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 «Интернет-магазин электронной техники»

Выполнил студент Трубач Дмитрий Сергеевич

(Ф.И.О.)

Руководитель проекта асс. Нистюк О.А.

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

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

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

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 5](#_Toc166246287)

[1 Постановка задачи 6](#_Toc166246288)

[1.1 Аналитический обзор аналогов 6](#_Toc166246289)

[1.1.1 5 элемент 6](#_Toc166246290)

[1.1.2 Wildberries 7](#_Toc166246291)

[1.1.3 Электросила 8](#_Toc166246292)

[1.2 Разработка функциональных требование и вариантов использования 9](#_Toc166246293)

[1.3 Выводы по разделу 10](#_Toc166246294)

[2 Проектирование web-приложения 11](#_Toc166246295)

[2.1 Обобщенная структура управлением приложения 11](#_Toc166246296)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 11](#_Toc166246297)

[2.4 Выводы по разделу 13](#_Toc166246298)

[3 Разработка web-приложения 14](#_Toc166246299)

[3.1 Разработка серверной части web-приложения 14](#_Toc166246300)

[3.2 Разработка клиентской части web-приложения 19](#_Toc166246301)

[3.3 Выводы по разделу 20](#_Toc166246302)

[4 Тестирование web-приложения 21](#_Toc166246303)

[5 Руководство пользователя 23](#_Toc166246304)

[5.1 Руководство пользователя 23](#_Toc166246305)

[5.2 Установка приложения 28](#_Toc166246306)

[5.3 Выводы по разделу 28](#_Toc166246307)

[Заключение 29](#_Toc166246308)

[Список используемых источников 30](#_Toc166246309)

[Приложение А 31](#_Toc166246310)

# Введение

Целью работы является разработка кроссплатформенного веб-приложения на тему «Интернет-магазин электронной техники». Приложение представляет из себя интернет-магазин, предоставляющий пользователю удобный функционал для ознакомления с ассортиментом магазина и оформления заказов.

Node.js – программная платформа, основанная на движке Chrome V8. Node.js асинхронен и событийно-ориентирован. 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.

В качестве СУБД была выбрана MSSQL, поскольку данная база данных обладает необходимой простотой, производительностью и надежностью.

Написанное приложение строится на клиент-серверной архитектуре. В качестве клиента выступает приложение, написанное на HTML и JavaScript, которое отображается в браузере. Для сервера была выбрана платформа Node.js в связи с кроссплатформенностью, простотой развертки сервера и гибкости работы за счёт npm и пакетов.

В пояснительной записке содержится краткая информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию и тестированию приложения.

# 1 Постановка задачи

Важным этапом разработки любого нового продукта является анализ существующих на рынке аналогов. Он позволяет определить, какие решения уже представлены потребителям, выявить их сильные и слабые стороны, а также определиться с направлениями для усовершенствований.

Интернет-магазин - это форма электронной коммерции, которая позволяет потребителям прямо покупать товары или услуги от продавца через Интернет. Интернет-магазины стали важной частью розничной торговли. В сфере электроники они используются для продажи различных товаров, от смартфонов и ноутбуков до комплектующих и аксессуаров.

На текущий момент рынок интернет-магазинов представляет собой множество платформ с различным уровнем функционала и безопасности. Однако, многие из них не могут предложить полноценного комплекса инструментов, сочетающего в себе удобство, гибкость и высокие стандарты безопасности. Это оставляет простор для новых разработок в этой сфере.

Задача проекта: разработать архитектуру приложения, взаимодействие с которой будет понятно любому пользователю. Построить базу данных и выполнить тестирование готового продукта.

Должны быть выполнены следующие требования:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли администратора и пользователя;
* добавление и изменение товаров администратором;
* фильтрация товаров;
* оформление заказа пользователем;
* предоставление возможности доступа к таблицам БД администратору;
* поддержка беседы через чат с клиентом со стороны администратора.

## 1.1 Аналитический обзор аналогов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

### 1.1.1 5 элемент

Одно из самых популярных альтернативных решений – интернет-магазин «5 элемент». «5 элемент» – это виртуальный каталог с интерактивным поиском, детальными характеристиками и подробными описаниями для тех, кто ищет новую электронику.

Интернет-магазин представляет собой виртуальный каталог с широким выбором электроники, обеспечивающий удобную систему фильтрации и сравнения товаров. Пользователи могут легко добавлять товары в «Избранные» и получать онлайн-техподдержку через удобный чат. Стильный дизайн и система оформления заказов делают покупки приятным процессом. Однако, некоторые пользователи могут столкнуться с трудностями в поиске нужного товара, а также возможны технические ошибки при обновлении каталога.

Интерфейс интернет-ресурса представлен на рисунке 1.1.

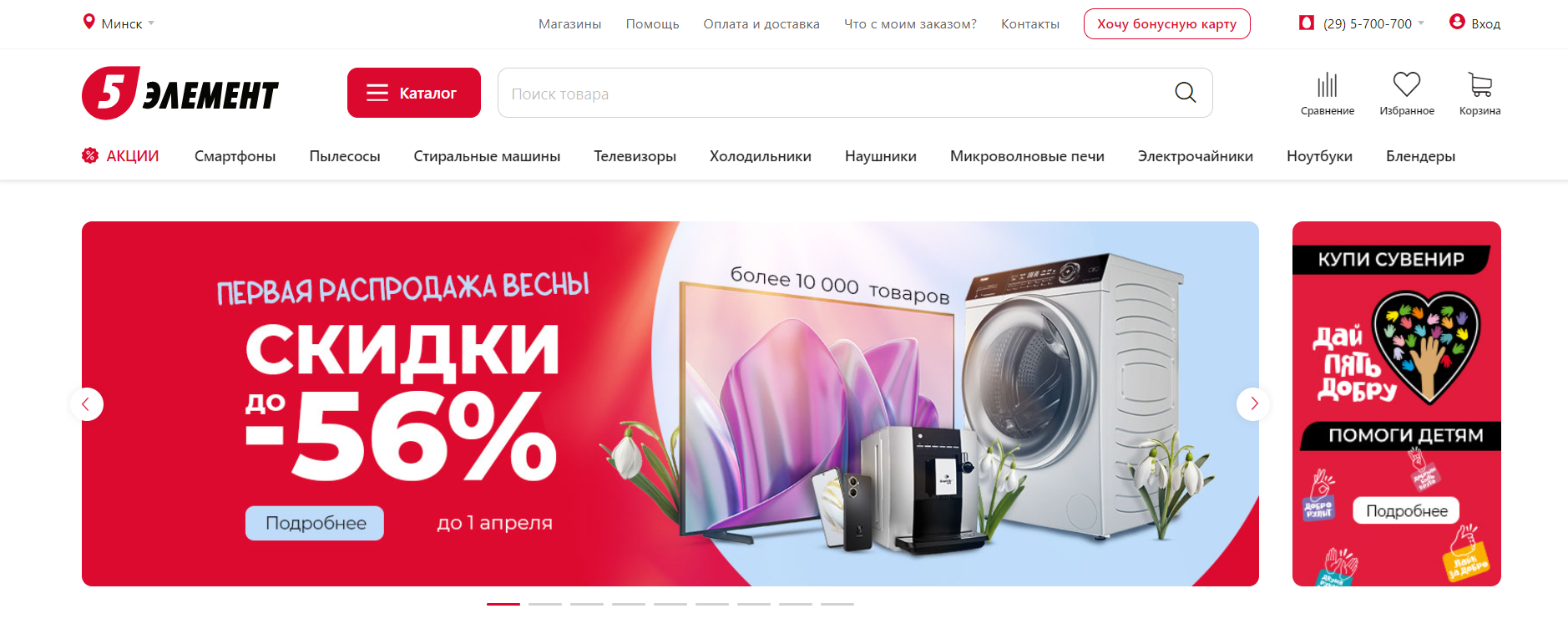


Рисунок 1.1 – Интернет-ресурс «5 элемент»

Из обзора такого аналога, как «5 элемент», для приложения можно взять идею чата с техподдержкой.

### 1.1.2 Wildberries

Wildberries – один из ведущих интернет-магазинов в России и странах СНГ, предоставляющий широкий ассортимент товаров от одежды и обуви до товаров для дома и электроники.

Главный экран приложения Wildberries отображает список последних просмотренных товаров, а также кнопки для быстрого доступа к главным разделам магазина и функциям. В верхней части экрана расположены кнопки для перехода к категориям товаров, просмотру акций и персональным настройкам пользователя. В целом, интерфейс Wildberries является интуитивно понятным и удобным в использовании.

Дизайн Wildberries отличается современным и стильным оформлением с использованием ярких цветов и понятных иконок. Минималистичный дизайн делает приложение легким и привлекательным для пользователей, обеспечивая комфортное взаимодействие с платформой.

Страница товара представляет из себя информативный карточка товара с изображением товара, ценой и основными характеристиками. Кнопки для добавления товара в корзину, сравнения с другими товарами и оформления заказа выделяются на фоне, что облегчает пользователю принятие решения и быстрое совершение покупки.

Интерфейс приложения представлен на рисунке 1.2.

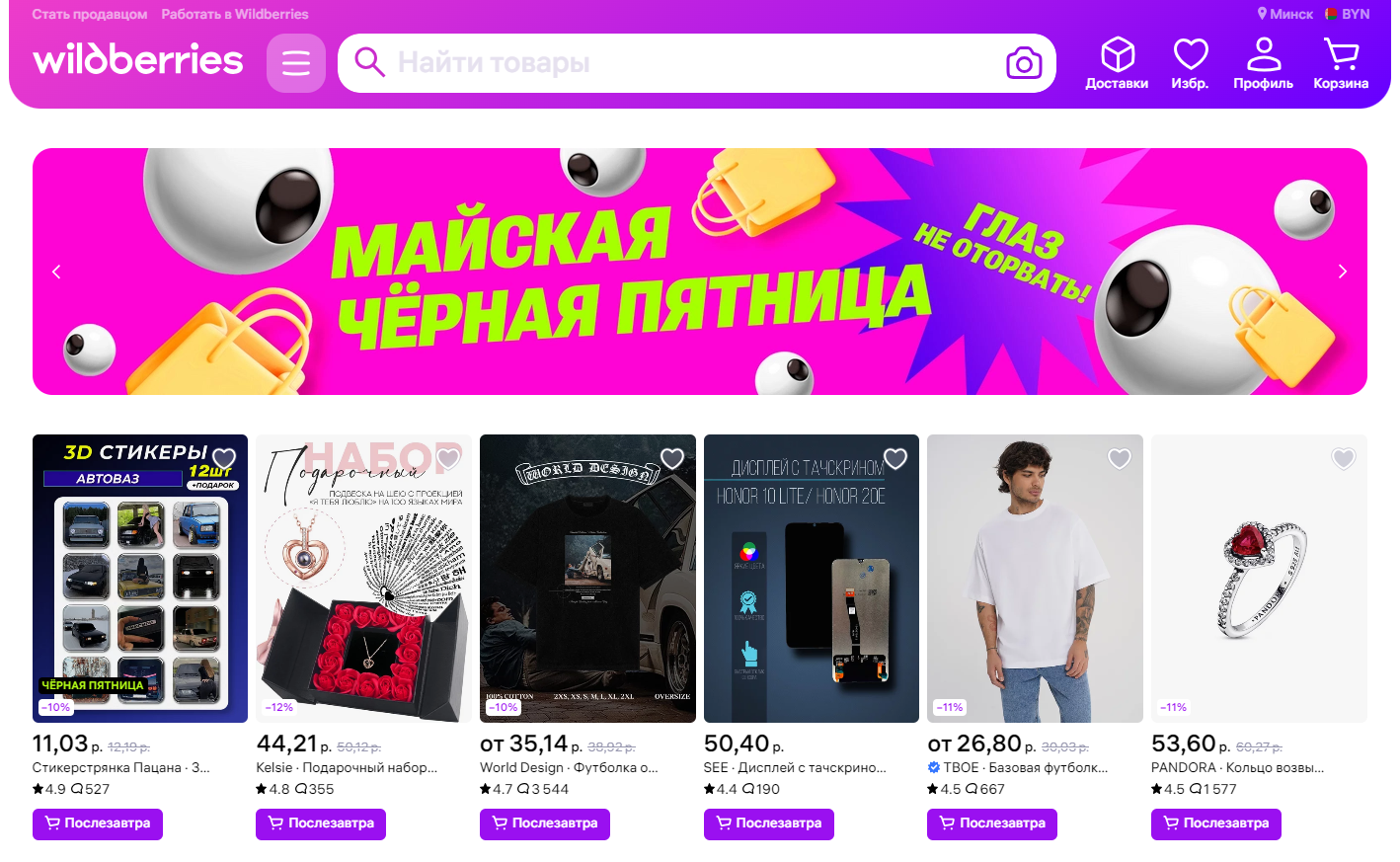


Рисунок 1.2 – Интернет-ресурс «Wildberries»

Из обзора такого аналога, как Wildberries, для приложения можно взять идею – реализовать приятный и лаконичный интерфейс карточки товара.

### 1.1.3 Электросила

Сайт «Электросила» представляет собой интернет-магазин, специализирующийся на продаже электроники. Он обладает широким ассортиментом товаров, включая ноутбуки, смартфоны, планшеты, телевизоры, аудио- и видеоустройства, а также аксессуары и периферийные устройства. Сайт разработан с учетом удобства использования и предлагает функциональный и интуитивно понятный интерфейс.

Основная страница сайта содержит разделы с различными категориями товаров, что позволяет пользователям легко найти нужные им продукты. Каждый товар представлен с подробной информацией, включая описание, характеристики, цены и фотографии. Также на сайте присутствуют разделы с акциями и распродажами, а также с новинками и популярными товарами.

Для удобства покупателей сайт предлагает функцию сравнения товаров, возможность добавления товаров в корзину и оформления заказа онлайн. Пользователи могут выбирать способ доставки и оплаты, а также следить за статусом своего заказа через личный кабинет на сайте.

Сайт «Электросила» также акцентирует внимание на поддержке клиентов, предлагая различные способы связи, включая телефон, электронную почту и форму обратной связи на сайте. Это позволяет быстро решать возникающие вопросы и проблемы, связанные с покупкой и доставкой товаров.

Интерфейс приложения представлен на рисунке 1.3.

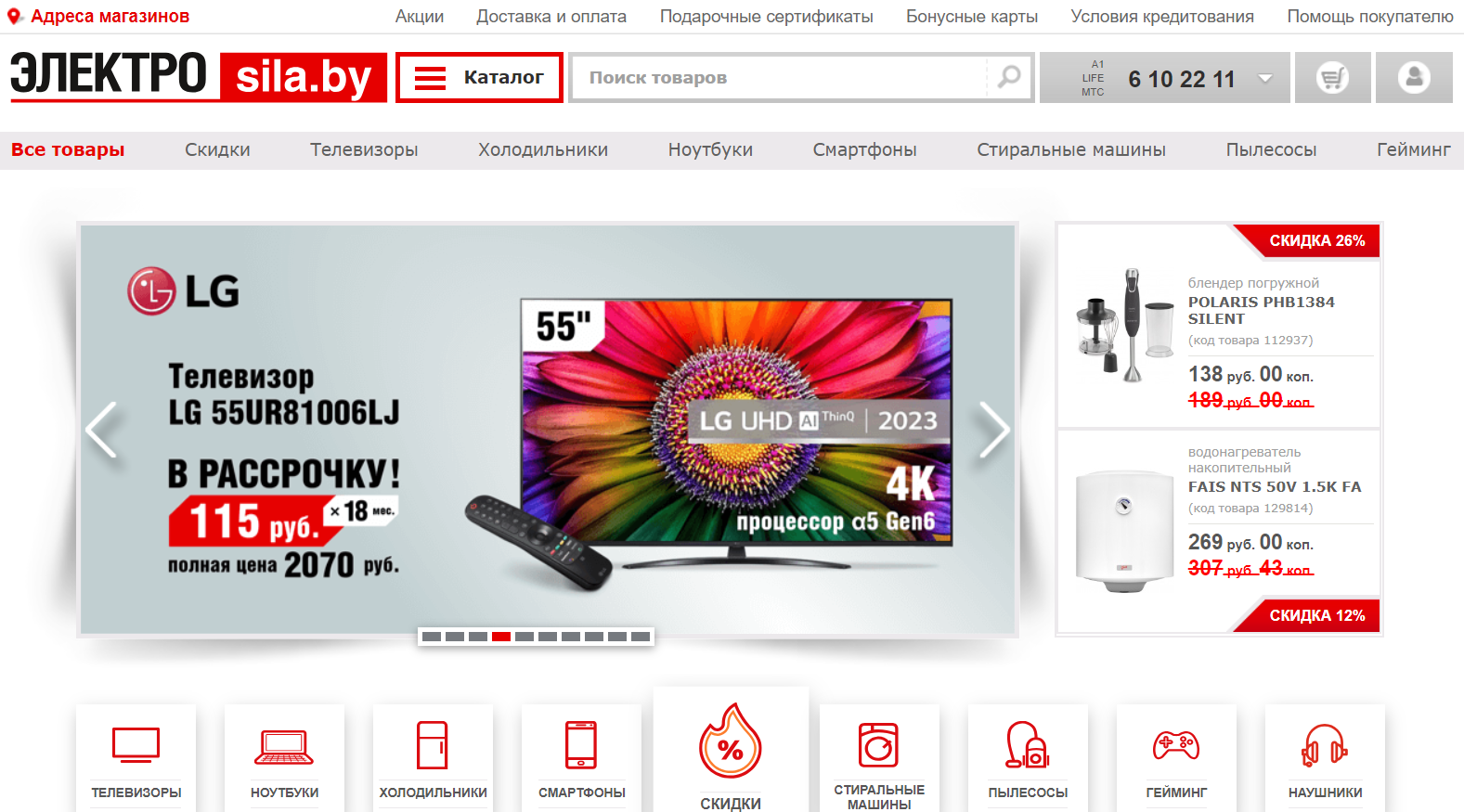


Рисунок 1.3 – Интернет-ресурс «Электросила»

«Электросила» обладает минималистичным стилем и быстрым откликом от сервера, что также должно обеспечивать разрабатываемое web-приложение.

## 1.2 Разработка функциональных требование и вариантов использования

Приложение должно обеспечивать функционал для регистрации авторизации, который позволит идентифицировать пользователей. Также приложение должно обеспечить функционал просмотра различных видов товара, с последующими фильтрами. Для работы с товарами должны быть созданы функции удаления и редактирования сообщений. Удаление должно быть сделано полным, чтобы товар полностью удалялось из базы данных. Также должен быть реализован механизм работы WebDav-сервера для хранения изображений товара. Администратор должен иметь функции для контроля как за товаром, так и за всеми пользователями. Диаграмма вариантов использования представлена на рисунке 1.4.

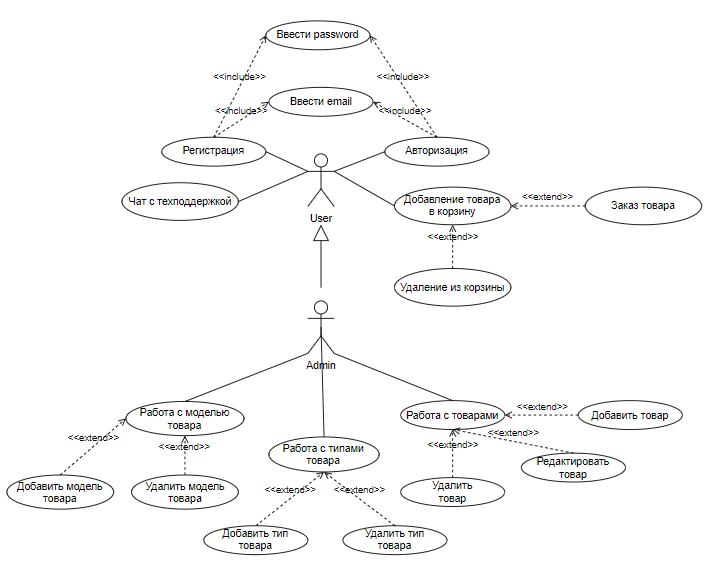


Рисунок 1.4 – Диаграмма вариантов использования

Диаграмма UML вариантов использования позволяет понять, что доступно каждой роли, доступной в данном веб-приложении.

## 1.3 Выводы по разделу

В данном разделе мы рассмотрели аналогичные веб-приложения. Исходя из этих аналогов можно взять пару идей для своего веб-приложения. Например, данный мессенджер должен обеспечивать чат с техподдержкой, как и 5 элемент, данный мессенджер должен предоставлять красивый и информативный интерфейс карточки товара, как Wildberries.

Также в данном разделе рассмотрели весь основной функционал, который должен присутствовать в конечном приложении. Данный функционал рассчитан на то, что он позволит довольно-таки просто начать пользоваться этим веб-приложением, а также что этот функционал позволит покупать товары или услуги. Была приведена UML диаграмма, на которой все и приведено.

# 2 Проектирование web-приложения

## 2.1 Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения интернет-магазина должны входить функции для удобного поиска людей в приложении, функции добавления в друзья, различные виды сообщение и другие подобные функции.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных представлена на рисунке 2.1.

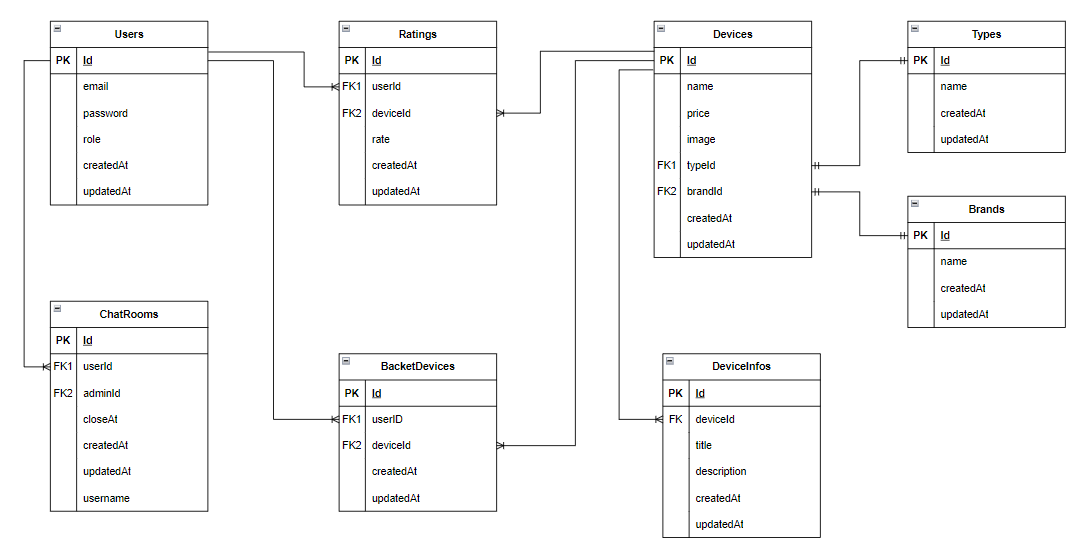


Рисунок 2.1 – Логическая схема базы данных

В базе данных находится 8 таблиц. В таблице Users хранятся данные для идентификации пользователей, зарегистрированных в приложении, столбец email содержит почту пользователя, которая используется как логин в процессе авторизации, password – хешированный алгоритмом SHA-1 пароль, role – роль пользователя, createdAt – дата создания записи, updatedAt – дата последней модификации записи.

CharRooms – информация о сеансах техподдержки, столбец userId содержится идентификатор пользователя, которому соответствует запись в таблице, adminId – содержит идентификатор администратора, которому соответствует запись в таблице, username – имя, под которым пользователь представился в чате, createdAt – дата создания записи, updatedAt – дата последней модификации записи, closedAt – дата и время закрытия сеанса чата.

BasketDevices – текущий список товаров в корзинах пользователей, в столбце userId содержится идентификатор пользователя, которому соответствует запись в таблице, deviceId – содержит идентификатор товара, которому соответствует запись в таблице, createdAt – дата создания записи, updatedAt – дата последней модификации записи.

Ratings – оценки товаров пользователями, в столбце userId содержится идентификатор пользователя, которому соответствует запись в таблице, deviceId – содержит идентификатор товара, которому соответствует запись в таблице, createdAt – дата создания записи, updatedAt – дата последней модификации записи, rate – значение оценки товара, поставленной пользователем.

Devices – список товаров или ассортимент магазина, в столбце name содержится наименование товара, price – содержит цену товара, которому соответствует запись в таблице, image – название изображения к товару, typeId – содержит идентификатор типа, которому соответствует запись в таблице категорий, brandId – содержит идентификатор типа, которому соответствует запись в таблице категорий, createdAt – дата создания записи, updatedAt – дата последней модификации записи.

Brands – список брендов товаров, в столбце name содержится наименование бренда, createdAt – дата создания записи, updatedAt – дата последней модификации записи.

Types – список категорий товаров, в столбце name содержится наименование типа, createdAt – дата создания записи, updatedAt – дата последней модификации записи, rate – значение оценки товара, поставленной пользователем.

DeviceInfos – список характеристик товаров, в столбце deviceId содержится идентификатор товара, которому соответствует запись в таблице товаров, title – наименование характеристики, description – значение характеристики, createdAt – дата создания записи, updatedAt – дата последней модификации записи.

Для того понимания структуры данного приложения и как его разворачивать нам может помочь диаграмма развертывания. Данная диаграмма показана на рисунке 2.2.

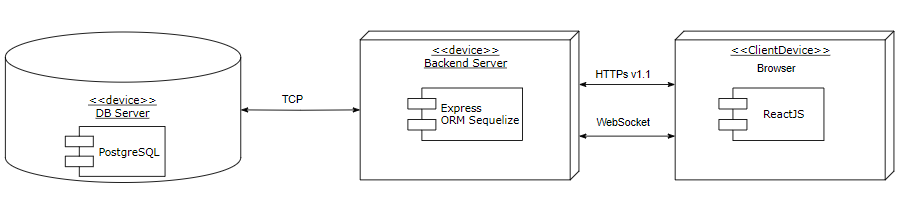


Рисунок 2.2 – Диаграмма развертывания

По данной диаграмме развертывания видно, что веб-приложение состоит из трех основных частей. Первая часть – это клиентская часть нашего приложения. Эта часть реализована при помощи одноименной библиотеки для языка программирования JavaScript ReactJS. Данная библиотека поможет создать динамически SPA приложение, которое будет делать запросы по протоколу HTTP к серверной части приложения. Также оно будет отправлять и получать сообщения по протоколу WebSocket. Вторая часть данного веб-приложения – это серверная часть, разработанная на платформе NodeJS при помощи фреймворка Express, который позволяет построить структуру из всех основных модулей нашего приложения. Данная часть приложения будет обращаться к базе данных по протоколу TCP. Третья часть нашего приложения – это сам сервер базы данных PostgreSQL, который будет отвечать за хранение всех данных, способствующих работе нашего приложения.

## 2.4 Выводы по разделу

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой.

В данном разделе рассмотрели структуру базы данных PostgreSQL, ее основные сущности и как они связаны. Также рассмотрели, какие есть основные части нашего веб-приложения, точнее, во что оно развертывается.

# 3 Разработка web-приложения

## 3.1 Разработка серверной части web-приложения

Серверная часть написана на фреймворке Express для платформы NodeJS, а также были использованы различные пакеты, такие как cors, express-fileupload, ws, dotenv, path, bcrypt, jsonwebtoken, tedious и sequelize. Непосредственно для взаимодействия с Postgres нам необходимы только пакеты tedious и sequelize. Схема взаимодействия Node.js с базой данных представлена на рисунке 3.1.

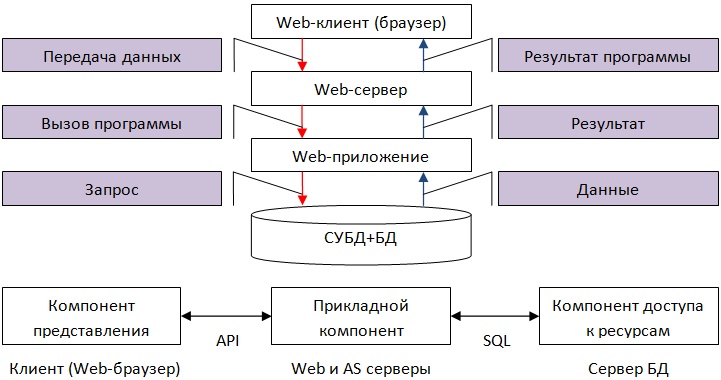


Рисунок 3.1 – Схема взаимодействия NodeJS и базы данных

Структура сервера была разработана соответствии с архитектурным паттерном Model-View-Controller (далее - MVC), что делает её читабельной и легко расширяемой.

MVC подразумевает под собой разделение проекта на три независимые компоненты:

Model – модель, предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

View – представление, отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Controller – контроллер, интерпретирует действия пользователя, оповещая модель о необходимости изменений.

В данном проекте модель работает с базой данных. Каждый объект базы данных имеет свой класс-сущность, сформированный при помощи Sequelize. Эти классы реализованы в отдельных модулях, что позволяет при необходимости вызвать какой-либо метод репозитория в контроллере.

Контроллер выполняет функцию маршрутизатора запросов к серверу. Контроллер (как часть MVC) состоит из двух частей:

* маршрутизатора index, который принимает запросы и передает их определенным контроллерам;
* контроллеров репозитория controllers, каждый из которых обрабатывает свой определенный путь маршрутизатора.

Дерево маршрутизации можно увидеть в листинге 3.1.

router.use('/user', userRouter)

router.use('/type', typeRouter)

router.use('/brand', brandRouter)

router.use('/device', deviceRouter)

router.use('/basketdevice', basketDeviceRouter)

router.use('/rating', ratingRouter)

router.use('/chat', chatRouter)

module.exports = router

Листинг 3.1 – Дерево маршрутизации

Архитектуру файлов контроллера MVC можно увидеть на рисунке 3.2.

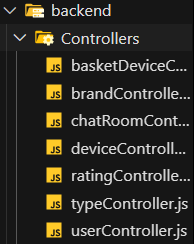


Рисунок 3.2 – Архитектура файлов контроллера MVC

Проект является одностраничным приложением (Single Page Application, далее – SPA), поэтому каждый контроллер производит определенные операции над моделью, формирует ответ и отправляет данные клиентской части приложения, написанной на ReactJS, где динамически формируется HTML разметка с использованием полученных данных.

Так как приложение является браузерным, нужно позаботиться о механизме авторизации пользователей. В приложении реализована JWT-авторизация с использованием пакета jsonwebtoken и bcrypt для шифрования пароля пользователя. Элемент конвейера обработки запросов, который отвечает за проверку пользователя на факт авторизации представлен в листинге 3.2.

module.exports = function(request,response,next)

{try {const token = request.headers.authorization.split(' ')[1];

if(!token) {return next(ApiError.notAuthorized("Неавторизованный доступ"));}

const decoded = jwt.verify(token,process.env.SECRET\_KEY);request.user = decoded;return next();}

catch(error){return next(ApiError.notAuthorized("Неавторизованный доступ"));}}

Листинг 3.2 – Реализация middleware проверки авторизации

Проверка пользователя на наличие прав доступа к ресурсу представлена в листинге 3.3.

module.exports = function(role)

{

return function (request, response, next) {

try {

const token = request.headers.authorization.split(' ')[1] if (!token) {return next(ApiError.notAuthorized("Неавторизованный доступ"));}

const decoded = jwt.verify(token, process.env.SECRET\_KEY)

if (decoded.role !== role) {return next(ApiError.notAuthorized("Недостаточно прав"));}request.user = decoded;return next()}

catch (e) {

return next(ApiError.notAuthorized("Неавторизованный доступ"));}

};

}

Листинг 3.3 – Реализация middleware проверки роли пользователя

При заполнении соответствующих полей в окне логина или регистрации устанавливается заголовок «authorization», который является индикатором авторизации пользователя. При выходе из приложения соответствующее значение перестаёт помещаться в данный заголовок, так как локальное хранилище данных пользователя на стороне клиента очищается.

Для определения правил процесса авторизации и регистрации пользователя было реализован соответствующий программный интерфейс. Функции, делающие запрос на сервер к ресурсу авторизации, регистрации и проверки данных авторизованного пользователя представлены в листинге 3.4.

export const registration = async (email, password) => {

const {data} = await $host.post('api/user/registration', {email, password, role: 'USER'})localStorage.setItem('token', data.token)

return jwt\_decode(data.token)

}

export const login = async (email, password) => {

const {data} = await $host.post('api/user/login', {email, password})localStorage.setItem('token', data.token)

return jwt\_decode(data.token)

}

export const check = async () => {

const {data} = await $authHost.get('api/user/auth' )localStorage.setItem('token', data.token)

return jwt\_decode(data.token)

}

Листинг 3.4 – API авторизации и регистрации

В приложении также реализован функционал чата техподдержки на основании протокола WebSocket. Клиентская и серверная части приложения обмениваются между собой сообщениями определённого содержимого, в соответствии с которым происходит реализация механизмов хранения данных о чатах в базе данных и подключения к чатам на основании событийной обработки получаемых клиентом и сервером сообщений. Пример функции и структуры сообщений, отправляемых серверу, представлены на в листинге 3.5.

const sendMessage = async () => {

let message = {id: socket.current.chatId,message: value,date: Date.now(),userId: user? user.user.id : null,username: props.username,adminId: socket.current.adminId ? socket.current.adminId : null,adminname: socket.current.adminname !== ''?socket.current.adminname : null,event: 'message',state: socket.current.userState,from: 'client',to: 'server'}

if(socket.current.userState === "answering")message.to = 'admin' socket.current.socket.send(JSON.stringify(message));setValue('')}

Листинг 3.5 – Пример функции отправки сообщения серверу

Для хранение статических ресурсов использовался удалённый веб-сервер, взаимодействие с которым происходит на основе протокола WebDav.

При загрузке, удалении или создании ресурса на удалённой сервере происходит обращение к соответствующей функции API для взаимодействия с сервером. Пример функции получения файла показан на листинге 3.6.

const getFile = async (request, response, next) => {

const { filename } = request.params;

const filePath = "/static/images/" + filename;

await client.exists(filePath).then((alreadyExists) => {if (alreadyExists) {let rs = client.createReadStream(filePath);rs.pipe(response);} else {next(ApiError.notFound("Файл не существует!"));}}).catch((err) => {next(ApiError.badRequest("Файл не существует!"));});};

Листинг 3.6 – Пример функции взаимодействия с удаленным сервером

Этот код демонстрирует асинхронную функцию getFile, которая используется для обработки запросов на получение файла с удаленного сервера.

Также, при разработке сервера был создан и подписан сертификат на ресурсы с помощью OpenSSL. На рисунке 3.3 показан сертификат и вспомогательные файлы сертификата.

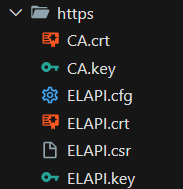


Рисунок 3.3 – Сертификат с помощью OpenSSL

На рисунке 3.4 показан результат выдачи сертификата на сервер.

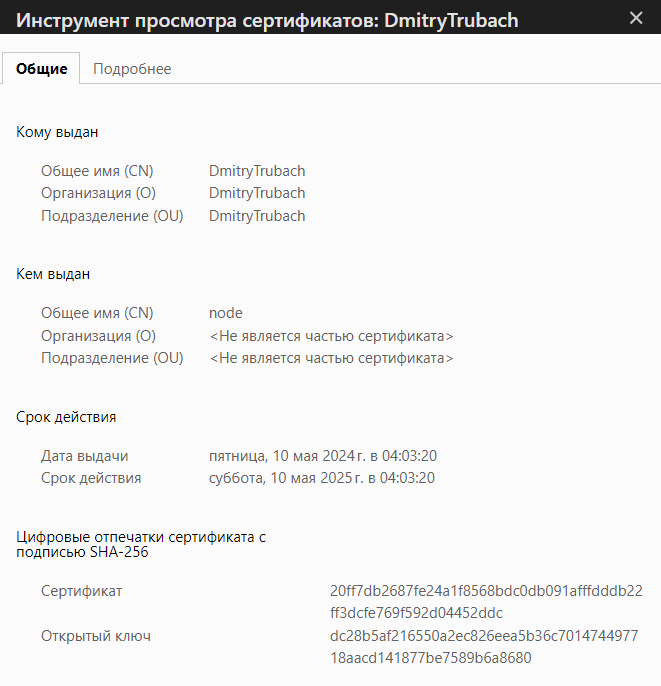


Рисунок 3.4 – Сертификат в браузере

## 3.2 Разработка клиентской части web-приложения

Так как проект является кроссплатформенным, клиентская часть была разработана для использования в браузере. Было принято решение реализовать SPA приложение, поэтому оно должно уметь запрашивать у сервера необходимые данные и динамически изменять представление в зависимости от полученных данных. Чтобы клиент не видел задержки при запросе у сервера необходимых данных, клиент запрашивает данные асинхронно с использованием promise-ориентированных AJAX и Axios, что позволяет отправить данные при помощи определенного метода на строго фиксированный путь. Пример реализации Axios-запросов представлен в листинге 3.7.

export const fetchBasketDevices = async (page, limit = 10) => {

const { data } = await $authHost.get('api/basketdevice', {params: {page, limit}})return data}

Листинг 3.7 – Пример функции с Axios-запросом

После каждого успешно выполненного запроса на сервере и получения данных, происходит рендеринг компонента в соответствии с содержимым ответа.

На листинге 3.8 представлен пример применения функции запроса и использования данных для рендеринга в хуке «useEffect».

const {device} = useContext(Context)

useEffect(() =>{fetchAllTypes().then(data=>device.setTypes(data))fetchAllBrands().then(data =>fetchDevices(null, null,device.selectedRate, 1, device.limit).then(data => {device.setDevices(data.rows)device.setTotalCount(data.count)})

Листинг 3.8 – Пример использования хука «useEffect»

Данные, которые необходимо передавать между компонентами, хранятся в объектах контекста, синтаксис определения которого представлен в листинге 3.9. Пример класса объекта контекста представлен в [приложении А](#_Приложение_А).

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<Context.Provider value={{

user : new UserStore(),device: new DeviceStore(),brand: new BrandStore(),}}><BrowserRouter><AxiosInterceptor><App className="gradientBackground"/></AxiosInterceptor></BrowserRouter>

</Context.Provider>

)

Листинг 3.9 – Пример синтаксиса определения контекста приложения

Для помещения JWT-токена в каждый запрос на взаимодействие с ресурсами, доступными только для авторизированных пользователей, разработан компонент «AxiosInterceptor», реализация которого приведена в [приложении А](#_Приложение_А).

Данный компонент также реализует функционал по обработке ошибок и выводу страницы ошибки, которая имеет различное содержимое, зависящее от номер передаваемой в ответе от сервера ошибки.

## 3.3 Выводы по разделу

В данном разделе мы рассмотрели основные момент разработки веб-приложения. Рассмотрели, как построено взаимодействие клиента с сервером по протоколу HTTP и по протоколу WebSocket. Рассмотрели, как хранится информация и как отображается на клиенте. Рассмотрели, как построена безопасность данного веб-приложения.

# 4 Тестирование web-приложения

Вначале нужно протестировать форму регистрации, введя некорректные данные. Результат проделанного текста показан на рисунке 4.1.

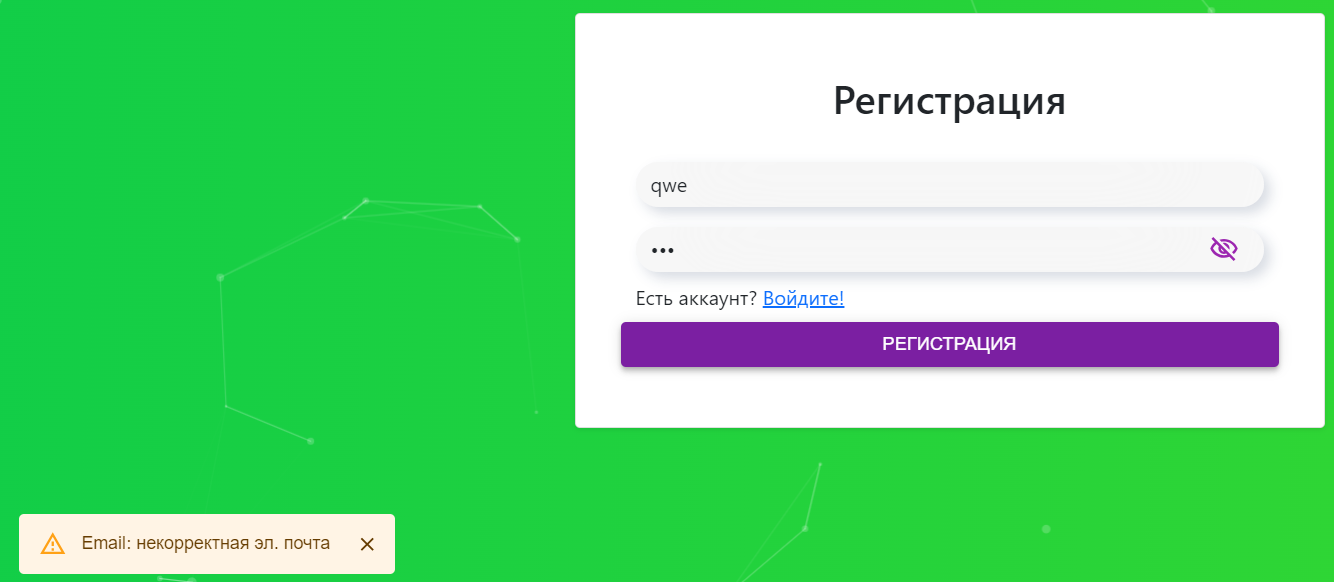


Рисунок 4.1 – Результат проделывания первого теста

По картинке 4.1 видно, что система отработала правильно и указала на ошибку.

Второй тест будет связан с проверкой на поле «Пароль». Если отправить пустое пароль, пользователь не должен зарегистрироваться. Результат данного теста показан на рисунке 4.2.

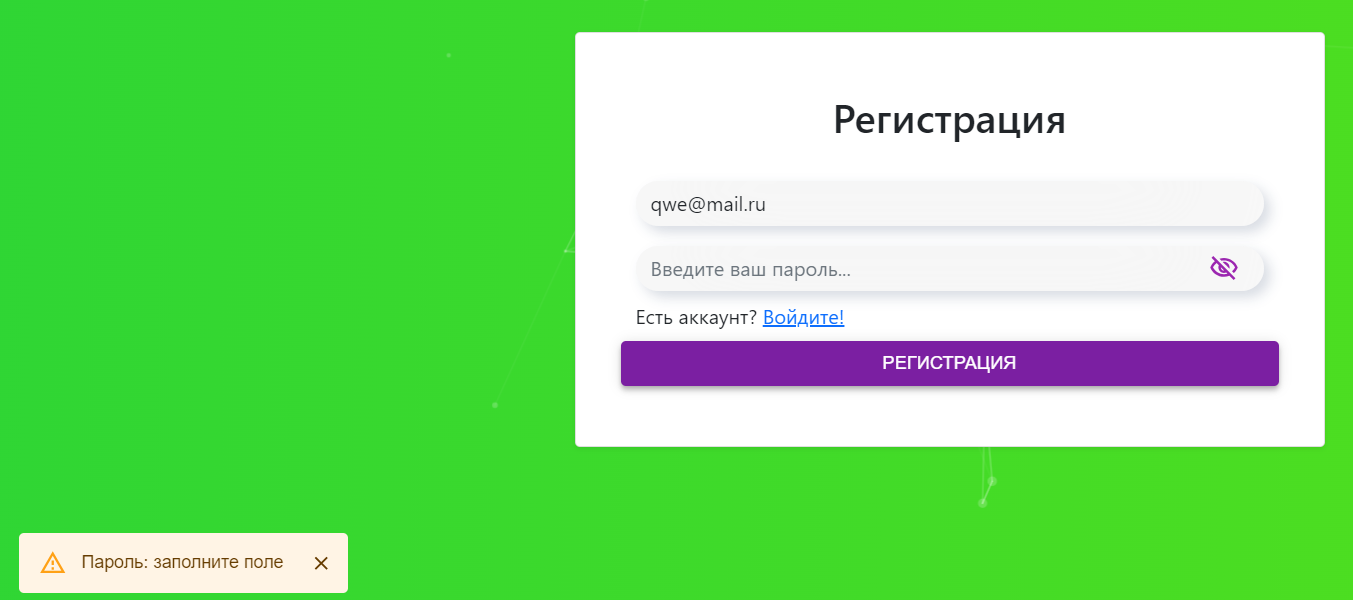


Рисунок 4.2 – Результат проделывания второго теста

По результатам второго теста, видно, что приложение отработало верно.

Третий тест будет заключатся в проверке регистрации по существующим данным. Мы воспользуемся уже использованной электронной почтой и попробуем зарегистрироваться снова. Ожидаемый результат – это вывод сообщении о ошибке. Результат проделывания теста три показан на рисунке 4.3.

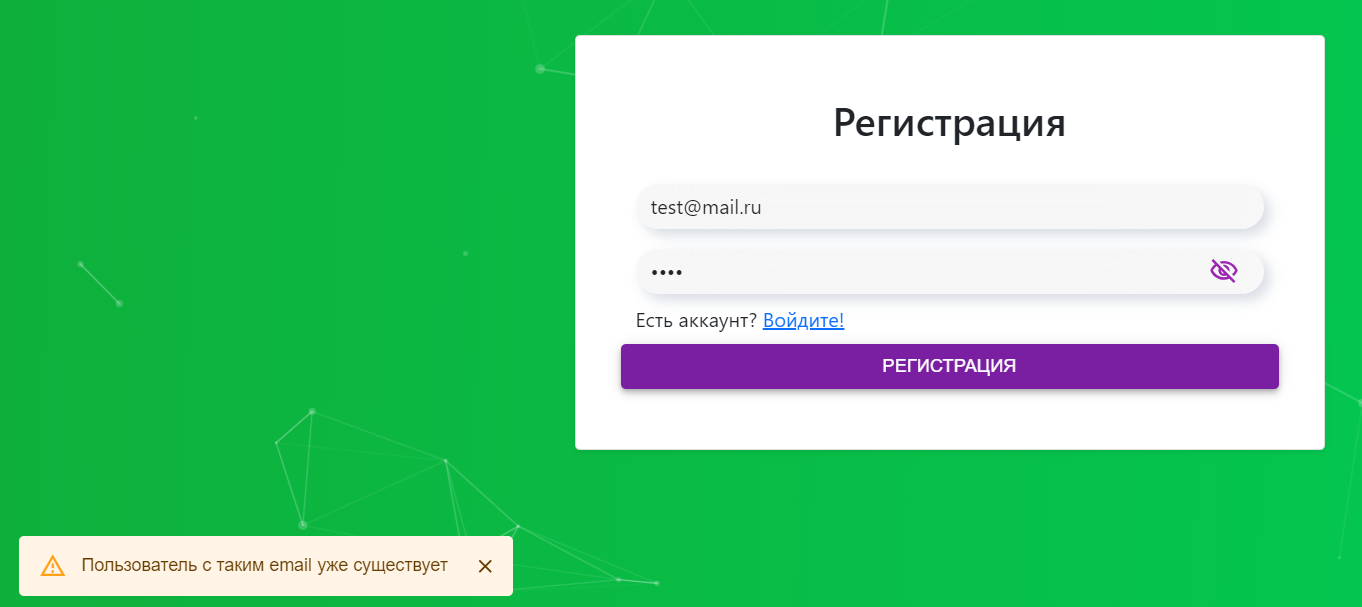


Рисунок 4.3 – Результат проделывания третьего теста

По результатам третьего теста, видно, что приложение отработало верно.

Четвертый тест будет заключатся в том, что мы попробуем изменить контент товара на пустую строку. Ожидаемый результат – редактирование не пройдет. Результат данного тест показан на рисунке 4.4.

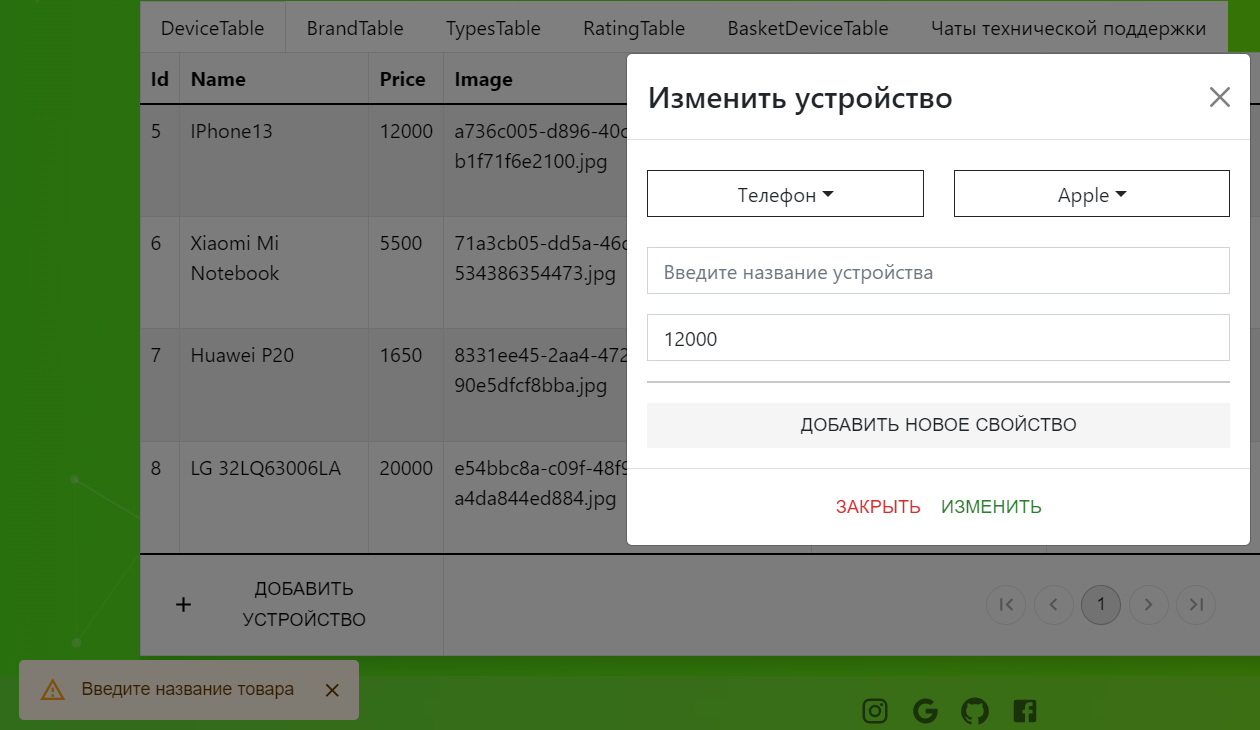


Рисунок 4.4 – Результат проделования четвертого теста

Результат четвертого теста соответствует ожидаемому результату. Данное приложение готово к публикации и работе в сети Интернет.

# 5 Руководство пользователя

## 5.1 Руководство пользователя

При открытии веб-приложения вас будет встречать гостевая страница, на которой вы можете некоторые факты о магазине или же вы можете осуществить регистрацию или авторизацию. Рисунок формы авторизации представлен на рисунке 5.1.

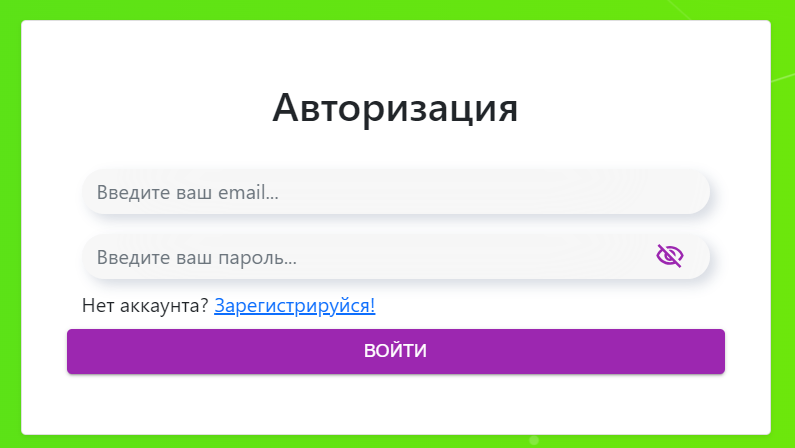


Рисунок 5.1 – Форма авторизации

После входа под определенным пользователем вам откроется страница со всеми товарами. В левой части страницы расположена панель с доступными фильтрами товаров: фильтрация по категории, бренду или рейтингу товара. По необходимости, пользователь может сбросить все фильтры нажатием на соответствующую кнопку. Главная страница представлена на рисунке 5.2.

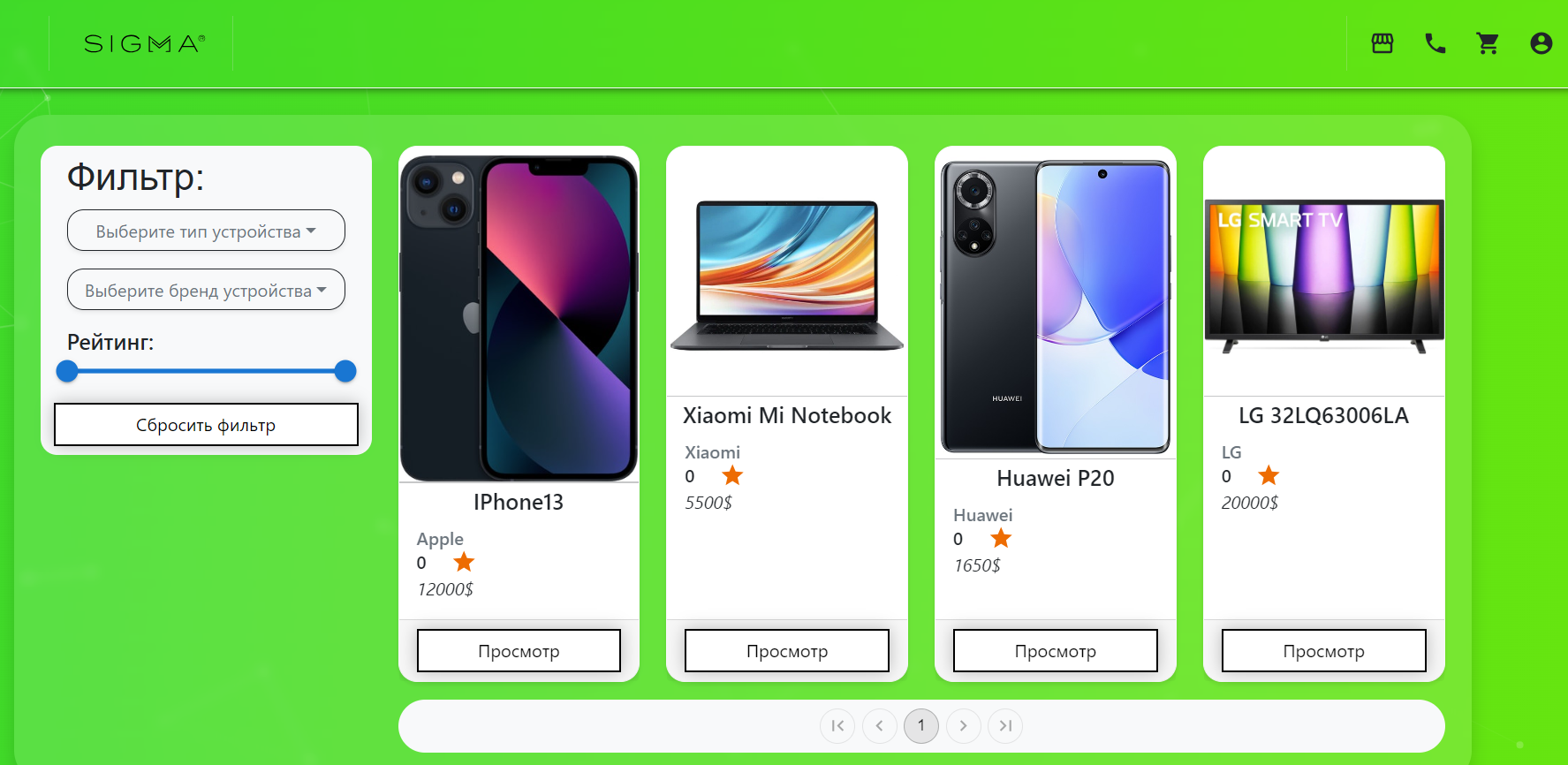


Рисунок 5.2 – Главная страница

Пользователь может создать новый аккаунт. Для этого ему надо перейти на страницу регистрации, где ему будет доступна интуитивно понятная форма. Рисунок формы регистрации представлен на рисунке 5.3.

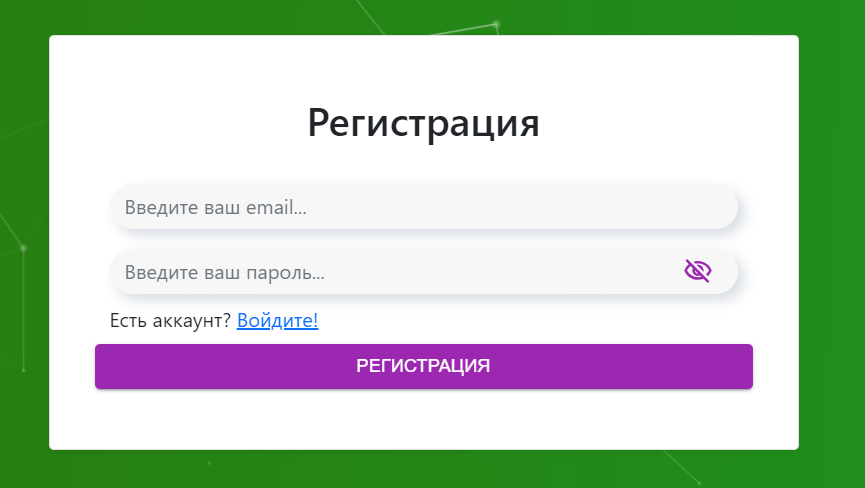


Рисунок 5.3 – Страница регистрации

На странице товаров отображена информация о цене, бренде, характеристиках товара. Пользователь нажатием кнопки "Добавить в корзину" может добавить одну единицу товара в свою корзину в случае, если она авторизован. При нажатии на данную кнопку неавторизованным пользователем произойдёт переадресация на страницу авторизации. Страница товара представлена на рисунке 5.4.

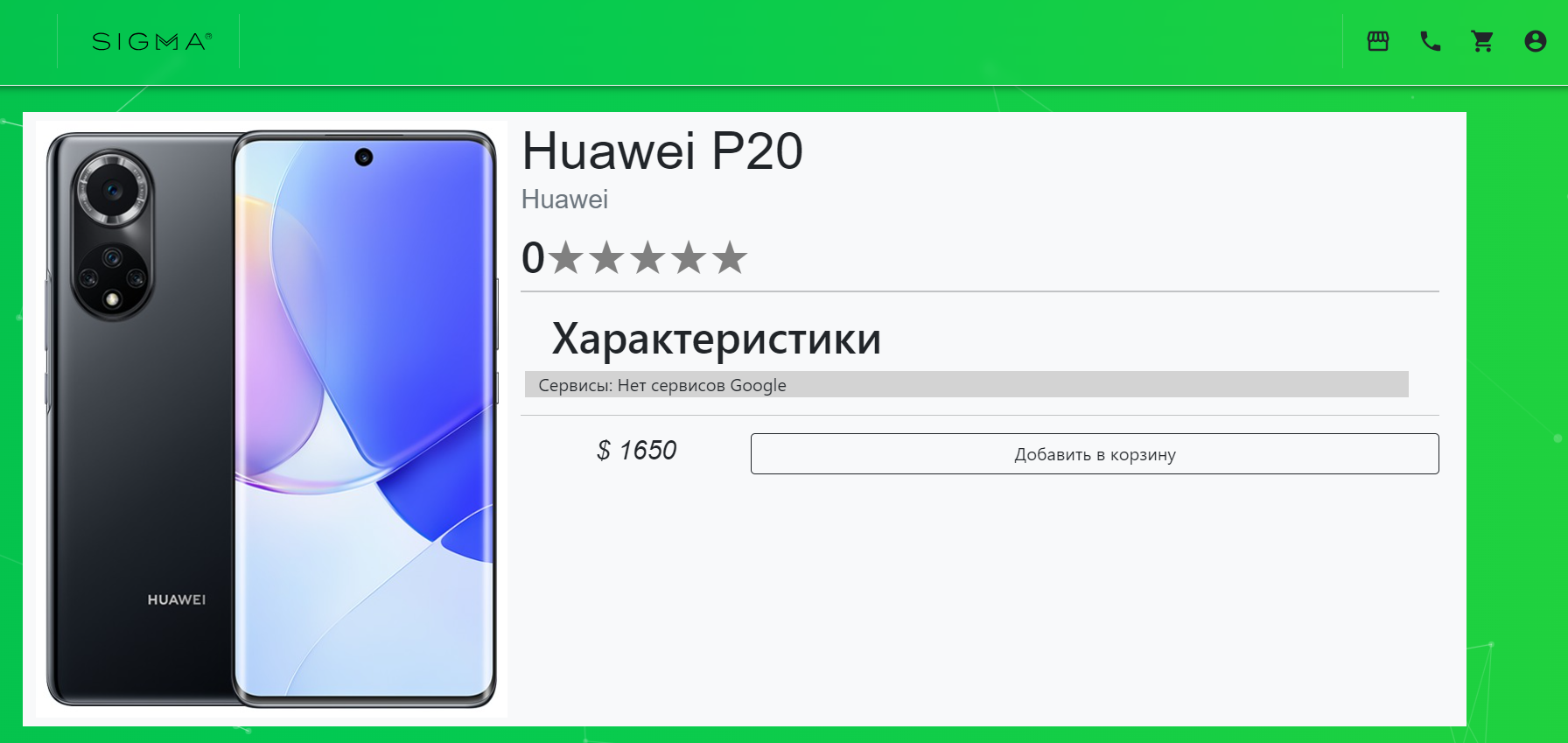


Рисунок 5.4 – Карточка товара

Страница корзины отображает таблицу с товарами, добавленными пользователем, краткую информацию о сумме заказов, доступных скидках и количеством товаров. Ниже располагается меню для выбора варианта оплаты товаров и кнопка оформления заказа.

Таблица с товарами корзины представлена на рисунке 5.5.

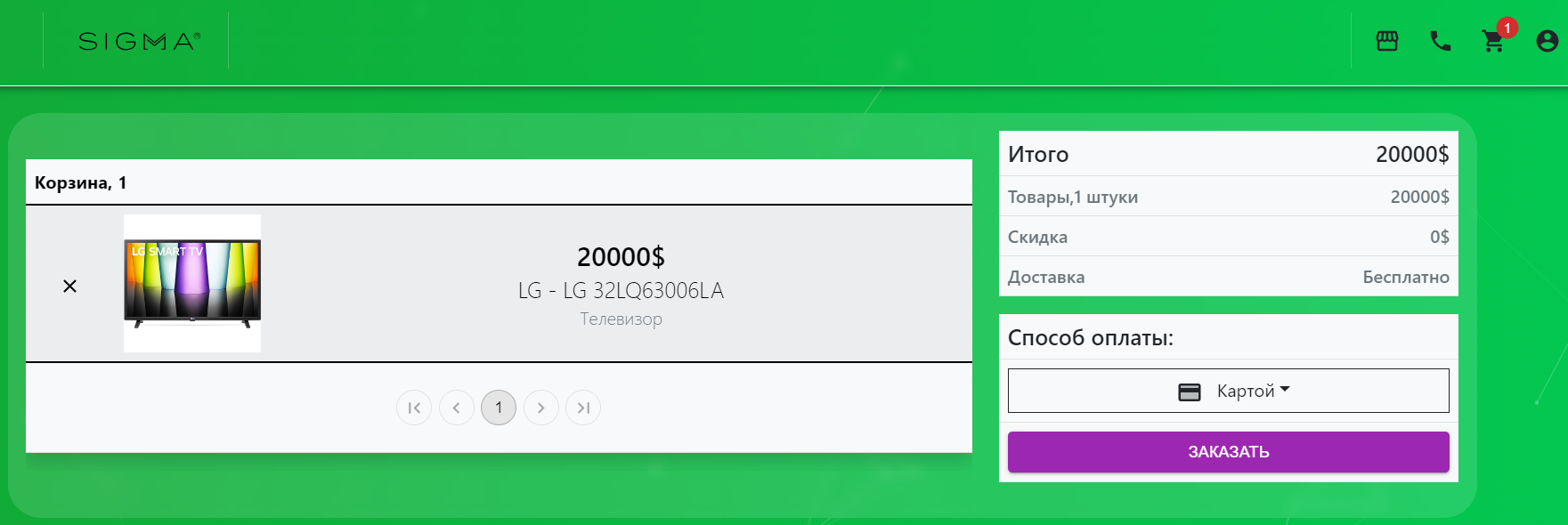


Рисунок 5.5 – Содержимое корзины

У администратора есть возможность взаимодействовать с таблицами базы данных, получать записи из некоторых значащих таблиц, добавлять товары, типы и бренды, а также взаимодействовать с пользователями посредством чата технической поддержки. Пример таблицы администратора представлен на рисунке 5.6.

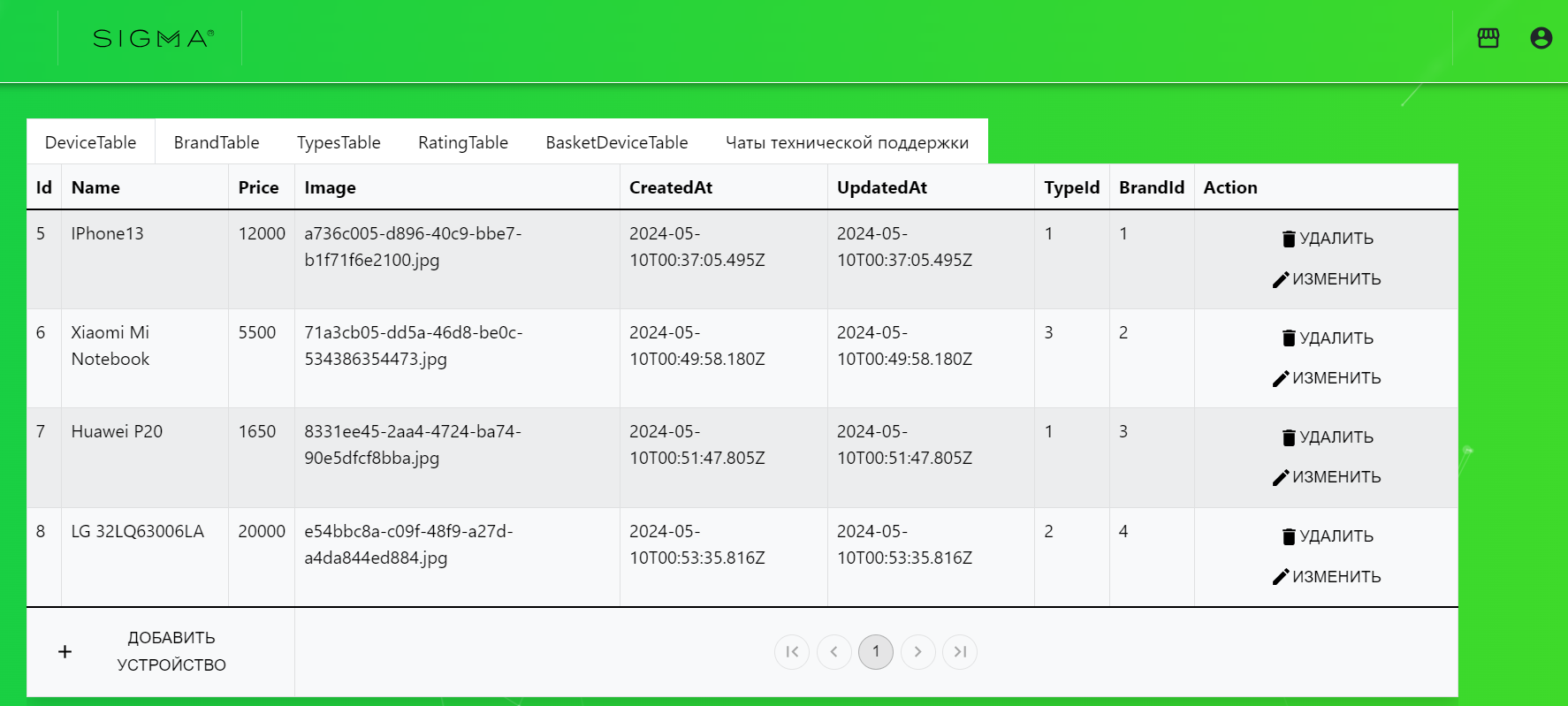


Рисунок 5.6 – Таблица администратора

Пользователь приложения может воспользоваться функционалом чата техподдержки. Перед входом в чат необходимо указать имя, под которым пользователь хочет в него войти. Данное окно также отображается администратору при попытке ответить на запрос пользователя. Окно ввода имени пользователя изображено на рисунке 5.7.

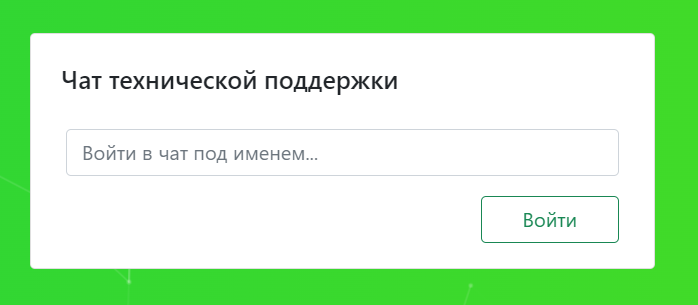


Рисунок 5.7 – Окно ввода имени пользователя

После ввода имени, пользователь переходит непосредственно в чат технической поддержки. До того момента, пока к чату не будет подключен администратор, пользователь не может отправить сообщение. Это связано с тем, каким образом реализован механизм обмена сообщений: они не хранятся на сервере или в базе данных. Все сообщения хранятся на стороне клиента и сервера до того момента, пока сеанс чата не будет завершён, и лишь для того, чтобы быть отображённым в окне чата клиента и администратора. Окно чата клиента до подключения представлено на рисунке 5.8.

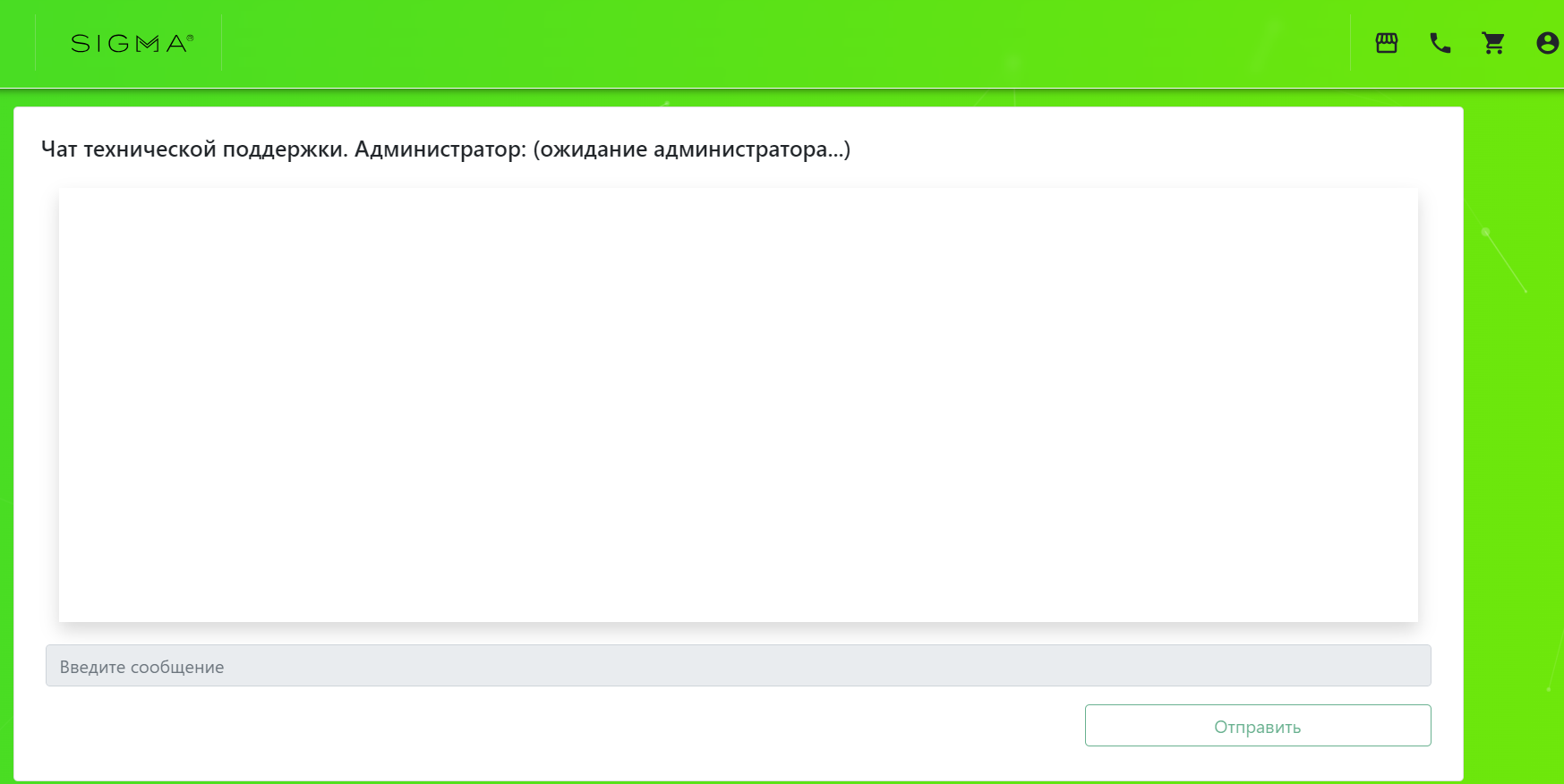


Рисунок 5.8 – Окно чата клиента

У администратора есть возможность добавлять и изменять товары. Модульное окно добавление товара представлено на рисунке 5.9.

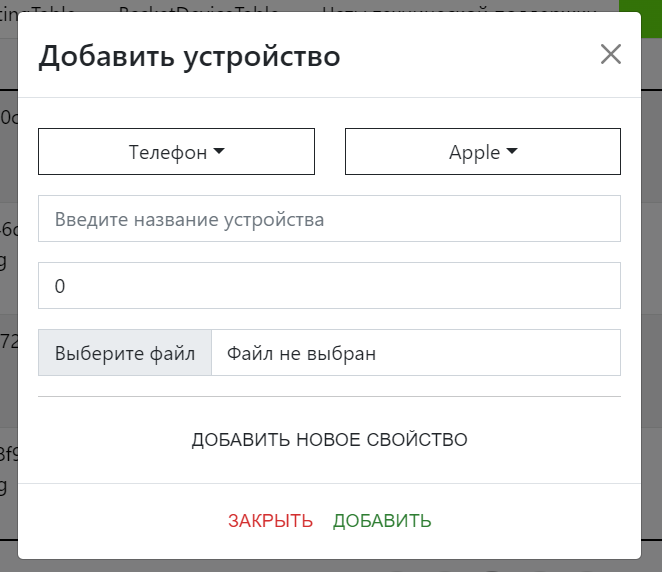


Рисунок 5.9 – Форма добавления

Модульное окно обновления товара представлено на рисунке 5.10.

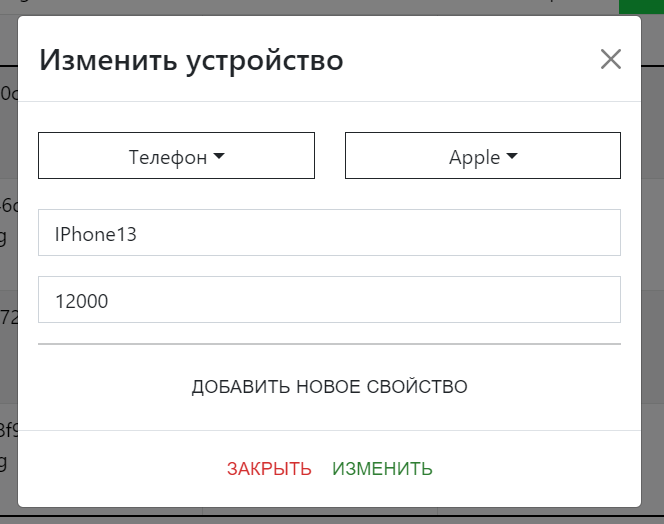


Рисунок 5.10 – Форма редактирования

Если пользователь попытается перейти на страницу или ресурс, к которому у него нет доступа, он будет перенаправлен на страницу ошибки, которая изменяет свой вид в зависимости от кода возникнувшей ошибки. Страница с сообщением о недостаточных правах для доступа к ресурсу изображена на рисунке 5.11.

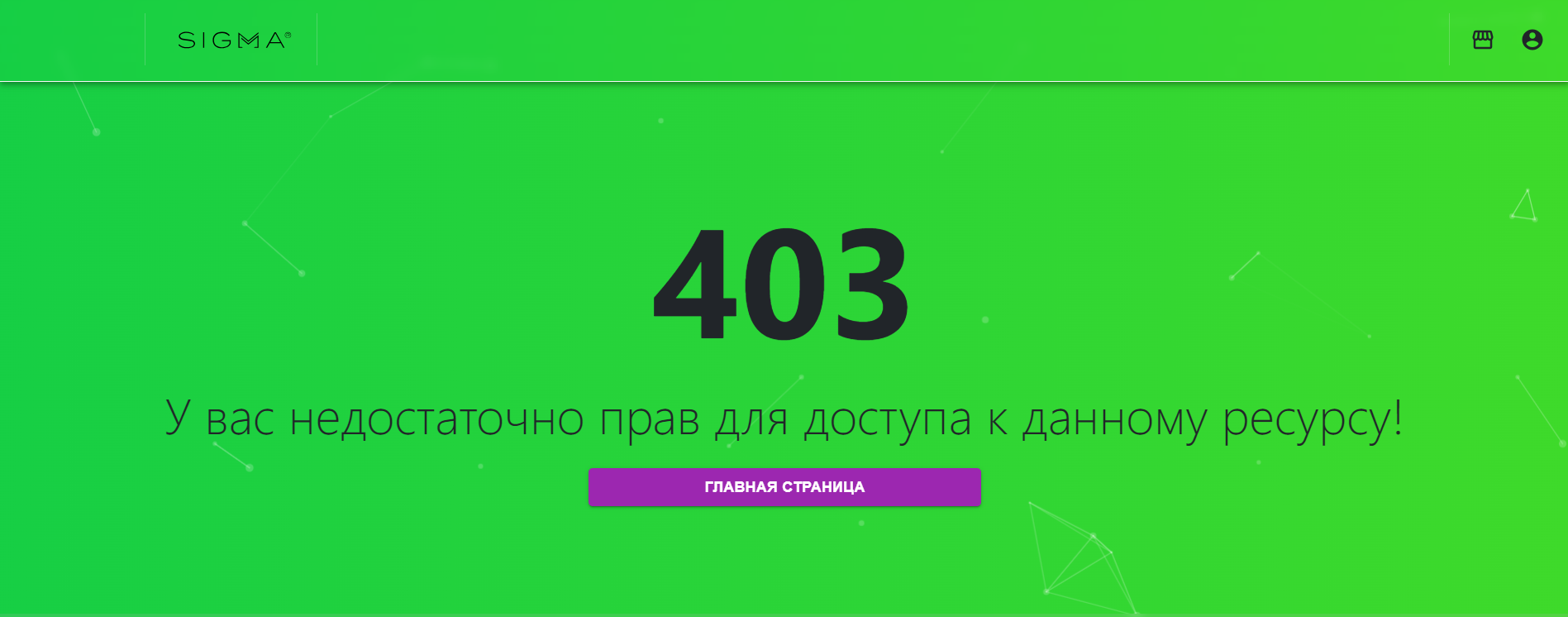


Рисунок 5.11 – Окно ошибки 403

Страница с сообщением о несуществующем ресурсе изображена на рисунке 5.12.

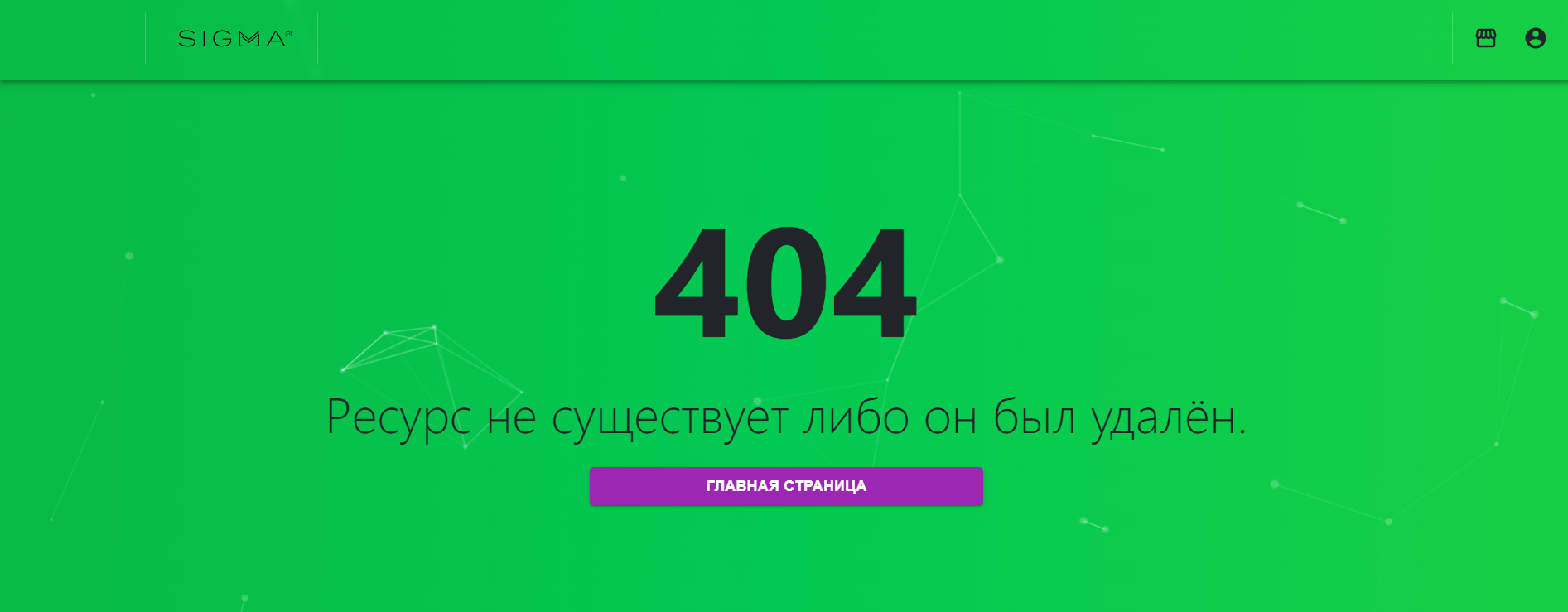


Рисунок 5.12 – Пользователь от лица администратора

Мы рассмотрели основной функционал данного приложения. Он должен помочь освоится новому пользователю. Мы увидели, что интерфейс более-менее понятный и отзывчивый. Дизайн приложения довольно-таки привлекательный.

## 5.2 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить серверную часть приложения, которая соединяет базу данных и React приложение. Для этого необходимо запустить скрипт, который настроит соединение с базой данных и запустит сервер.

2. Запустить React приложение, которое будет обрабатывать пользовательские запросы и взаимодействовать с сервером. Для этого необходимо запустить команду для сборки и запуска React приложения.

После выполнения этих шагов приложение будет полностью готово к работе и пользователь сможет начать использовать его функционал.

## 5.3 Выводы по разделу

В данном разделе мы рассмотрели, как пользоваться приложением. Рассмотрели, какими функциями обладает обычный пользователь, а какими функционалом обладает администратор. Рассмотрели, как администратор может добавлять новые товары, где администратор может посмотреть содержимое всей базы данных данного приложения. Рассмотрели способ связи пользователя с администратором с помощью чата. Также рассмотрели, как пользователь может заносить товары в корзину.

# Заключение

В процессе решения поставленной задачи была достигнута поставленная цель по созданию приложения «Магазин электроники». Основой целью курсового проекта стало проектирование кроссплатформенного приложение, которое помогло облегчить взаимодействие с пользователем. Это было достигнуто за счёт гибкой программной платформы сервера на Node.js и адаптивности сайта, в качестве клиентской части.

При разработке выполнены следующие пункты:

* авторизация и регистрация пользователя;
* создание новых учетных записей пользователя;
* оформление заказа пользователем;
* создание чата с представителем техподдержки;
* предоставление возможности доступа к таблицам БД администратору;
* поддержка беседы через чат с клиентом со стороны администратора.

В курсовом проекте были реализованы следующие задачи:

* создание базы данных;
* создание серверной части приложения;
* создание клиенткой части с пользовательским интерфейсом;
* реализация функций приложения;
* тестирование программного продукта.

Данный проект является показательным примером программного продукта, реализующего интернет-магазин.

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

Тестирование проекта показало, что приложение функционирует правильно и все поставленные тесты прошло. В ходе тестирования ошибок не было найдено.

Данной приложение готово к эксплуатации и использованию в сети Интернет.

# Список используемых источников

1. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 18.03.2024.
2. Все, что вы хотели знать о Sequelize [Электронный ресурс] / Режим доступа: https://habr.com/ru/post/566036/ – Дата доступа: 19.03.2024.
3. Руководство по React [Электронный ресурс] – Режим доступа: https://metanit.com/web/react/– Дата доступа: 21.03.2024.
4. Model Instances [Электронный ресурс] – Режим доступа: https://sequelize.org/master/manual/model-instances.html – Дата доступа: 26.03.2024.
5. How to Build and Structure a Node.js MVC Application [Электронный ресурс] – Режим доступа: <https://www.sitepoint.com/node-js-mvc-application/> – Дата доступа: 01.04.2024.
6. React hooks in Axios interceptors [Электронный ресурс] – Режим доступа: https://dev.to/arianhamdi/react-hooks-in-axios-interceptors-3e1h – Дата доступа: 25.04.2024.
7. How to Set Up a Websocket Server with Node.js and Express [Электронный ресурс] – Режим доступа: https://cheatcode.co/tutorials/how-to-set-up-a-websocket-server-with-node-js-and-express – Дата доступа: 28.04.2024.

# Приложение А

import {makeAutoObservable} from "mobx";

export default class DeviceStore{

constructor() {

this.\_updateName = ""

this.\_updatePrice = 0

this.\_updateId = 0

this.\_updateInfo = []

this.\_types = []

this.\_brands = []

this.\_devices = []

this.\_selectedType = {}

this.\_selectedBrand = {}

this.\_selectedRate =[0,100]

this.\_page = 1

this.\_totalCount = 0

this.\_limit = 10

makeAutoObservable(this)

}

setUpdateName(updateName)

{

this.\_updateName = updateName

}

setUpdateId(updateId)

{

this.\_updateId = updateId

}

setUpdatePrice(updatePrice)

{

this.\_updatePrice = updatePrice

}

setUpdateInfo(updateInfo)

{

this.\_updateInfo = updateInfo

}

setTypes(types) {

this.\_types = types

}

setBrands(brands) {

this.\_brands = brands

}

setDevices(devices) {

this.\_devices = devices

}

setSelectedType(type) {

this.setPage(1)

this.\_selectedType = type

}

setSelectedBrand(brand) {

this.setPage(1)

this.\_selectedBrand = brand

}

setPage(page) {

this.\_page = page

}

setTotalCount(count) {

this.\_totalCount = count

}

setSelectedRate(selectedRate)

{

this.\_selectedRate = selectedRate

}

get types() {

return this.\_types

}

get brands() {

return this.\_brands

}

get devices() {

return this.\_devices

}

get selectedType() {

return this.\_selectedType

}

get selectedBrand() {

return this.\_selectedBrand

}

get selectedRate() {

return this.\_selectedRate

}

get totalCount() {

return this.\_totalCount

}

get page() {

return this.\_page

}

get updateId() {

return this.\_updateId

}

get limit() {

return this.\_limit

}

get updateName()

{

return this.\_updateName

}

get updatePrice()

{

return this.\_updatePrice

}

get updateInfo()

{

return this.\_updateInfo

}

}

Листинг 1 – Пример класса объекта контекста

const AxiosInterceptor = ({ children }) => {

const {errorResult} = useContext(Context);

const navigate = useNavigate();

useEffect(() => {

const resInterceptor = response => {

return response;

}

const errInterceptor = error =>

{

if(error.response !== undefined){

if (error.response.status === 401){

navigate(LOGIN\_ROUTE);

}

else if(error.response.status === 403){

navigate(ERROR\_ROUTE,{state:{errorCode:403}});

}

errorResult.setMessage(error.response.data.message);

}

else{

errorResult.setMessage("Ошибка на стороне сервера");

}

return Promise.reject(error);

}

const authResponseInterceptor = $authHost.interceptors.response.use(resInterceptor,errInterceptor);

const responseInterceptor = $host.interceptors.response.use(resInterceptor,errInterceptor);

return () =>

{

$authHost.interceptors.response.eject(authResponseInterceptor);

$host.interceptors.response.eject(responseInterceptor);

}

},[navigate])

return children;

}

Листинг 2 – Компонент «AxiosInterceptor»

import {*Injectable*, Scope} from "@nestjs/common";  
import \* as *fs* from "fs";  
import {MemoryStoredFile} from "nestjs-form-data";  
import \* as path from "path";  
  
@Injectable({  
 scope: Scope.*REQUEST*})  
export class FilesService {  
 static readonly *stickersPath*: string = path.resolve(\_\_dirname, '../../public/img/stickers');  
 static readonly *usersPath*: string = path.resolve(\_\_dirname, '../../public/img/users');  
 static readonly *reactionsPath*: string = path.resolve(\_\_dirname, '../../public/img/reactions');  
  
 async saveUserFile(*file*: MemoryStoredFile): *Promise*<void> {  
 return fs.promises.*writeFile*(path.resolve(FilesService.*usersPath*, *file*.originalName), *file*.buffer);  
 }  
  
 async saveStickerFile(*file*: MemoryStoredFile): *Promise*<void> {  
 return fs.promises.*writeFile*(path.resolve(FilesService.*stickersPath*, *file*.originalName), *file*.buffer);  
 }  
  
 async saveReactionFile(*file*: MemoryStoredFile): *Promise*<void> {  
 return fs.promises.*writeFile*(path.resolve(FilesService.*reactionsPath*, *file*.originalName), *file*.buffer);  
 }  
  
 rename(*file*: MemoryStoredFile, *newName*: string): void {  
 *file*.originalName = *newName* + '.' + this.getExt(*file*)  
 }  
  
 getExt(*fileName*: MemoryStoredFile): string {  
 return *fileName*['fileType'].ext;  
 }  
  
 async deleteUserFile(*name*: string): *Promise*<void> {  
 return fs.promises.*unlink*(path.resolve(FilesService.*usersPath*, *name*))  
 }  
  
 async deleteStickerFile(*name*: string): *Promise*<void> {  
 return fs.promises.*unlink*(path.resolve(FilesService.*stickersPath*, *name*));  
 }  
  
 async deleteReactionFile(*name*: string): *Promise*<void> {  
 return fs.promises.*unlink*(path.resolve(FilesService.*reactionsPath*, *name*));  
 }  
  
 getNameForUserImg(*name*: string): string {  
 return `/static/img/users/${*name*}`  
 }  
  
 async saveFile(*path*: string, *buffer*: *Buffer*): *Promise*<void> {  
 return fs.promises.*writeFile*(*path*, *buffer*);  
 }  
  
 async isFileExist(*path*: string): *Promise*<boolean> {  
 try {  
 await fs.promises.*access*(*path*);  
 return true;  
 } catch (*e*: unknown) {  
 return false;  
 }  
 }  
  
 async deleteFile(*path*: string): *Promise*<void> {  
 return fs.promises.*unlink*(*path*);  
 }  
  
 async getFile(*path*: string): *Promise*<*Buffer*> {  
 return fs.promises.*readFile*(*path*);  
 }  
}

Листинг 3 – FilesService