МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация web-приложения магазина одежды»

Выполнил студент Яшный Никита Сергеевич

(Ф.И.О.)

Руководитель работы ст. преп. Дубовик Марина Владимировна

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

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

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

Курсовая работа защищена с оценкой

Минск 2024

**Содержание**

[Введение 4](#_Toc164868225)

[1 Постановка задачи 5](#_Toc164868226)

[1.1 Аналитический обзор аналогов 5](#_Toc164868227)

[1.1.1 Zara 5](#_Toc164868228)

[1.1.2 Bershka 6](#_Toc164868229)

[1.1.3 Pull & Bear 7](#_Toc164868230)

[1.1.4 Lamoda 8](#_Toc164868231)

[1.2 Разработка функциональных требований определение вариантов использования 8](#_Toc164868232)

[1.3 Выводы по разделу 10](#_Toc164868233)

[2 Проектирование web-приложения 11](#_Toc164868234)

[2.1 Архитектура приложения 11](#_Toc164868235)

[2.2 Общая структура приложения 11](#_Toc164868236)

[2.3 Структура базы данных 12](#_Toc164868237)

[2.4 Вывод 14](#_Toc164868238)

[3 Разработка web-приложения 15](#_Toc164868239)

[3.1 Разработка клиентской части web-приложения 15](#_Toc164868240)

[3.2 Разработка серверной части web-приложения 19](#_Toc164868241)

[3.3 Вывод 23](#_Toc164868242)

[4 Тестирование web-приложения 24](#_Toc164868243)

[5 Руководство пользователя 26](#_Toc164868244)

[5.1 Руководство пользователя 26](#_Toc164868245)

[5.2 Установка приложения 31](#_Toc164868246)

[5.3 Вывод 31](#_Toc164868247)

[Заключение 32](#_Toc164868248)

[Список используемых источников 33](#_Toc164868249)

# 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 позволяет решать многочисленные проблемы пользователей, включая ознакомление с ассортиментом бренда и подробной информацией по каждой позиции, актуальными предложениями, наличием определенных товаров в магазинах и дает возможность сравнивать цены.

Возможность создания личного кабинета позволяет пользователю добавлять понравившиеся модели в избранное, сравнивать их, оформлять заказ, указывая адрес доставки, получать актуальную информацию по электронной почте и иметь доступ к истории заказов.

Данное программное средство незаменимо для администратора, который имеет удаленный доступ к базе данных, может проверять и изменять ее наполнение, выполнять поиск и сортировку товаров, отслеживать заказы и осуществлять помощь покупателям, имея доступ к их личному кабинету.

Главная цель данного курсового проектирования – это разработка программного средства, которое реализует все вышеперечисленные функции и решает поставленные задачи.

Для разработки серверной части данного веб-приложения был выбран фреймворк Express, а для разработки клиентской части был выбран React. Для работы с БД будет использоваться ORM Sequelize для взаимодействия с PostrgreSQL.

В пояснительной записке вы сможете найти краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

# 1 Постановка задачи

## 1.1 Аналитический обзор аналогов

Были проанализированы цель 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.

# Zara

Одним из самых популярных альтернативных решений является интернет-ресурс «Zara», выступающая известным брендом молодёжной одежды, собравшим 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.

Из выявленных недостатков данного ресурса необходимо выделить отсутствие возможности доставки по Беларуси. А также отсутствие адаптивного дизайна под все виды устройств.

Интерфейс интернет-ресурса представлен на рисунке 1.1.

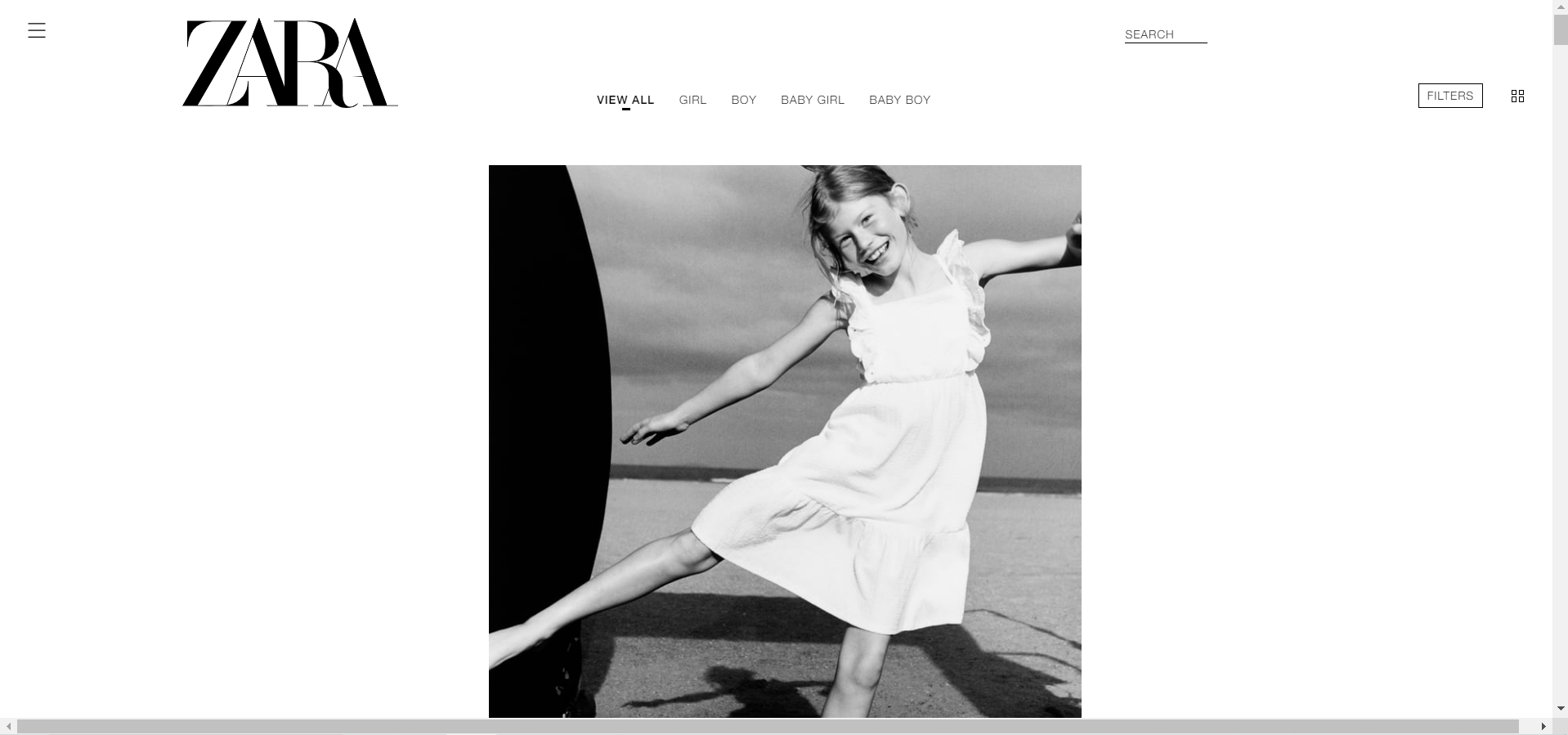


Рисунок 1.1 – Интернет-ресурс «Zara»

Из выявленных недостатков данного ресурса можно выделить 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.

Интерфейс интернет-ресурса представлен на рисунке 1.2.

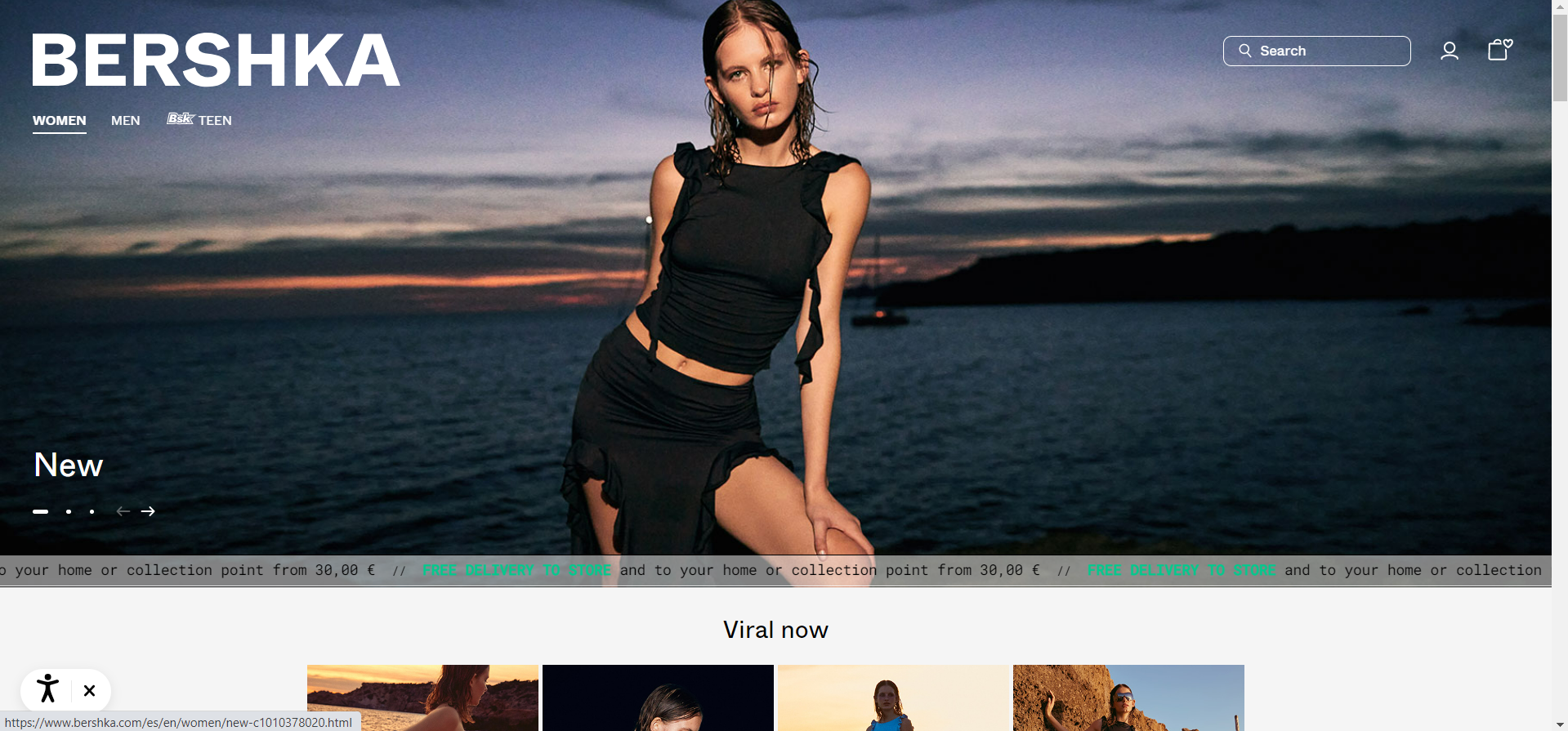


Рисунок 1.2 – Интернет-ресурс «Bershka»

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

Интерфейс интернет-ресурса представлен на рисунке 1.3.

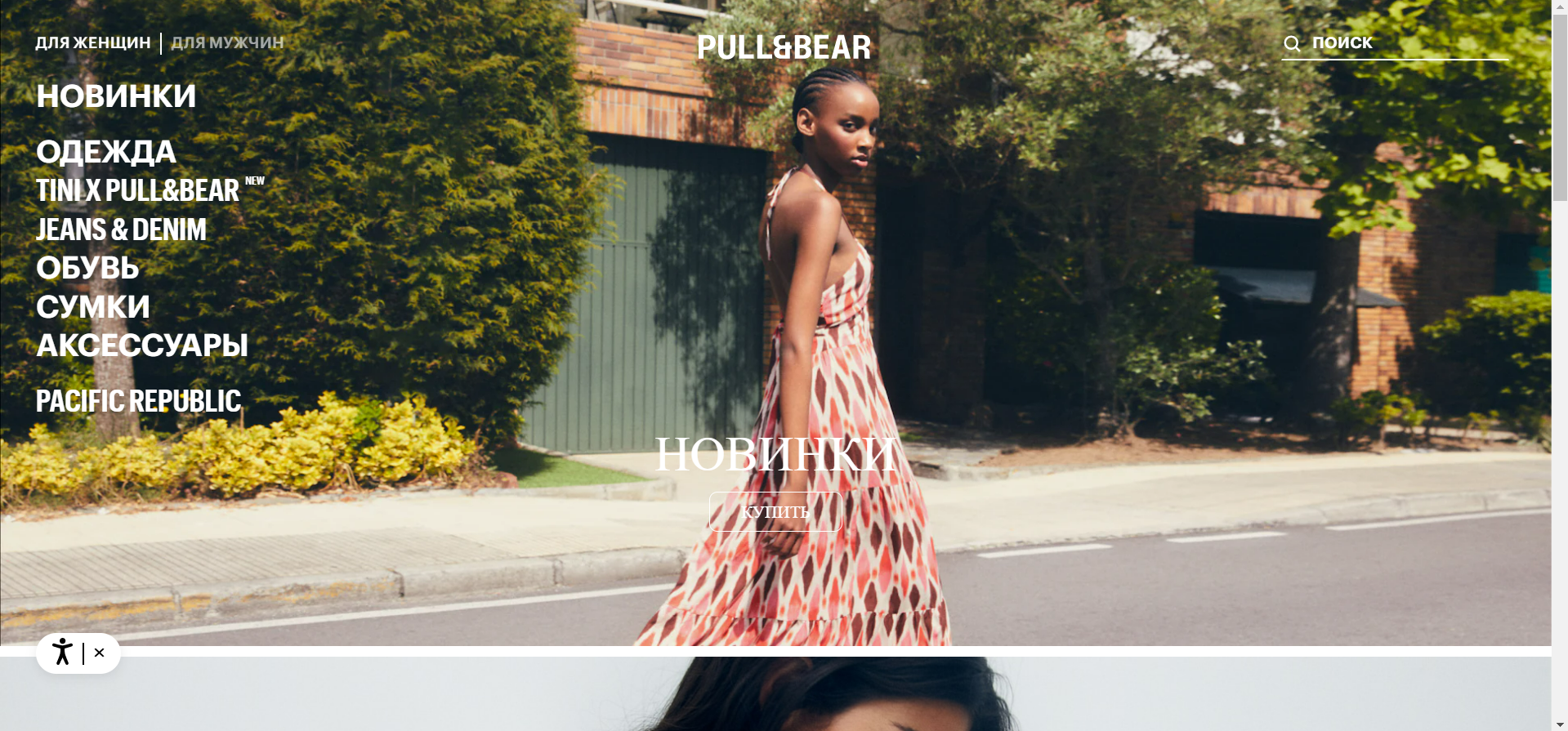


Рисунок 1.3 – Интернет-ресурс «Pull&Bear»

Однако, недостатком данной альтернативы являются отсутствие качественной оптимизации, из-за чего даже при наличии высокоскоростного интернета многие разделы долго загружаются. Также, как и у приведённых ранее примеров альтернатив, на сайте отсутствует возможность резервирования и просмотра популярных позиций.

# 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 не только по классическим категориям вроде пола и размера, но и по брендам и популярности, что является крайне удобным при поиске необходимого товара.

Интерфейс интернет-ресурса представлен на рисунке 1.4.

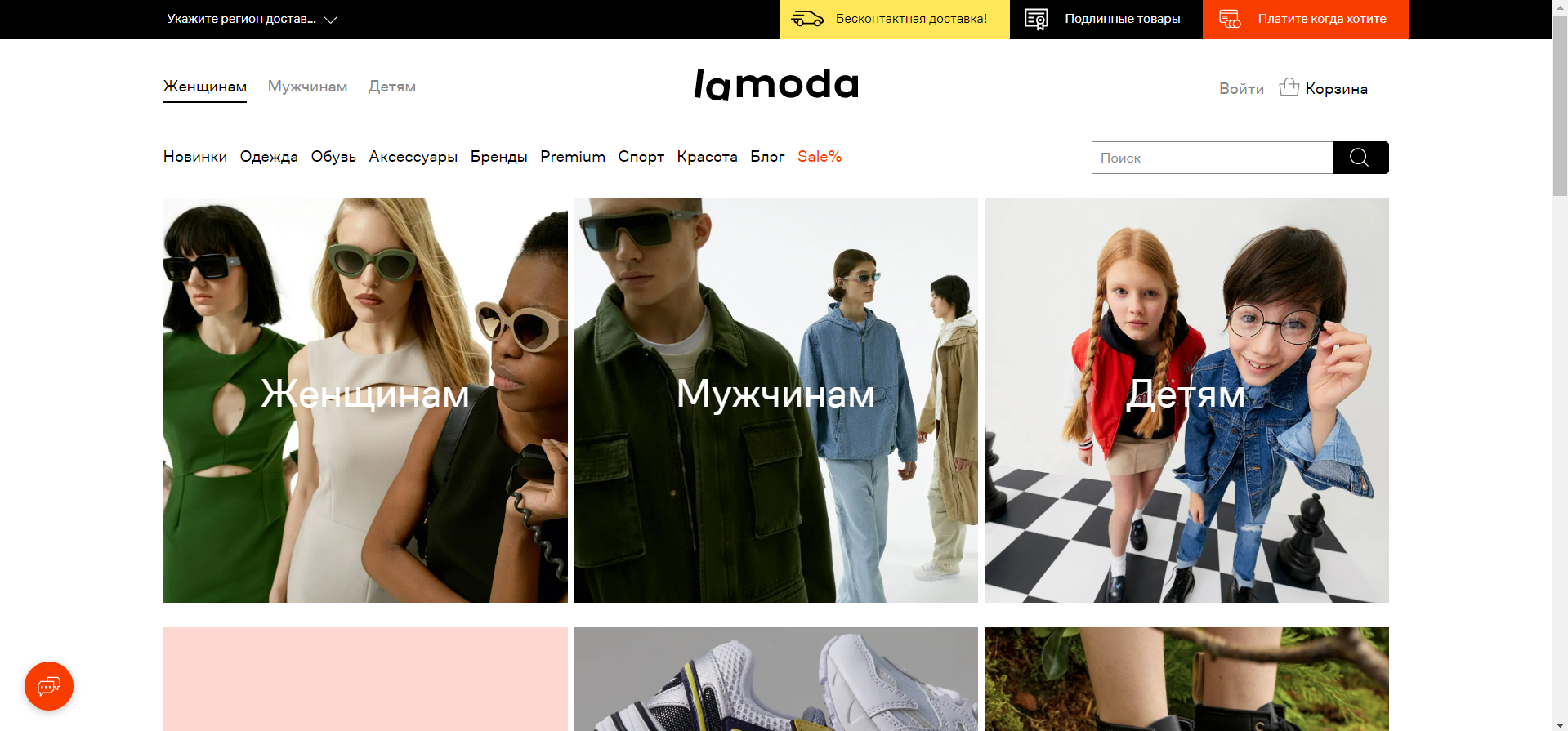


Рисунок 1.4 – Интернет-ресурс «Lamoda»

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 и пользователя;
* 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.

Роли пользователя – это набор прав, которые пользователь может получить в системе. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* GUEST;
* USER;
* ADMIN;

Роль GUEST позволяет только просмотреть отзывы и каталог, а также создать новый аккаунт. После создания аккаунта пользователь получает роль USER.

Роль USER дает возможность просматривать каталог, оформлять заказы, добавлять товары в корзину.

Роль ADMIN также может просматривать информацию о товарах, помимо этого у него есть доступ к статистике продаж. Также он может добавлять новые позиции товара, изменять имеющиеся и изменять статусы заказов.

Приложение обладает тремя ролями, на которые разложены все основные функции для корректного использования.

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

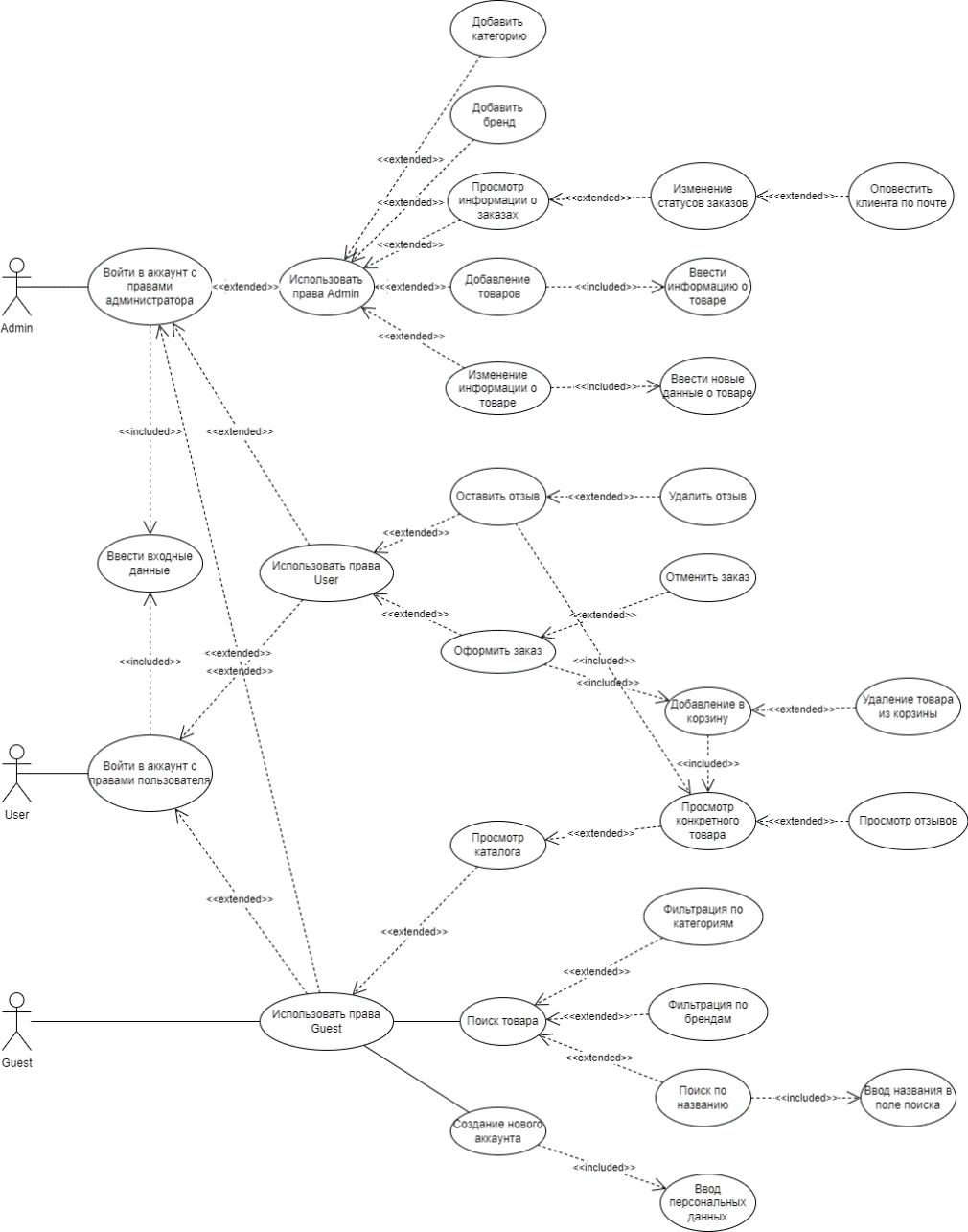


Рисунок 1.5 – Диаграмма вариантов использования

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. Этот обзор позволил определить основные характеристики и функциональные возможности, которые необходимо предусмотреть в разрабатываемой системе. Также были определены функциональные требования приложения, а также роли пользователей и варианты использования системы в зависимости от этих 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 база данных PostgreSQL.

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 HTTP-запросы на сервер для получения 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. Он облегчает разработку маршрутов, запросов и ответов, а также управление сеансами и промежуточным программным обеспечением. Сервер обрабатывает запросы от клиента, включая маршрутизацию запросов к соответствующим обработчикам.

Выбор СУБД остановился на PostgreSQL, она является реляционной базой данных с открытым исходным кодом, предоставляет надежное хранение и управление структурированными данными. Node.js сервер использует ORM Sequelize для CRUD операций.

Взаимодействие между клиентской и серверной частью приложения осуществляется следующим образом: клиент отправляет HTTP-запросы на сервер, используя AJAX. Сервер обрабатывает запросы, выполняет требуемые операции в базе данных и формирует HTTP-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. 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 нашего 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 работы приложения.

Диаграмма развёртывания приложения представлена на рисунке 2.1.

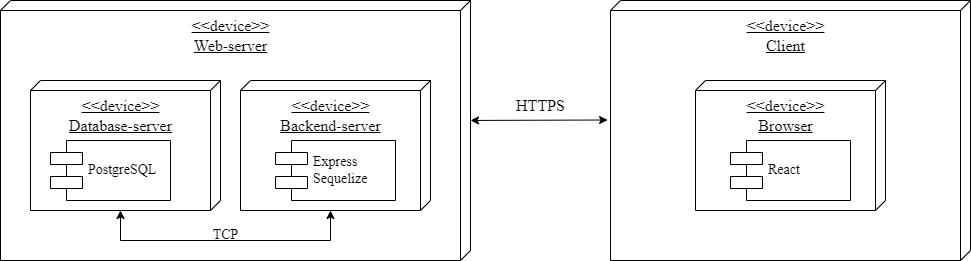


Рисунок 2.1 – Диаграмма развёртывания приложения

На диаграмме представлены все вышеописанные связи основных модулей приложения.

## 2.3 Структура базы данных

Диаграмма базы данных таблиц (Database Table Diagram) – 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.

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.

Диаграмма базы данных представлена на рисунке 2.2.

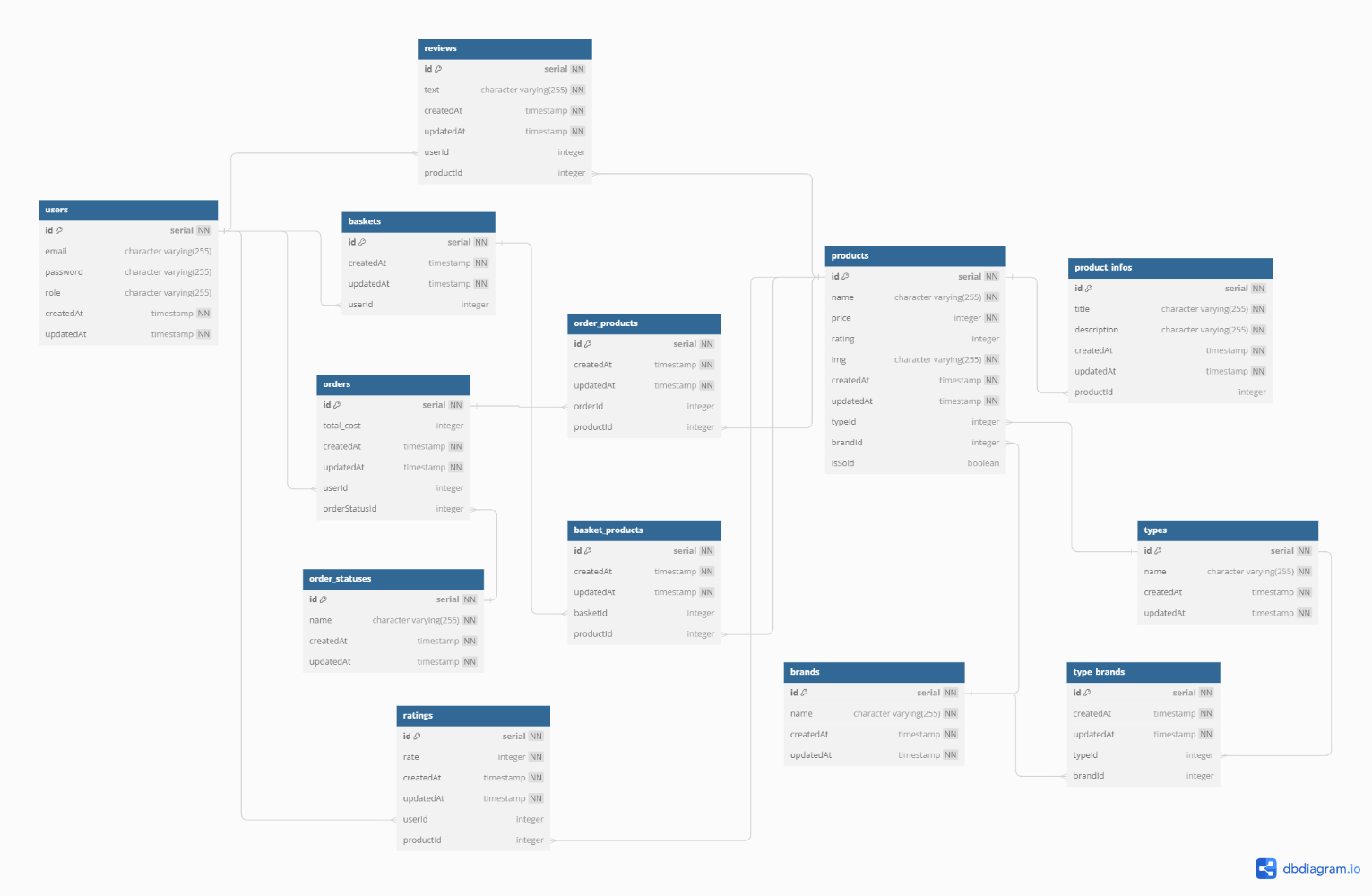


Рисунок 2.2 – Диаграмма базы данных

Диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-одному", "один-ко-многим", "многие-ко-многим".

## 2.4 Вывод

В этой главе была рассмотрена архитектура приложения, основанная на клиент-серверной модели. Описаны основные компоненты: клиентская часть, реализованная с использованием React, и серверная часть, основанная на Node.js и Express. Также было упомянуто, что для хранения и управления данными применяется база данных PostgreSQL. Описано взаимодействие между клиентской и серверной частью через HTTPS-запросы. 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 описание 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 листинги 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 3.1.

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 в приложения начинается в файле index.js в основной директории проекта, на которой 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 загрузки 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 представляющим собой список товаров содержится массив из компонентов ProductItem для загрузки которого используется хук useState(), позволяюзий 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. 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 с 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 моделью базы данных. Содержимое файла представлено в листинге 3.7.

export require('dotenv').config()

const express = require('express')

const https = require('https')

const fs = require('fs')

const sequelize = require('./db')

const models = require('./models/models')

const cors = require('cors')

const fileUpload = require('express-fileupload')

const router = require('./routes/index')

const errorHandling = require('./middleware/ErrorHandlingMiddleware')

const path = require('path')

const PORT = process.env.PORT

const SSL\_KEY\_PATH = '../private.key'

const SSL\_CERT\_PATH = '../certificate.crt'

const app = express()

app.use(cors())

app.use(express.json())

app.use('/api', router)

app.use(fileUpload({}))

app.use(express.static(path.resolve(\_\_dirname, 'static')))

app.use(errorHandling)

const options = {

key: fs.readFileSync(SSL\_KEY\_PATH),

cert: fs.readFileSync(SSL\_CERT\_PATH)

}

const start = async () => {

try {

await sequelize.authenticate()

await sequelize.sync()

https.createServer(options, app).listen(PORT, () => {

console.log(`Server started on port ${PORT}`);

});

} catch (e) {

console.log(e)

}

}

start()

Листинг 3.7 – Содержимое файла index.js

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 содержмимого файла представлен в листинге 3.8.

// описание моделей

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}

})

// описание связей

User.hasOne(Basket)

Basket.belongsTo(User)

User.hasOne(Rating)

Rating.belongsTo(User)}

Листинг 3.8 – Фрагмент файла model.js

Доступ к основному функционалу осуществляется через роутер, поставляемый модулем Router из express. Он позволяет описать маршруты к контроллерам содержащим функции приложения. Так же 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. Провека 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. В качетсве примера роутера представлен роутер обрабатывающий запросы для манипуляции товарами. 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 соответсвующий ему роутер и вызывается контроллер для его обработки. В качетсве пример 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) {

try { 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);

});

} catch (e) { next(ApiError.badRequest(e.message)); }

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 которого приложение отправляет сообщения клиентам. При смене статуса заказа при помощт пакета nodemailer пользователю на почту приходит уведомление. Реализация функция 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 createProduct

Для хранение 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 Вывод

В данном разделе были рассмотрели 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

Вначале 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.

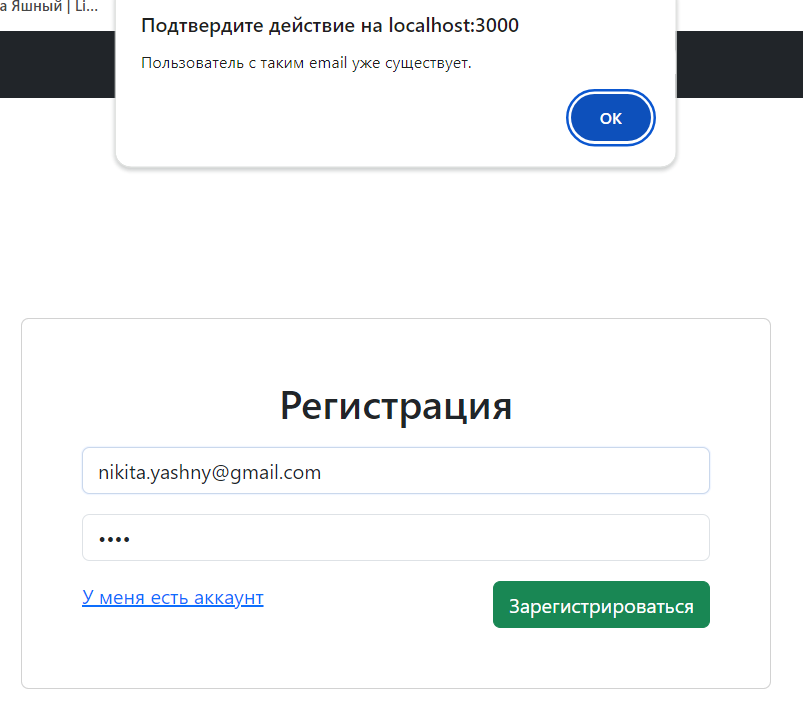


Рисунок 4.1 – Обработка ошибки связанной с вводом электронной, уже существующей в базе данных

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

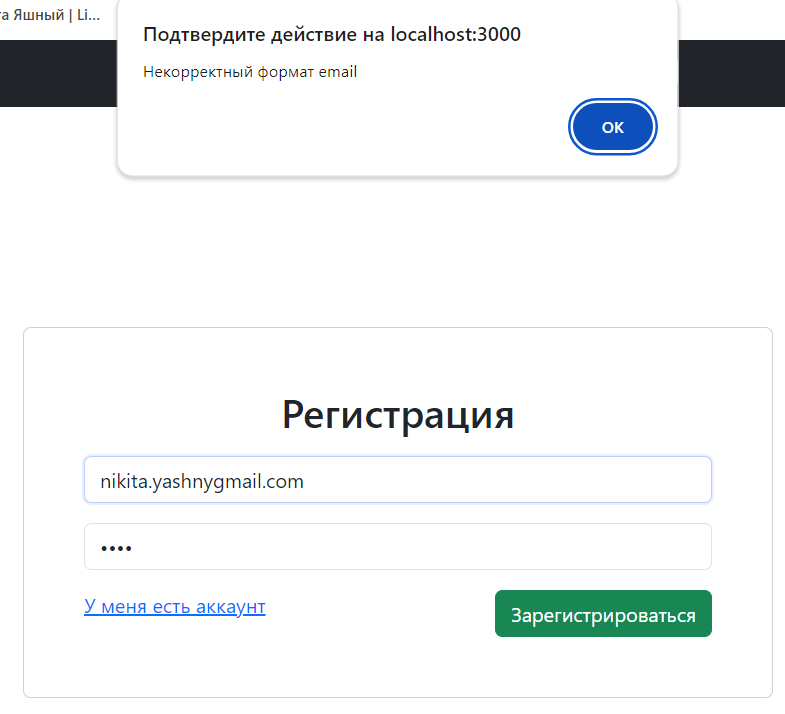


Рисунок 4.2 – Обработка ошибки связанной с вводом невалидной почты

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.

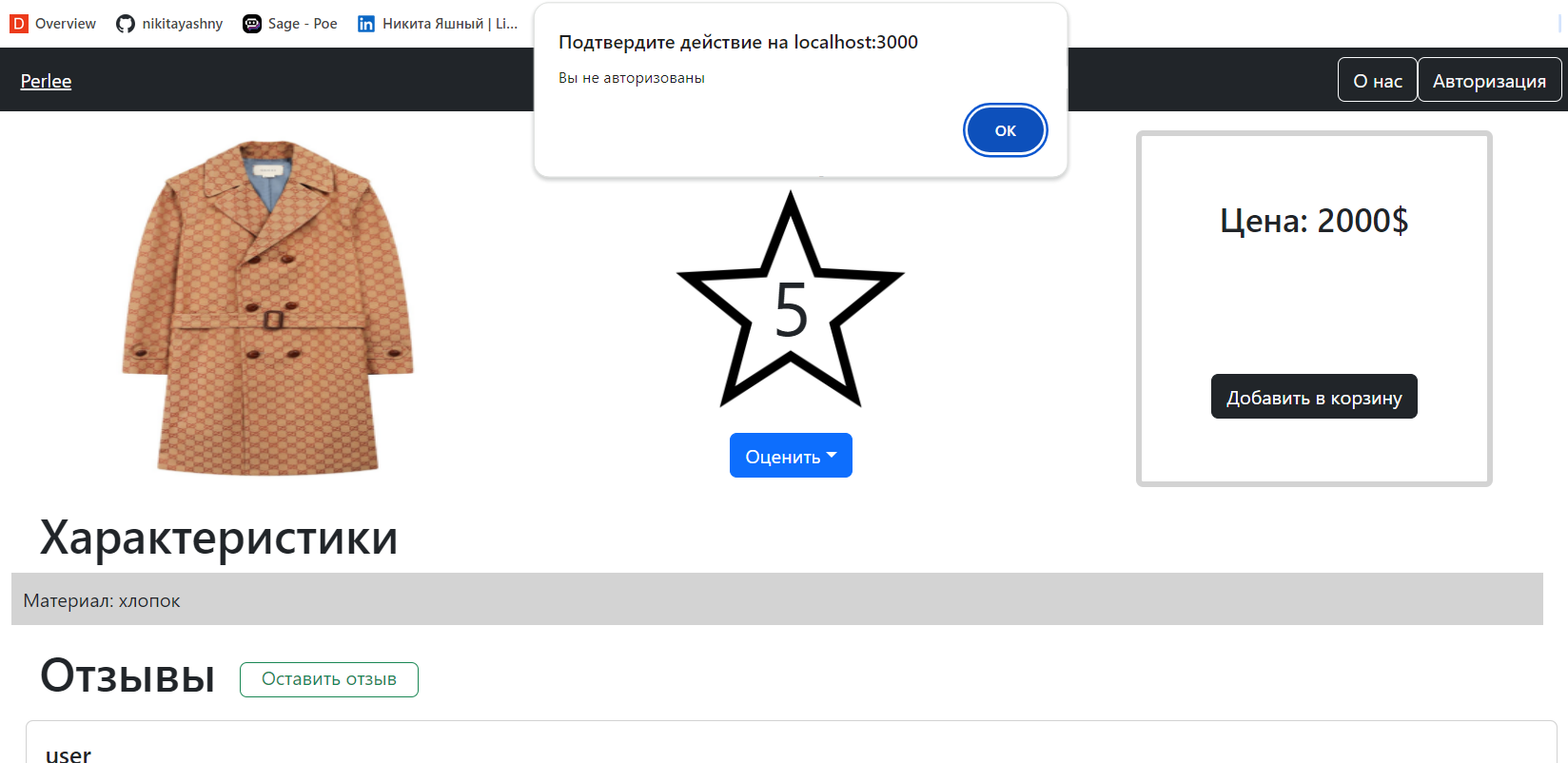


Рисунок 4.3 – Обработка ошибки связанной с попыткой добавления товара в корзину будучи неавторизованным

Проверка на попытку выполнение 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.

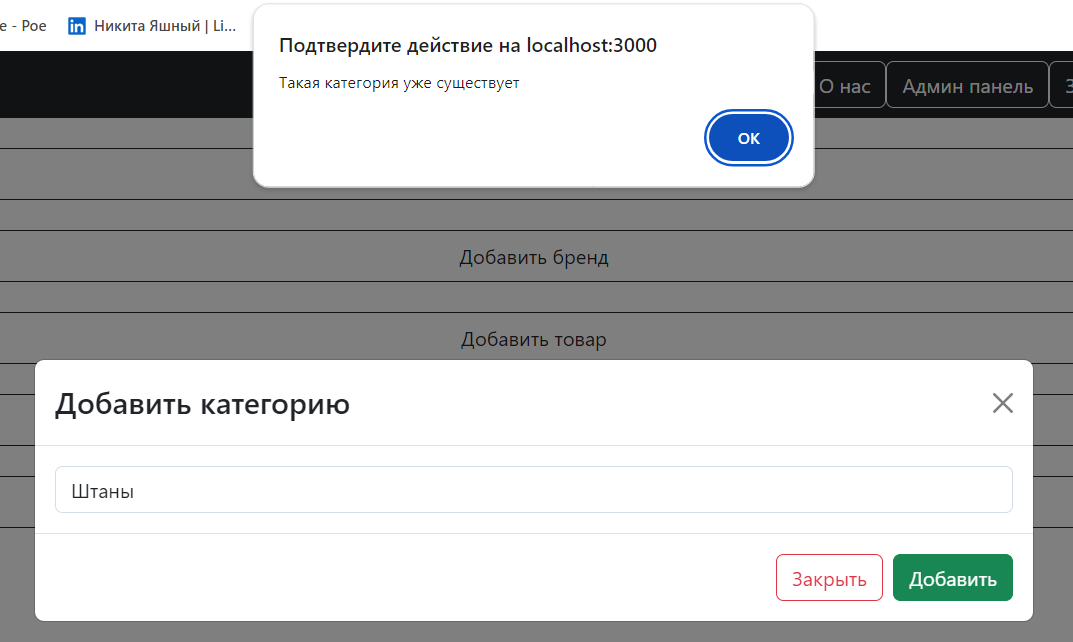


Рисунок 4.4 – Обработка ошибки связанной с попыткой добавления товара в корзину будучи неавторизованным

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

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.

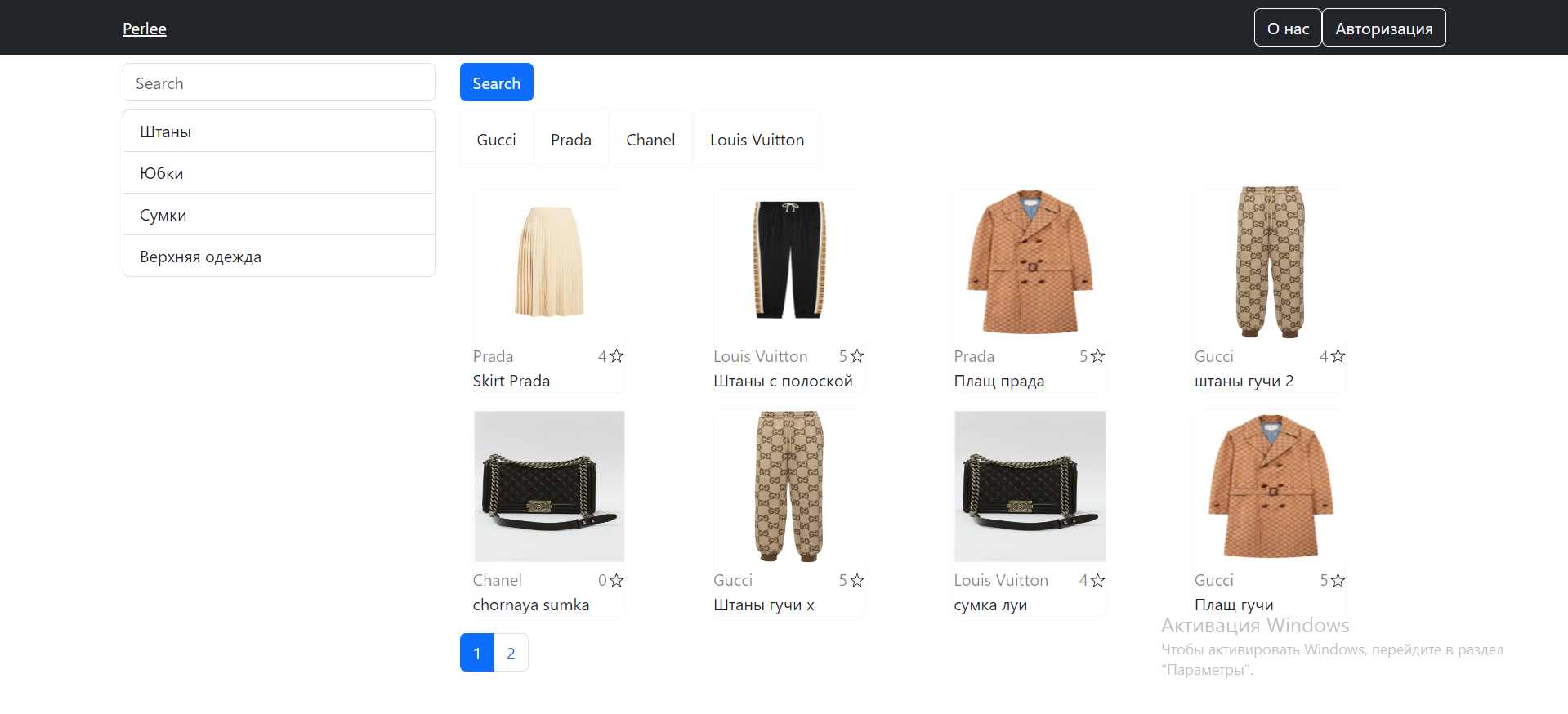


Рисунок 5.1 – Главная страница магазина

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.

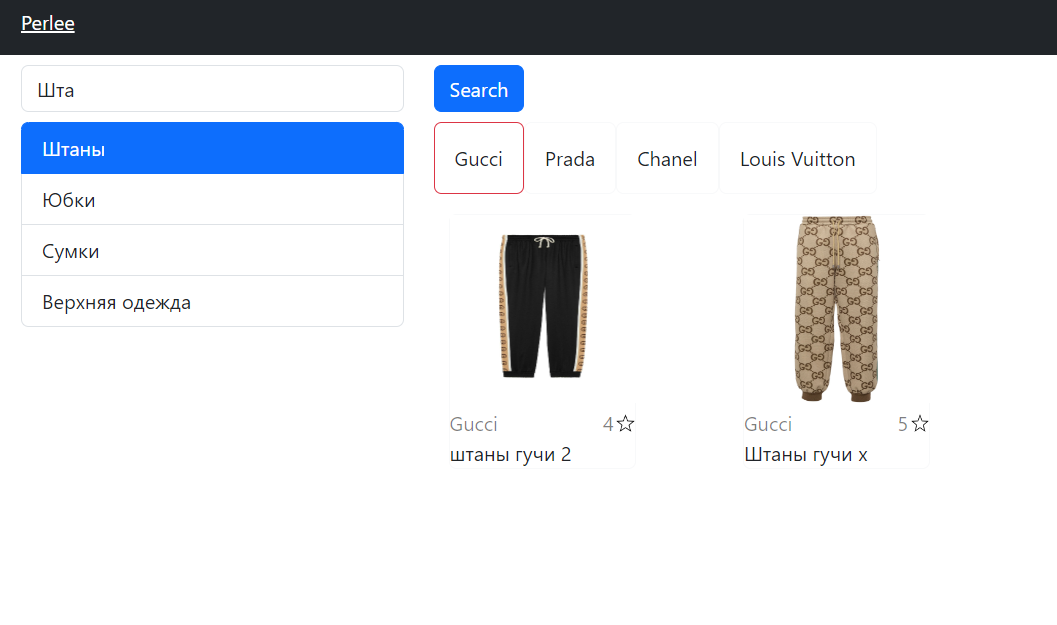


Рисунок 5.2 – Главная страница магазина

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.

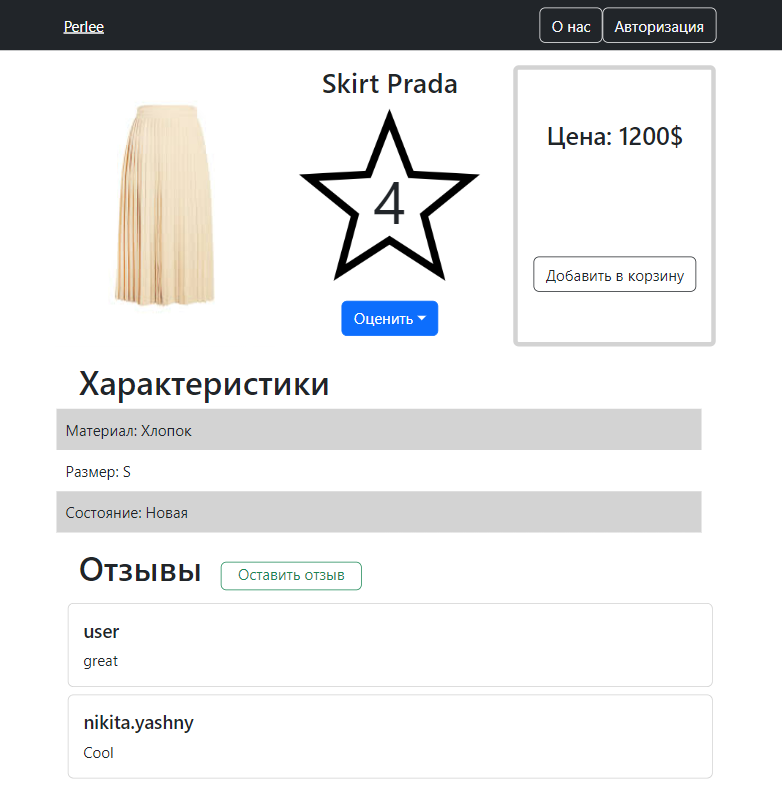


Рисунок 5.3 – Карточка товара

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.

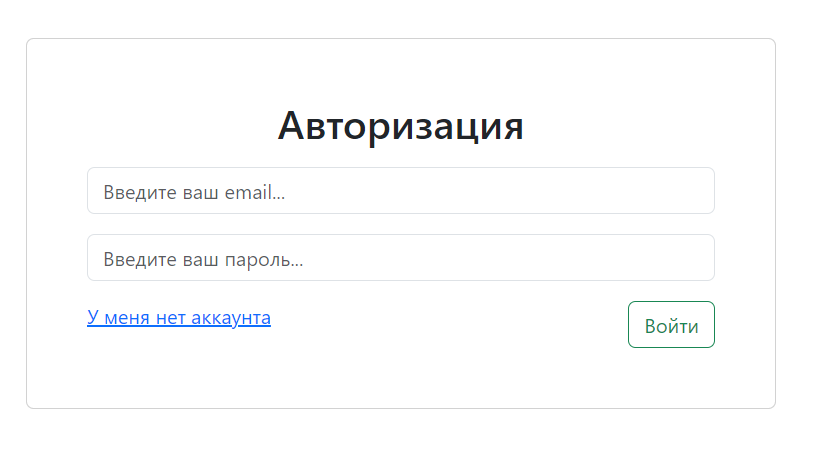


Рисунок 5.4 – Форма авторизации

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 просмотрите свою корзину и выберите то что желаете оставить, лишние можно удалить, перейдя в их карточку и нажав кнопку удаления товара из корзины. После вы можете оформить заказ, для этого перейдя на страницу корзины из панели навигации, нужно нажать кнопку оформления заказа. Пример оформления заказа приведён на рисунке 5.5.

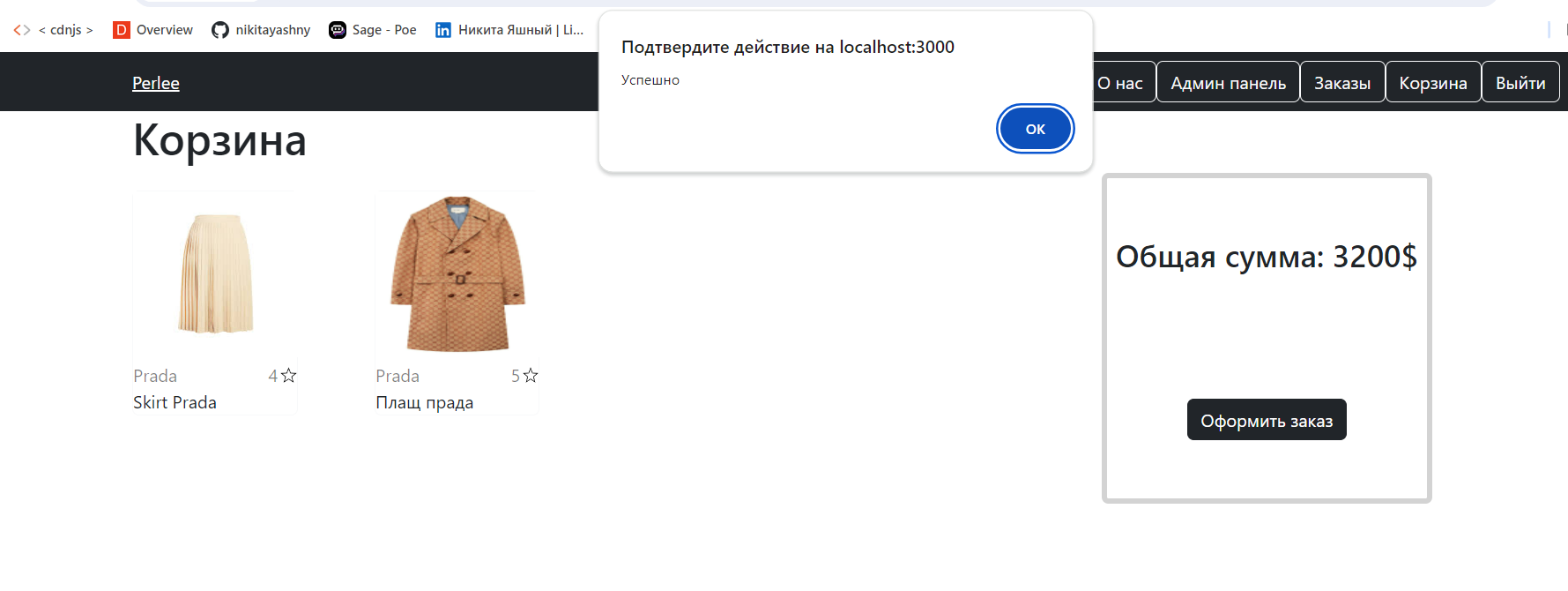


Рисунок 5.5 – Оформление заказа

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. Страница заказов представлена на рисунке 5.6.

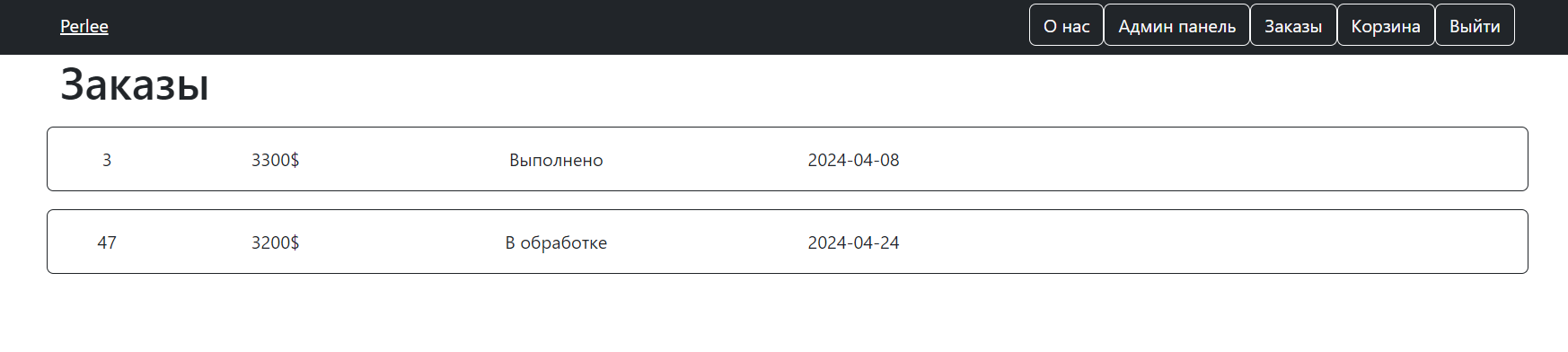


Рисунок 5.6 – Страница с заказами

Нажав на карточку вашего заказа, вы сможете перейти к подробной нформации о нем, состоящей из карточек товаров, входящих в заказ и информации, которая видна на карточке заказа. Также там имеется кнопка для отмены заказа. 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.

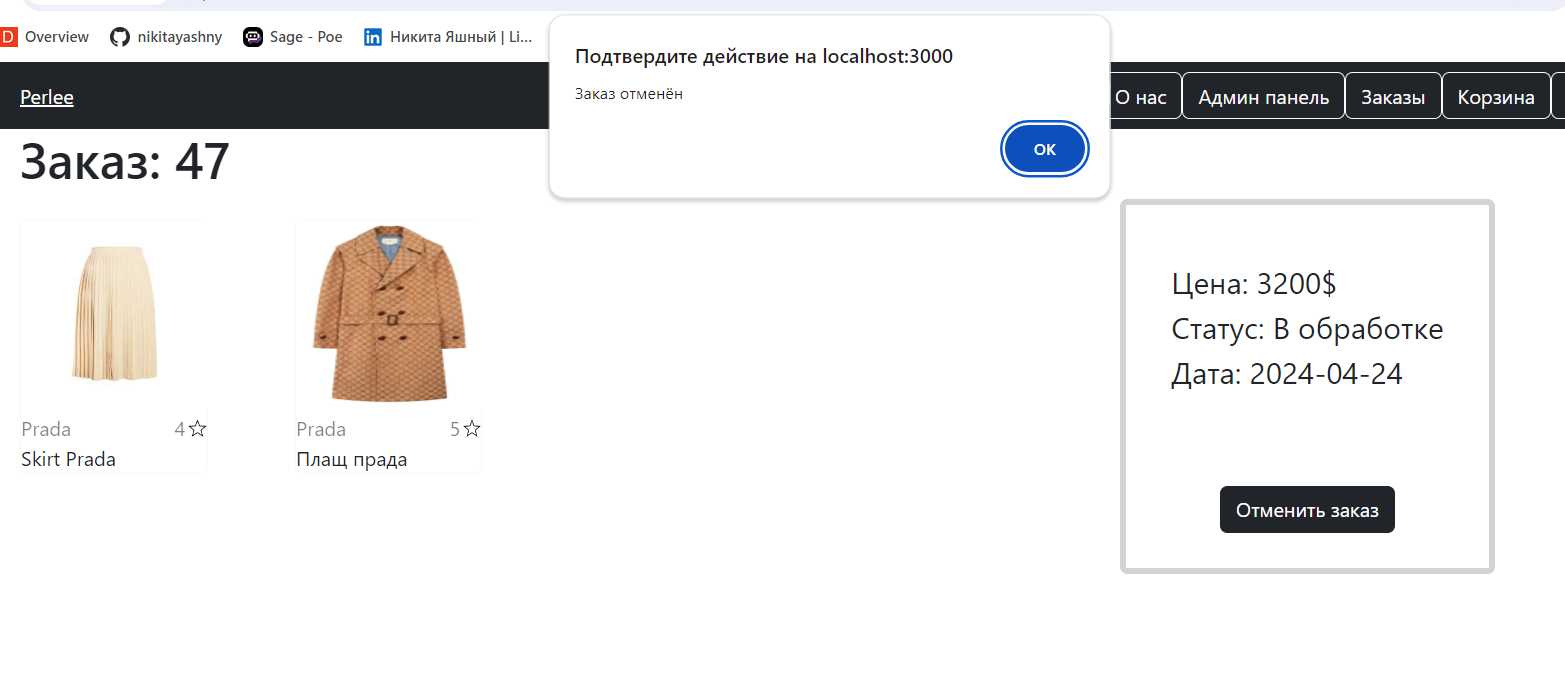


Рисунок 5.7 – Отмена заказа

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.

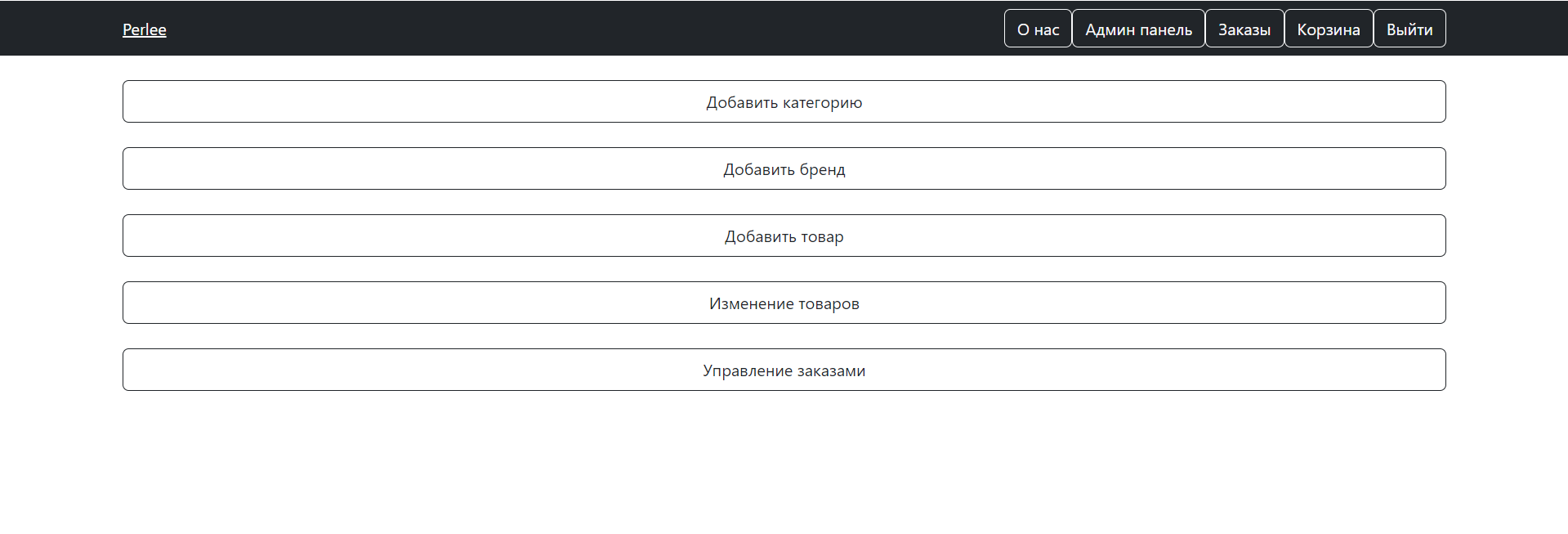


Рисунок 5.8 – Административная панель

Тут есть возможность добавить категорию, добавить бренд, добавить товар, изменить товар и панель управления заказами. Для добавления нового товара нужно нажать соответсвующую кнопку, после этого откроется модальное окно, в котором нужно заполнить форму, в неё входит название товара, наименование категории, наименовании 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.

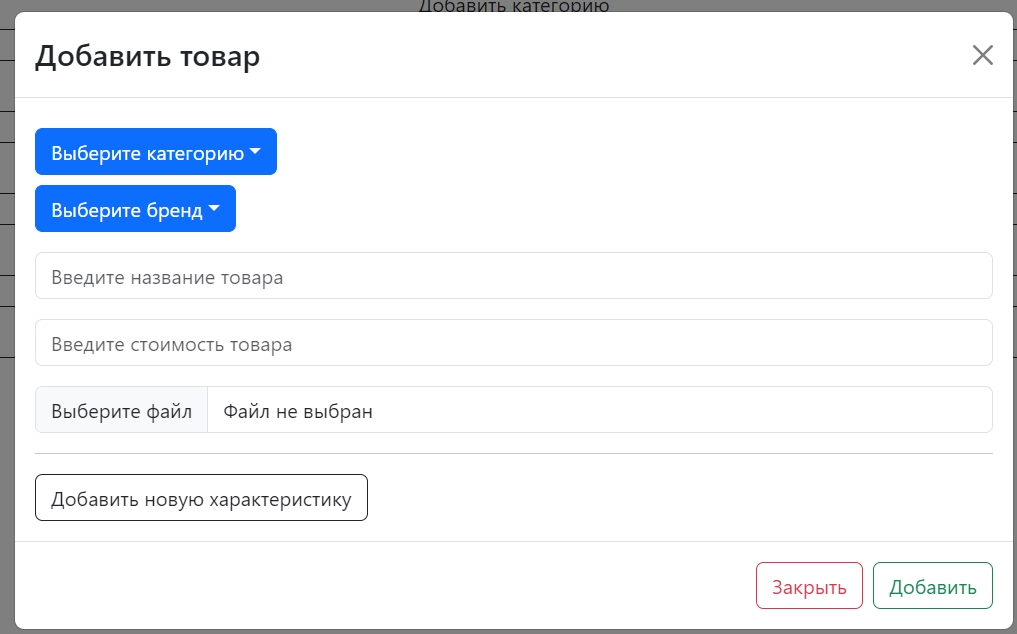


Рисунок 5.9 – Форма добавления нового товара

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

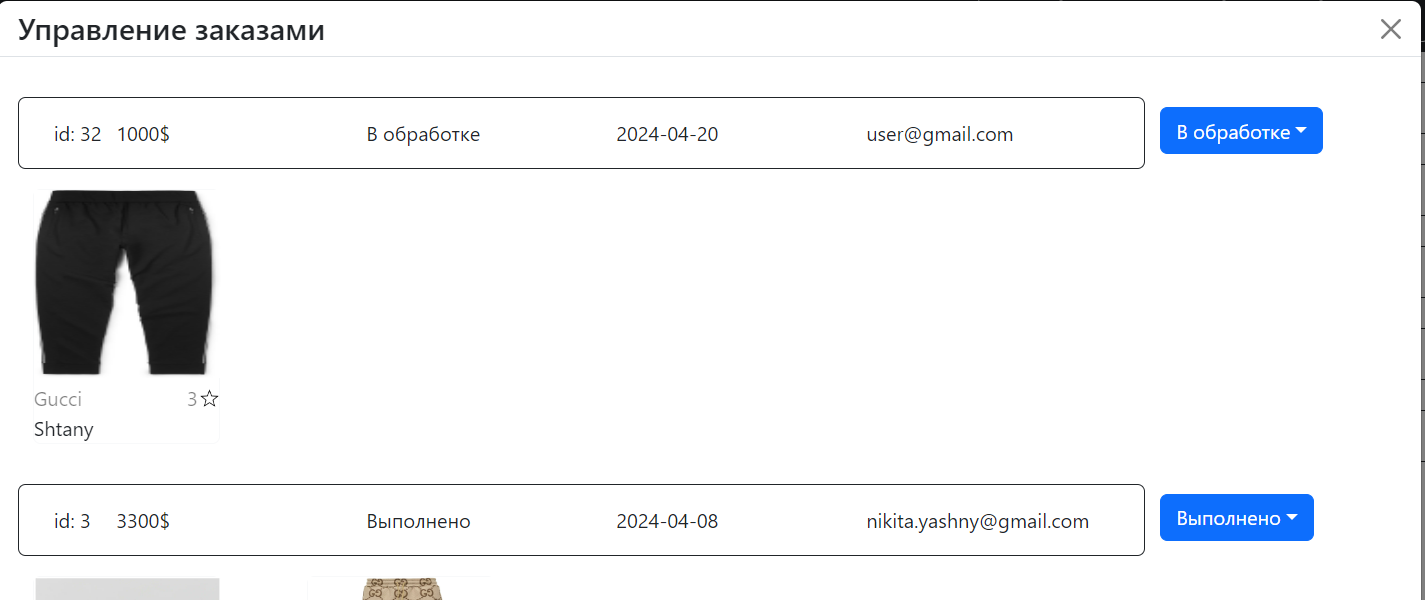


Рисунок 5.10– Модальное окно управления заказами

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 указывает адрес в котором можно его забрать.

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 5.11.

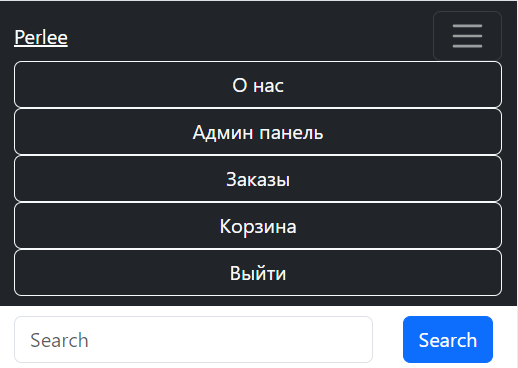


Рисунок 5.11– Панель навигации на мобильных устройствах

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

## 5.2 Установка приложения

Для запуска приложения необходимо выполнить следующие шаги:

1. Запустить серверную часть приложения, соединяющую базу данных и клиентскую часть приложения. Для этого необходимо запустить команду, которая настроит соединение с базой данных и запустит сервер. Сама команда выглядит так: “npm run dev”.

2. Запустить клиентскую часть приложение, которое будет обрабатывать пользовательские запросы, взаимодействовать с сервером и предоставлять пользовательский интерфейс. Для этого необходимо запустить команду для сборки и запуска React приложения. Сама команда выглядит так: “npm start”.

После выполнения этих шагов приложение будет полностью готово к работе.

## 5.3 Вывод

В данном разделе мы рассмотрели, как пользоваться приложением «Perlee». Рассмотрели, какие действия доступны гостю, авторизированному пользователю и администратору. Рассмотрели, как администратор может добавлять новые товары, и как он может изменять статус заказа. Рассмотрели, как пользоваться поиском для более комфортного изучения каталога для пользователей. Также рассмотрели, как пользователь может добавить товар в корзину, оформить заказ и оставить отзыв о товаре.

# Заключение

Приложение «Perlee» сайт интернет-магазина – полностью функциональное средство для продажи товаров. Оно позволяет пользователям просматривать каталог товаров, добавлять товары в корзину, совершать заказы, также можно просмотреть и оставить отзывы и поставить оценку товару. Администраторы могут добавлять новые позиции и изменять старые, также им доступна функция смены статуса заказа и отправки сообщения пользователю по электронной почте.

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.

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. React позволил легко построить структуру клиентской части веб-приложения.

База данных PostgreSQL позволила создать структуру для эффективного хранения взаимосвязанных данных. В базе данных хранятся все необходимые для работы приложения данные о товарах и пользователях.

В результате тестирования мы пришли к выводу, что приложение функционирует правильно и все выполненные оказались успешными. 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: 20.03.2024.
2. Postgresqltutorial.com [Электронный ресурс] / Режим доступа: https://www.postgresqltutorial.com/ – Дата доступа: 20.03.2024.
3. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 02.04.2024.
4. Документация на фреймворк Expess [Электронный ресурс] / Режим доступа: https://expressjs.com – Дата доступа: 03.04.2024.