVASCRIPT, REACT, REST, MSSQL.

Цель курсового проекта: разработка веб-приложения, которое бы смогло собирать необходимые данные с систем для их последующего анализа.

Первая глава описывает постановку задачи и анализ аналогов приложения.

Во второй главе изложено проектирование web-приложения.

Третья глава описывает реализацию web-приложения.

Четвёртая глава описывает проведенное тестирование web-приложения.

В пятой главе изложено руководство пользователя.

В заключении приведены результаты проделанной работы.

# 1 Постановка задачи

## **1.1 Анализ аналогов**

В данном разделе был проведен анализ конкурентов, предоставляющих услуги интернет-магазина чая. Анализ проводился, основываясь на информации из открытых источников.

**1.1.1 Teashop.by**

Сайт teashop.by [1] предоставляет возможность покупки чая, посуды, благовоний и других товаров. Первое, что встречает пользователя, это главная страница.

Главная страница представлена на рисунке 1.1.



Рисунок 1.1 – Главная станица сервиса teashop.by

В навигационном меню присутствует возможность перехода к различным сортам чая.

Навигационное меню представлено на рисунке 1.2.

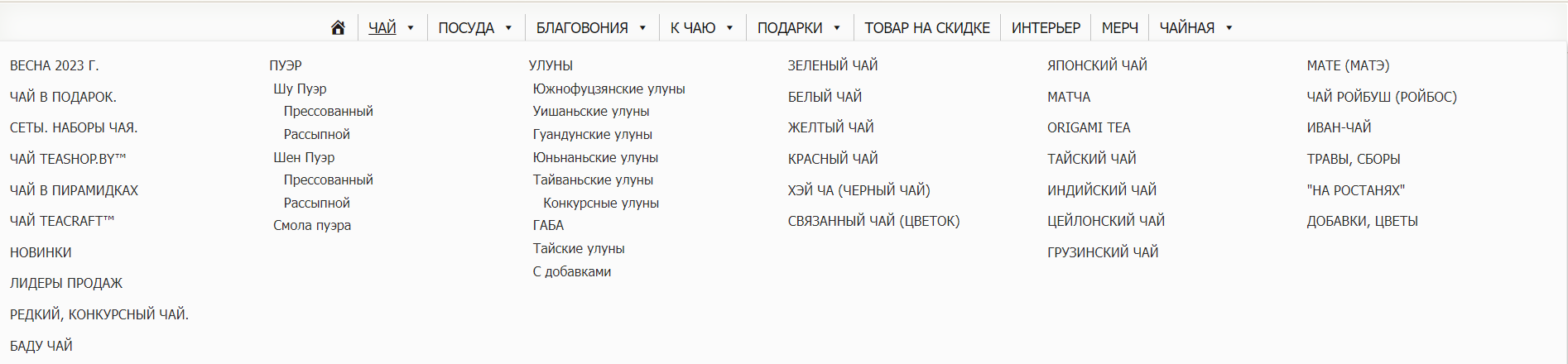


Рисунок 1.2 – Навигационное меню сервиса teashop.by

В данном выпадающем списке вы можете увидеть различные сорта чая, их категории и даже их агрегатное состояние.

Перейдя в каталог, можно увидеть продукты, их цену, граммовку и добавить их в корзину.

Каталог представлен на рисунке 1.3.

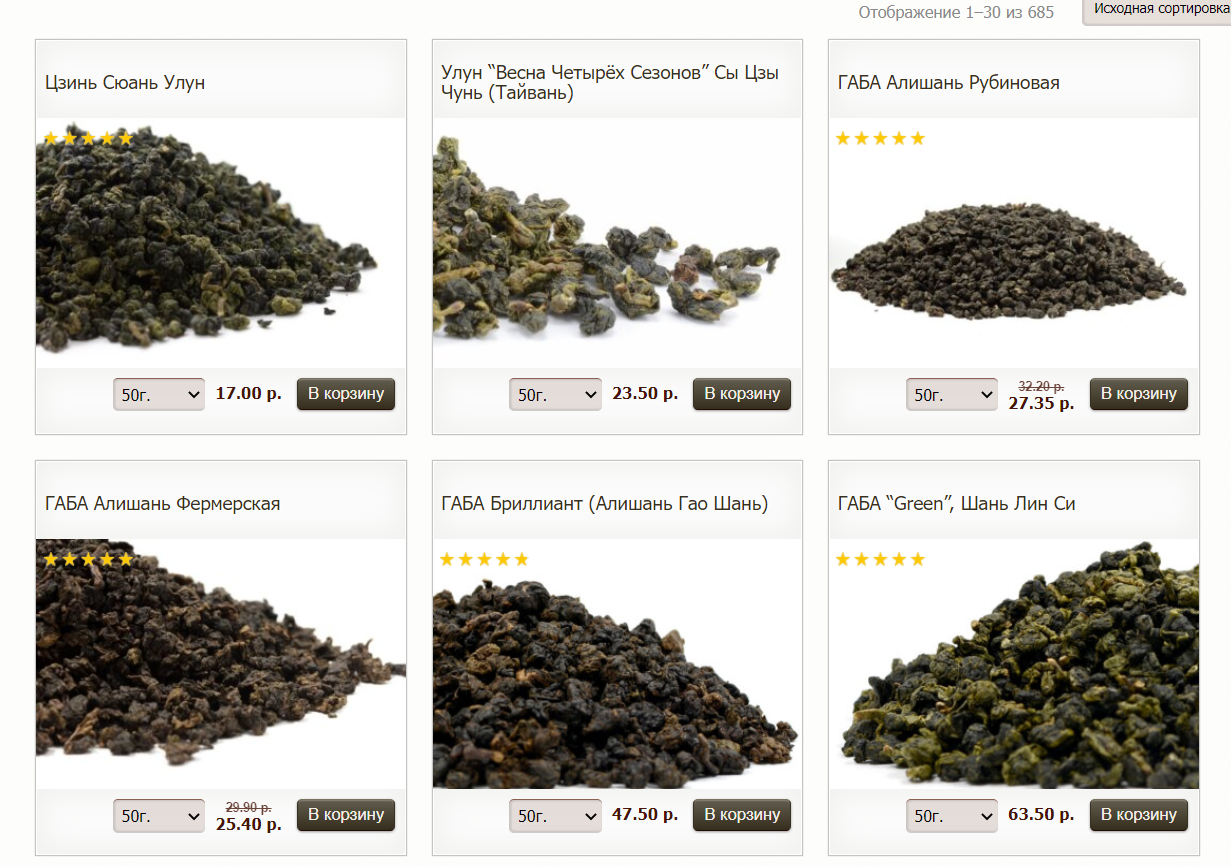


Рисунок 1.3 – Каталог сервиса teashop.by

Также на каждом элементе каталога есть возможность выбора грамм для заказа.

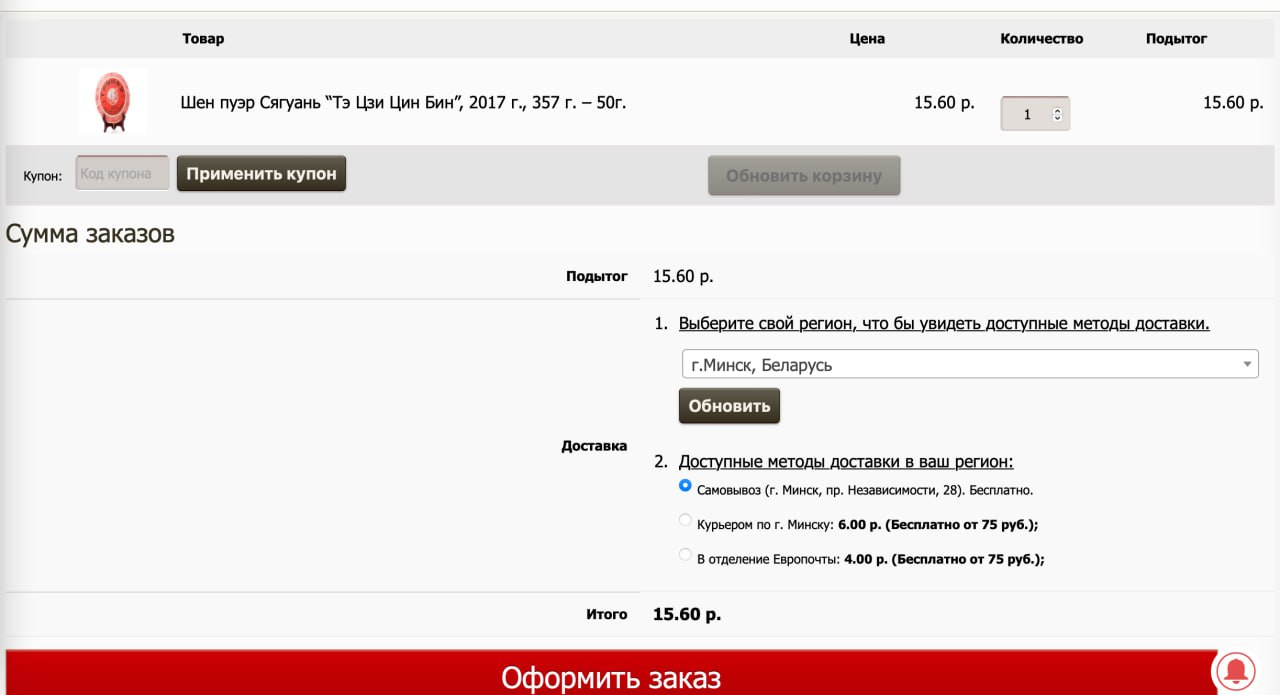


Рисунок 1.4 – Корзина сервиса teashop.by

Корзина включает в себя информацию о товаре, способ доставки и кнопку оформления заказа. Корзина представлена на рисунке 1.4.

Данный сервис практически не имеет недостатков, за исключением морально-устаревшего дизайна и большой загруженности пространства на странице. Данный сервис покрывает весь необходимый функционал для пользователя.

**1.1.2 Teaplantation.by**

Teaplantation.by – интернет-магазин для покупки чая [2].

Магазин предлагает вам уникальную возможность погрузиться в мир ароматного и вкусного чая прямо из уюта вашего дома. В магазине вы найдете широкий ассортимент чая, от классических сортов до эксклюзивных коллекций, чтобы удовлетворить самые изысканные вкусы и предпочтения.

Ваше путешествие в мир чая начинается с нашего интернет-магазина. Откройте для себя новые вкусы, насладитесь ароматами и создайте уютные моменты, наливая себе чашечку вашего любимого чая.

При входе на сайт пользователя встречает главная страница, которая представлена на рисунке 1.5.

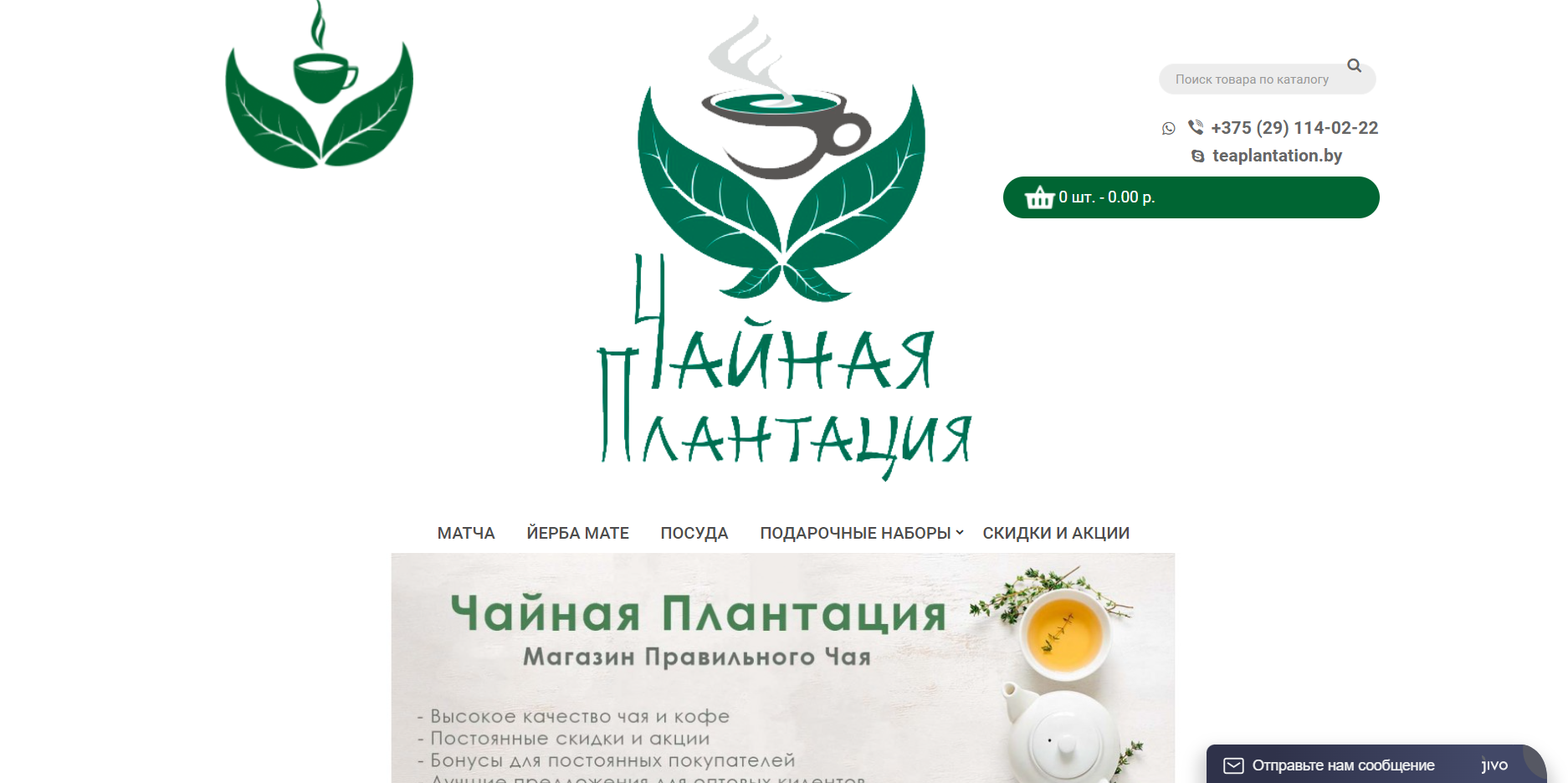


Рисунок 1.5 – Главная страница

В этом сервисе каталог, как таковой, отсутствует. Можно перейти лишь к одному виду чая.

Нажимая на единственный вид чая, осуществляется переход на страницу с подвидами этого чая, где размещается главная информация о чае и кнопка добавления чая в корзину. Данная вкладка представлена на рисунке 1.6.

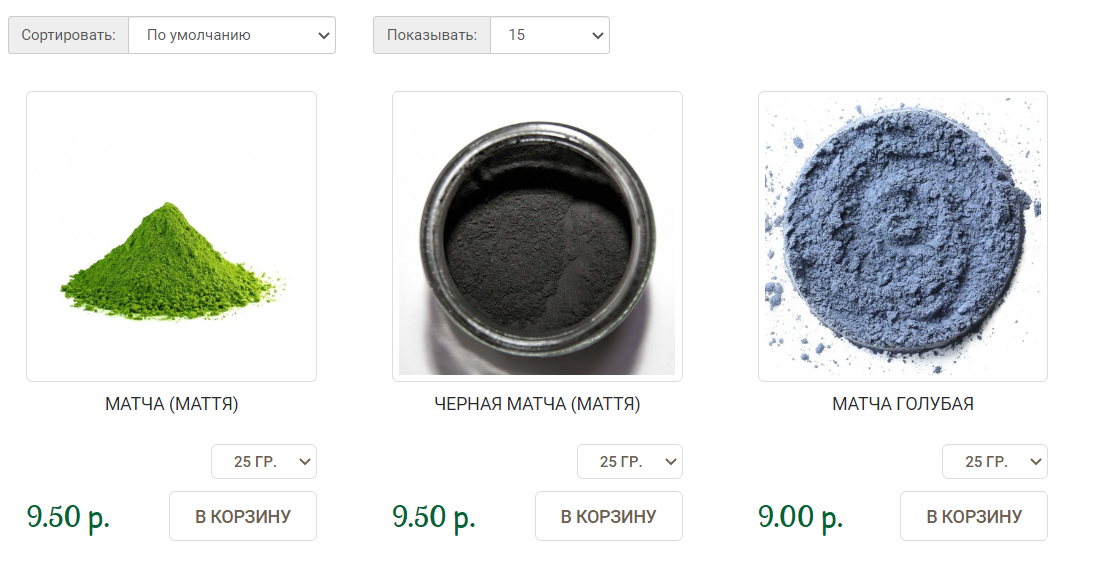


Рисунок 1.6 – Вкладка матча

При переходе на вкладку продукта выводится информация о товаре и его описание. Данная вкладка представлена на рисунке 1.7.

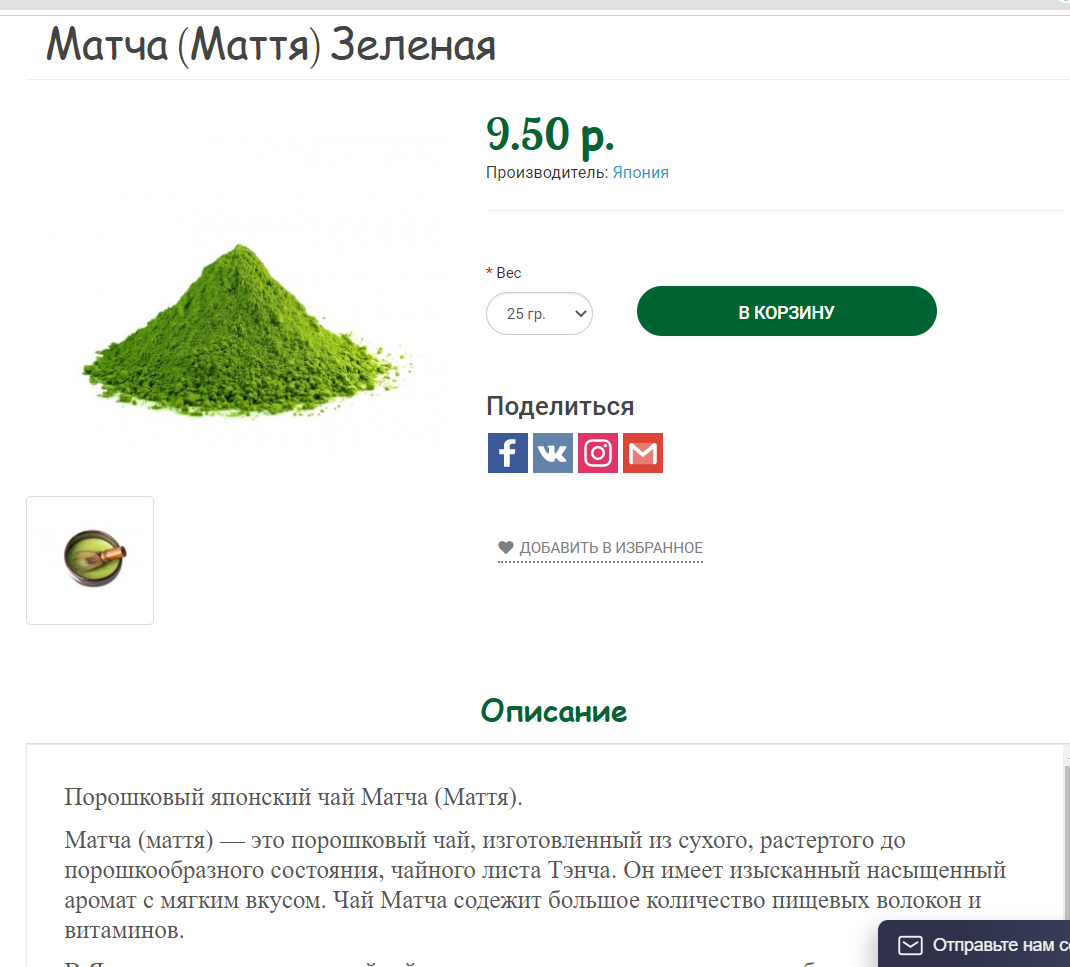


Рисунок 1.7 – Вкладка матча зеленая

При добавлении товара в корзину ничего не происходит, но пролистав страницу вверх видно, что товар был добавлен в корзину.

Перейдя в корзину можно увидеть добавленный товар и оформить заказ или продолжить покупки, но отсутствует кнопка удаления товара из корзины, товар можно удалить только переведя его счетчик количества в 0 и нажав на кнопку обновить корзину. Вкладка корзина, на которой можно это сделать представлена на рисунке 1.8.

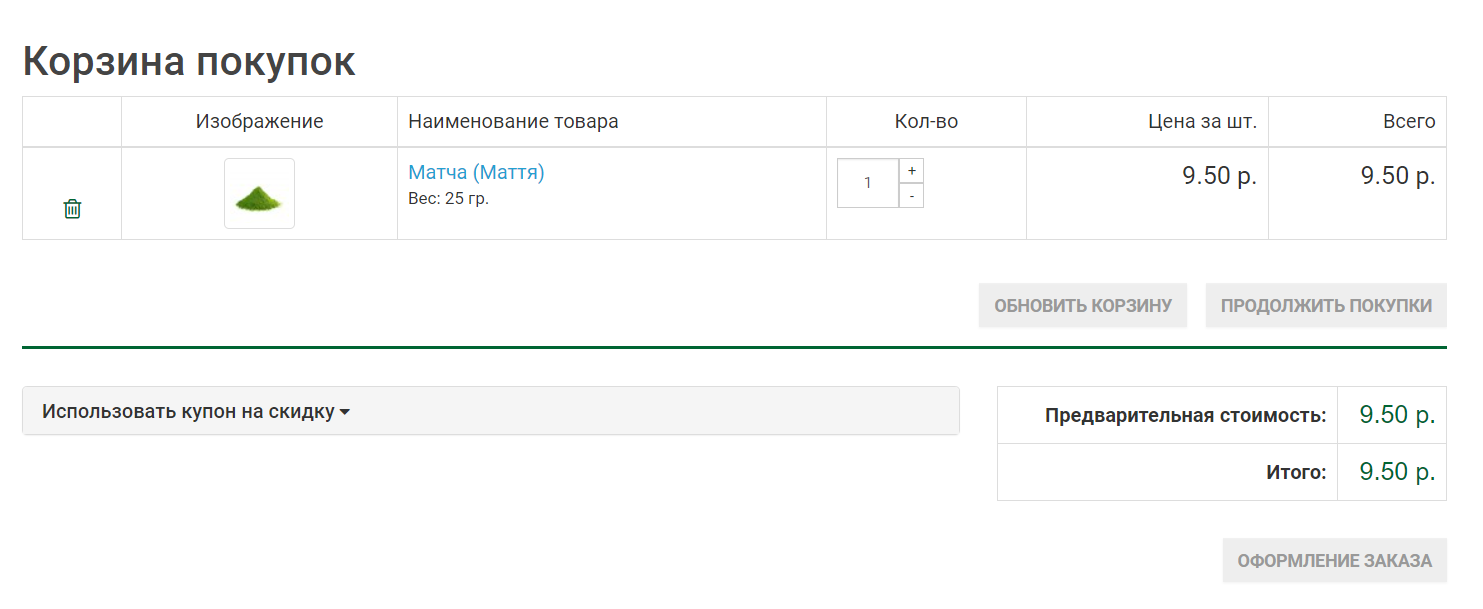


Рисунок 1.8 – Вкладка корзина

Данный сервис имеет ряд недостатков: отсутствие полноценного каталога, кнопки удаления товара из корзины и оповещений о действиях пользователя, также интерфейс не является дружелюбным и имеет громоздкий дизайн.

**1.1.3 Результаты анализа**

Анализ сравнения конкурентов интернет-магазинов чая поможет нам лучше понять текущую ситуацию на рынке и выявить особенности каждого конкурента. Ниже представлен обзор нескольких интернет-магазинов чая и их особенностей.

После проведенного анализа, были сформирован функционал, которым должно обладать приложение.

Приложение должно обладать следующим функционалом: просмотр каталога товаров, система отзывов, добавление товара в корзину, удаление товара из корзины, просмотр заказанных товаров в личном кабинете.

Приложение должно содержать две роли (администратор и пользователь). Администратор имеет возможность удалять и добавлять товары в каталог. Пользователь, в свою очередь, может просматривать каталог товаров, добавлять товары в корзину, удалять товары из корзины, оставлять отзывы, просматривать отзывы, просматривать в личном кабинете заказанные товары.

# 2 Проектирование web-приложения

## **2.1 Проектирование архитектуры базы данных**

В качестве СУБД была выбрана MSSQL. MSSQL – является реляционной системой управления базами данных (СУБД), разработанной компанией Microsoft. Она предоставляет надежное и масштабируемое решение для хранения, управления и обработки больших объемов структурированных данных.

В базе данных находится 4 таблицы, схема базы данных представлена на рисунке 2.1.

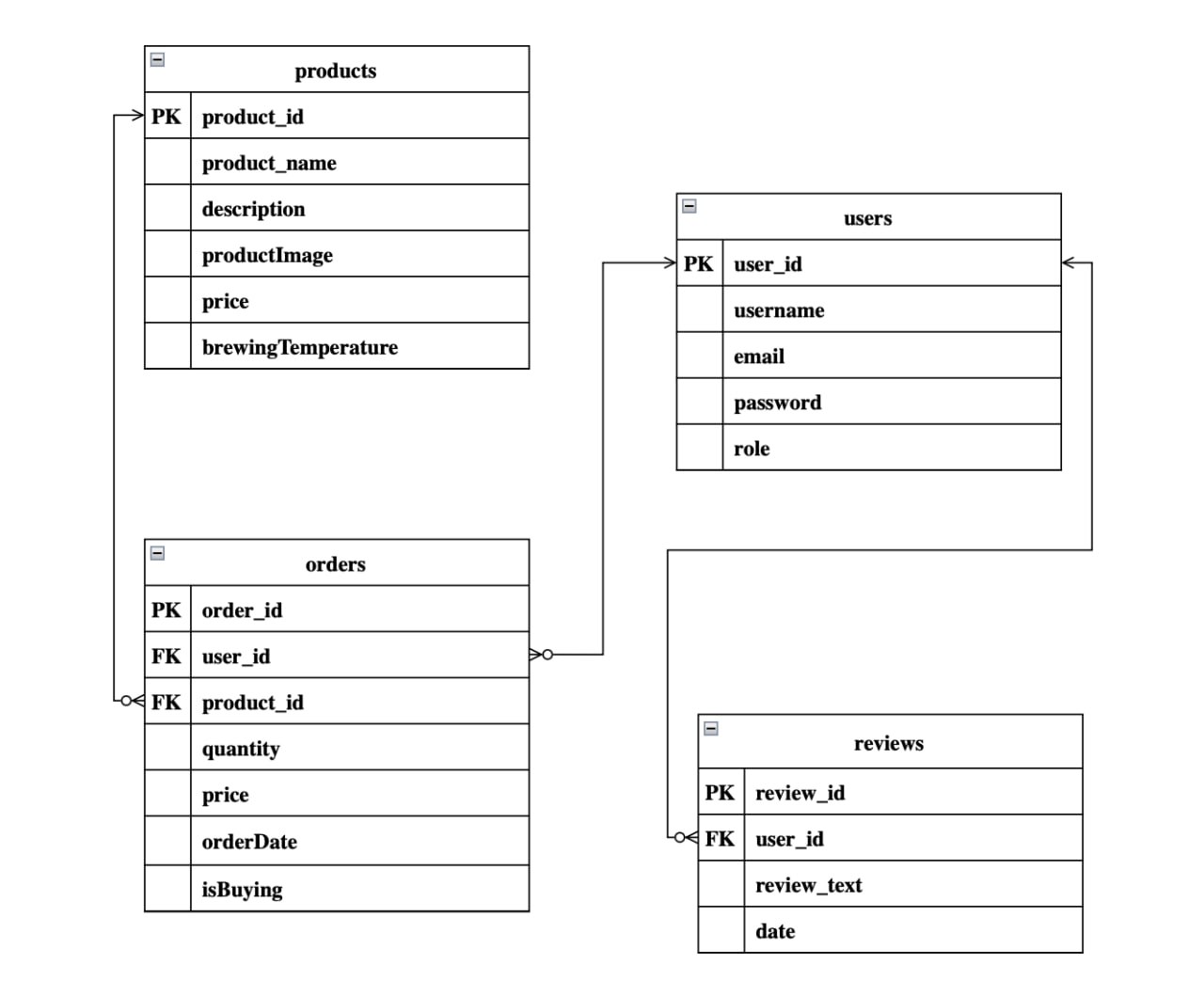


Рисунок 2.1 – Схема базы данных

В таблице users хранится информация о пользователях приложения, products хранит все продукты, которые есть в ассортименте, в orders находятся заказы конкретного пользователя, reviews хранит информацию о отзывах, оставленных пользователями.

Таблица users содержит следующие поля:

* user\_id – тип данных int, содержит идентификатор пользователя;
* username – тип данных varchar, содержит никнейм пользователя;
* email – тип данных varchar, содержит электронную почту пользователя, с помощью которой он аунтефицируется;
* password – тип данных varchar, содержит пароль пользователя;

Таблица products содержит следующие поля:

* product\_id – тип данных int, содержит идентификатор продукта;
* product\_name – тип данных varchar, содержит название продукта;
* description – тип данных varchar, содержит описание продукта;
* productImage – тип данных varchar, содержит изображение продукта;
* price – тип данных decimal, содержит цену продукта;
* brewingTemperature – тип данных varchar, содержит температуру заваривания чая;

Таблица reviews содержит следующие поля:

* review\_id – тип данных int, содержит идентификатор отзыва;
* user\_id – тип данных int, содержит идентификатор пользователя, разместившего отзыв;
* releaseDate – тип данных Date, содержит дату, когда система была выпущена;
* review\_text – тип данных varchar, содержит отзыв пользователя;
* date – тип данных varchar, содержит дату и время публикации отзыва;

Таблица orders содержит следующие поля:

* order\_id – тип данных int, содержит идентификатор заказа;
* user\_id – тип данных int, содержит идентификатор пользователя, сделавшего заказ;
* product\_id – тип данных int, содержит идентификатор заказанного товара;
* quantity – тип данных int, содержит количество заказанного товара;
* price – тип данных decimal, содержит цену заказа;
* orderDate – тип данных varchar, содержит дату оформления заказа;
* isBuying– тип данных varchar, содержит информацию о том, был ли заказ оформлен или нет;

Таким образом, была рассмотрена каждая из коллекций базы данных, описаны поля документов.

## **2.2 Проектирование структуры сервера**

NestJS – это фреймворк для разработки серверных приложений на языке TypeScript, который использует концепцию модульности и подход, основанный на Angular. Архитектура сервера NestJS построена на нескольких основных концепциях, обеспечивающих гибкость, масштабируемость и легкость в разработке.

### **2.2.1 Модули**

NestJS разделяет приложение на модули, которые представляют собой группы связанных компонентов, сервисов, контроллеров и провайдеров. Каждый модуль имеет свою область видимости и определяет связи между компонентами.

В NestJS модуль представляет собой организационную единицу приложения, которая объединяет компоненты, провайдеры и контроллеры в логическую группу. Модуль служит для организации кода и определения зависимостей между компонентами приложения.

Модули в NestJS обычно отражают различные функциональности или части приложения, такие как пользователи, заказы, аутентификация и другие. Каждый модуль имеет свою собственную область видимости, что позволяет изолировать компоненты и провайдеры внутри модуля и предоставлять их только внутри этого модуля или его дочерних модулей.

### **2.2.2 Контроллеры**

NestJS контроллер представляет собой класс, который отвечает за обработку HTTP-запросов. Контроллер определяет различные маршруты и их обработчики для выполнения определенных операций или возврата данных клиенту.

Основная задача контроллера - принять входящий запрос, обработать его и вернуть ответ. Контроллеры в NestJS обычно организованы вокруг определенной функциональности или ресурса приложения.

### **2.2.3 Провайдеры**

Провайдеры предоставляют различные сервисы и компоненты приложения. Они могут включать сервисы базы данных, сервисы для работы с внешними API, сервисы для бизнес-логики и т. д. Провайдеры инъектируются в контроллеры или другие провайдеры для реализации зависимостей и повышения переиспользуемости кода.

В NestJS провайдеры являются основным механизмом инверсии управления (Inversion of Control, IoC) и внедрения зависимостей (Dependency Injection, DI). Они предоставляют экземпляры классов и других значений, которые могут быть инжектированы в другие компоненты приложения, такие как контроллеры, сервисы и другие провайдеры.

Провайдеры в NestJS могут быть классами, фабриками или значениями, которые регистрируются в модуле и доступны для инжектирования в другие компоненты приложения. Они играют важную роль в обеспечении модульности, повторного использования и тестируемости кода.

### **2.2.4 Middleware**

NestJS поддерживает промежуточное программное обеспечение (middleware), которое позволяет обрабатывать запросы до и после обработки контроллерами. Middleware может выполнять дополнительную обработку запросов, применять авторизацию, логирование и другие операции.

Middleware в NestJS позволяет добавлять дополнительную функциональность и влиять на обработку запросов и ответов в приложении. Они предоставляют гибкую архитектуру для обработки различных аспектов запросов, таких как логирование, аутентификация, авторизация и другие операции.

### **2.2.5 Фильтры и Интерсепторы**

Фильтры позволяют обрабатывать исключения и ошибки на уровне приложения. Они могут выполнять преобразования данных или возвращать определенные ответы в случае ошибок. Интерсепторы позволяют изменять входящие и исходящие данные, добавлять дополнительную логику перед и после обработки запросов.

Фильтры (Filters) в NestJS - это компоненты, которые позволяют обрабатывать исключения и другие исходящие данные до того, как они будут отправлены клиенту. Они являются частью обработки исключений и фильтрации и предоставляют механизм для преобразования и изменения ответов сервера.

Интерсепторы (Interceptors) в NestJS - это компоненты, которые позволяют перехватывать и изменять запросы до их обработки обработчиками маршрутов (route handlers) и ответы до их отправки клиенту.

## **2.3 Диаграмма развертывания приложения**

Диаграмма развертывания – это тип диаграммы UML (Unified Modeling Language), который показывает физическое размещение компонентов системы и связи между ними в рамках аппаратной инфраструктуры. Диаграмма развертывания может быть использована для представления как физических, так и логических элементов системы, таких как серверы, устройства хранения данных, сетевые устройства, приложения и т.д. Компоненты системы отображаются в виде узлов, а связи между ними – в виде линий. Диаграмма развертывания позволяет понять, как компоненты системы взаимодействуют между собой и как они связаны с аппаратным обеспечением. Диаграмма развертывания приложения представлена на рисунке 2.2.

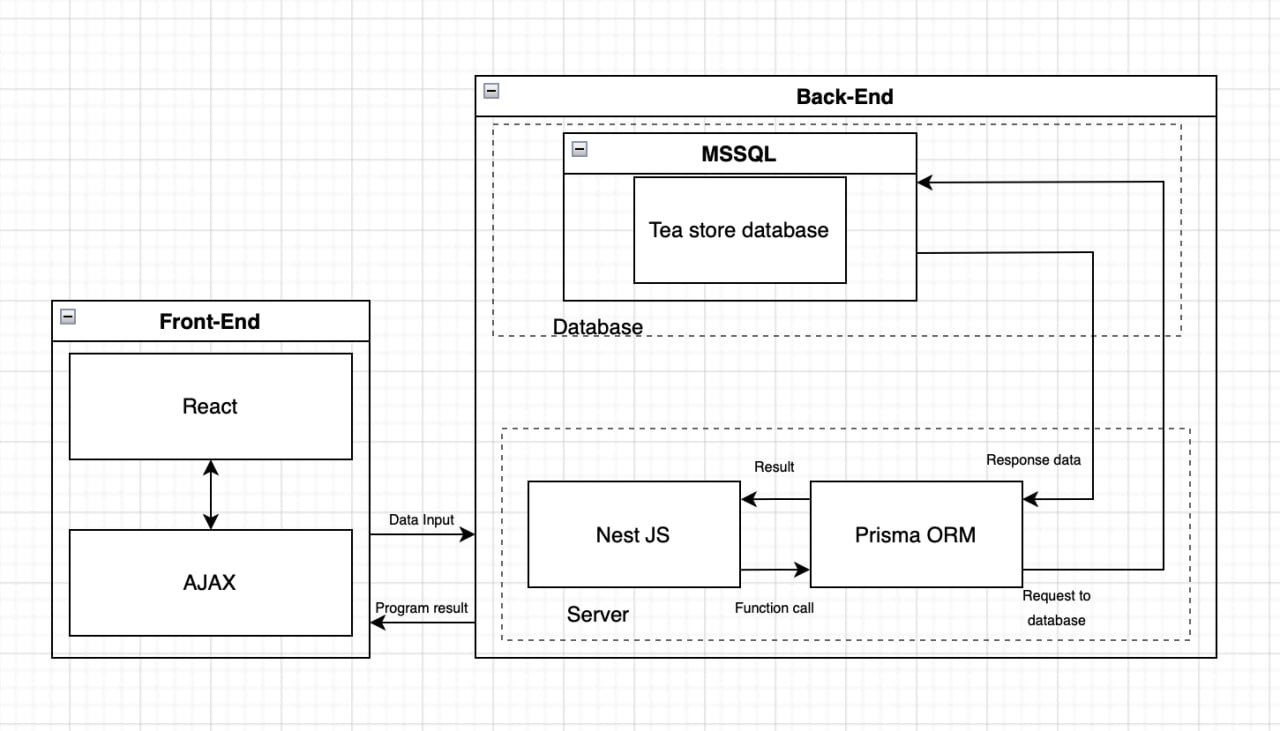


Рисунок 2.2 – Диаграмма развертывания приложения

В данной диаграмме представлены основные компоненты и их взаимодействие в развертываемом веб-приложении:

Серверная часть приложения, разработана с использованием фреймворка NestJS. Он обрабатывает входящие HTTP/HTTPS запросы от клиентского приложения и осуществляет маршрутизацию к соответствующим контроллерам.

Контроллеры и модели входят в состав серверного приложения на NestJS. Бизнес-логика реализована в сервисах, контроллеры определяют точки входа для обработки запросов, а модели представляют сущности данных и взаимодействуют с СУБД.

Prisma – это объектно-реляционное отображение (ORM), которое облегчает взаимодействие с базой данных. Оно предоставляет средства для создания, чтения, обновления и удаления данных в СУБД MSSQL. Prisma позволяет выполнять запросы и манипулировать данными с использованием типобезопасного API.

СУБД MSSQL используется для хранения данных приложения. Она обрабатывает запросы на чтение и запись данных, предоставляет эффективное хранение и управление информацией.

Взаимодействие происходит следующим образом:

Клиентское приложение отправляет HTTP-запросы на Web-сервер (NestJS) для выполнения различных операций, таких как получение данных, создание, обновление или удаление информации.

Web-сервер принимает запросы, маршрутизирует их к соответствующим контроллерам и передает данные в модели для обработки и взаимодействия с СУБД MSSQL.

Модели взаимодействуют с СУБД MSSQL, используя Prisma для выполнения ORM-запросов. Prisma предоставляет удобный и типобезопасный интерфейс для работы с данными.

СУБД MSSQL выполняет запросы к базе данных, обрабатывает данные и возвращает результаты на Web-сервер.

Web-сервер отправляет ответы на клиентское приложение, которое отображает полученные данные или обрабатывает результаты операций.

Таким образом, данная диаграмма развертывания иллюстрирует взаимодействие между компонентами и поток данных в веб-приложении, написанном на NestJS и React с использованием СУБД MSSQL и Prisma.

Web-приложение развернуто на локальной машине, для общения между клиентом и сервером используются протоколы WebSocket и Https.

## **2.4 Диаграмма вариантов использования**

Диаграмма вариантов использования – это тип диаграммы UML, который показывает взаимодействие между системой и ее окружением, описывая различные сценарии использования системы со стороны ее пользователей или других систем.

Web-приложение, в зависимости от роли, предоставляет пользователю функциональные возможности, которые показаны в use-case диаграмме, которая представлена на рисунке 2.3.

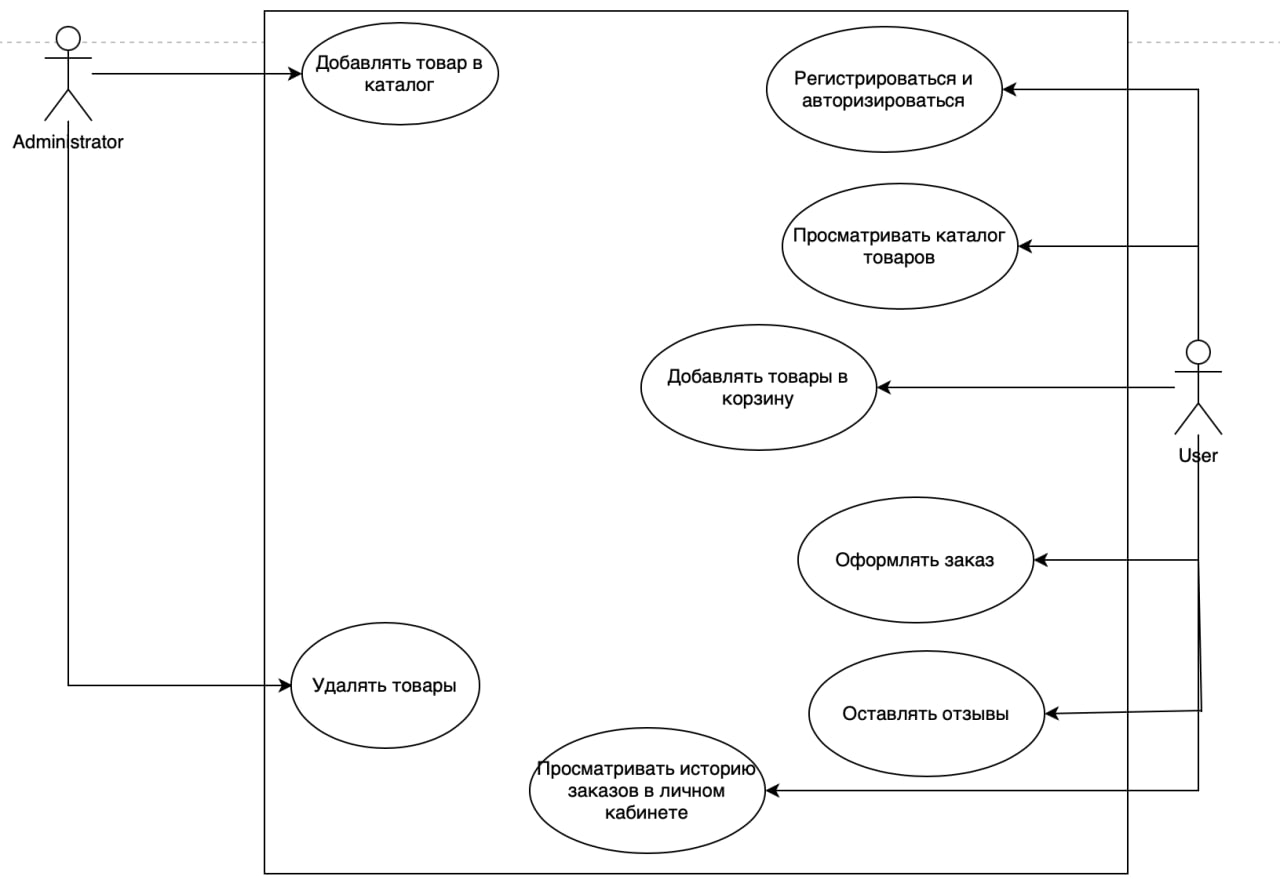


Рисунок 2.3 – Use-case диаграмма приложения

Для того, чтобы пользователь мог использовать возможности приложения, он должен быть зарегистрирован в нем. Для этого у пользователя есть возможность регистрации. Кроме того, у администратора есть функционал для добавления новых товаров в магазин, и их удаления.

Каждый пользователь приложения имеет возможность осуществлять переход к конкретному элементу системы, чтобы добавлять товар в корзину, оставлять отзывы и оформлять заказ.

# 3 Разработка web-приложения

## **3.1 Реализация серверной части**

### **3.1.1 Структура проекта**

Проект был разработан с помощью фреймворка Nest [3]. Для реализации встроенной архитектуры фреймворка файлы программы были распределены по соответствующим директориям и реализовали соответствующие функции. Разделение проекта на логические модули представлено на рисунке 3.1.

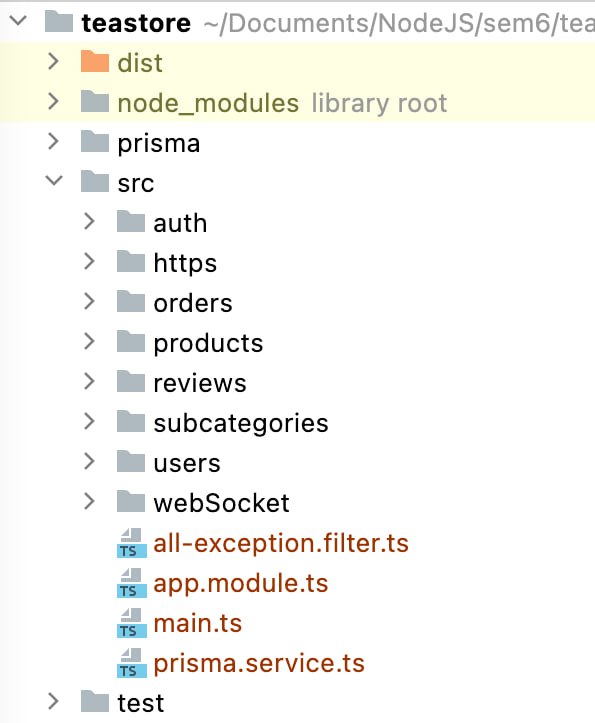


Рисунок 3.1 – Структура проекта

В директории orders находятся контроллер, сервис, сущность, модуль, и классы для манипуляций с заказами, содержимое папки представлено на рисунке 3.2.

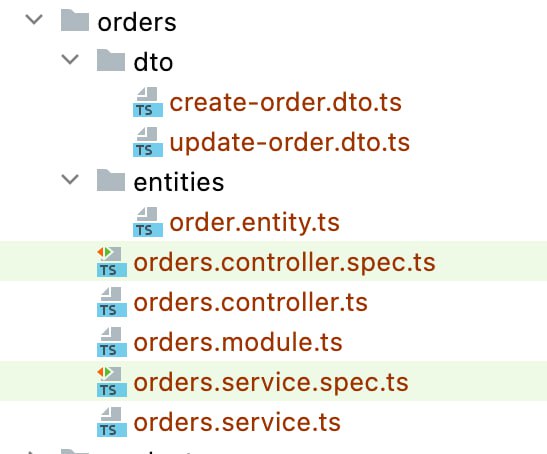


Рисунок 3.2 – Структура папки orders

В директории products находятся контроллер, сервис, сущность, модуль, и классы для манипуляций с продуктами, содержимое папки представлено на рисунке 3.3.

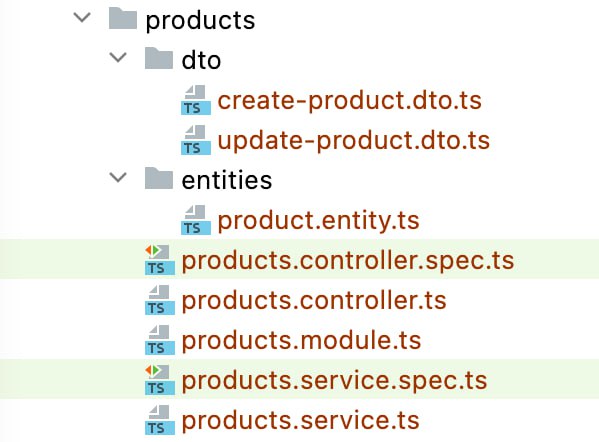


Рисунок 3.3 – Структура папки products

В директории auth находятся контроллер, сервисы, модуль и декоратор для обеспечения безопасности приложения, содержимое папки представлено на рисунке 3.4.

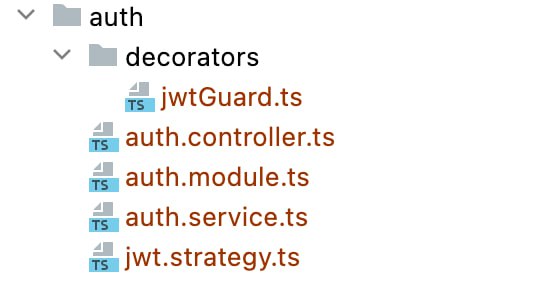


Рисунок 3.4 – Структура папки auth

В директории https находятся файлы для подключения связывания приложения по протоколу https безопасности приложения, содержимое папки представлено на рисунке 3.5.



Рисунок 3.5 – Структура папки https

Таким образом была рассмотрена структура серверной части проекта.

### **3.1.2 Реализация взаимодействия с prisma ORM**

Для взаимодействия с базой данных был выбран пакет prisma, чтобы его использовать было необходимо создать класс, который наследуется от PrismaClient, код представлен в листинге 3.1.

@Injectable()

export class PrismaService extends PrismaClient {}

Листинг 3.1 – Реализация класса PrismaService

Модели для работы с базой данных находятся в файле schema.prisma, который был сгенерирован самой призмой, путём вытягивания данных из базы данных. Файл schema.prisma представлен в листинге 3.2.

model orders {

order\_id Int @id(map: "PK\_\_orders\_\_46596229C9B5CB96") @default(autoincrement())

user\_id Int

product\_id Int

quantity Int

price Decimal @db.Decimal(10, 2)

orderDate String @db.VarChar(250)

isBuying String @default("false", map: "DF\_\_orders\_\_isBuying\_\_47DBAE45") @db.VarChar(250)

users users @relation(fields: [user\_id], references: [user\_id], onDelete: Cascade, onUpdate: NoAction, map: "orders\_ibfk\_1")

products products @relation(fields: [product\_id], references: [product\_id], onDelete: Cascade, onUpdate: NoAction, map: "orders\_ibfk\_2")

}

model products {

product\_id Int @id(map: "PK\_\_products\_\_47027DF594358D23") @default(autoincrement())

product\_name String @unique(map: "uproduct\_name") @db.VarChar(50)

description String @db.VarChar(500)

productImage String @db.VarChar(255)

price Decimal @db.Decimal(10, 2)

brewingTemperature String @db.VarChar(100)

orders orders[]

subcategories subcategories @relation(fields: [category\_id], references: [category\_id], onDelete: Cascade, onUpdate: NoAction, map: "products\_ibfk\_1")

}

Листинг 3.2 – Файл schema.prisma

Таким образом была рассмотрена реализация взаимодействия с prisma ORM.

### **3.1.3 Реализация слоя контроллеров**

В качестве примера реализации будет рассматриваться контроллер для заказов. В аннотации @Controller указывается url по которому мы будем обращаться к контроллеру. Далее идут аннотации для методов, которые будут отправляться с клиента на сервер. Также над контроллером используется декоратор @UseGuards(), который обеспечивает безопасный доступ к ресурсам. Реализацию контроллера приведена в листинге 3.3.

@Resolve@Controller('orders')

@UseGuards(JwtAuthGuard)

export class OrdersController {

constructor(private readonly ordersService: OrdersService, private readonly authService: AuthService) {}

@Post('/addToCart/:token')

async addToCart(@Body() createOrderDto: CreateOrderDto, @Param('token') token: string) {

const payload = await this.authService.getPayloadFromToken(token);

createOrderDto.user\_id = +payload.sub;

return await this.ordersService.createInCart(createOrderDto);

}

@Get('ord/:token')

async findAllInCabinet(@Param('token') token: string) {

const payload = await this.authService.getPayloadFromToken(token);

return await this.ordersService.findAllOrders(+payload.sub);

}

@Patch('uptToBuy/:token')

async updateToBuy(@Param('token') token: string, @Body() updateOrderDto: UpdateOrderDto) {

const payload = await this.authService.getPayloadFromToken(token);

return await this.ordersService.updateOrder(+payload.sub, updateOrderDto);

}

@Delete('removeCart/:id/:token')

async removeCart(@Param('id') id: string, @Param('token') token: string) {

const payload = await this.authService.getPayloadFromToken(token);

return await this.ordersService.removeFromCart(+id, +payload.sub);

}

}

Листинг 3.3 – Реализация контроллера

Таким образом была рассмотрена реализация слоя контроллеров.

### **3.1.4 Реализация слоя сервисов**

В качестве примера реализации будет рассматриваться сервис для заказов. Методы сервиса позволяют работать с базой данных, используя пакет prisma. Частичная реализация сервиса приведена в листинге 3.4.

export class OrdersService {

constructor(private readonly prisma: PrismaService) {

}

async create(createOrderDto: CreateOrderDto) {

return await this.prisma.orders.create({

data: {

...createOrderDto

}

});

}

async findAll() {

return await this.prisma.orders.findMany();

}

async findOne(id: number) {

return await this.prisma.orders.findUnique({

where: {

order\_id: id

}

});

}

async update(id: number, updateOrderDto: UpdateOrderDto) {

return await this.prisma.orders.update({

where: {

order\_id: id

},

data: {

...updateOrderDto

}

});

}

async remove(id: number) {

return await this.prisma.orders.delete({

where: {

order\_id: id

}

});

}

Листинг 3.4 – Реализация сервиса OrdersService

Таким образом была рассмотрена реализация слоя сервисов.

### **3.1.5 Реализация слоя модулей**

В качестве примера реализации будет рассматриваться модуль для заказов. Реализация модуля приведена в листинге 3.5.

@Module({

imports: [AuthModule],

controllers: [OrdersController],

providers: [OrdersService, PrismaService],

})

export class OrdersModule {}

Листинг 3.5 – Реализация модуля OrdersModule

Таким образом была рассмотрена реализация слоя модулей.

## **3.2 Реализация клиентской части**

Клиентская часть реализована с использованием библиотеки React [3]. React – это библиотека JavaScript, которая используется для создания пользовательских интерфейсов (UI) веб-приложений. Она позволяет разработчикам создавать динамические и масштабируемые приложения с помощью компонентного подхода.

### **3.2.1 Структура проекта**

Проект был реализован с помощью JavaScript библиотеки React [4]. При реализации клиентской части, использовался модульный подход, пользовательский интерфейс разбит на отдельные, переиспользуемые компоненты, которые находятся в папке src. Структура react-проекта client представлена на рисунке 3.6.

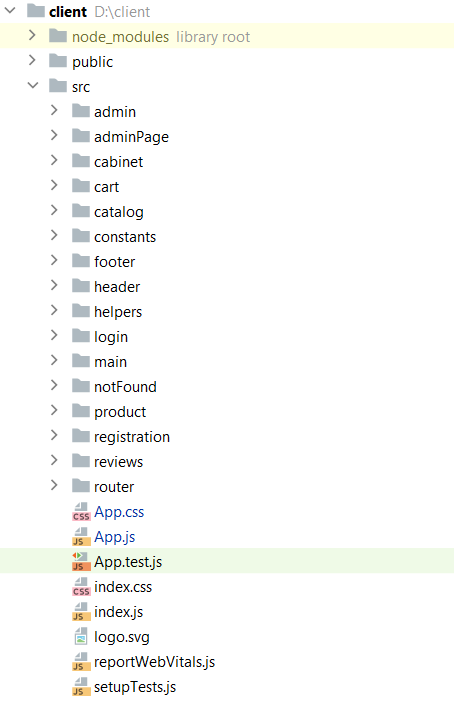


Рисунок 3.6 – Структура проекта client

В директории src находятся все react-компоненты, используемые в приложении, папка helpers содержит часть реализации логики приложения, в constants находятся постоянные переменные приложения.

### **3.2.2 Компоненты**

Реакт (React) компонент – это основная строительная единица веб-приложений, разработанных с использованием библиотеки React. Компонент представляет собой независимую и переиспользуемую часть пользовательского интерфейса, которая может содержать разметку (HTML), логику и стили.

Реакт компоненты являются разделенными, изолированными блоками кода, которые можно использовать для создания сложных интерфейсов. Каждый компонент может иметь свое состояние (state) и свои свойства (props), которые определяют его поведение и внешний вид.

Реакт компоненты обладают свойством повторного использования, что позволяет создавать небольшие и самодостаточные блоки кода и собирать их вместе для создания более сложных интерфейсов. Это позволяет упростить разработку, обслуживание и масштабирование веб-приложений на React.

Для того, чтобы связать react-приложение с сервером в папку к каждому компоненту был добавлен js файл с функциями, которые посылают запросы на сервер. Реализация одного из таких файлов, а именно cabinetFunctions.js приведена в листинге 3.6.

export function getOrders() {

return fetch(${API\_URL}/orders/ord/${window.localStorage.getItem('token') || ''}, {

method: 'GET',

headers: { 'Accept': 'application/json', 'Authorization': Bearer ${window.localStorage.getItem('token') || ''}}

});

}

export function getUser() {

return fetch(${API\_URL}/users/${window.localStorage.getItem('token') || ''}, {

method: 'GET',

headers: { 'Accept': 'application/json', 'Authorization': Bearer ${window.localStorage.getItem('token') || ''}}

});

}

Листинг 3.6 – Реализация файла cabinetFunctions.js

Сам же компонент отображает верстку и монтирует полученные данные в html страницу, с помощью метода map(). Хук useEffect используется для рендера компонента при загрузке страницы, динамически отображая данные из базы данных на странице. Для парсинга респонса применяетсяя метод responseHandler из папки helpers. Реализация компонента Cabinet представлена в листинге 3.7.

export function getOrders() {

export default function Cabinet() {

const [data, setData] = useState([]);

useEffect(() => {

async function fetchData() {

const response = await getOrders();

setData(await responseHandler(response));

}

fetchData();

}, [])

return (

<div className="cabinetMain">

<Header/>

<div className="cabinetContent">

<div>Заказанные товары:</div>

<div className="orderCardBuying">

{

data.map((item, index) => (

<div className="prCardBuying" key={index}>

<div className="ordIndo">

<img src={'./' + item.products.productImage} alt="not found"/>

<div className="prInfoBuying">

<div className="prMainInfoBuying">

<div>{item.products.product\_name}</div>

<div>{item.price} BYN</div>

<div>100гр.</div>

</div>

</div>

</div>

</div>

))

}

</div>

</div>

<Footer/>

</div>

)

}

Листинг 3.7 – Реализация компонента Cabinet

Таким образом была рассмотрена реализация компонентов react-приложения.

# 4 Тестирование web-приложения

## **4.1 Тестирование серверной части**

Для тестирования серверной части приложения было использовано ручное тестирование.

### **4.1.1 Ручное тестирование**

Ручное тестирование веб-приложения - это процесс проверки функциональности, пользовательского интерфейса, производительности и других аспектов веб-приложения путем ручного взаимодействия с ним.

Во время ручного тестирования тестировщик выполняет следующие действия:

Проверка функциональности: Тестировщик проверяет, работают ли все функции веб-приложения согласно заданным требованиям. Он проверяет, что каждая функция выполняет свою задачу корректно и без ошибок.

Проверка пользовательского интерфейса: Тестировщик проверяет, что пользовательский интерфейс (UI) веб-приложения выглядит правильно и функционирует должным образом. Он проверяет расположение элементов, взаимодействие с элементами, отображение ошибок и другие аспекты интерфейса.

Проверка совместимости: Тестировщик проверяет, как веб-приложение работает на разных веб-браузерах, операционных системах и устройствах. Он проверяет, что приложение корректно отображается и функционирует на различных платформах.

Проверка производительности: Тестировщик оценивает производительность веб-приложения, например, проверяет скорость загрузки страниц, время отклика и обработки запросов, а также его способность обрабатывать большую нагрузку.

Проверка безопасности: Тестировщик проверяет уязвимости безопасности веб-приложения, такие как защита от взлома, проверка прав доступа, обработка и валидация ввода данных и другие механизмы безопасности.

Воспроизведение ошибок: Если тестировщик обнаруживает ошибку во время тестирования, он должен воспроизвести ее и заполнить соответствующий отчет о проблеме. Это позволяет разработчикам исправить ошибку и улучшить качество приложения.

Ручное тестирование веб-приложения является важной частью процесса разработки и помогает обеспечить качество и надежность приложения перед его выпуском. Однако, для повышения эффективности тестирования и сокращения времени, часто применяются автоматизированные тесты, которые могут автоматически выполнять множество тестовых сценариев.

Для проверки взаимодействия клиентской и серверной части приложения было проведено ручное тестирование. Будут предоставлены скриншоты с описанием проведенных тестов.

При попытке регистрации и указания не валидного адреса электронной почты будет появляться соответствующее предупреждение, которое приведено на рисунке 4.1.

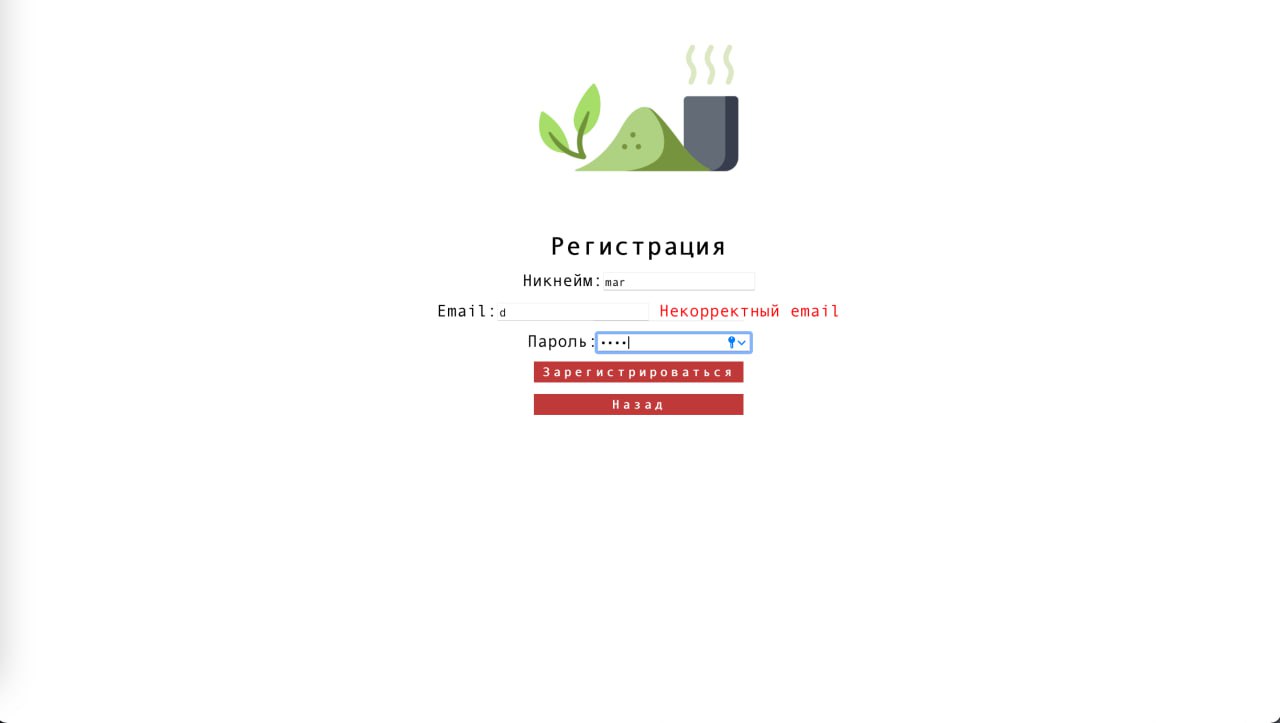


Рисунок 4.1 – Результат ввода некорректного адреса электронной почты при регистрации

При попытке авторизации и указания не валидного адреса электронной почты будет появляться соответствующее предупреждение, которое приведено на рисунке 4.2.

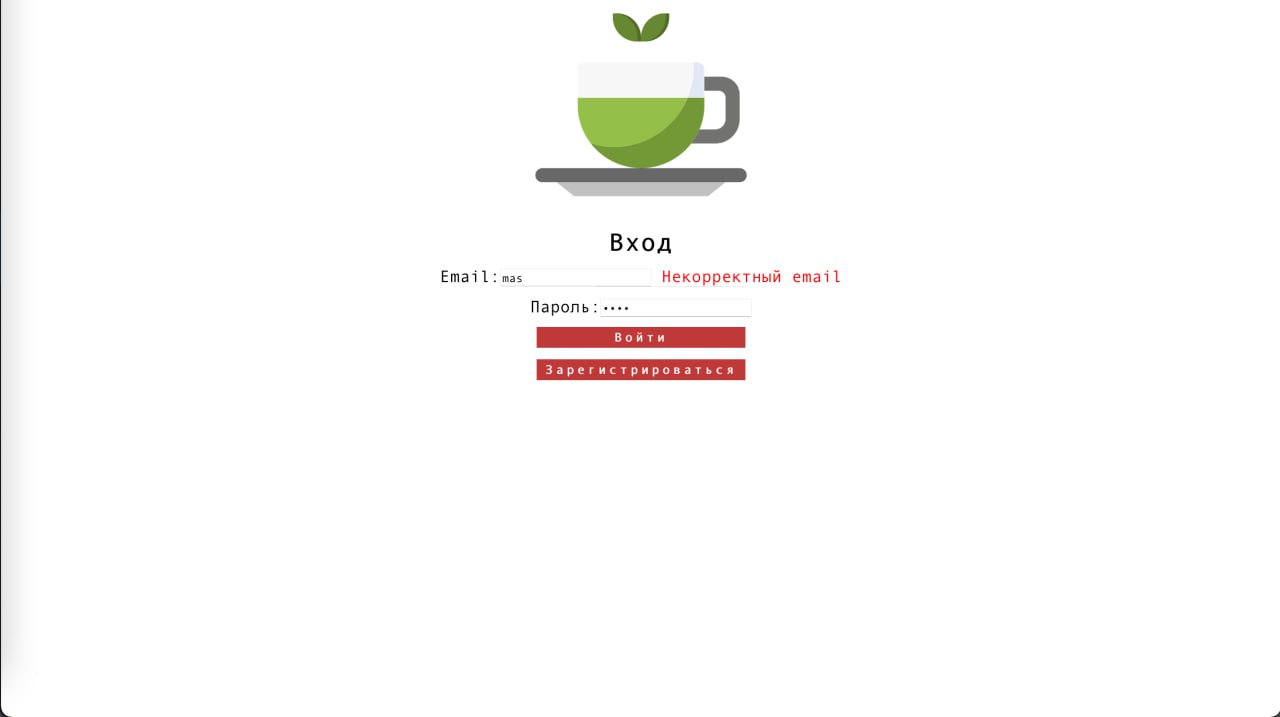


Рисунок 4.2 – Результат ввода некорректного адреса электронной почты при авторизации

При попытке авторизации и указания не существующего адреса электронной почты будет появляться соответствующее предупреждение, которое приведено на рисунке 4.3.

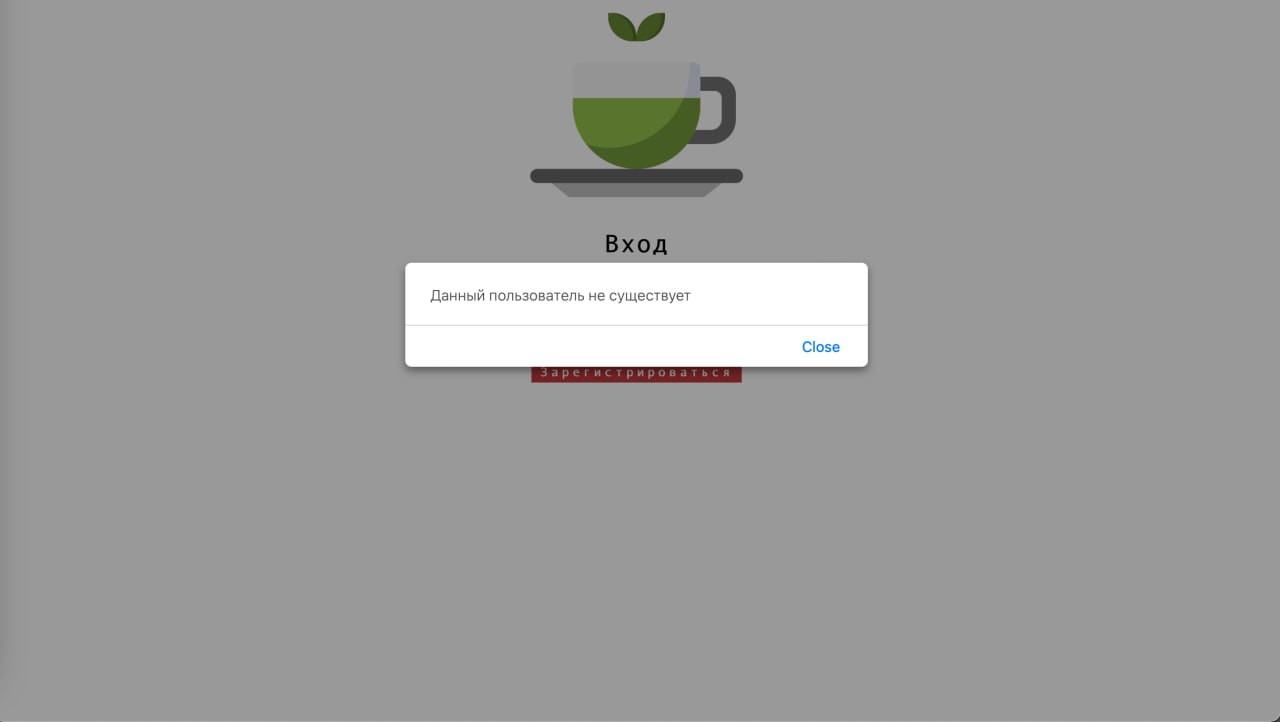


Рисунок 4.3 – Результат ввода несуществующего адреса электронной почты при авторизации

Если токен был удален или изменен, то следующий запрос на сервер перенаправит пользователя на страницу авторизации.

При попытке сделать запрос без авторизации будет кидать на 404 страницу, как показано на рисунке 4.4.

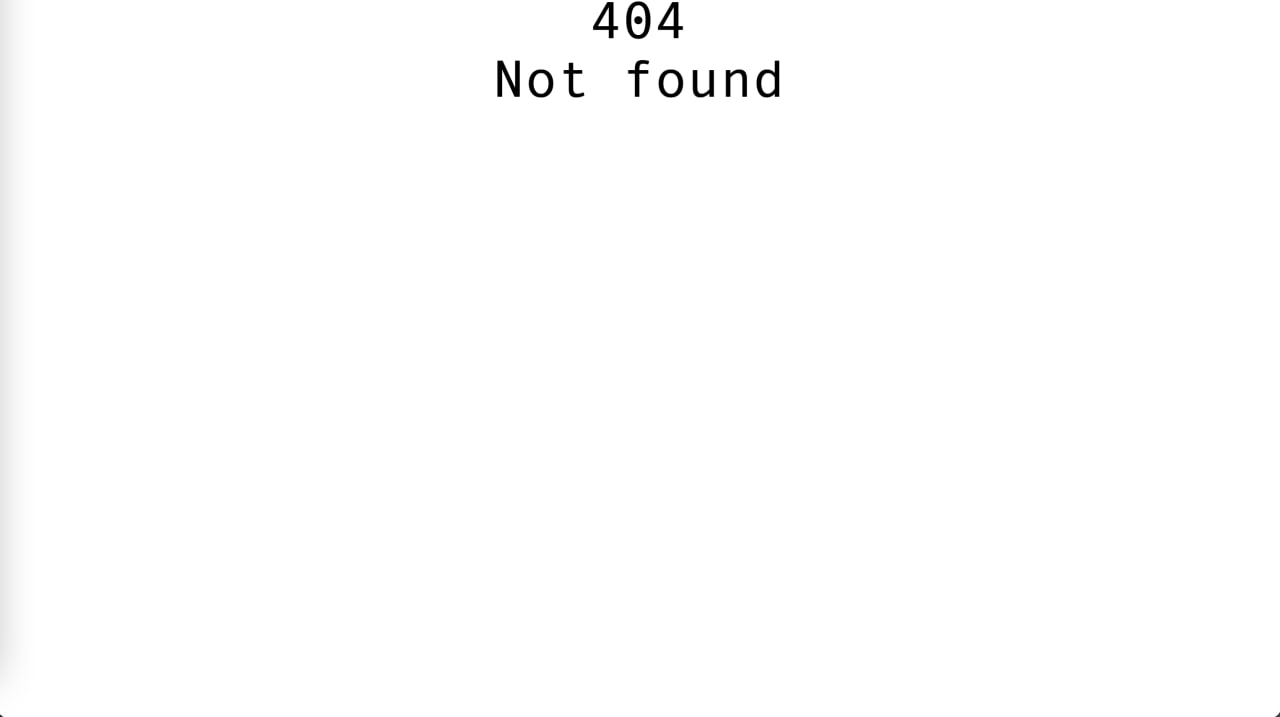


Рисунок 4.4 – Результат обращения к существующему пути без авторизации

При попытке сделать запрос без авторизации будет кидать на 404 страницу, как показано на рисунке 4.5.

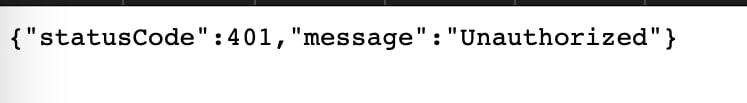


Рисунок 4.4 – Результат обращения к серверу через браузер без авторизации

Таким образом, было проведено ручное тестирование приложения.

# 5 Руководство пользователя

Первое, что встречает пользователь, – страница для входа в приложение, приведена на рисунке 5.1.



Рисунок 5.1 – Страница входа в приложение

Чтобы пользователь попал в систему, ему необходимо ввести свой логин и пароль. Если у пользователя нет аккаунта, он может его зарегистрировать. Страница регистрации представлена на рисунке 5.2.

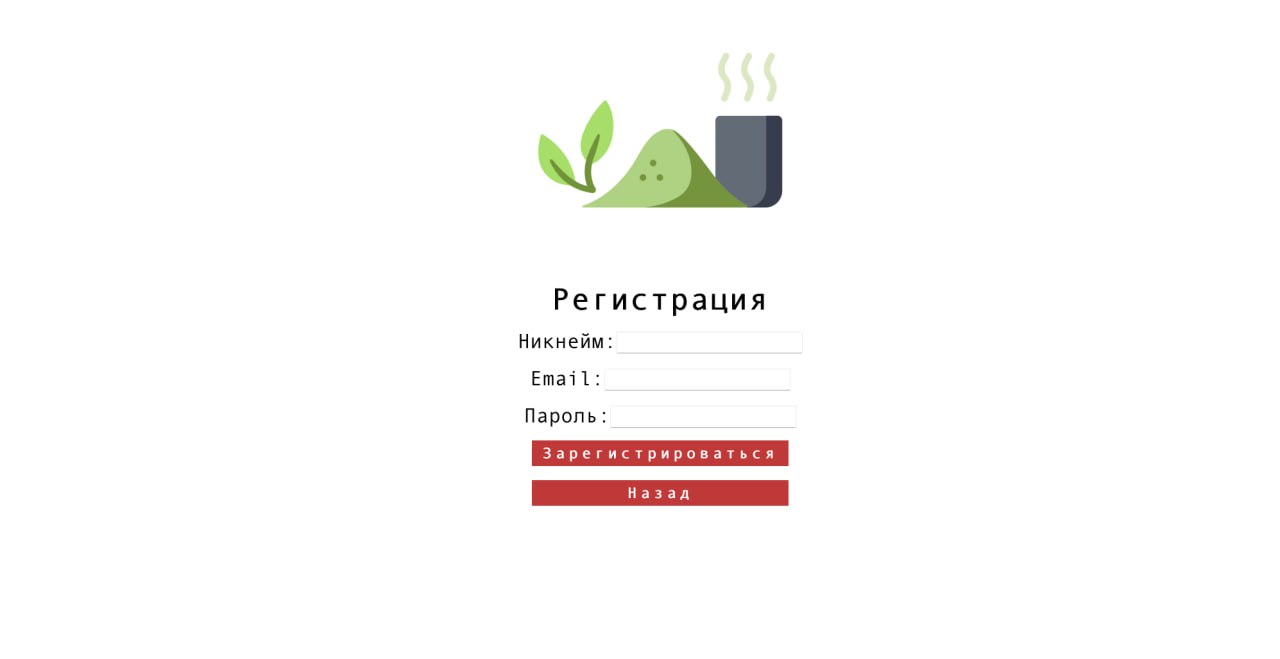


Рисунок 5.2 – Страница регистрации

После входа в приложение, пользователь попадает на главную страницу. На этой странице пользователь может прочитать информацию о магазине и с помощью навигационной панели использовать функционал приложения. Главная страница представлена на рисунке 5.3.



Рисунок 5.3 – Главная страница

У пользователя есть возможность посмотреть каталог и добавить понравившийся товар в корзину, каталог представлен на рисунке 5.4.

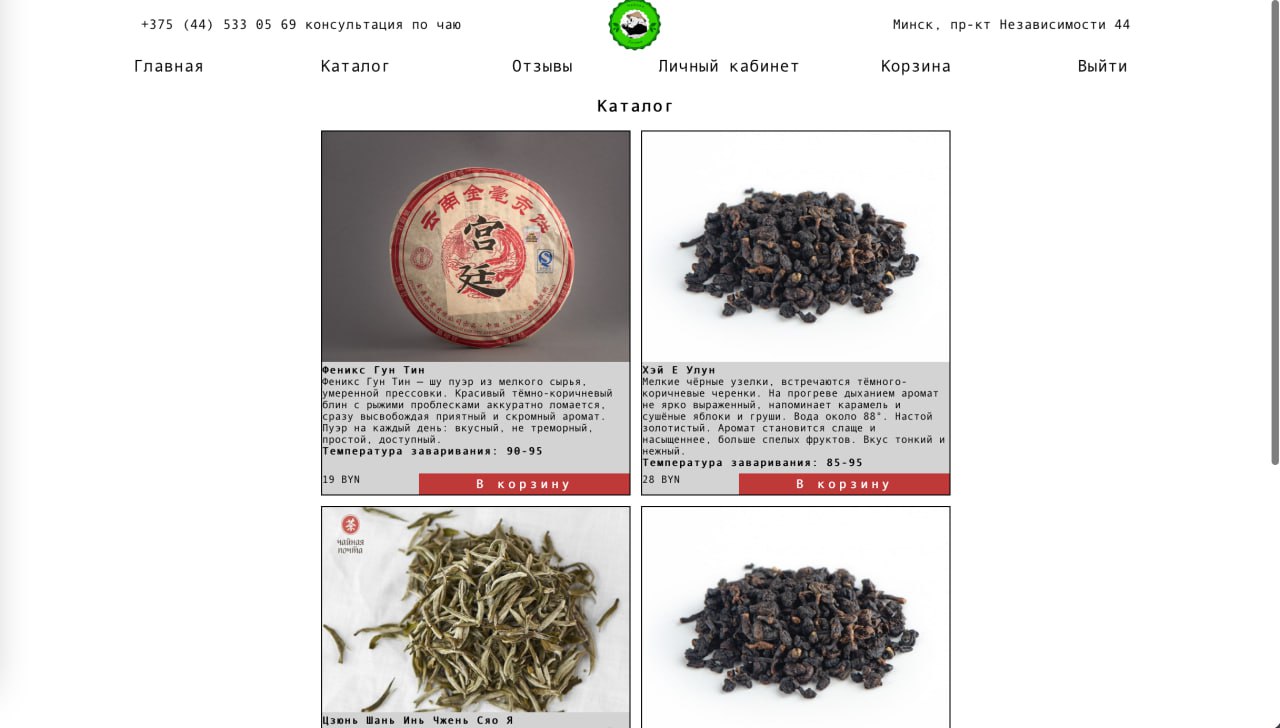


Рисунок 5.4 – Каталог

Попав в корзину, пользователь увидит товары, которые он добавил в корзину , она представлена на рисунке 5.5.

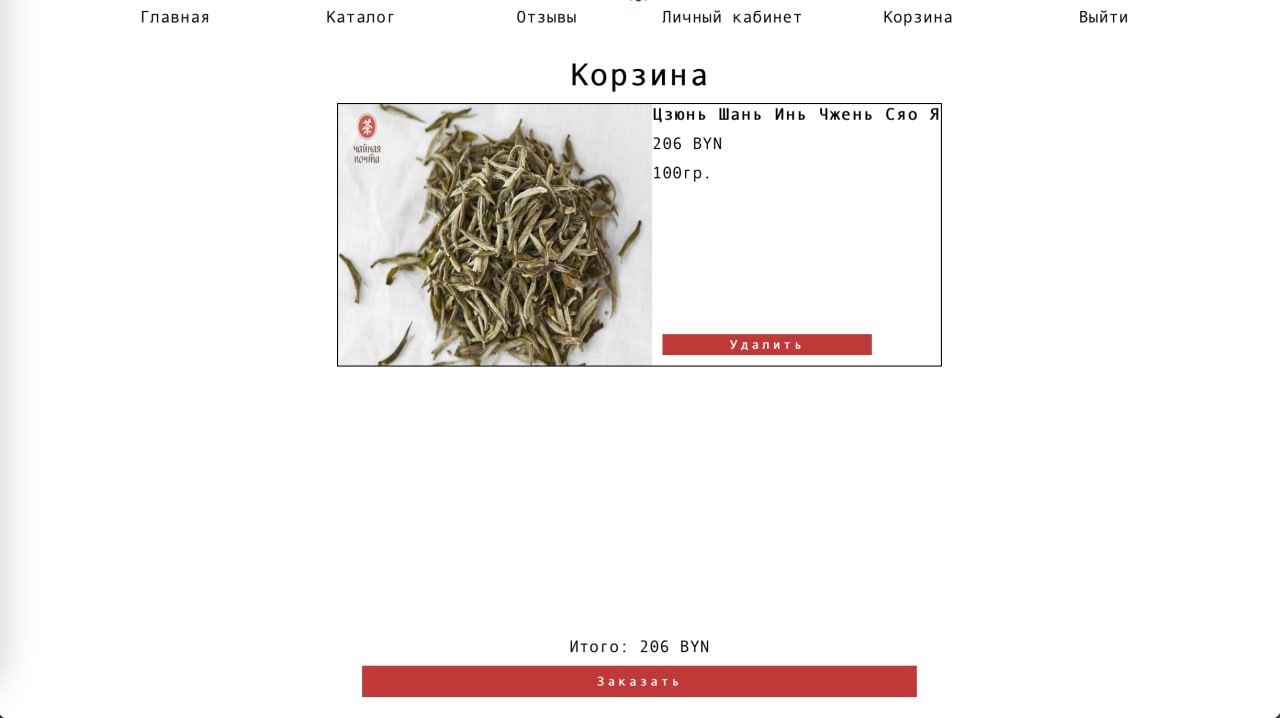


Рисунок 5.5 – Корзина

В личном кабинете пользователь может увидеть товары, которые он заказал из корзины. Личный кабинет представлен на рисунке 5.6.

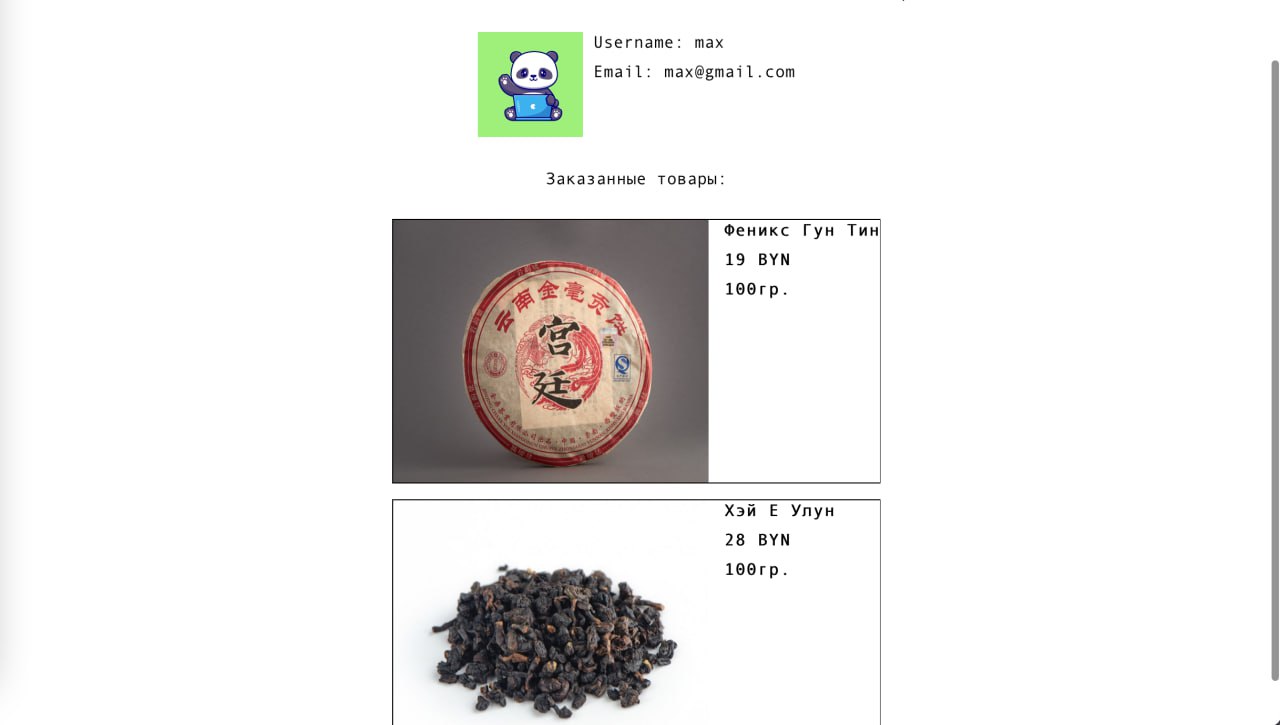


Рисунок 5.6 – Личный кабинет

Следующая страница – отзывы, пользователь может оставлять свои отзывы и просматривать отзывы других юзеров. Страница отзывов приведена на рисунке 5.7.

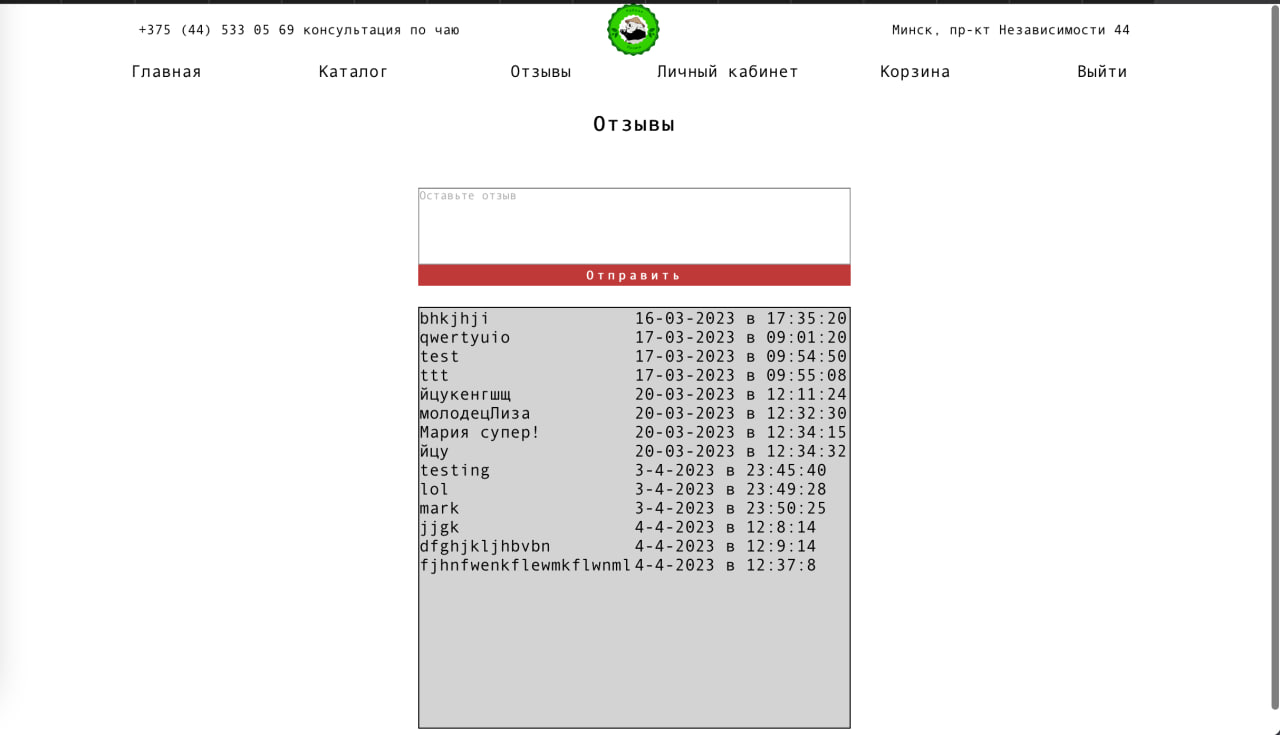


Рисунок 5.7 – Страница отзывов

В области с отзывами отображаются элементы содержащие сообщение, и время создания отзыва.

Пользователь имеет возможность обмена сообщениями с другими пользователями в режиме реального времени, чат представлен на рисунке 5.8.

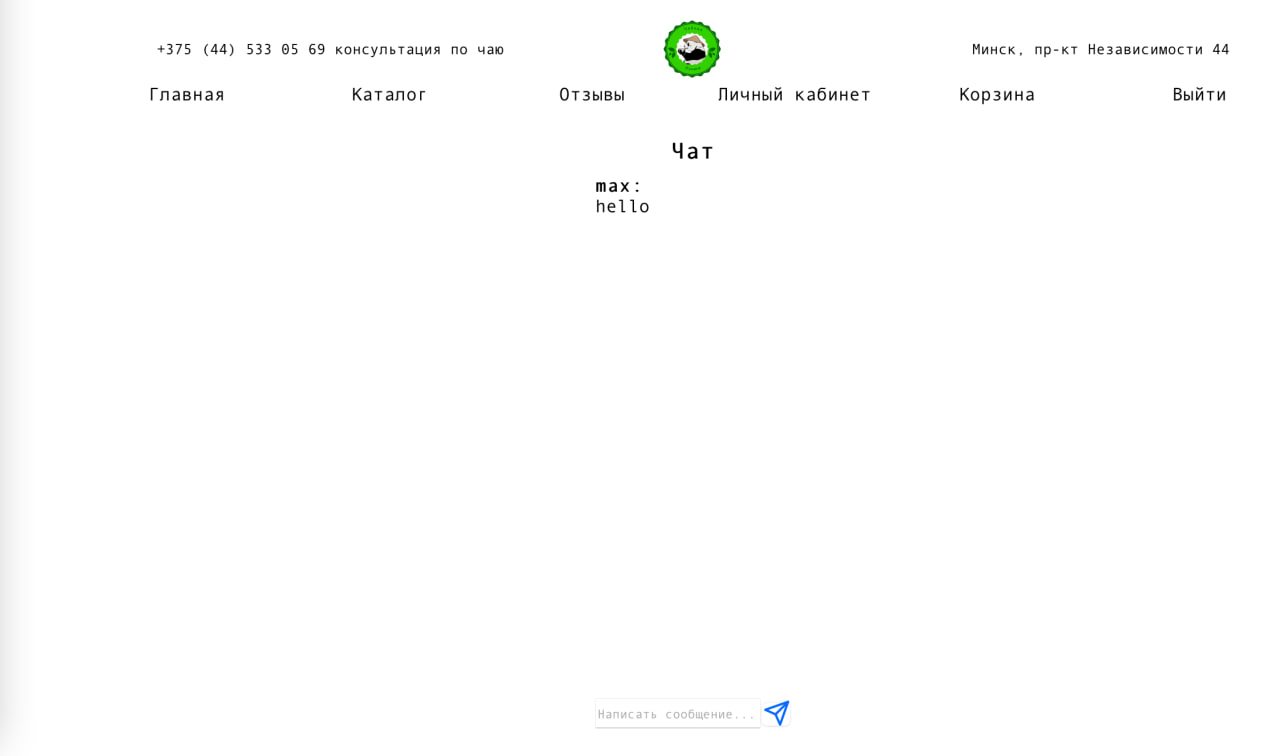


Рисунок 5.8 – Чат

Таким образом, в этой главе было дано краткое руководство пользователя для быстрого старта в использовании данного приложения.

# Заключение

В данном курсовом проекте были рассмотрены основные аналоги, были рассмотрены их преимущества и недостатки, изучены возможности рассматриваемых веб-приложений. Произведен обзор теоретического материала по теме курсового проекта.

Веб-приложение выполнено с использованием асинхронного программирования. Оно реализовано на языке JavaScript на платформе Node.js, что делает его кроссплатформенным. В качестве системы управления базой данных использована MSSQL. Для разработки клиентской части использовалась библиотека React. Разработанный программный продукт предоставляет API (REST).

Разработанное веб-приложение обладает следующим функционалом:

* регистрация и авторизация пользователей;
* поддержка ролей администратора и пользователя;
* просмотр каталога;
* просмотр отзывов;
* добавление товара в корзину;
* заказ товаров из корзины;
* удаление товаров из корзины;
* добавление новых товаров.

При составлении руководства пользователя была подробно описана работа с веб-приложением. Так же было проведено тестирование (ручное), показавшее что оно соблюдает все ранее установленные требования.

При разработке программного средства была спроектирована структура базы данных, спроектирован и разработан интерфейс для пользователя и администратора.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает, верно, поставленная цель достигнута, а задачи выполнены.

# Список используемых источников

1. teashop.by [Электронный ресурс] / teashop.by. – Режим доступа: https://www.teashop.by. – Дата доступа: 18.03.2023.
2. teaplantation.by [Электронный ресурс] / teaplantation.by. – Режим доступа: https://teaplantation.by. – Дата доступа: 18.03.2023.
3. Nest documentation [Электронный ресурс] / nestjs.com. – Режим доступа: https://nestjs.com. – Дата доступа: 18.04.2023.
4. React documentation [Электронный ресурс] / legacy.reactjs.org. –  Режим доступа: https://legacy.reactjs.org/docs – Дата доступа: 19.05.2023.

# ПРИЛОЖЕНИЕ А

Реализация класса OrdersController

@Controller('orders')

@UseGuards(JwtAuthGuard)

export class OrdersController {

constructor(private readonly ordersService: OrdersService, private readonly authService: AuthService) {}

@Post()

async create(@Body() createOrderDto: CreateOrderDto) {

return await this.ordersService.create(createOrderDto);

}

@Post('/addToCart/:token')

async addToCart(@Body() createOrderDto: CreateOrderDto, @Param('token') token: string) {

const payload = await this.authService.getPayloadFromToken(token);

createOrderDto.user\_id = +payload.sub;

return await this.ordersService.createInCart(createOrderDto);

}

@Get()

async findAll() {

return await this.ordersService.findAll();

}

@Get(':id')

async findOne(@Param('id') id: string) {

return await this.ordersService.findOne(+id);

}

@Get('ord/:token')

async findAllInCabinet(@Param('token') token: string) {

const payload = await this.authService.getPayloadFromToken(token);

return await this.ordersService.findAllOrders(+payload.sub);

}

@Get('ordInCart/:token')

async findAllInCart(@Param('token') token: string) {

const payload = await this.authService.getPayloadFromToken(token);

return await this.ordersService.findAllOrdersInCart(+payload.sub);

}

@Patch(':id')

async update(@Param('id') id: string, @Body() updateOrderDto: UpdateOrderDto) {

return await this.ordersService.update(+id, updateOrderDto);

@Patch('uptToBuy/:token')

async updateToBuy(@Param('token') token: string, @Body() updateOrderDto: UpdateOrderDto) {

const payload = await this.authService.getPayloadFromToken(token);

return await this.ordersService.updateOrder(+payload.sub, updateOrderDto);

}

@Delete(':id')

async remove(@Param('id') id: string) {

return await this.ordersService.remove(+id);

}

@Delete('removeCart/:id/:token')

async removeCart(@Param('id') id: string, @Param('token') token: string) {

const payload = await this.authService.getPayloadFromToken(token);

return await this.ordersService.removeFromCart(+id, +payload.sub);

}

}

# ПРИЛОЖЕНИЕ Б

Реализация класса OrdersService

@Injectable()

export class OrdersService {

constructor(private readonly prisma: PrismaService) {

}

async create(createOrderDto: CreateOrderDto) {

return await this.prisma.orders.create({

data: {

...createOrderDto

}

});

}

async createInCart(createOrderDto: CreateOrderDto) {

return await this.prisma.orders.create({

data: {

user\_id: createOrderDto.user\_id,

product\_id: createOrderDto.product\_id,

quantity: createOrderDto.quantity,

price: createOrderDto.price,

orderDate: "",

isBuying: "false"

}

});

}

async findAll() {

return await this.prisma.orders.findMany();

}

async findAllOrders(id: number) {

return await this.prisma.orders.findMany({

select: {

order\_id: true,

user\_id: true,

product\_id: true,

quantity: true,

price: true,

orderDate: true,

isBuying: true,

products: {

select: {

product\_name: true,

productImage: true

}

}

},

where: {

user\_id: id,

isBuying: "true"

}

});

}

async findAllOrdersInCart(id: number) {

return await this.prisma.orders.findMany({

select: {

order\_id: true,

user\_id: true,

product\_id: true,

quantity: true,

price: true,

orderDate: true,

isBuying: true,

products: {

select: {

product\_name: true,

productImage: true

}

}

},

where: {

user\_id: id,

isBuying: "false"

}

});

}

async findOne(id: number) {

return await this.prisma.orders.findUnique({

where: {

order\_id: id

}

});

}

async update(id: number, updateOrderDto: UpdateOrderDto) {

return await this.prisma.orders.update({

where: {

order\_id: id

},

data: {

...updateOrderDto

}

});

}

async updateOrder(id: number, updateOrderDto: UpdateOrderDto) {

return await this.prisma.orders.updateMany({

where: {

user\_id: id,

isBuying: "false"

},

data: {

orderDate: updateOrderDto.orderDate,

isBuying: "true"

}

});

}

async remove(id: number) {

return await this.prisma.orders.delete({

where: {

order\_id: id

}

});

}

async removeFromCart(id: number, userId: number) {

return await this.prisma.orders.deleteMany({

where: {

order\_id: id,

user\_id: userId,

isBuying: "false"

}

});

}

}

# ПРИЛОЖЕНИЕ В

Реализация компонента Review

export default function Review() {

const [data, setData] = useState([]);

const [review, setReview] = useState('');

useEffect(() => {

async function fetchData() {

const response = await getReviews();

setData(await responseHandler(response));

}

fetchData();

}, [data])

function handleCommentChange(e) {

setReview(e.target.value);

};

function handleSubmit (e) {

e.preventDefault();

const date = new Date();

const parseDate = ${date.getDay()}-${date.getMonth()}-${date.getFullYear()} в ${date.getHours()}:${date.getMinutes()}:${date.getSeconds()}

postReview(review, parseDate);

setReview('');

};

return (

<div className="reviewMain">

<div>

<Header/>

</div>

<div className="reviewContent">

<div>Отзывы</div>

<div className="userInfo">

<div></div>

<div></div>

</div>

<div>

<form onSubmit={handleSubmit} className="formReview">

<textarea

value={review}

onChange={handleCommentChange}

placeholder="Оставьте отзыв"

className="textAr"

/>

<button type="submit" className="buttonStl">Отправить</button>

</form>

</div>

<div className="allReviews">

{

data.map((item, index) => (

<div className="oneReview" key={index}>

<div>{item.review\_text}</div>

<div>{item.date}</div>

</div>

))

}

</div>

</div>

<div>

<Footer/>

</div>

</div>

)

}

# ПРИЛОЖЕНИЕ Г

Реализация компонента Catalog

export default function Catalog() {

const [data, setData] = useState([]);

useEffect(() => {

async function fetchData() {

const response = await getProducts();

setData(await responseHandler(response));

}

fetchData();

}, [])

async function addToCart(item) {

await postProductToCart(item);

alert('Товар добавлен в корзину');

}

return (

<div className="catalogMain">

<Header/>

<div className="catalogContent">

<div className="catalogHead">Каталог</div>

<div className="productsList">

{

data.map((item, index) => (

<div className="productCard" key={index}>

<div className="mainCardInfo">

<div>

<img src={'./' + item.productImage} alt="not found"/>

</div>

<div>{item.product\_name}</div>

<div>{item.description}</div>

<div>Температура заваривания: {item.brewingTemperature}</div>

</div>

<div className="lastRow">

<div>{item.price} BYN</div>

<div>

<button className="buttonStl" onClick={() => addToCart(item)}>В корзину</button>

</div>

</div>

</div>

))

}

</div>

</div>

<Footer/>

</div>

)

}