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 магазина одежды

Выполнил студент Яшный Никита Сергеевич

(Ф.И.О.)

Руководитель проекта ст.преп. Дубовик М.В.

(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.)

Консультанты ст.преп. Дубовик М.В.

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

Нормоконтролер ст.преп. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 3](#_Toc165549346)

[1 Постановка задачи 4](#_Toc165549347)

[1.1 Аналитический обзор аналогов 4](#_Toc165549348)

[1.1.1 Zara 4](#_Toc165549349)

[1.1.2 Bershka 4](#_Toc165549350)

[1.1.3 Pull&Bear 5](#_Toc165549351)

[1.1.4 Lamoda 6](#_Toc165549352)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc165549353)

[1.3 Выводы по разделу 8](#_Toc165549354)

[2 Проектирование web-приложения 9](#_Toc165549355)

[2.1 Архитектура приложения 9](#_Toc165549356)

[2.2 Общая структура приложения 9](#_Toc165549357)

[2.3 Структура базы данных 10](#_Toc165549358)

[2.4 Выводы по разделу 11](#_Toc165549359)

[3 Разработка web-приложения 12](#_Toc165549360)

[3.1 Разработка клиентской части web-приложения 12](#_Toc165549361)

[3.2 Разработка серверной части web-приложения 16](#_Toc165549362)

[3.3 Выводы по разделу 19](#_Toc165549363)

[4 Тестирование web-приложения 20](#_Toc165549364)

[4.1 Тестирование web-приложения 20](#_Toc165549365)

[4.2 Выводы по разделу 21](#_Toc165549366)

[5 Руководство пользователя 22](#_Toc165549367)

[5.1 Руководство гостя 22](#_Toc165549368)

[5.2 Руководство пользователя 23](#_Toc165549369)

[5.3 Руководство администратора 25](#_Toc165549370)

[5.4 Установка приложения 28](#_Toc165549371)

[5.5 Выводы по разделу 28](#_Toc165549372)

[Заключение 29](#_Toc165549373)

[Список используемых источников 30](#_Toc165549374)

[Приложение А 31](#_Toc165549375)

# Введение

В современных реалиях самым ценным ресурсом человека является время. Новый темп жизни не позволяет тратить этот ограниченный ресурс на посещение физических магазинов для ознакомления с ассортиментом или же для поиска необходимого предмета гардероба. Постоянное развитие и увеличение количества молодых брендов является доступной роскошью для людей, проживающих в больших городах, однако население, живущее в населенных пунктах малого размера, не имеет данных привилегий.

Данное программное средство позволит решать многочисленные проблемы пользователей, включая ознакомление с ассортиментом бренда и подробной информацией по каждой позиции, актуальными предложениями, наличием определенных товаров в магазинах и даcт возможность сравнивать цены.

Возможность создания личного аккаунта пользователя позволит ему добавлять понравившиеся модели в корзину, сравнивать их, оформлять заказ, получать актуальную информацию по электронной почте и иметь доступ к истории заказов. Также у него будет возможность оставлять отзывы о товарах и ставить им оценки.

Данное программное средство станет незаменимо для администратора, который будет иметь удаленный доступ к базе данных, сможет добавлять и изменять позиции товаров, добавлять новые категории и бренды, выполнять поиск и сортировку товаров, отслеживать заказы и изменять их статус, оповещая клиентов по электронной почте.

Главная цель данного курсового проектирования – это разработка программного средства, предназначенного для обеспечения функциональности интернет-магазина одежды. Для достижения поставленной цели необходимо выполнить следующие задачи:

* рассмотреть и проанализировать существующие аналоги;
* спроектировать web-приложение;
* разработать web-приложение;
* протестировать web-приложение;
* описать руководство пользователя.

В пояснительной записке находится подробная информация о выполнении каждой из поставленных задач.

# 1 Постановка задачи

## 1.1 Аналитический обзор аналогов

В данном разделе будут приведены аналогичные программные продукты. Приведены их плюсы и минусы, что позволит нам подобрать наиболее эффективный подход к решению поставленной цели.

# Zara

Одним из самых популярных альтернативных решений является интернет-ресурс «Zara», являющаяся известным брендом молодёжной одежды, имеющим огромную клиентскую базу во всём мире. Главным преимуществом данного ресурса является дизайн, передающий эстетику бренда. Также у сайта широкий функционал, доступный любому пользователю: удобный поиск товара, наличие сортировки по категории, полу, размеру. Данное программное средство предоставляет пользователю возможность легко найти нужный товар, добавить его в корзину и совершить покупку. Интерфейс интернет-ресурса представлен на рисунке 1.1.

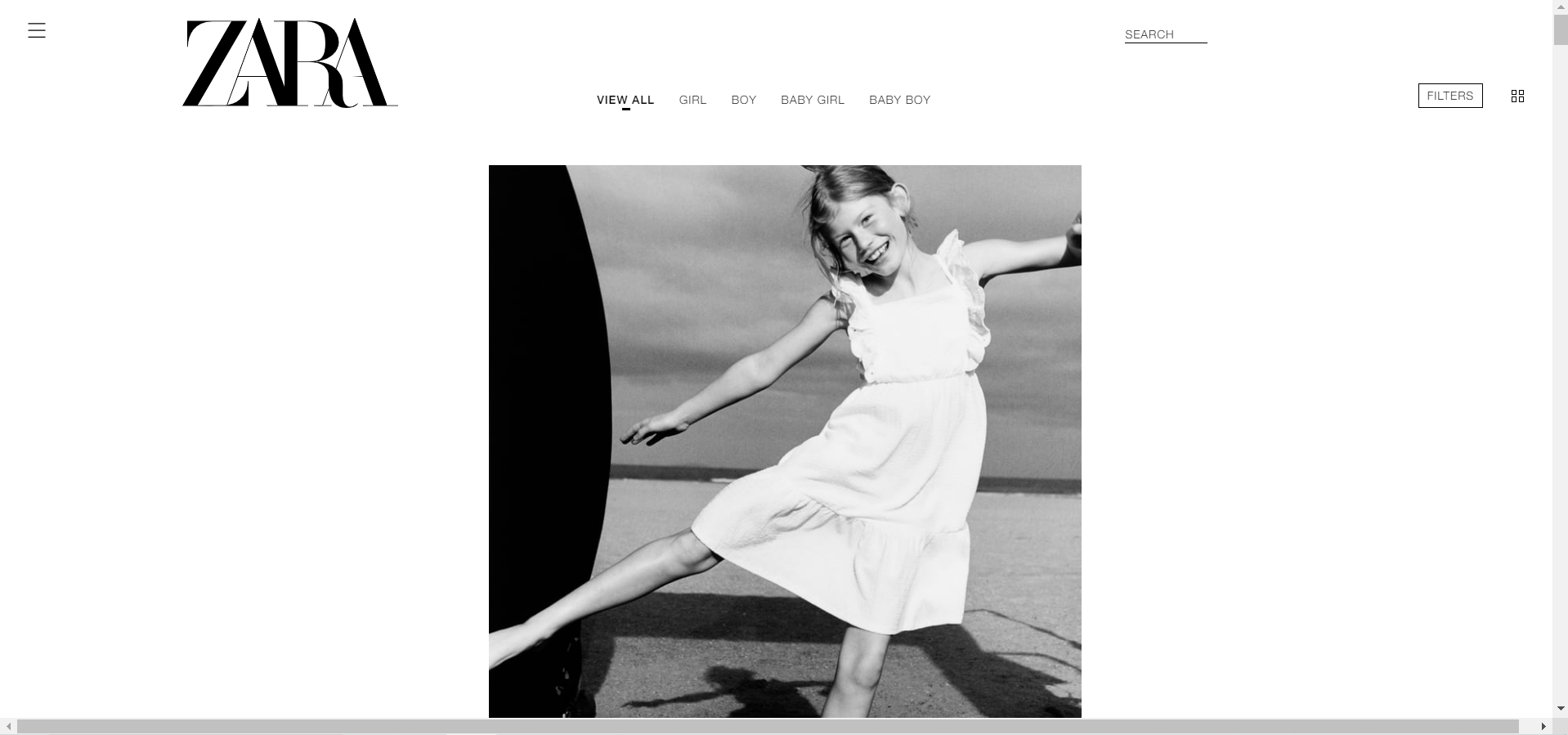


Рисунок 1.1 – Интернет-ресурс «Zara»

К недостаткам данного ресурса можно отнести отсутствие возможности зарезервировать товар на какое-то время, что гарантирует пользователю, что товар не исчезнет после попадания в корзину. Также на сайте нет возможности просмотреть популярные товары и количество продаж за определённые периоды.

# Bershka

Ещё одним альтернативным решением задач, поставленных в данном курсовом проекте, является интернет-ресурс «Bershka». С помощью данного программного средства пользователь может ознакомиться с каталогом данного бренда одежды, выбрать для себя подходящий по вкусу товар, а также оформить доставку, введя данные своей банковской карты.

Магазин позиционирует себя как молодёжный, что явно видно из его стилистического оформления и настроения. Это положительным образом сказывается на опыте пребывания на данном сайте.

Интерфейс интернет-ресурса представлен на рисунке 1.2.

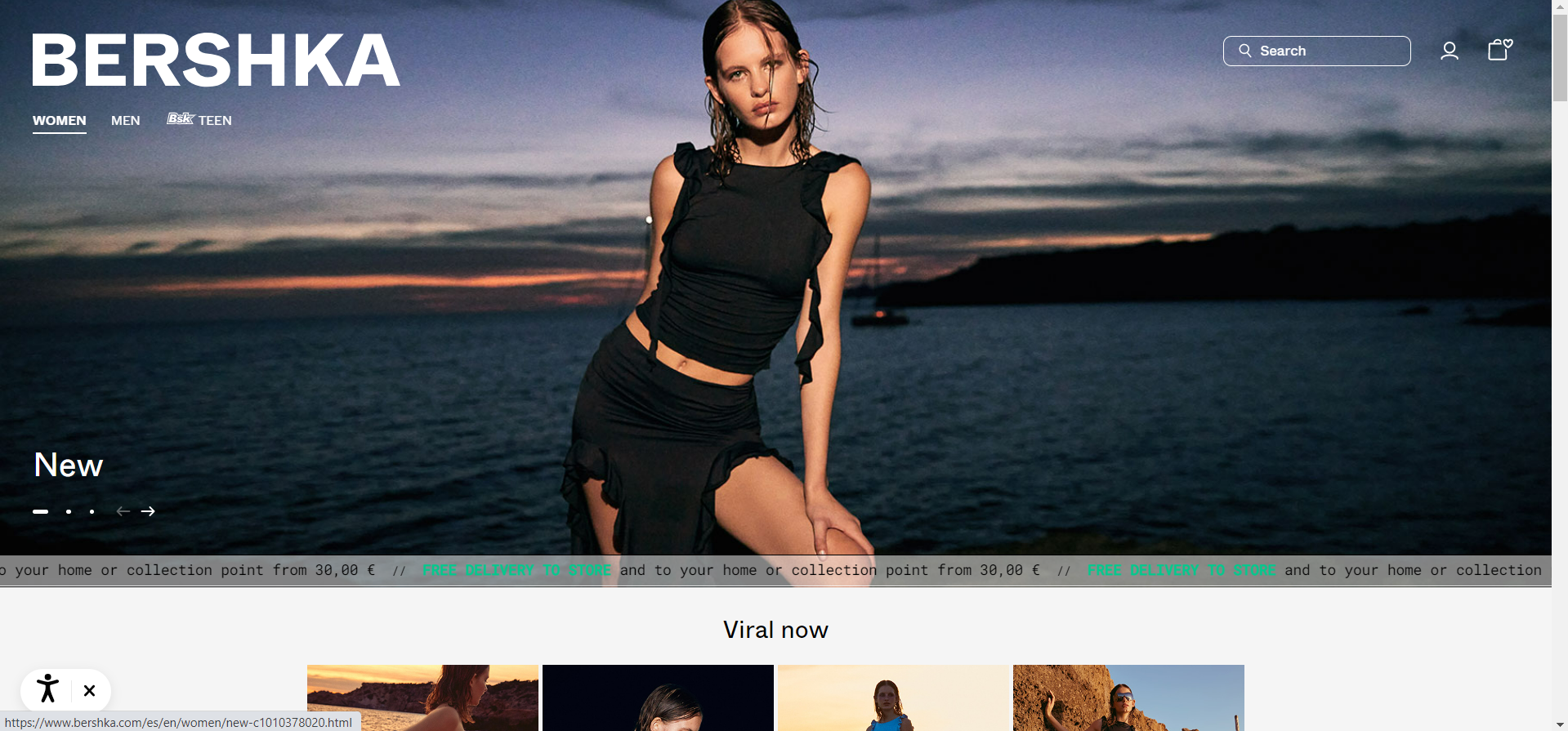


Рисунок 1.2 – Интернет-ресурс «Bershka»

Одним из главных недостатков такой альтернативы является отсутствие возможности посмотреть количество оставшихся на складе товаров, а также невозможность получить отзыв от других пользователей. В данном интернет-ресурсе тоже отсутствует возможность зарезервировать понравившийся товар на какое-то время.

# Pull & Bear

Ещё одной альтернативой решения поставленных задач является интернет-ресурс «Pull&Bear». С помощью данного программного средства пользователь может ознакомиться со списком доступных для покупки товаров, выбрать доступную для любого бюджета и стиля одежду и аксессуары.

Немаловажным преимуществом является то, что пользователь может оценить понравившийся образ на многочисленных фотографиях моделей. Также немаловажна возможность быть уведомлённым о заказе путём получения письма по электронной почте. На сайте удобное разделение на мужскую, женскую и детскую категории.

Как и в предыдущих примерах пользователь может оформить доставку на понравившийся товар, введя личные данные в кабинете.

Однако, недостатком данной альтернативы являются отсутствие качественной оптимизации, из-за чего даже при наличии высокоскоростного интернета и мощного устройства многие разделы долго загружаются. Также, как и у приведённых ранее примеров альтернатив, на сайте отсутствует возможность резервирования и просмотра популярных позиций товаров. Интерфейс интернет-ресурса «Pull&Bear» представлен на рисунке 1.3.

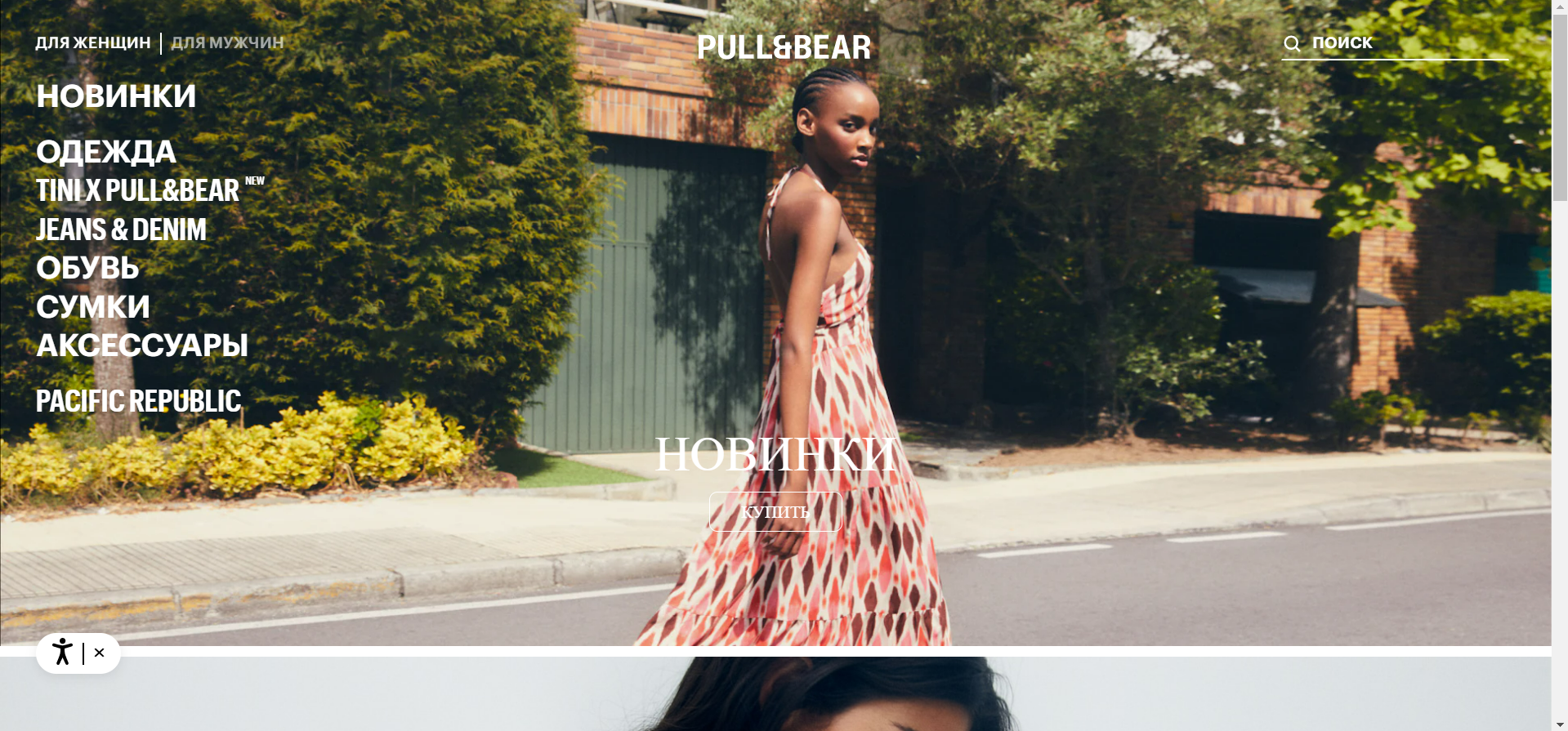


Рисунок 1.3 – Интернет-ресурс «Pull&Bear»

Данный интернет-ресурс является качественным примером решения поставленных задач.

# Lamoda

В список альтернатив, рассматриваемых для решения поставленных задач, также можно включить интернет-ресурс «Lamoda». Данное программное средство позволяет ознакомиться с предложениями от различных брендов в рамках одного маркетплейса. Приложение позволяет выбирать товар, сортируя по брендам и популярности, что является крайне удобным при поиске необходимого товара.

Интерфейс интернет-ресурса представлен на рисунке 1.4.

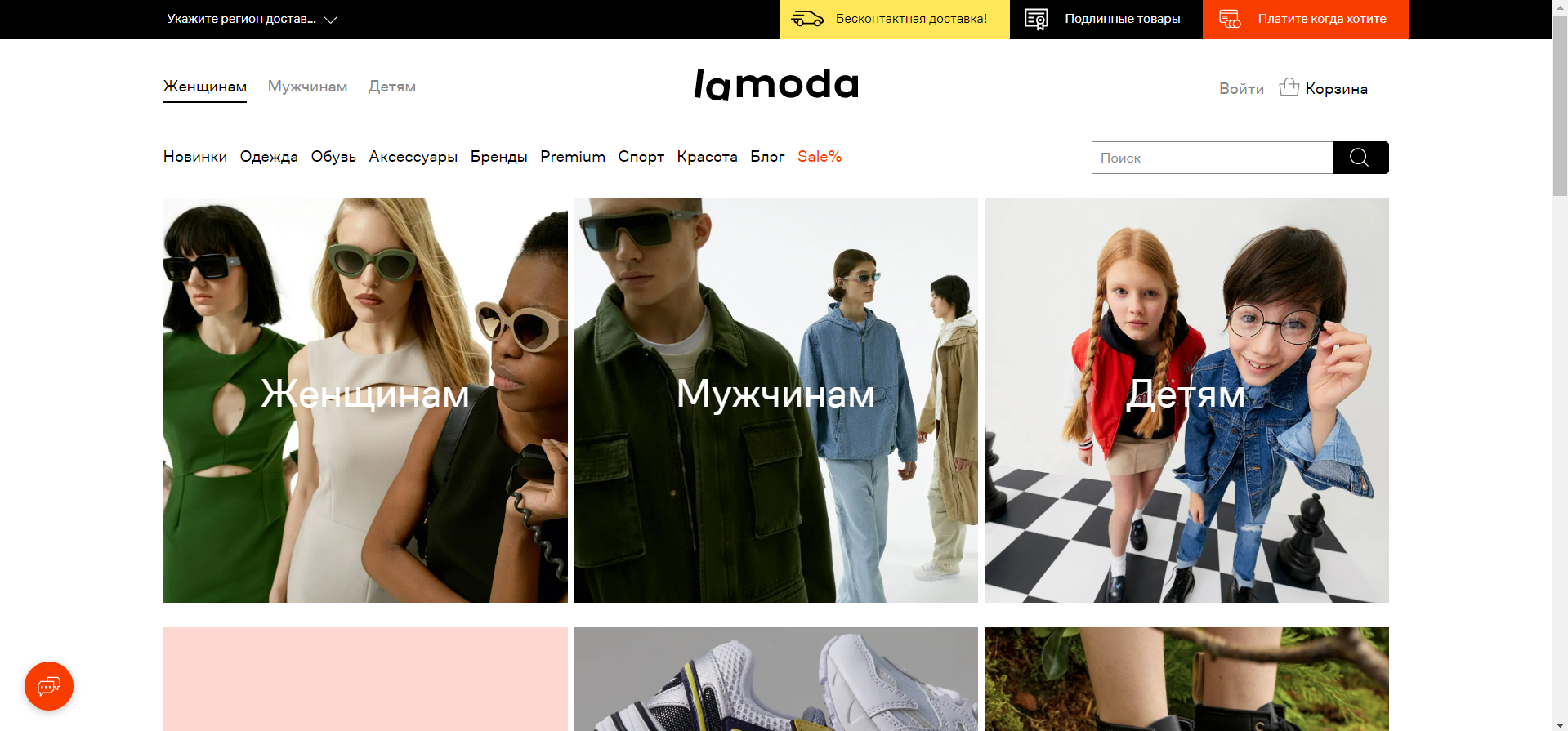


Рисунок 1.4 – Интернет-ресурс «Lamoda»

Недостатком данного программного средства является отсутствие возможности легко отменить свой заказ прямо с сайта без связи с продавцом и так же, как у конкурентов отсутствует возможность резервирования.

## 1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования приложения определяют, как оно должно обрабатывать данные и запросы пользователя и предоставлять ему необходимый пользовательский интерфейс. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные.

Основные задачи к приложению:

* обеспечение возможности регистрации и авторизации;
* поддержка ролей администратора, пользователя и гостя;
* наличие оповещения клиента о выполнении заказа по email;
* обеспечение возможности выполнять поисковые запросы;
* обеспечение предоставления каталога товаров;
* обеспечение возможности добавления товаров администратору;
* наличие корзины;
* наличие фильтра по категориям;
* обеспечение возможности оставлять отзывы;
* обеспечение возможности изменять статус заказа администратору;
* обеспечение возможности редактирования товаров администратору.

Для хранения данных приложение должно использовать базу данных. Страницы должны обновляться без перезагрузки. Вход в приложение должен осуществляться только после авторизации.

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли.

В данном проекте роли пользователей будут следующими:

* GUEST;
* USER;
* ADMIN;

Роль GUEST позволяет просмотреть отзывы и каталог, осуществить поиск товара, применив фильтрацию по бренду или категории, либо искать товар по названию, введя фрагмент в поисковое поле, а также создать новый аккаунт. После создания аккаунта пользователь получает роль USER.

Роль USER дает возможность использования всех прав роль GUEST. Помимо этого, ему открывается возможность оставить или удалить свой отзыв, добавить товар в корзину, поставить оценку товару и оформить заказ.

Роль ADMIN также позволяет использовать права ролей GUEST и USER, а также добавить новую категорию товаров, новый бренд, изменять статусы заказов клиентов, присылая им сообщение на электронную почту, добавлять новые товары и изменять о них информацию.

На основе предоставленного списка ролей была построена uml-диаграмма вариантов использования. Диаграмма использования изображена на рисунке 1.5.

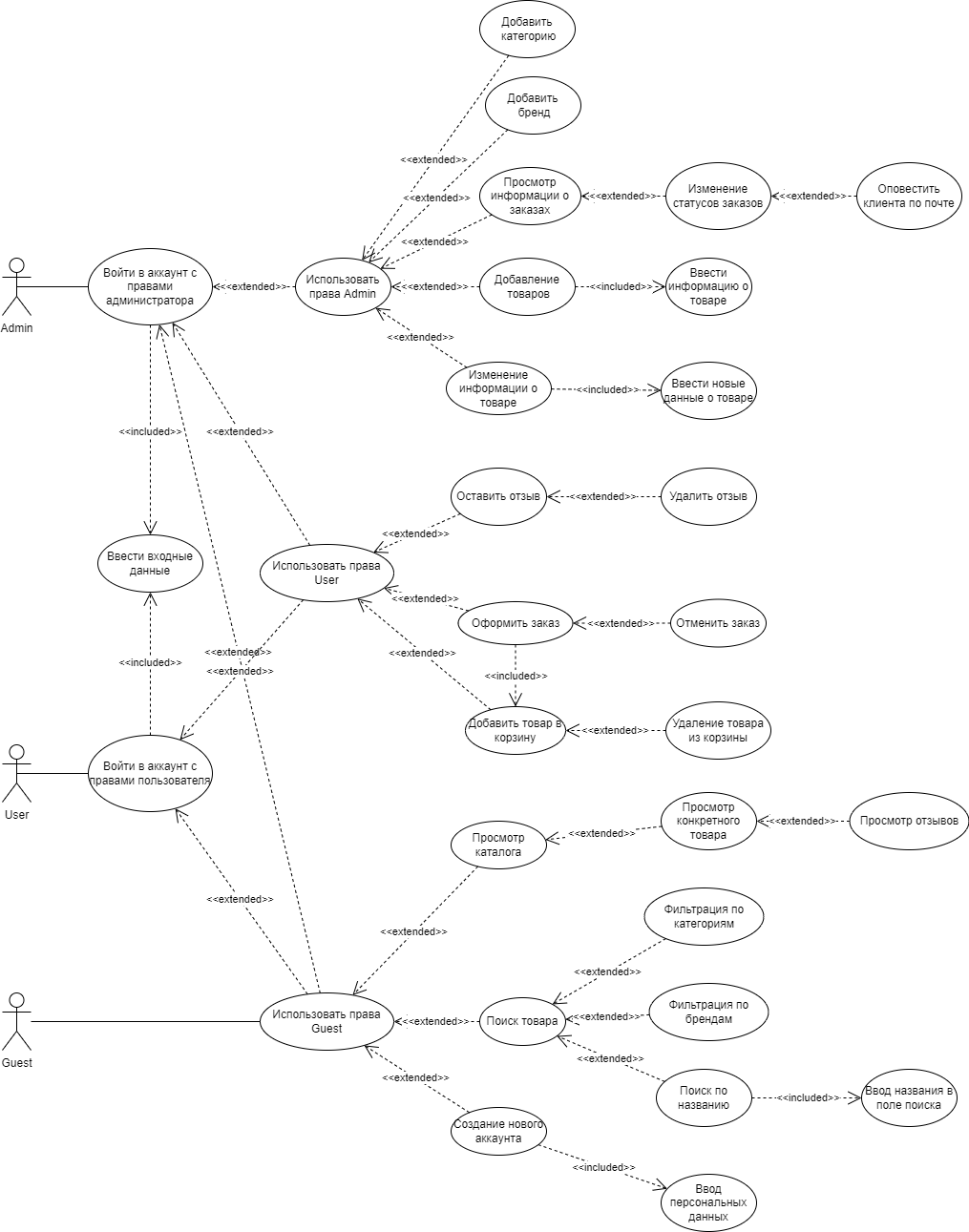


Рисунок 1.5 – Диаграмма вариантов использования

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

## 1.3 Выводы по разделу

Итого, был проведен аналитический обзор аналогов, которые уже существуют на рынке, выделены их главные плюсы и минусы. Этот обзор позволил определить основные характеристики и функциональные возможности, которые необходимо предусмотреть в разрабатываемой системе. Были определены функциональные требования приложения. Роли пользователей включают гостя, авторизированного пользователя и администратора. На основе каждой роли пользователю выделяется ряд функциональных возможностей. Была построена UML-диаграмма, демонстрирующая привилегии каждой из ролей

# 2 Проектирование web-приложения

## 2.1 Архитектура приложения

Приложение построено на основе клиент-серверной архитектуры. Клиентская часть приложения разработана с использованием React, а серверная часть основана на Node.js и Express. Для хранения и управления данными используется система управления базами данных PostgreSQL [1].

Клиентскую часть называют frontend. Для реализации пользовательского интерфейса был выбран React. Он предоставляет эффективное модульное решение для создания пользовательского интерфейса. React-компоненты отвечают за отображение данных и взаимодействие с пользователем. Клиентская часть отправляет HTTPS-запросы на сервер для получения актуальных данных и обновления информации.

Серверную часть называют backend. Для её реализации был выбран Node.js, он является средой выполнения JavaScript на стороне сервера, обеспечивает высокую производительность и масштабируемость. Express – это минималистичный и гибкий фреймворк для создания web-приложений на основе Node.js [2]. Он облегчает разработку маршрутов, а также управление сеансами и промежуточным программным обеспечением. Сервер обрабатывает запросы от клиента, включая маршрутизацию запросов к соответствующим обработчикам.

Выбор системы управления базами данных остановился на PostgreSQL [3], она является реляционной СУБД с открытым исходным кодом, предоставляет надежное хранение и управление структурированными данными. Node.js сервер использует ORM Sequelize [4] для CRUD операций.

Взаимодействие между клиентской и серверной частью приложения осуществляется следующим образом: клиент отправляет HTTPS-запросы на сервер, используя AJAX. Сервер обрабатывает запросы, выполняет требуемые операции в базе данных и формирует HTTPS-ответы. Ответы сервера в формате JSON. Клиентская часть обрабатывает ответы от сервера и обновляет пользовательский интерфейс соответствующим образом.

## 2.2 Общая структура приложения

Приложение состоит из основных частей. Первая – это клиентская часть приложения. Эта часть при помощи библиотеки React, а также модуля React-bootstrap, который облегчает вёрстку приложения. Данная библиотека позволяет создать приложение, которое будет делать запросы по протоколу HTTPS к серверной части приложения. Вторая часть приложения – серверная, разработана на платформе NodeJS при помощи фреймворка Express, который позволяет построить структуру из всех модулей приложения. Данная часть будет обращаться к базе данных по протоколу TCP. Обращение к базе данных будет упрощено использованием модуля ORM Sequelize, который позволяет создать модели объектов базы данных и работать с ними. Сама база данных будет работать на сервере PostgreSQL, который отвечает за хранение всех данных, необходимых для корректной работы приложения. Диаграмма развёртывания приложения представлена на рисунке 2.1.

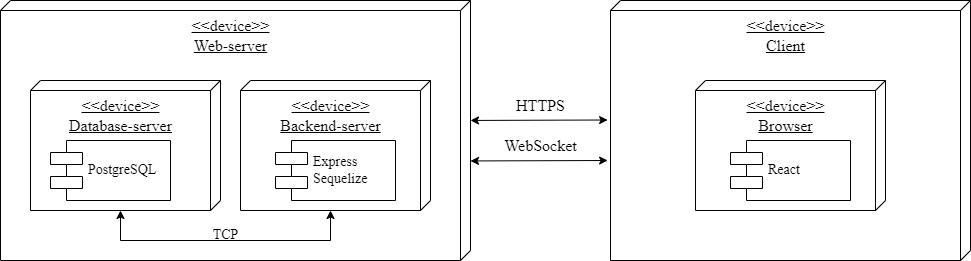


Рисунок 2.1 – Диаграмма развёртывания приложения

На диаграмме представлены все вышеописанные связи основных модулей приложения.

## 2.3 Структура базы данных

Логическая схема базы данных – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных.

База данных состоит из 13 таблиц, связанных между собой внешними ключами. Диаграмма показывает связи между таблицами и полями. Логическая схема базы данных представлена на рисунке 2.2.

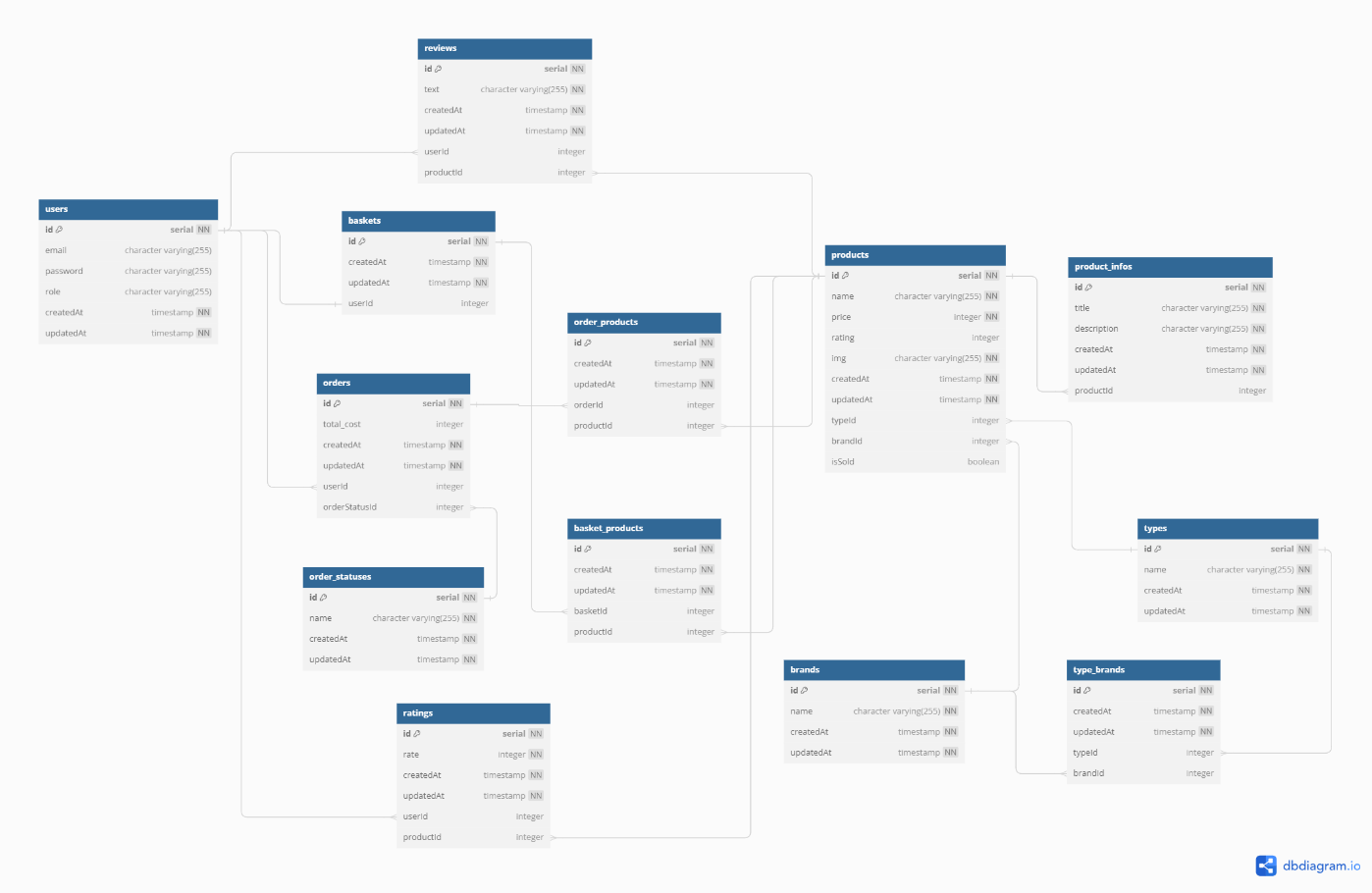


Рисунок 2.2 – Логическая схема базы данных

Таблица «USER» хранит информацию о пользователях, в ней хранится идентификатор пользователя, его email, пароль и роль.

Таблица «PRODUCT» хранит информацию о товарах, состоит из идентификатора, названия, цены, рейтинга, изображения, идентификатора типа, идентификатора бренда и статуса isSold, показывающего продан товар или нет. Связана с таблицами «TYPE» и «BRAND» по столбцам typeId и brandId соответственно.

Таблица «PRODUCT\_INFO» хранит информацию о характеристиках товаров, состоит из идентификатора, идентификатора товара, названия и описания. Связана внешним ключом с таблицей «PRODUCT» по столбцу product\_id.

Таблица «REVIEW» хранит информацию об отзывах, состоит из идентификатора, идентификатора пользователя, идентификатора товара и текста. Связана с таблицами «USER» и «PRODUCT» по столбцам user\_id и product\_id соответственно.

Таблица «BASKET» хранит информацию о корзине, состоит из идентификатора и идентификатора пользователя. Связана с таблицей «USER» по столбцу user\_id.

Таблица «BASKET\_PRODUCT» хранит информацию о наполнении корзины, состоит из идентификатора, идентификатора корзины и идентификатора товара. Связана с таблицами «BASKET» и «PRODUCT» по столбцам basket\_id и product\_id соответственно.

Таблица «ORDER» хранит информацию о заказах, состоит из идентификатора, идентификатора пользователя и идентификатора статуса заказа. Связана с таблицей «USER» и «ORDER\_STATUS» по столбцам user\_id и status\_id соответственно.

Таблица «ORDER\_PRODUCT» хранит информацию о составе заказа, состоит из идентификатора, идентификатора заказа, идентификатора товара и количестве. Связана с таблицами «ORDER» и «PRODUCT» по столбцам order\_id и product\_id соответственно.

Таблица «ORDER\_STATUS» хранит информацию о статусах заказа, состоит из идентификатора и названия статуса.

Таблица «RATING» хранит информацию о рейтинге товара, состоит из идентификатора, идентификатора пользователя, идентификатора товара и оценки. Связана с таблицами «USER» и «PRODUCT» по столбцам user\_id и product\_id соответственно.

Таблица «TYPE» хранит информацию о категориях товара, состоит из идентификатора и названия типа.

Таблица «BRAND» хранит информацию о брендах, состоит из идентификатора и названия бренда.

Таблица «TYPE\_BRAND» хранит информацию о связи бренда и категории, состоит из идентификатора, идентификатора бренда и идентификатора категории.

## 2.4 Выводы по разделу

В этой главе была рассмотрена архитектура приложения. Описаны основные компоненты: клиентская часть, реализованная с использованием React, и серверная часть, основанная на Node.js и Express. Описано взаимодействие между клиентской и серверной частью. Была представлена общая структура приложения, а также детально описана структура базы данных

# 3 Разработка web-приложения

## 3.1 Разработка клиентской части web-приложения

Основной частью приложения является клиентская, ведь именно с ней взаимодействует конечный пользователь. Она должна иметь интуитивно понятный интерфейс. Клиентская часть будет построена с помощью библиотеки React, а основные интерактивные элементы реализованы с помощью React-bootstrap.

При входе в приложение неавторизированному пользователю будут доступны только 4 из 7 основных страниц, а именно главная страница магазина, на которой он сможет ознакомиться с каталогом товаров, страница товаров с полной информации о нём, страница с описанием приложения и страница регистрации. После регистрации пользователю становится доступна страница с его корзиной, страница с заказами и страница с подробным описанием каждого товара в заказе. Также в приложении предусмотрена страница с административной панелью, доступ к которой имеют только пользователи с правами администратора.

Для маршрутизации между страницами в React используется компонент Router. В файле routes.js описаны маршруты доступные каждой роли. Код файла routes.js представлен в листинге 3.1.

export const authRoutes = [{

path: BASKET\_ROUTE,

Component: Basket}, {

path: ORDER\_ROUTE,

Component: Orders }, {

path: ORDER\_ROUTE + '/:userId' + '/:orderId',

Component: OrderPage }]

export const adminRoutes = [ {

path: ADMIN\_ROUTE,

Component: Admin

},]

export const publicRoutes = [ {

path: SHOP\_ROUTE,

Component: Shop }, {

path: LOGIN\_ROUTE,

Component: Auth }, {

path: REGISTRATION\_ROUTE,

Component: Auth }, {

path: PRODUCT\_ROUTE + '/:id',

Component: ProductPage }, {

path: ABOUTUS\_ROUTE,

Component: Aboutus }]

Листинг 3.1 – Содержимое файла routes.js

Из листинга видно, какие маршруты доступны каждому из трёх видов пользователей.

Для взаимойдействия с серверной частью используется модуль axios, содержащий интерцептор, проверяющий авторизирован пользователь или нет. Проверка авторизации осуществляется через jwt-токен. В случае если у пользователя нет jwt-токена, то ему присвается $host, иначе $authHost. Описана данная логика в файле index.js из папки http. Содержимое файла представлено в листинге 3.2.

const $host = axios.create({

baseURL: process.env.REACT\_APP\_API\_URL

})

const $authHost = axios.create({

baseURL: process.env.REACT\_APP\_API\_URL

})

const authInterceptor = config => {

config.headers.authorization = `Bearer ${localStorage.getItem('token')}`

return config

}

$authHost.interceptors.request.use(authInterceptor)

export {

$host,

$authHost,

}

Листинг 3.2 – Содержимое файла index.js папки http

Начало работы приложения начинается в файле index.js в основной директории проекта, в котором происходит загрузка основного компонента App. Также в нем используется Context.Provider, позволяющий пользоваться хранилищами user и product для хранения их состояний и последующего их использования в приложении. Содержимое index.js представлено в листинге 3.3.

const Context = createContext(null)

createRoot(document.getElementById('root')).render(

<Context.Provider value={{

user: new UserStore(),

product: new ProductStore()

}}>

<App />

</Context.Provider>),}

Листинг 3.3 – Содержимое файла index.js

Хранилище user содержит состояния пользователя, а именно авторизован ли он, его основную информацию из базы данных, включая идентификатор, адрес электронной почты, является ли он администратором. Хранилище product содержит информацию о товарах, такую как список брендов, список категорий, список самих товаров, выбранный бренд, выбранную категорию, текущую страницу, общее количество страниц товаров, ограничение на количество товаров на странице. Доступ к данным полям осуществляется с помощью геттеров и сеттеров. В качестве примера приведено содержимое файла хранища ProductStore.js в листине 3.4.

export default class ProductStore {

constructor()

{

this.\_types = []

this.\_brands = []

this.\_products = []

this.\_selectedType = {}

this.\_selectedBrand = {}

this.\_page = 1

this.\_totalCount = 0

this.\_limit = 8

makeAutoObservable(this)

}

setTypes(types) {this.\_types = types}

setBrands(brands) {this.\_brands = brands}

setProducts(products) {this.\_products = products}

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; }

setLimit(limit) {this.\_limit = limit; }

get types() {return this.\_types}

get brands() {return this.\_brands}

get products() {return this.\_products}

get selectedType() {return this.\_selectedType}

get selectedBrand() {return this.\_selectedBrand}

get totalCount() {return this.\_totalCount}

get page() {return this.\_page}

get limit() {return this.\_limit}

}

Листинг 3.4 – Содержимое файла ProductStore.js

Первая страница которая открывается пользователю это главная страница магазина – Shop. При её загрузке с помощью useEffect() сразу заполняются состояния брендов и категорий, а также происходит загрузка товаров. На ней расположен каталог товаров, окно поиска, область фильтрации товаров по брендам и по категориям. Также сверху находится панель навигации по сайту, откуда можно перейти на страницу авторизации, корзину, заказам, а администратору ещё и в администраторскую панель. Для размещения элементов используется React-bootstrap позволяющий размещать компоненты по строкам и колонкам, что значительно упрощает вёрстку и адаптацию приложения под различные устройства. Так же в нём имеются стили для кастомизации основных элементов управления таких как кнопки, панели ввода и загрузки данных. Все компоненты оборачиваются в контейнер для более удобного их размещения. Код реализации главной страницы Shop представлен в листинге 3.5.

const Shop = observer(() => {

const {product} = useContext(Context)

useEffect(() => {

fetchTypes().then(data => product.setTypes(data))

fetchBrands().then(data => product.setBrands(data))

}, [])

useEffect(() => {

fetchProducts(product.selectedType.id, product.selectedBrand.id, false, product.page, product.limit).then(data => {

product.setProducts(data.rows)

product.setTotalCount(data.count)

})

}, [product.page, product.selectedType, product.selectedBrand])

return (

<Container>

<SearchBar/>

<Row className="mt-2">

<Col md={3}>

<TypeBar/>

</Col>

<Col

md={9}

>

<BrandBar/>

<ProductList/>

<Pages/>

</Col>

</Row>

</Container>

)

})

export default Shop;}

Листинг 3.5 – Содержимое файла Shop.js

В компоненте ProductList, который представляет собой список товаров, содержится массив из компонентов ProductItem, для загрузки которого используется хук useState(), позволяющий загрузить данные из хранилища. Для перехода между страницами используется useNavigate() из пакета mobx. В администраторской панели взаимодействие с товарами и заказами пользователей реализовано через модальные окна предоставляемые React-bootstrap. Функции, отправляющие запросы на сервер, используя axios, находятся в файлах productAPI.js и userAPI.js, а строка, позволяющая установить соединение с сервером находится в файле .env. Содержимое файла userAPI.js представлено в листинге 3.6.

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

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

localStorage.setItem('token', data.token)

return jwtDecode(data.token)

}

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

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

localStorage.setItem('token', data.token)

return jwtDecode(data.token)

}

export const check = async () => {

try {

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

localStorage.setItem('token', data.token)

return jwtDecode(data.token)

}

catch (e) {

console.log('неавторизован')

}

}

Листинг 3.6 – Содержимое файла userAPI.js

Из листинга видно, что для проверки пользователя используется jwt-токен сохраняемый в localStorage браузера, что позволяет возвращаться на сайт без необходимости повторной авторизации.

Соединение устанавливается по протоколу https. Для этого был установлен самоподписанный сертификат.

## 3.2 Разработка серверной части web-приложения

Серверная часть написана с использованием фреймворка Express платформы Node.js.

Первым и самым главным файлом является index.js. В нём происходит загрузка основых модулей и осуществляется запуск приложения. Основными модулями являются cors, позволяющий использовать ресурсы с другого домена, express.json для обработки данных в JSON формате, router для маршрутизации и sequelize для работы с базой данных. Файлы, содержащие сертификат и приватный ключ, находятся в главной директории проекта, они позволяют использовать https соединение вместо обычного http. Также подключён модуль ws, позволяющий создать websocket-сервер, необходимый для отправки подключившимся клиентам сообщений. Поскольку используется модуль cors и https протокол, необходимо разрешить заголовки и порт для установления соединения WebSocket. Часть кода создания https-сервера и web-socket сервера приведена в листинге 3.7.

await sequelize.authenticate()

await sequelize.sync()

const server = https.createServer(options, app).listen(PORT, () => { console.log(`Server started on port ${PORT}`);

}); const wss = new WebSocket.Server({ server });

wss.on('headers', (headers, request) => {

headers.push('Access-Control-Allow-Origin: https://localhost:3000');

headers.push('Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept');

headers.push('Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS');

});

wss.on('connection', (socket) => {

const clientId = uuidv4();

console.log('New client connected:', clientId);

setTimeout(() => {

const message = 'Благодарим за использование нашего сервиса, удачных покупок!';

socket.send(message);

}, 60000); socket.on('close', () => {

console.log('Client disconnected:', clientId);

}); });

Листинг 3.7 – Фрагмент файла index.js, создающий https-сервер и websocket-сервер

Для взаимодействия с базой данных PostgreSQL используется модуль Sequelize. Описываются сущности базы данных используя define(), устанавливаются их атрибуты и ограничения. Для настройки связей между сущностями используются методы hasOne, hasMany, belongsTo. Описание происходит в файле models.js. Фрагмент содержимого файла представлен в листинге 3.8.

const Product = sequelize.define('product', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true}, name: {type: DataTypes.STRING, unique: true, allowNull: false}, price: {type: DataTypes.INTEGER, allowNull: false},

rating: {type: DataTypes.INTEGER, defaultValue: 0},

img: {type: DataTypes.STRING, allowNull: false},

isSold: {type: DataTypes.BOOLEAN, defaultValue: false}})

const Basket = sequelize.define('basket', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true}})User.hasOne(Basket)Basket.belongsTo(User)

User.hasOne(Rating)Rating.belongsTo(User)}

Листинг 3.8 – Фрагмент файла model.js

Полный код файла model.js представлен в приложении А. Доступ к основному функционалу осуществляется через роутер, поставляемый модулем Router из express. Он позволяет описать маршруты к контроллерам, содержащим функции приложения. Также позволяет добавить промежуточное программное обеспечение, такое как проверка авторизации, проверка роли и обработка ошибок. Промежуточное программное обеспечение находится в папке middleware в файлах authMiddleware.js, checkRoleMiddleware.js и ApiErrorMiddleware.js. Проверка авторизации осуществляется через jwt-токен, в котором закодирована информация о пользователе, включая его роль, что позволяет, используя jwtDecode, получать о ней информацию и определять, можно ли предоставить пользователю доступ к функции. Содержимое файла productRouter.js представлено в листинге 3.9.

const Router = require('express');const router = new Router()

const productController =require('../controllers/productController')

const multer = require('multer');const upload = multer({ dest: 'static/' });

const checkRole = require('../middleware/checkRoleMiddleware')

router.post('/', checkRole('ADMIN'), upload.single('img'), productController.create)router.get('/', productController.getAll)

router.get('/:id', productController.getOne)router.delete('/:id', checkRole('ADMIN'), productController.deleteOneByName)

router.put('/:id', checkRole('ADMIN'), upload.single('img'), productController.update)

Листинг 3.9 – Содержимое файла productRouter.js

При получении запроса он попадает в соответствующий ему роутер и вызывается контроллер для его обработки. В качестве примера приведена функция createProduct из контроллера ProductController, позволяющая создать новый товар в базе данных. Для этого все необходимые данные извлекаются из тела запроса и с помощью метода create() из пакета sequelize создаётся объект. Для загрузки файлов изображений используется multer, который позволяет считывать файлы в теле post запроса. Помимо основного товара, добавляется его описание в модели ProductInfo. Далее в качестве ответа возвращается данный товар в формате json. Функция createProduct приведена в листинге 3.10.

async create(req, res, next) {let { name, price, brandId, typeId, info } = req.body; const fileName = uuid.v4() + '.jpg'; const img = req.file;const uploadPath = path.resolve(\_\_dirname, '..', 'static', fileName); fs.rename(img.path, uploadPath, async (err) => {if (err) { console.error(err);return res.status(500).json({ message: 'Failed to upload file' });} const product = await Product.create({ name, price, brandId, typeId, img: fileName });

if (info) {info = JSON.parse(info)info.forEach(i => ProductInfo.create({title: i.title, description: i.description,productId: product.id}))} return res.json(product);});}

Листинг 3.10 – Код функции createProduct

Приложение работает по протоколу https, для этого был сгенерирован сертификат и подключен пакет https. Вся конфиденциальная информация хранится в файле .env, что повышает безопасность использования приложения. В файле .env хранится пароль базы данных, секретный ключ jwt, email и пароль почты, с которой приложение отправляет сообщения клиентам.

При смене статуса заказа при помощи пакета nodemailer пользователю на почту приходит уведомление. Реализация функции, отвечающей за это, находится в контроллере orderController. Код функции представлен в листинге 3.11.

async changeOrderStatus(req, res) {

try { const { orderId, orderStatusId } = req.body;

const order = await Order.findOne({where: {id: orderId },

include: Product }); let orderProducts = '';

order.products.forEach(product => { orderProducts += product.name + ', ' }); const orderProductsnew = orderProducts.slice(0, -2); await Order.update(

{ orderStatusId: orderStatusId },{ where: { id: {

[Op.in]: [orderId], },},} ); const user = await User.findOne({where: {id: {Op.in]:[order.userId], }},});

const statusMap = {1: 'В обработке',2: 'Подтверждено',

3: 'Выполнено' }; const status = statusMap[orderStatusId]; const transporter = nodemailer.createTransport({ service: 'Gmail',

auth: { user: process.env.USER\_EMAIL,

pass: process.env.PASS\_EMAIL }});

let mailOptions; if (status == 'Выполнено') {

mailOptions = { from: process.env.USER\_EMAIL,

to: user.email, subject: 'Изменение статуса заказа', html: 'Ваш заказ: ' + orderProductsnew + '\nТекущий статус заказа: ' + status + '.\nЗабрать заказ вы можете по адресу ... с 9:00 до 21:00' }; }

else { mailOptions = {

from: process.env.USER\_EMAIL, to: user.email, subject: 'Изменение статуса заказа',

html: 'Ваш заказ: ' + orderProductsnew + '\nТекущий статус заказа: ' + status }; }

transporter.sendMail(mailOptions, (error, info) => {

}); return res.json(order); } catch (e) { return res.json(e); } }

Листинг 3.11 – Код функции changeOrderStatus

Для хранения файлов используется папка static, куда попадают все загружаемые файлы. Для этого используется модуль express.static. Листинг, содержащий весь, код предоставлен в приложении A.

## 3.3 Выводы по разделу

В данном разделе были рассмотрены основные моменты разработки приложения. Было рассмотрено то, как взаимодействуют между собой клиентская и серверная часть по протоколу HTTPS, то, как и где хранится и отображается клиенту информация, и то, как реализована безопасность приложения.

# 4 Тестирование web-приложения

## 4.1 Тестирование web-приложения

Сперва нужно протестировать форму регистрации. Для этого введём уже существующую и невалидную электронную почту. Будем ожидать ошибку. Результат ввода электронной почты, уже существующей в базе данных представлен на рисунке 4.1.

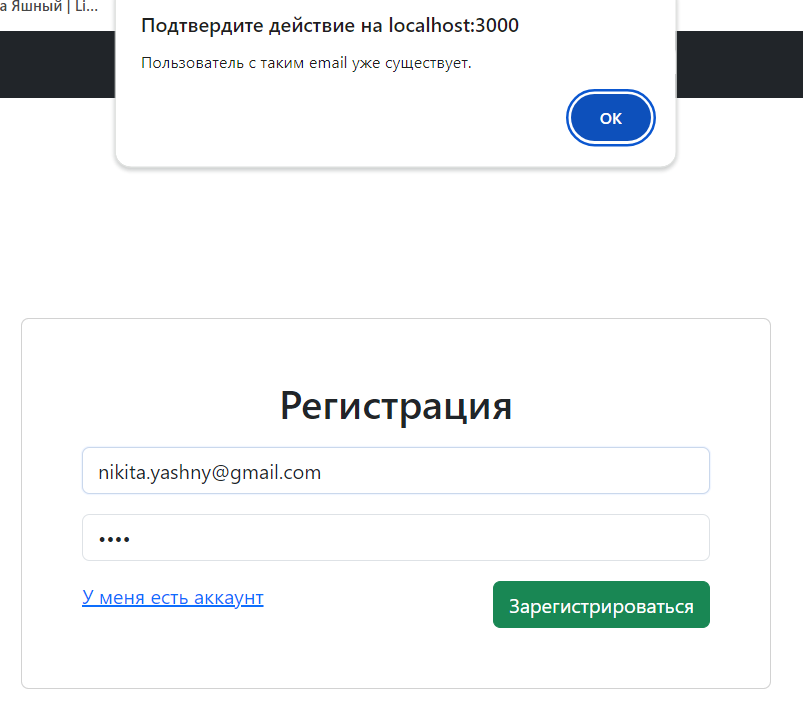


Рисунок 4.1 – Обработка ошибки, связанной с вводом электронной почты, уже существующей в базе данных

На рисунке видно, что ошибка обработана корректно. Результат обработки ввода невалидной почты представлен на рисунке 4.2.

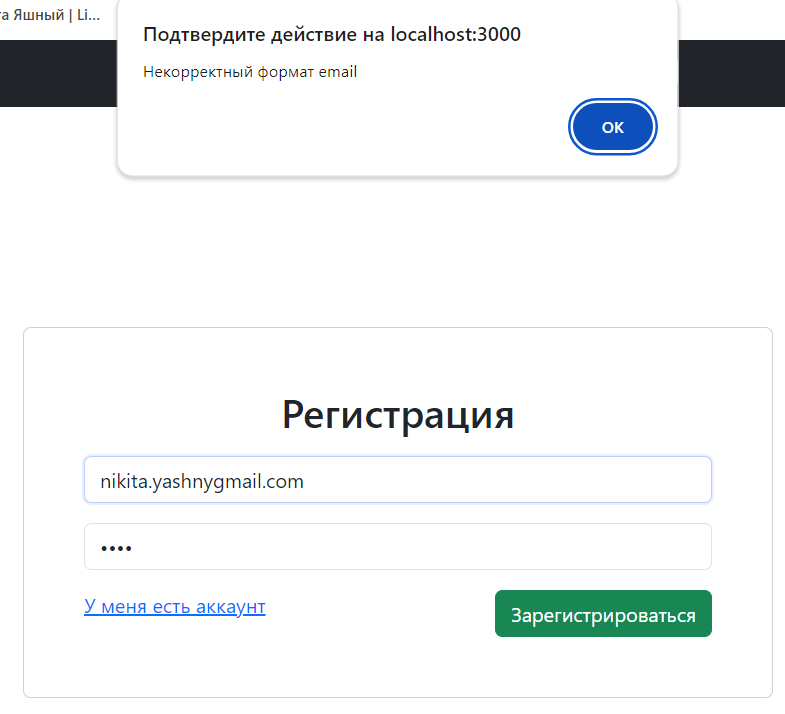


Рисунок 4.2 – Обработка ошибки, связанной с вводом невалидной почты

Далее попробуем добавить товар в корзину будучи неавторизованным пользователем. Приложение должно отреагировать на это и вывести сообщение о том, что мы не авторизованы.

Обработка ошибки, связанной с попыткой добавления товара в корзину будучи неавторизованным, представлена на рисунке 4.3.

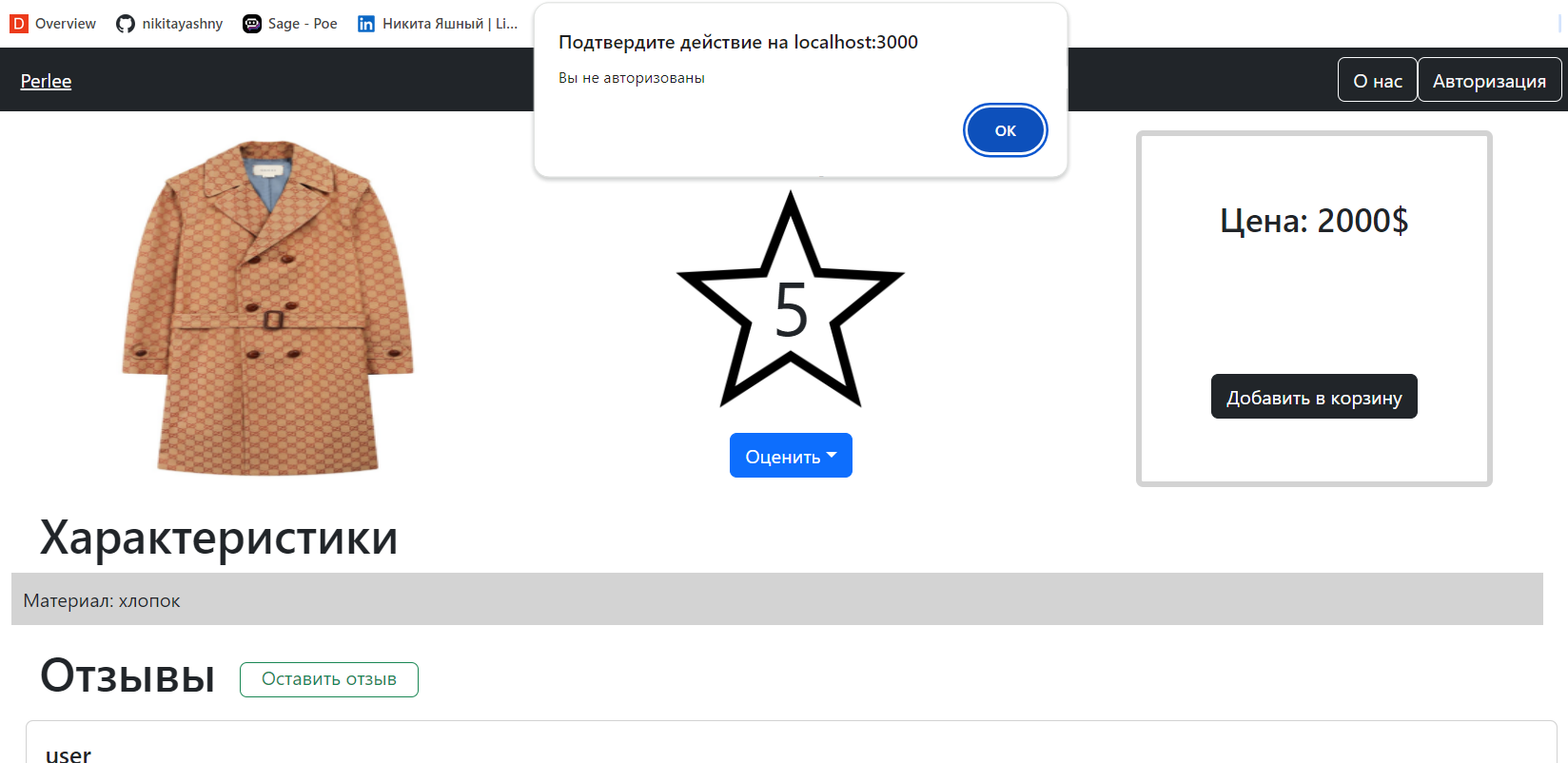


Рисунок 4.3 – Обработка ошибки, связанной с попыткой добавления товара в корзину будучи неавторизованным

Проверка на попытку выполнения действий, требующих авторизации, выполнена. Теперь нужно проверить попытку доступа к административной панели будучи не администратором. Поскольку кнопка для доступа не отображается у обычных пользователей, введём адрес в url. В результате нас просто отправило на главную страницу магазина.

Теперь проверим возможные ошибки со стороны администратора. Попробуем добавить категорию уже существующую в базе данных. Обработка ошибки представлена на рисунке 4.4.

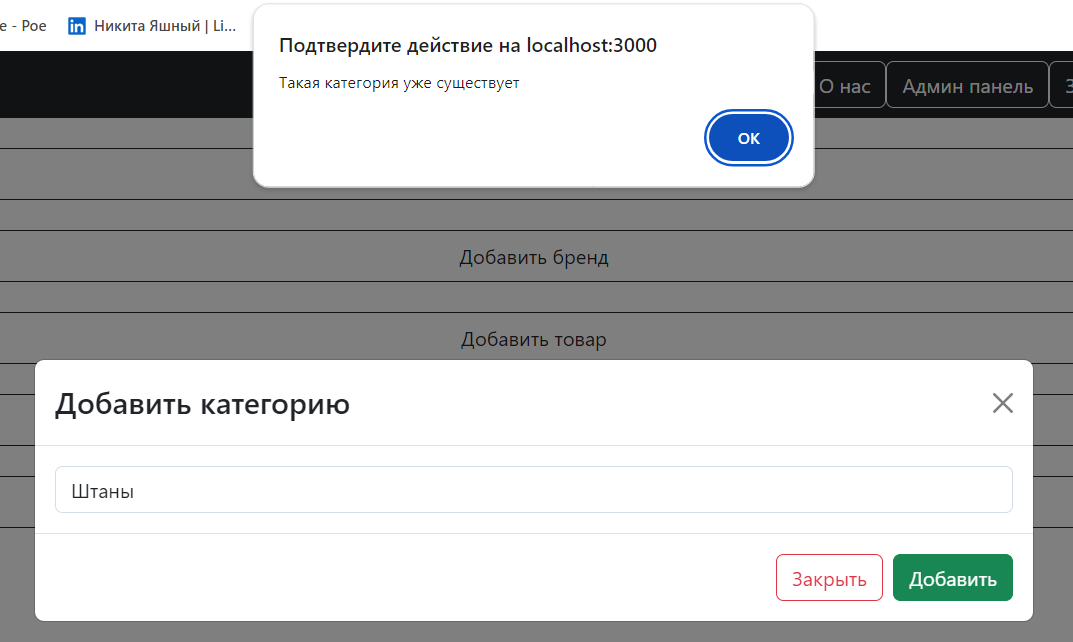


Рисунок 4.4 – Обработка ошибки, связанной с попыткой добавления товара в корзину будучи неавторизованным

Результат пятого теста показал, что эта ошибка, также успешно обработана.

## 4.2 Выводы по разделу

Были опробованы разнообразные сценарии использования приложения. Все ошибки не вызывают сбоев работы приложения, следовательно, приложение работает правильно и готово к работе.

# 5 Руководство пользователя

## 5.1 Руководство гостя

При открытии веб-приложения Perlee вы попадаете на главную страницу магазина, где вы сможете ознакомиться с ассортиментом товаров. Главная страница магазина представлена на рисунке 5.1.

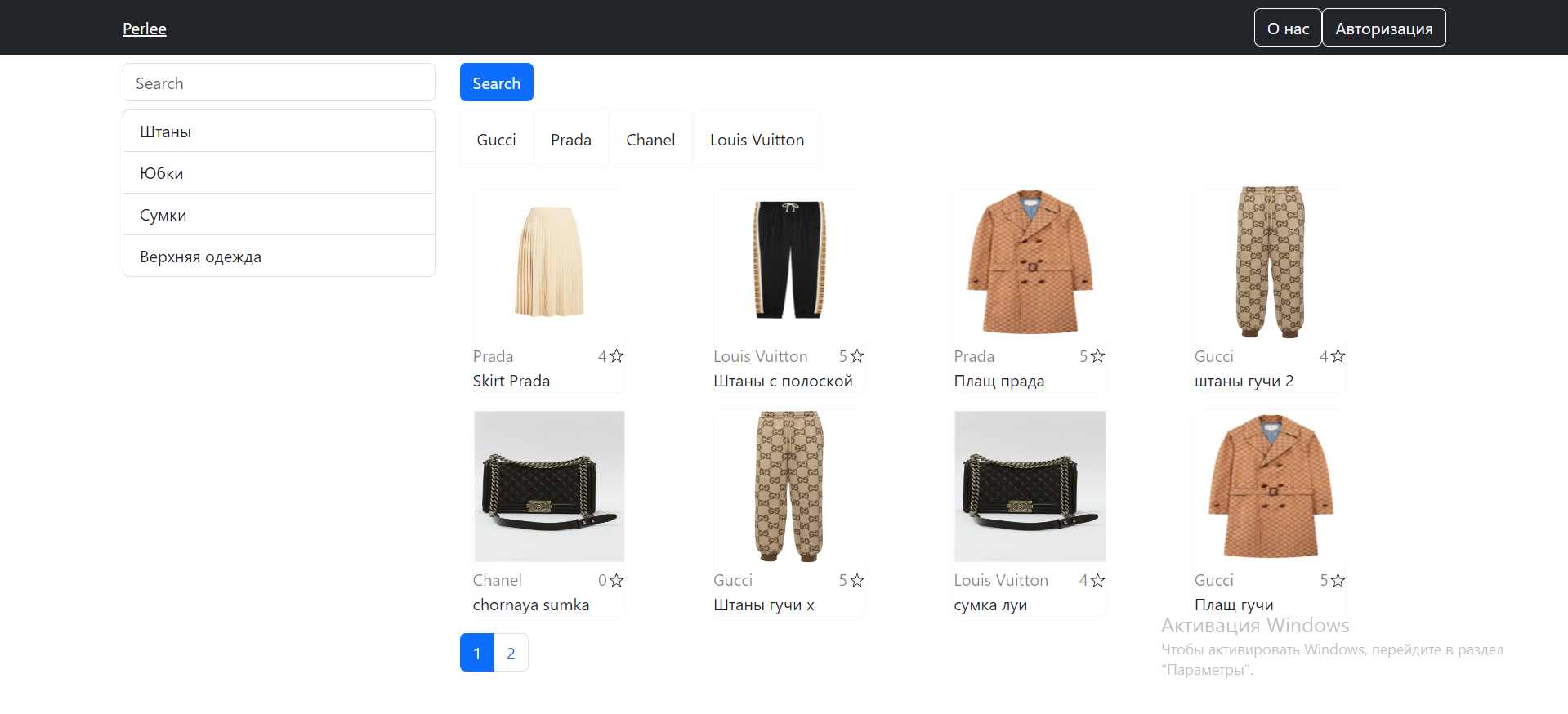


Рисунок 5.1 – Главная страница магазина

Гость может воспользоваться фильтрацией и поиском, всячески комбинируя их, например выбрав категорию, бренд и указав часть названия в поиске применятся все фильтры одновременно. Пример поиска товаров приведён на рисунке 5.2.

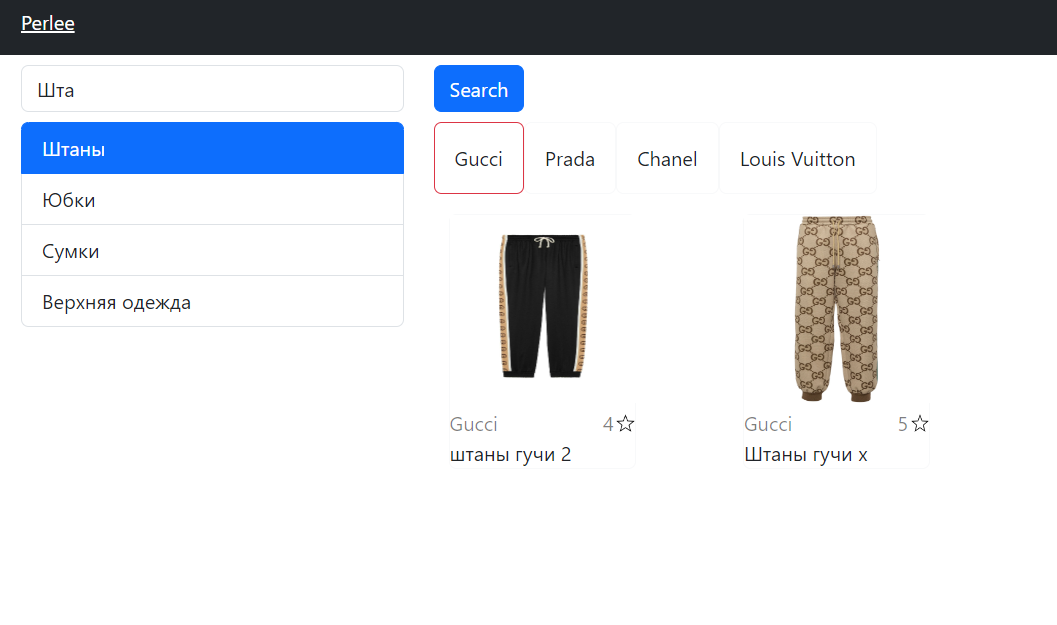


Рисунок 5.2 – Результат фильтрации и поиска

Далее вы можете открыть карточку товара, на которой представлена полная информация о нем, включая изображение, стоимость, оценку, характеристики и отзывы других пользователей. Авторизованным пользователям доступно оценить товар, добавить его в корзину и оставить отзыв. Карточка товара представлена на рисунке 5.3.

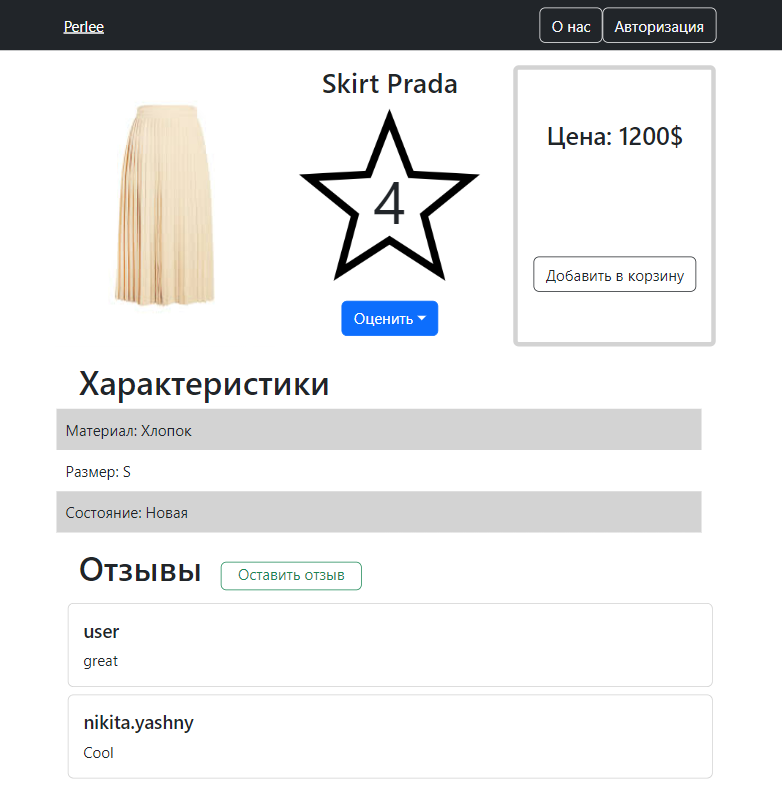


Рисунок 5.3 – Карточка товара

Для доступа ко всем остальным функциям магазина нужно авторизироваться, для этого нужно нажать на кнопку «Авторизация» в панели навигации и заполнить форму, введя свою электронную почту и пароль. Форма авторизации приведена на рисунке 5.4.

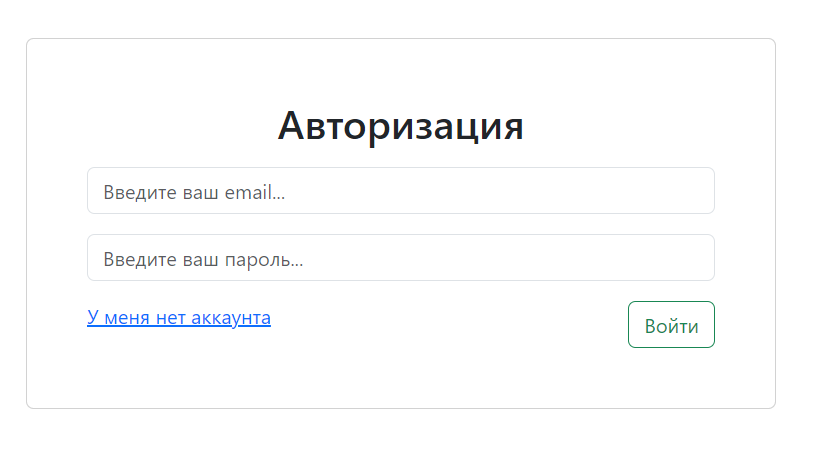


Рисунок 5.4 – Форма авторизации

В случае, если аккаунта ещё нет, нужно нажать на кнопку «У меня нет аккаунта» и заполнить форму.

## 5.2 Руководство пользователя

После авторизации вы сможете оценить товар, оставить отзыв и добавить товар в корзину, для этого нужно нажать на кнопку «Добавить в корзину». Туда вы сможете добавить любое количество товаров, и они не пропадут из каталога. Далее можно просмотреть свою корзину и выбрать то, что желаете оставить, лишние можно удалить, перейдя в их карточку и нажав кнопку «Удалить из корзины». После вы можете оформить заказ, для этого перейдя на страницу «Корзина» из панели навигации, нужно нажать кнопку оформления заказа. Пример оформления заказа приведён на рисунке 5.5.

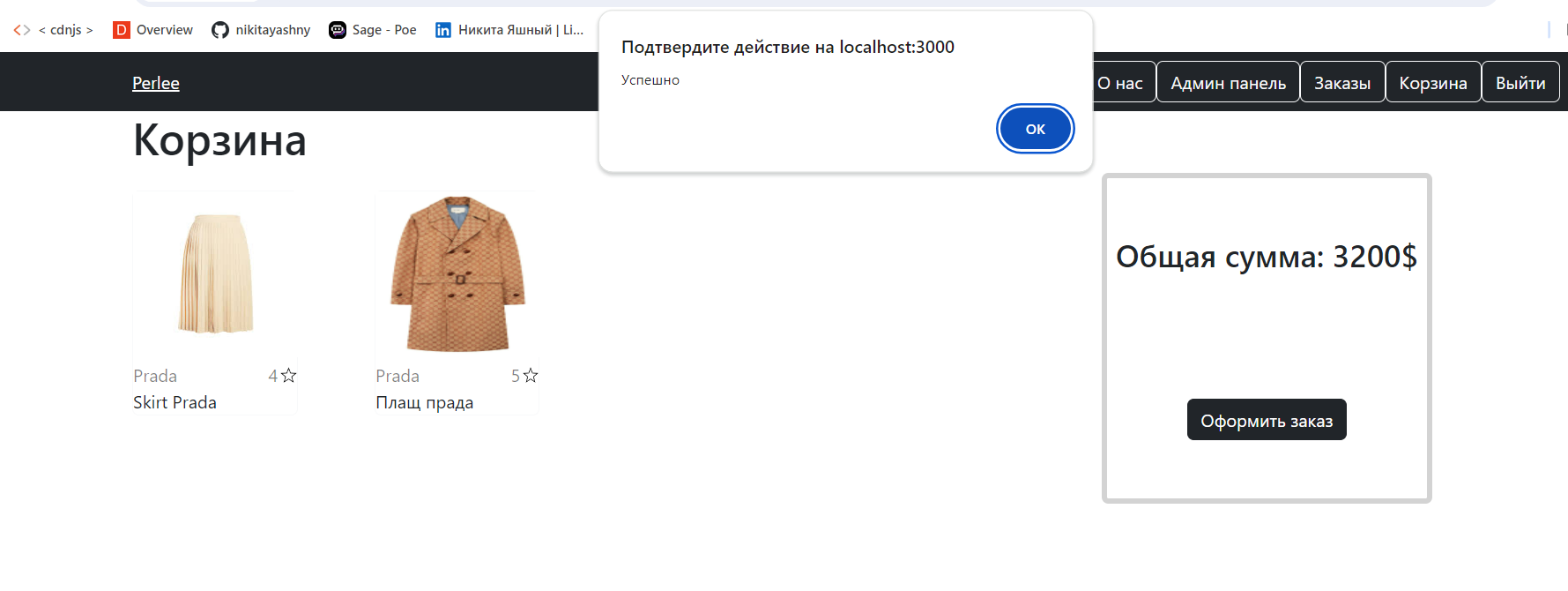


Рисунок 5.5 – Оформление заказа

После этого действия вы сможете просмотреть свои заказы на странице заказов, перейдя на неё через панель навигации. Там вы сможете просмотреть идентификатор вашего заказа, его общую стоимость, статус и дату оформления. После того как администратор изменит статус вашего заказа, вам на электронную почту придёт сообщение об этом, а во вкладке «Заказы» произойдут соответствующие изменения.

Страница заказов представлена на рисунке 5.6.

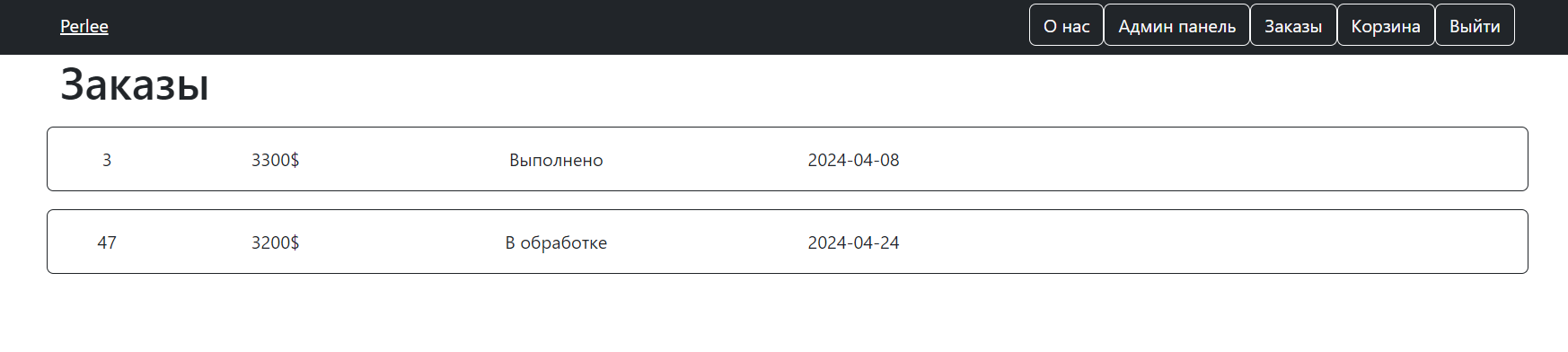


Рисунок 5.6 – Страница с заказами

Нажав на карточку вашего заказа, вы сможете перейти к подробной информации о нем, состоящей из карточек товаров, входящих в заказ, и информации, которая видна на карточке заказа. Если открыть карточку товара, то на ней будет находится информация о том, что данный товар продан. Также там имеется кнопка для отмены заказа. В случае если статус заказа – «Выполнено», кнопка для его отмены пропадает, и отменить его становится невозможным. После отмены заказа товары возвращаются в каталог магазина и становятся доступны к добавлению в корзину всем остальным пользователям.

Пример отмены заказа приведён на рисунке 5.7.

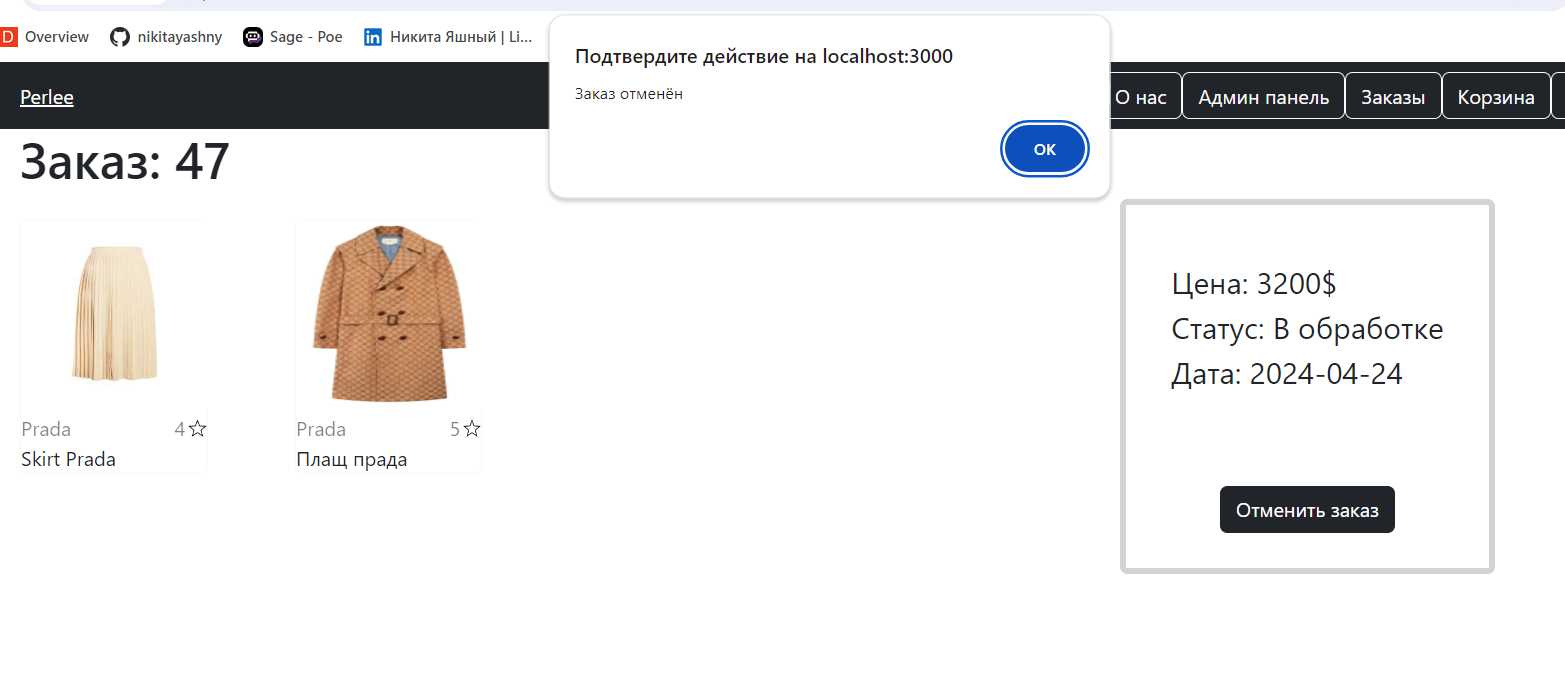


Рисунок 5.7 – Отмена заказа

Будучи администратором, вы также можете проделывать описанные выше действия.

## 5.3 Руководство администратора

Теперь разберём функции доступные администратору магазина. Первой рассмотрим возможность добавления нового товара. Для этого нужно перейти в раздел «Админ панель», находящийся в навигационном меню. Внешний вид административной панели приведён на рисунке 5.8.

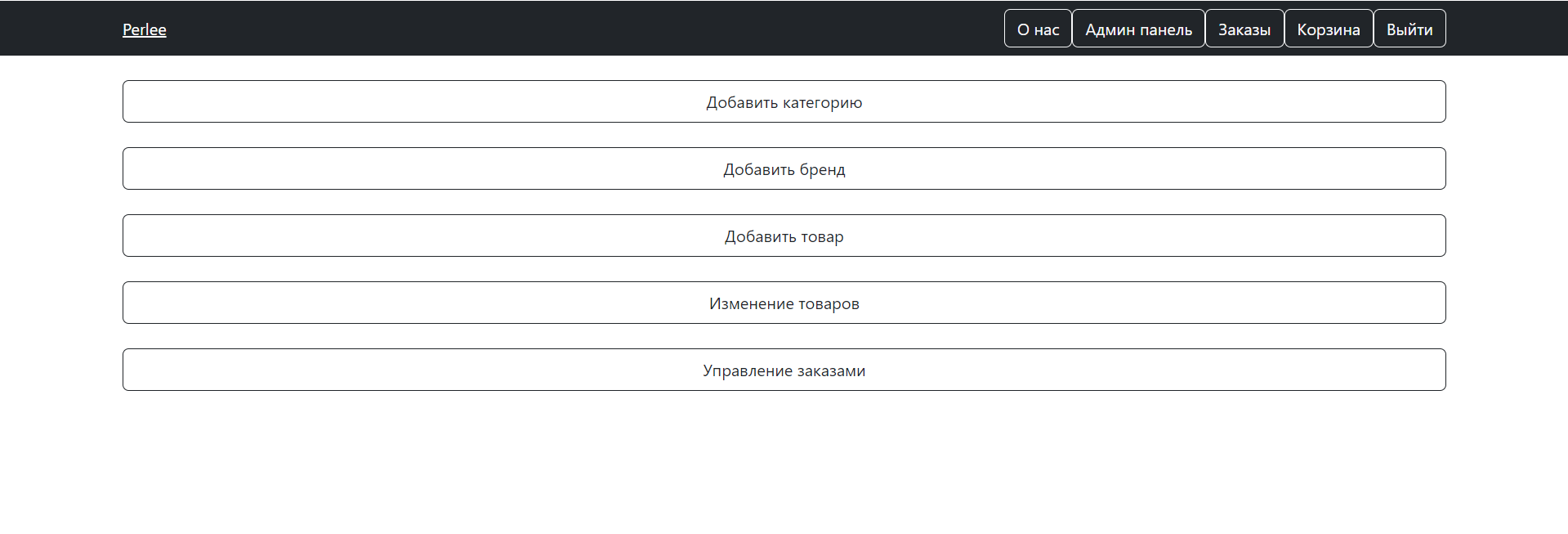


Рисунок 5.8 – Административная панель

Тут есть возможность добавить категорию, добавить бренд, добавить товар, изменить товар и панель управления заказами. Для добавления нового товара нужно нажать соответствующую кнопку, после этого откроется модальное окно, в котором нужно заполнить форму, в неё входит название товара, наименование категории, наименование бренда, изображение товара, также есть возможность добавить характеристики. После добавления товар появится в каталоге и будет доступен для добавления в корзину пользователями. Форма добавления нового товара приведена на рисунке 5.9.

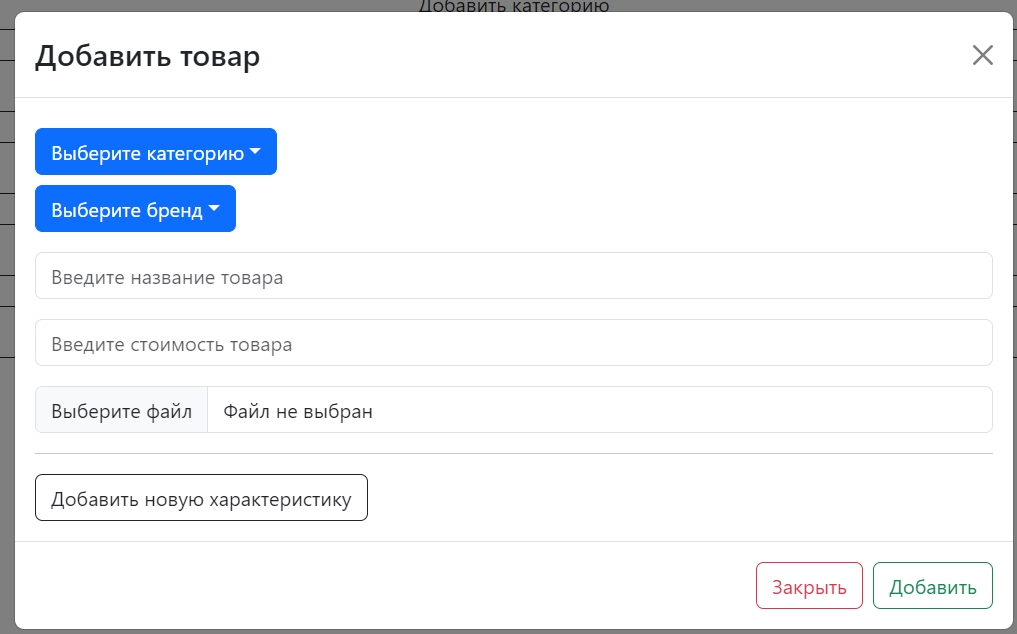


Рисунок 5.9 – Форма добавления нового товара

Помимо манипуляций товарами администратор также может просматривать заказы пользователей и изменять их статус, для этого в панели администратора необходимо выбрать вкладку «Управление заказами», найти нужный заказ и через выпадающее меню изменить его статус. Модальное окно управления заказами приведено на рисунке 5.10.

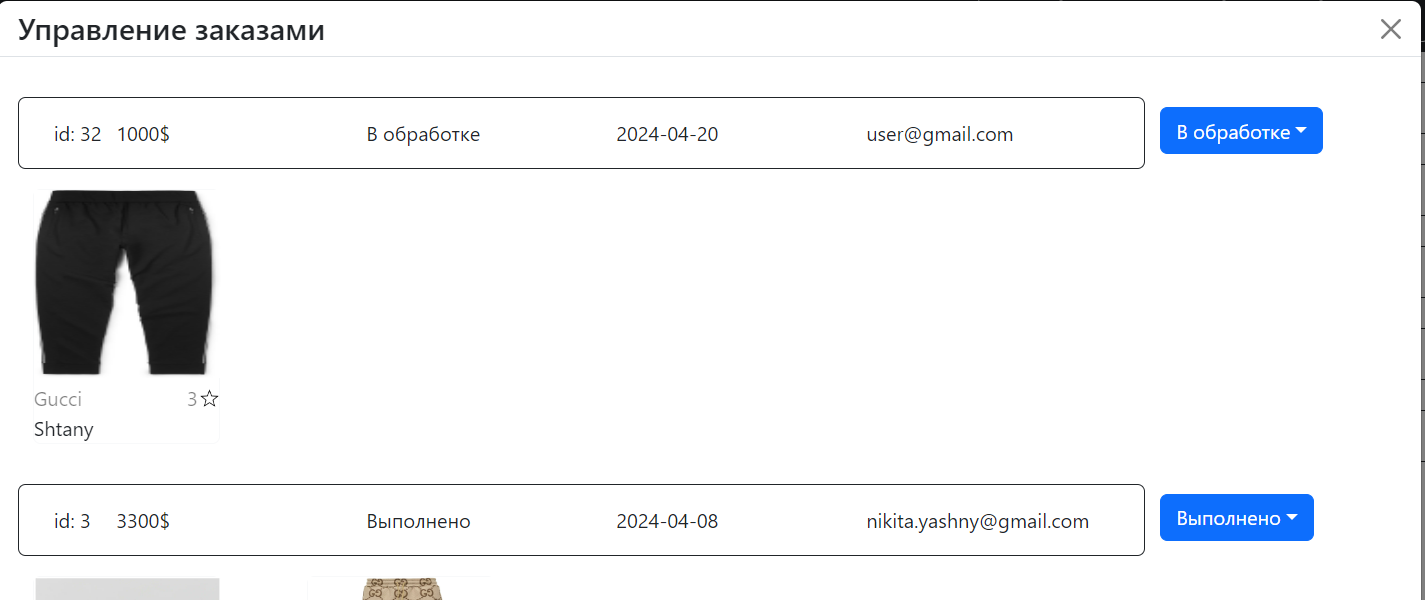


Рисунок 5.10 – Модальное окно управления заказами

После изменения статуса заказа пользователю на электронную почту приходит сообщение об этом, в сообщении указан сам заказ, его содержимое и его статус, если статус заказа – выполнено, то также в сообщении указывается адрес, в котором можно его забрать. Пример сообщения приведён на рисунке 5.11.

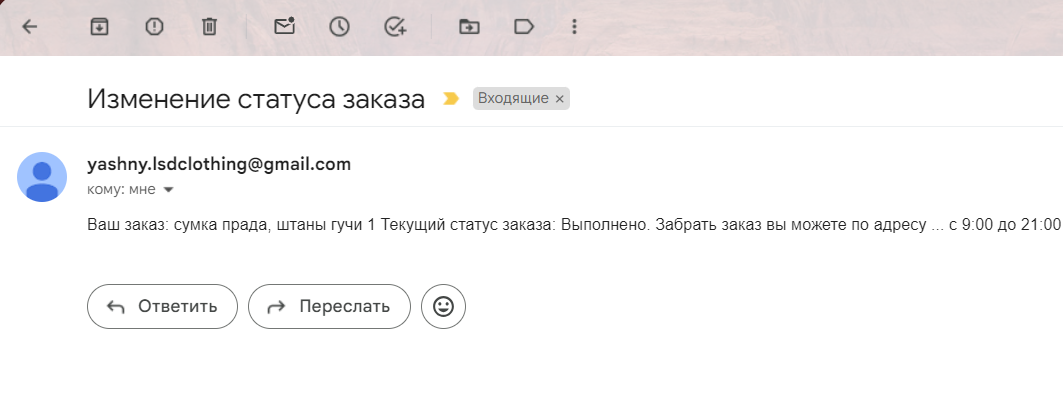


Рисунок 5.11 – Сообщение об изменении статуса заказа

Также у приложения есть мобильная версия, в которой весь интерфейс приводится к виду удобному для мобильных устройств, одним из главных изменений становится то, что панель навигации превращается в меню-бургер. Новая панель навигации приведена на рисунке 5.12.

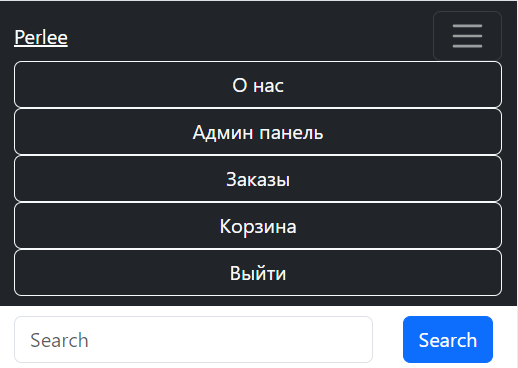


Рисунок 5.12 – Панель навигации на мобильных устройствах

Как мы можем заметить, использование приложения на мобильных устройствах также является доступным и весьма удобным.

## 5.4 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить серверную часть приложения, соединяющую базу данных и клиентскую часть приложения. Для этого необходимо выполнить команду, которая настроит соединение с базой данных и запустит сервер. Сама команда выглядит так: «npm run dev». Серверная часть приложения находиться в директории проекта под названием «Server».

2. Запустить клиентскую часть приложения, которая будет обрабатывать пользовательские запросы, взаимодействовать с сервером и предоставлять клиенту пользовательский интерфейс. Для этого необходимо выполнить команду для сборки и запуска React приложения. Сама команда выглядит так: «npm start». Клиентская часть приложения находится в директории проекта под названием «Client».

Необходимые https сертификат и приватный ключ находятся в главной директории проекта.

После выполнения этих шагов приложение будет полностью готово к работе. Сайт станет доступен в браузере по url https://localhost:3000.

## 5.5 Выводы по разделу

В данном разделе было рассмотрено, как пользоваться приложением интернет-магазина «Perlee». Рассмотрели, что гостю доступны просмотр каталога товаров, осуществление поиска и фильтрации, также возможность просмотра оценок товаров и отзывов. Авторизированному пользователю становится доступным добавление товара в корзину и оформления заказов, также у него есть возможности оставить отзыв о товаре и поставить ему оценку. Администратору доступны функции добавления новых категорий товара, новых брендов и новых позиций. Также он может изменять информацию о товарах и ставить новый статус заказам пользователей, отправляя им сообщение на электронную почту. Письмо, приходящее на электронную почту, также было продемонстрировано.

Также был рассмотрен процесс запуска приложения со всеми необходимыми шагами.

После ознакомления с данным руководством пользователю должно стать понятно, как взаимодействовать с данным программным продуктом.

# Заключение

Приложение «Perlee» – полностью функциональное средство для продажи товаров. Оно позволяет пользователям просматривать каталог товаров, пользуясь удобной фильтрацией и поиском, добавлять товары в корзину, совершать заказы, также можно просмотреть и оставить отзывы и поставить оценку товару. Администраторы могут добавлять новые позиции товаров и изменять старые, им доступна функция смены статуса заказа и отправки сообщения пользователю по электронной почте. Также они могут добавлять новые категории товаров и их бренды.

За серверную часть отвечала платформа NodeJS, а именно фреймворк Express, который позволил реализовать быструю асинхронную серверную часть. Структура приложения состоит из роутера и контроллера, а также модели базы данных позволяющей получить над ней абстракцию.

Использование протокола WebSocket позволило реализовать рассылку сообщений от сервера клиентам, благодаря непрерывному соединению между клиентом и сервером.

Протокол HTTPS обеспечивает конфиденциальность данных, шифруя их с помощью SSL. Также использование HTTPS повышает доверие пользователей к приложению и может положительно сказаться на SEO-оптимизации.

Клиентская часть была написана при помощи библиотеки React для языка JavaScript. Благодаря модулю axios мы смогли отправлять запросы на сервер, а с помощью mobx обеспечили асинхронную навигацию по страницам приложения. Также mobx помогает управлять состоянием приложения, предоставляя простую модель наблюдаемых значений и возможность автоматического отслеживания зависимостей между ними. React позволил легко построить структуру клиентской части веб-приложения.

Система управления базами данных PostgreSQL позволила создать структуру для эффективного хранения взаимосвязанных данных. В базе данных хранятся все необходимые для работы приложения данные о товарах и пользователях. Работа с базой данных осуществляется с помощью ORM Sequelize, что значительно доступ к данным.

В результате тестирования мы пришли к выводу, что приложение функционирует правильно и все выполненные тесты оказались успешными. В ходе тестирования ошибок не было найдено.

Данное 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: 20.03.2024.
2. Документация на фреймворк Expess [Электронный ресурс] / Режим доступа: https://expressjs.com – Дата доступа: 03.04.2024.
3. Postgresqltutorial.com [Электронный ресурс] / Режим доступа: https://www.postgresqltutorial.com/ – Дата доступа: 20.03.2024.
4. Sequelize.org [Электронный ресурс] / Режим доступа: https://sequelize.org/docs/v6/getting-started – Дата доступа: 02.04.2024.

# Приложение А

const sequelize = require('../db')

const {DataTypes} = require('sequelize')

const User = sequelize.define('user', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

email: {type: DataTypes.STRING, unique: true},

password: {type: DataTypes.STRING},

role: {type: DataTypes.STRING, defaultValue: "USER"}

})

const Basket = sequelize.define('basket', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true}

})

const BasketProduct = sequelize.define('basket\_product', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true}

})

const Product = sequelize.define('product', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

name: {type: DataTypes.STRING, unique: true, allowNull: false},

price: {type: DataTypes.INTEGER, allowNull: false},

rating: {type: DataTypes.INTEGER, defaultValue: 0},

img: {type: DataTypes.STRING, allowNull: false},

isSold: {type: DataTypes.BOOLEAN, defaultValue: false}

})

const Type = sequelize.define('type', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

name: {type: DataTypes.STRING, unique: true, allowNull: false}

})

const Brand = sequelize.define('brand', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

name: {type: DataTypes.STRING, unique: true, allowNull: false}

})

const Rating = sequelize.define('rating', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

rate: {type: DataTypes.INTEGER, allowNull: false}

})

const ProductInfo = sequelize.define('product\_info', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

title: {type: DataTypes.STRING, allowNull: false},

description: {type: DataTypes.STRING, allowNull: false}

})

const TypeBrand = sequelize.define('type\_brand', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true}

})

const Order = sequelize.define('order', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

total\_cost: {type: DataTypes.INTEGER, defaultValue: 0}

})

const OrderProduct = sequelize.define('order\_product', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true}

})

const OrderStatus = sequelize.define('order\_status', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

name: {type: DataTypes.STRING, unique: true, allowNull: false}

})

const Review = sequelize.define('review', {

id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true},

text: {type: DataTypes.STRING, allowNull: false}

})

User.hasOne(Basket)

Basket.belongsTo(User)

User.hasOne(Rating)

Rating.belongsTo(User)

Basket.hasMany(BasketProduct)

BasketProduct.belongsTo(Basket)

Type.hasMany(Product)

Product.belongsTo(Type)

Brand.hasMany(Product)

Product.belongsTo(Brand)

Product.hasMany(Rating)

Rating.belongsTo(Product)

Product.hasMany(BasketProduct)

BasketProduct.belongsTo(Product)

Product.hasMany(ProductInfo, {as: 'info'})

ProductInfo.belongsTo(Product)

Type.belongsToMany(Brand, {through: TypeBrand})

Brand.belongsToMany(Type, {through: TypeBrand})

User.hasMany(Order)

Order.belongsTo(User)

Order.hasMany(OrderProduct)

OrderProduct.belongsTo(Order)

Product.hasMany(OrderProduct)

OrderProduct.belongsTo(Product)

OrderStatus.hasMany(Order)

Order.belongsTo(OrderStatus)

Order.belongsToMany(Product, { through: OrderProduct });

Product.belongsToMany(Order, { through: OrderProduct });

User.hasMany(Review)

Review.belongsTo(User)

Product.hasMany(Review)

Review.belongsTo(Product)

module.exports = {

User,

Basket,

BasketProduct,

Product,

Type,

Brand,

Rating,

TypeBrand,

ProductInfo,

Order,

OrderProduct,

OrderStatus,

Review

}}

Листинг 1 – Модель базы данных

const {Basket, Product, BasketProduct, Order, OrderProduct, User, OrderStatus} = require('../models/models')

const ApiError = require('../error/ApiError')

const { Op } = require('sequelize');

const nodemailer = require('nodemailer');

const { emit } = require('nodemon');

require('dotenv').config();

class OrderController {

async getAllOrders(req, res) {

try {

const orders = await Order.findAll({

include: [

{

model: Product,

through: {

model: OrderProduct,

},

},

{

model: User,

},

{

model: OrderStatus,

},

],

});

const orderList = [];

for (const order of orders) {

const products = order.products.map((product) => ({

id: product.id,

name: product.name,

price: product.price,

rating: product.rating,

img: product.img,

isSold: product.isSold,

brandId: product.brandId

}));

const orderData = {

id: order.id,

total\_cost: order.total\_cost,

userId: order.user.id,

date: order.createdAt,

status: order.order\_status.name,

products: products,

user: order.user

};

orderList.push(orderData);

}

return res.json(orderList);

} catch(e) {

return res.json(e)

}

}

async getUserOrders(req, res) {

try {

const { userId } = req.params

const orders = await Order.findAll({

include: [

{

model: Product,

through: {

model: OrderProduct,

},

},

{

model: User,

},

{

model: OrderStatus,

},

],

where: {userId}

});

const orderList = [];

for (const order of orders) {

const products = order.products.map((product) => ({

id: product.id,

name: product.name,

price: product.price,

rating: product.rating,

img: product.img,

isSold: product.isSold,

brandId: product.brandId

}));

const orderData = {

id: order.id,

total\_cost: order.total\_cost,

userId: order.user.id,

date: order.createdAt,

status: order.order\_status.name,

products: products,

};

orderList.push(orderData);

}

return res.json(orderList);

} catch(e) {

return res.json(e)

}

}

async getOneUserOrder(req, res) {

try {

const { userId, orderId } = req.params

const orders = await Order.findAll({

include: [

{

model: Product,

through: {

model: OrderProduct,

},

},

{

model: User,

},

{

model: OrderStatus,

},

],

where: {id: orderId, userId}

});

const orderList = [];

for (const order of orders) {

const products = order.products.map((product) => ({

id: product.id,

name: product.name,

price: product.price,

rating: product.rating,

img: product.img,

isSold: product.isSold,

brandId: product.brandId

}));

const orderData = {

id: order.id,

total\_cost: order.total\_cost,

userId: order.user.id,

date: order.createdAt,

status: order.order\_status.name,

products: products,

};

orderList.push(orderData);

}

return res.json(orderList);

} catch(e) {

return res.json(e)

}

}

async makeOrder(req, res) {

try {

const {userId} = req.params

const basket = await Basket.findOne({

where: {

userId: {

[Op.eq]: userId

}

}

})

const basketProducts = await BasketProduct.findAll({

where: {

basketId: {

[Op.eq]: basket.id

}

}

})

const productIds = basketProducts.map(basketProduct => basketProduct.productId);

const products = await Product.findAll({

where: {

id: {

[Op.in]: productIds

}

}

})

if (products.length == 0)

return res.json("Корзина пуста")

const total\_cost = products.reduce((sum, product) => sum + product.price, 0)

const order = await Order.create({ userId: userId, orderStatusId: 1, total\_cost: total\_cost });

let hasIsSold = false;

const orderProducts = products.map((product) => {

if (product.isSold == true)

hasIsSold = true;

return {

orderId: order.id,

productId: product.id,

};

});

if (hasIsSold)

return res.json("Товар продан")

await OrderProduct.bulkCreate(orderProducts);

await Product.update(

{ isSold: true },

{

where: {

id: {

[Op.in]: productIds,

},

},

}

);

await BasketProduct.destroy({

where: {

productId: {

[Op.in]: productIds,

},

},

});

return res.json(orderProducts);

} catch(e) {

return res.json(e)

}

}

async deleteOrder(req, res) {

try {

const { orderId } = req.params;

const order = await Order.findOne({

where: {

id: {

[Op.in]: [orderId],

},

},

});

if (order.orderStatusId != 1) {

return res.json({ message: "Отмена заказа невозможна" });

}

const orderProducts = await OrderProduct.findAll({

where: {

orderId: {

[Op.in]: [orderId],

},

},

});

const productIds = orderProducts.map((orderProduct) => orderProduct.productId);

await OrderProduct.destroy({

where: {

orderId: {

[Op.in]: [orderId],

},

},

});

await order.destroy();

await Product.update(

{ isSold: false },

{

where: {

id: {

[Op.in]: productIds,

},

},

}

);

return res.json({ order: order });

} catch (e) {

return res.json(e);

}

}

async changeOrderStatus(req, res) {

try {

const { orderId, orderStatusId } = req.body;

const order = await Order.findOne({

where: {

id: orderId

},

include: Product

});

let orderProducts = '';

order.products.forEach(product => {

orderProducts += product.name + ', '

});

const orderProductsnew = orderProducts.slice(0, -2);

await Order.update(

{ orderStatusId: orderStatusId },

{

where: {

id: {

[Op.in]: [orderId],

},

},

}

);

const user = await User.findOne({

where: {

id: {

[Op.in]: [order.userId],

}

},

});

const statusMap = {

1: 'В обработке',

2: 'Подтверждено',

3: 'Выполнено'

};

const status = statusMap[orderStatusId];

const transporter = nodemailer.createTransport({

service: 'Gmail',

auth: {

user: process.env.USER\_EMAIL,

pass: process.env.PASS\_EMAIL

}

});

let mailOptions;

if (status == 'Выполнено') {

mailOptions = {

from: process.env.USER\_EMAIL,

to: user.email,

subject: 'Изменение статуса заказа',

html: 'Ваш заказ: ' + orderProductsnew + '\nТекущий статус заказа: ' + status + '.\nЗабрать заказ вы можете по адресу ... с 9:00 до 21:00'

};

}

else {

mailOptions = {

from: process.env.USER\_EMAIL,

to: user.email,

subject: 'Изменение статуса заказа',

html: 'Ваш заказ: ' + orderProductsnew + '\nТекущий статус заказа: ' + status

};

}

transporter.sendMail(mailOptions, (error, info) => {

});

return res.json(order);

} catch (e) {

return res.json(e);

}

}

}

module.exports = new OrderController()

Листинг 2 – Контроллер OrderController