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:**

Web-приложение «Магазин по продаже одежды»

Выполнил студент Климович Антон Сергеевич

(Ф.И.О.)

Руководитель работы ассист. Нистюк Ольга Александровна (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 3](#_Toc166647086)

[1 Постановка задачи 4](#_Toc166647087)

[1.1 Анализ прототипов 4](#_Toc166647088)

[1.1.1 Интернет-ресурс «Poison» 4](#_Toc166647089)

[1.1.2 Интернет-ресурс «Street Cult» 4](#_Toc166647090)

[1.2 Постановка задачи 5](#_Toc166647091)

[2 Проектирование web-приложения 6](#_Toc166647092)

[2.1 Архитектура приложения 6](#_Toc166647093)

[2.2 Проектирование структуры базы данных 7](#_Toc166647094)

[3 Разработка web-приложения 9](#_Toc166647095)

[3.1 Разработка серверной части приложения 9](#_Toc166647096)

[3.2 Разработка клиентской части приложения 12](#_Toc166647097)

[4 Тестирование web-приложения 15](#_Toc166647098)

[5 Руководство пользователя 20](#_Toc166647099)

[Заключение 29](#_Toc166647100)

[Список используемых источников 30](#_Toc166647101)

[ПРИЛОЖЕНИЕ А 31](#_Toc166647102)

[ПРИЛОЖЕНИЕ Б 32](#_Toc166647103)

[ПРИЛОЖЕНИЕ В 33](#_Toc166647104)

[ПРИЛОЖЕНИЕ Г 34](#_Toc166647105)

Введение

В современном мире бизнес онлайн-торговли одеждой становится все более динамичным и требовательным к эффективному управлению. Поэтому возникает необходимость в разработке инновационных инструментов, способных оптимизировать процессы управления и обслуживания. В рамках данного проекта будет разработано web-приложение «Интернет-магазин одежды», которое позволит значительно упростить и улучшить управление ассортиментом, заказами и пользователями.

Основной целью курсового проекта является создание комплексного и интуитивно понятного веб-приложения, которое поможет повысить эффективность управления интернет-магазином одежды, оптимизировать процессы заказа, а также улучшить обслуживание клиентов.

Требования к проекту:

* приложение должно быть направлено на упрощение процесса покупки одежды для конечного пользователя, предоставляя удобный интерфейс для выбора товаров, оформления заказа;
* приложение должно обеспечивать гибкое управление ассортиментом магазина, его редактирование и администрирование, что позволит быстро реагировать на изменения в коллекциях и спросе;
* приложение должно предоставлять возможность уведомления пользователей о статусе их заказа, что поспособствует повышению удовлетворенности клиентов и созданию положительного опыта пользования сервисом.

Техническая реализация проекта основана на асинхронном программировании с использованием языка Javascript и платформы Node.js. Программное средство взаимодействует с базой данных и может быть запущено на различных платформах.

1 Постановка задачи

* 1. Анализ прототипов

Курсовой проект представляет из себя веб-приложение для интернет магазина одежды. В бизнес-сфере, в частности в сфере торговли одеждой, уже существуют некоторые аналогичные решения. В этом разделе будут приведены интернет-ресурсы, а также веб-приложения существующих магазинов одежды .

* + 1. Интернет-ресурс «Poison»

Одним из альтернативных решений приложения пиццерии является интернет-ресурс «poison.com». На веб-сайте можно найти широкий ассортимент товаров. Каталог товаров на сайте поделен на несколько категорий: кросовки, сумоки, кофты, аксессуары и другие товары.

Интерфейс интернет-ресурса «poison.com» представлен на рисунке 1.1.

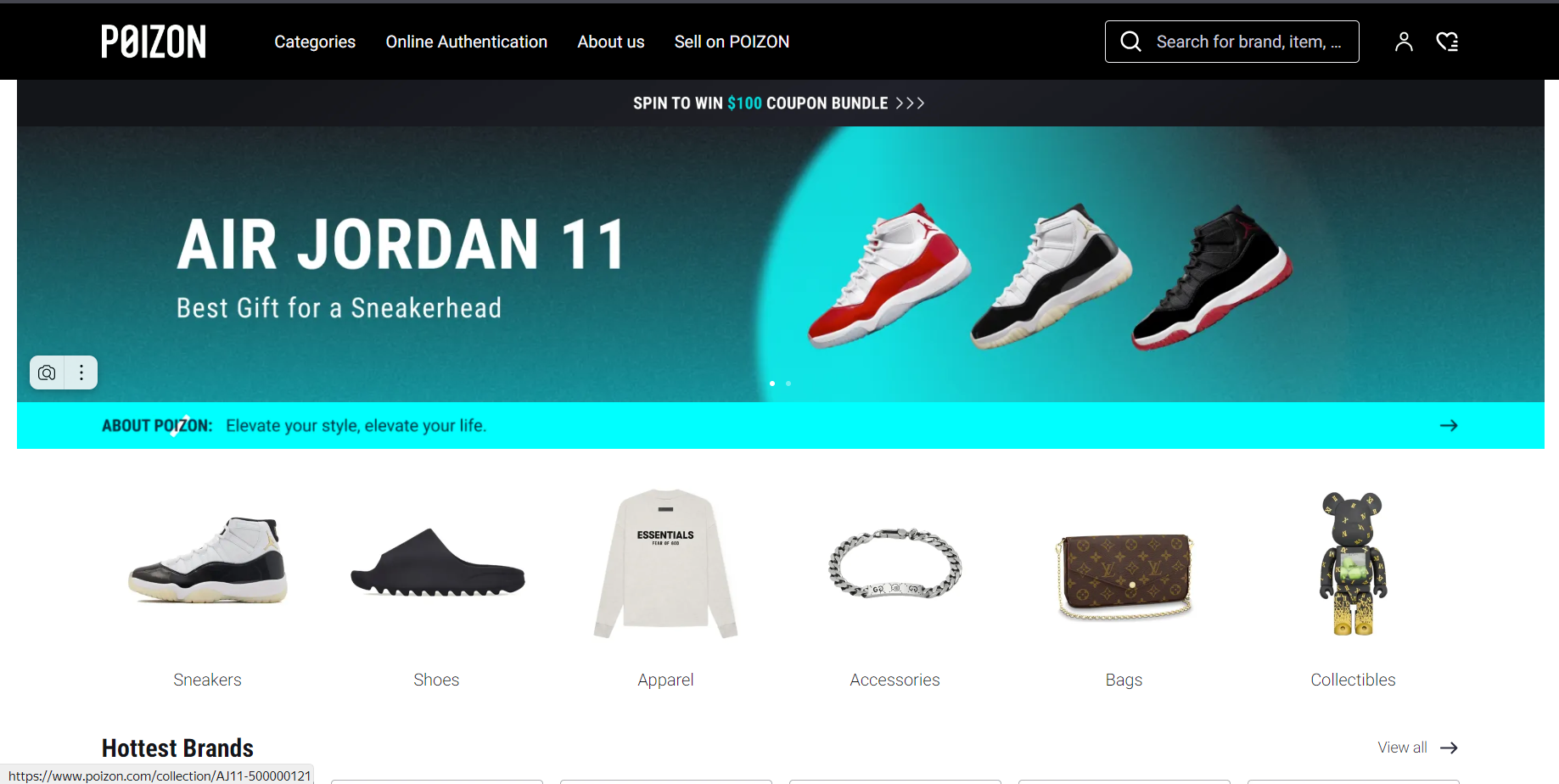


Рисунок 1.1 – Интерфейс «poison.com»

Преимуществами веб-сайта являются возможность совершения заказа с выбором опции доставки, редактирование корзины и сортировка по категориям.

К недостаткам можно отнести отсутствие избранного, а также отсутствие уведомлений об акциях.

* + 1. Интернет-ресурс «Street Cult»

Веб-сайт «Street Cult» предлагает удобный интерфейс для заказа онлайн, что упрощает процесс выбора одежды и других товаров, а также отображает информацию о размерах одежды, что помогает клиентам отследитьтовары нужные ему решения.

Интерфейс интернет-ресурса «Street Cult» представлен на рисунке 1.2.

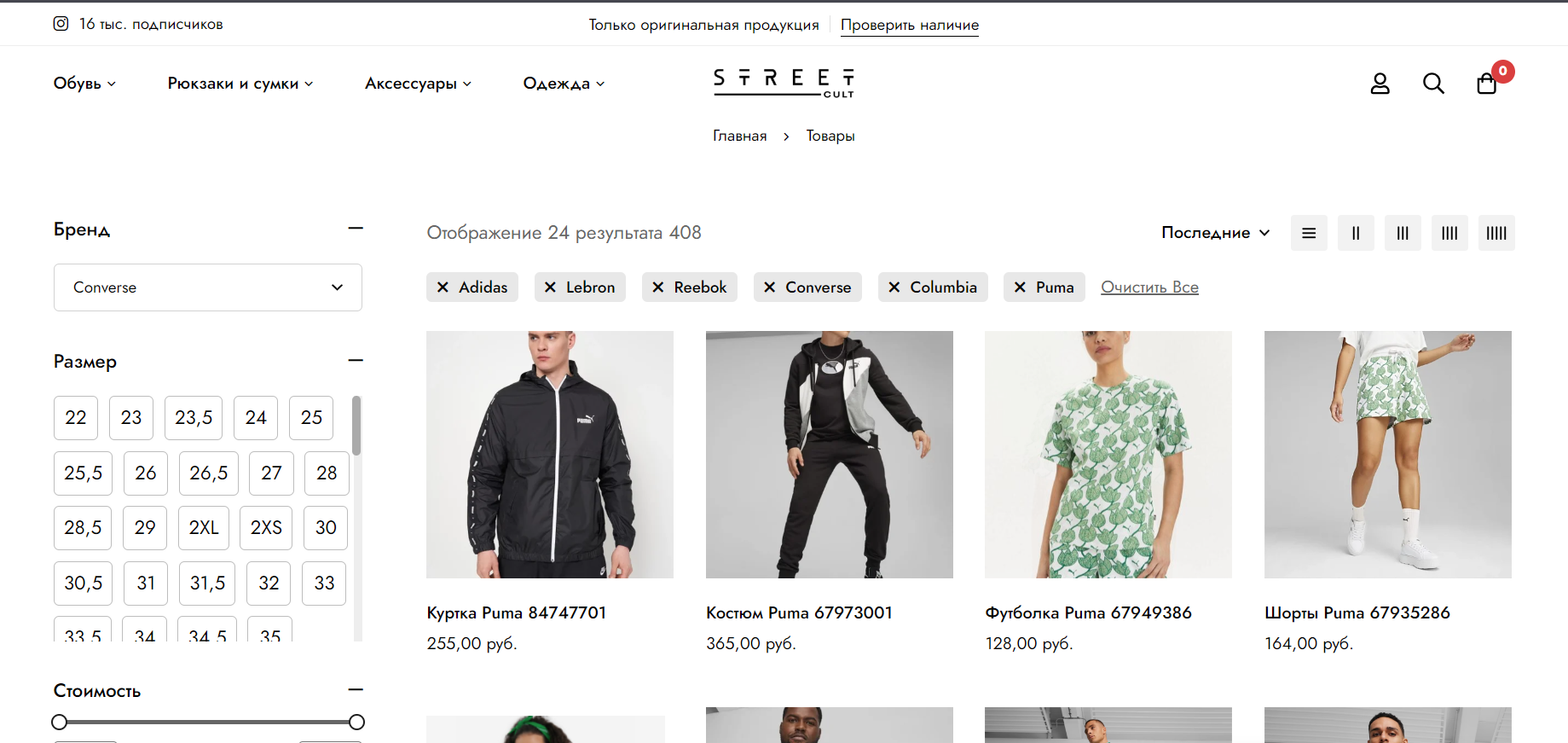


Рисунок 1.2 – Интерфейс «Street Cult»

К преимуществам веб-сайта можно отнести возможность регистрации пользователя, возможность работы с размерами товаров, управление корзиной, просмотр совершенных заказов.

Недостатками веб-сайта является отсутствие функции добавления в избранное и отсутсвие уведомлений пользователей.

* 1. Постановка задачи

На сегодняшний день актуальность разработки программного средства для управления сетью пиццерий велика, так как многие компании стремятся улучшить свои процессы производства, управления заказами и доставки, а также повысить удовлетворенность клиентов.

Обзор аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и сформулировать список требований, предъявляемых к разрабатываемому в данном курсовом проекте программному средству. Программное средство должно реализовывать следующие функции:

* управление корзиной товаров;
* управление избранным;
* совершение заказа;
* управление товарами;
* управление категориями;
* поиск по категорям;
* уведомление пользователя о заказе;
* удаление товаров из корзины и избранного;
* просмотр статистики продаж.

Диаграмма вариантов использования представлена в приложении А.

В данной главе были сформулированы основные требования к приложению на основе преимуществ и недостатков некоторых аналогов.

2 Проектирование web-приложения

2.1 Архитектура приложения

Серверная часть веб-приложения реализована с использованием фреймворка Express.js, который применяет архитектуру MVC (Model-View-Controller) для организации компонентов. Эта архитектура разделяет приложение на три основных уровня: модели (models), представления (views) и контроллеры (controllers).

Контроллеры (controllers) являются центральным звеном в архитектуре MVC. Они отвечают за обработку входящих запросов от клиентов и формирование соответствующих ответов. Контроллеры определяют конечные точки (endpoints) API и управляют логикой обработки запросов. Они принимают запросы от клиентов, вызывают соответствующие методы моделей для получения или изменения данных и передают данные в представления для отображения.

Модели (models) представляют собой структуры данных и логику работы с этими данными. Они отвечают за доступ к базе данных, выполнение операций CRUD (Create, Read, Update, Delete) и валидацию данных. Модели обычно содержат методы для работы с данными, такие как сохранение, поиск и удаление записей, а также бизнес-логику, связанную с этими данными.

Представления (views) отвечают за отображение данных пользователю. Они представляют собой HTML-шаблоны, которые могут быть заполнены данными из контроллеров или моделей. Представления определяют, как данные будут отображаться пользователю в браузере или другом клиентском приложении.

Архитектура MVC в Express.js помогает разделить логику приложения на отдельные компоненты, улучшая читаемость и поддерживаемость кода. Эта структура помогает упростить разработку, отладку и поддержку приложения.

Клиентская часть проекта использует Express.js, который представляет собой библиотеку JavaScript для разработки динамичных пользовательских интерфейсов. Express.js позволяет создавать переиспользуемые компоненты, которые эффективно обновляются и реагируют на изменения данных. Это достигается за счет использования виртуального DOM (Virtual DOM), который позволяет Express обновлять только необходимые элементы интерфейса без перерисовки всего дерева компонентов. Такой подход улучшает производительность взаимодействия с динамическими интерфейсами и значительно упрощает процесс разработки.

Взаимодействие между клиентской и серверной частями осуществляется посредством Web API. Сервер предоставляет API, определяющий доступные конечные точки и форматы данных для обмена информацией с клиентским приложением. Клиентское приложение, построенное на Express.js, отправляет HTTP-запросы на сервер, используя соответствующие конечные точки, и принимает ответы, обычно в формате JSON. Это обеспечивает надежное взаимодействие между клиентом и сервером, позволяя передавать данные и обновления между ними и обеспечивая функциональность и интерактивность всего приложения.

Для хранения данных используется база данных Postgres. База данных Postgres представляет собой мощный инструмент для хранения данных и их анализа.

Преимуществом использование данной СУБД является возможность интеграции геоданных в приложения. Postgres обеспечивает удобные средства для работы с данными как на серверной, так и на клиентской стороне. Это позволяет разработчикам создавать приложения, которые могут анализировать и визуализировать данные, что особенно полезно в информационных системах.

Также в приложении используется протокол WebSockets. Использование WebSockets в web-приложении позволяет создать интерактивный и динамичный пользовательский опыт. WebSockets обеспечивают постоянное соединение между клиентом и сервером, что позволяет мгновенно передавать данные в обоих направлениях без необходимости постоянного обновления страницы.

Диаграмма развертывания представлена в приложении Б.

2.2 Проектирование структуры базы данных

В данном разделе представлена концептуальная и логическая модель базы данных для проекта. На этапе разработки концептуальной модели было проведено детальное изучение бизнес-требований и основных процессов интернет магазинов. Основные сущности, выделенные в концептуальной модели, включают в себя:

* пользователи: представляет информацию о зарегистрированных пользователях системы;
* пользователи: содержит информацию о зарегестрированных пользователях;
* каталог: содержит информацию о товарах, предлагаемых магазином;
* категории: хранит информацию об категории товара;
* заказы: хранит данные о заказах, сделанных пользователями;
* корзина: хранит данные о товарах, которые находятся в корзине пользователя;
* избранное: хранит данные о товарах, которые находятся в избранном у пользователя;
* уведомления: хранит информацию уведомлениях пользователей.

Диаграмма базы данных представлена в приложении В.

По итогу были сформированы следующие таблицы: USER, CATEGORY, CLOTHING, CART, FAVORITE, NOTIFICATION, PAYMENT, SALES STATISTIC. Далее представлено описание каждой из них.

Таблица USER используется для хранения данных о пользователях, таких как хешированный пароль, роль пользователя, персональные данные. Ее структура представлена в таблице 2.1.

Таблица 2.1 – Структура таблицы USER

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| id | serial | Primary key |
| password\_hash | text | Not null |
| user\_role | int | Not null, foreign key |
| personal\_data | int | Not null, foreign key |
| role\_name | boolean | Not null |

Таблица CATEGORY используется для хранения информации о категориях товаров. Структура представлена в таблице 2.2.

Таблица 2.2 – Структура таблицы CATEGORY

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| category\_id | serial | Primary key |
| category\_name | text | Not null |

Таблица CLOTHING содержит информацию об одежде: название, описание, цену, адрес картинки. Структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы CLOTHING

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| clothing\_id | serial | Primary key |
| title | text | Not null |
| description | text | Not null |
| price | numeric(10, 2) | Not null |
| image\_url | text | Not null |

Таблица CART содержит информацию товарах в корзине пользователя: идентификатор корзины, количевство товаров в корзине. Ее структура представлена в таблице 2.4.

Таблица 2.4 – Структура таблицы CART

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| cart\_id | serial | Primary key |
| quantity | int | Not null |

Таблица FAVORITE содержит информацию товарах в избранном пользователя: идентификатор избранного. Ее структура представлена в таблице 2.5.

Таблица 2.5 – Структура таблицы FAVORITE

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| favorite\_id | serial | Primary key |

Таблица NOTIFICATION содержит информацию об уведомлениях пользователя. Таблица имеет поля: идентификатор уведомления, сообщение, статус сообщения . Структура представлена в таблице 2.6.

Таблица 2.6 – Структура таблицы NOTIFICATION

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| notification\_id | serial | Primary key |
| message | text | Not null |
| status | boolean | Not null |

Таблица PAYMENT представляет информацию об оплате. Ее структура представлена в таблице 2.7.

Таблица 2.7 – Структура таблицы PAYMENT

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| payment\_id | serial | Primary key |
| payment\_date | date | Not null |
| amount | numeric(10, 2) | Not null |

Таблица SALES STATISTIC представляет информацию о корзине пользователя. Ее структура представлена в таблице 2.8.

Таблица 2.8 – Структура таблицы SALES STATISTIC

|  |  |  |
| --- | --- | --- |
| Столбец | Тип данных | Ограничение целостности |
| statistic\_id | int | Primary key |
| quantity\_sold | int | Not null |
| sales\_date | date | Not null |
| total\_revenue\_ | numeric(10, 2) | Not null |

В процессе проектирования базы данных было выявлено необходимость в восьми таблицах. Postgres идеально соответствует нашим потребностям благодаря своей простоте, удобству использования и возможности хранения данных, что позволяет эффективно управлять данными. Это значительно облегчает дальнейшую разработку приложения. Помимо этого, Postgres обеспечивает высокий уровень безопасности, доступности и масштабируемости, что является важным преимуществом при последующем расширении функционала приложения.

3 Разработка web-приложения

3.1 Разработка серверной части приложения

Серверная часть приложения обеспечивает обработку клиентских запросов, выполняет запросы к базе данных, обеспечивает аутентификацию и авторизацию, а также поддерживает websocket сервер.

Структура серверного приложения представлена на рисунке 3.1.

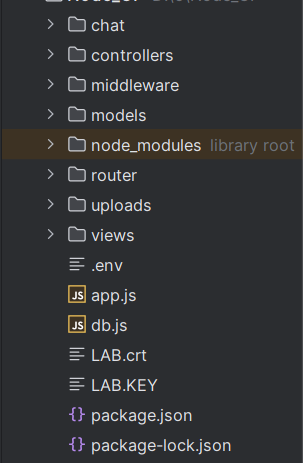


Рисунок 3.1 – Структура проекта серверного приложения

Входной точкой приложения является файл app.js. В нем происходит инициализация express-сервера, websocket сервера, а также установка сертификатов для обеспечения защищенного соединения с сервером. Мы также определяем обработчики событий для подключения клиента к веб-сокет серверу, получения сообщений от клиента и закрытия соединения. Этот подход обеспечивает нам надежное и эффективное взаимодействие с клиентами в реальном времени.

В директории models находится схема моделей, созданная ORM-фреймворком sequelize. Модели описывают сущности базы данных и представляют интерфейс для взаимодействия с ней. Схема моделей представлена в приложении Г.

В директории router находятся маршрутизаторы для каждой сущности. Пример маршрутизатора для сущности «корзина» представлен в листинге 3.2.

|  |
| --- |
| const express = require('express');  const router = express.Router();  const cartController = require('../controllers/cartController'); router.use(express.json());  const {checkAuth} = require("../middleware/checkAuth");    router.get('/cart', checkAuth, cartController.getCart); router.post('/cart/decrease/:cartItemId',checkAuth, cartController.decreaseQuantity); router.post('/cart/increase/:cartItemId',checkAuth, cartController.increaseQuantity); router.delete('/cart/:cartItemId',checkAuth, cartController.deleteCartItem); router.post('/cart/purchase/all', checkAuth, cartController.purchaseAllItems) router.get('/cart/totalAmount', checkAuth , cartController.getTotalAmount); router.post('/cart/purchase/item', checkAuth, cartController.purchaseItem);  module.exports = router; |

Листинг 3.1 – Маршрутизатор для сущности «корзина»

С каждым маршрутом сопоставляется определенный метод контроллера, который непосредственно осуществляет обработку запроса. Пример метода контроллера addToCart, который возвращает информацию об одном курьере представлен в листинге 3.2.

|  |
| --- |
| exports.addToCart = async (req, res) => {  try {  const userId = req.id;  const {itemId} = req.body;  const clothing = await Clothing.findByPk(itemId);  let cartItem = await CartItem.findOne({where: {UserUserId: userId, ClothingClothingId: itemId}});  if (cartItem) { cartItem.quantity += 1;  await cartItem.save(); }  else {  await CartItem.create({  quantity: 1,  UserUserId: userId,  ClothingClothingId: itemId,  title: clothing.title,  image\_url: clothing.image\_url  });  }  res.status(200).send('Item added to cart successfully');  }  catch (error) { console.error('Error adding item to cart:', error); res.status(500).send('Internal Server Error'); }  }; |

Листинг 3.2 – Метод addToCart контроллера cartController

Кроме конечных точек приложения, маршрутизаторы также используют промежуточную обработку запроса за счет использования middleware.

В приложении реализовано несколько middleware. Первый тип – это middleware для аутентификации и авторизации. Middleware checkAuth необходим для того, чтобы защитить некоторые маршруты от не аутентифицированных пользователей. Код middleware представлен в листинге 3.3.

|  |
| --- |
| exports.checkAuth = async (req, res, next) => {  try {  if (req.cookies.token) {  const token = req.cookies.token;  const decoded = jwt.verify(token, 'secret');  req.id = decoded.id;  req.role = decoded.role;  next();  } else {  res.redirect('/auth/login');  }  } catch (err) {  res.status(500).json({ message: 'Internal Server Error' });  }  }; |

Листинг 3.3 – Middleware checkAuth

Middleware checkAdmin предназначен для проверки на роль администратора. Код middleware представлен в листинге 3.4.

|  |
| --- |
| exports.checkAdmin = async (req, res, next) => {  try {  if (req.cookies.token) {  const token = req.cookies.token;  const decoded = jwt.verify(token, 'secret');  req.id = decoded.id;  req.role = decoded.role;  if (req.role === true) next();  else res.redirect('/user/homePage');  } else {  res.redirect('/auth/login');  }  } catch (err) {  res.status(500).json({ message: 'Internal Server Error' });  }  }; |

Листинг 3.4 – Middleware checkAdmin

3.2 Разработка клиентской части приложения

Разработка клиентской части приложения является ключевым этапом в создании пользовательского опыта высокого качества. Этот процесс включает в себя создание привлекательного и интуитивно понятного интерфейса пользователя (UI), который будет легко восприниматься и использоваться конечными пользователями. Важной частью разработки является также реализация взаимодействия с сервером, что обеспечивает получение и отправку данных между клиентской и серверной частями приложения.

Структура клиентского приложения представлена на рисунке 3.2.

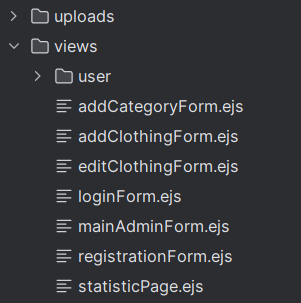


Рисунок 3.2 – Структура клиентского приложения

Клиентская часть приложения разработана с использование шаблонизатора EJS, такое решение упрощает передачу данных клиенту, а также предоставляет все те преимущества которыми обладают шаблонизаторы, к таким преимуществам относятся: экономия траффика, т.к. передаются только данные ввиде json-объекта, шаблон кэшируется и берется из кэша, снижение нагрузки на сервер, т.к. шаблонизация происходит в браузере, MVC структура, с выводом «V» за пределы сервера.

Директория «view» содержит основные представления, а также ещё несколько директив, содержимое тех директориев рассмотрим дальше. Содержимое представлено на рисунке 3.3.

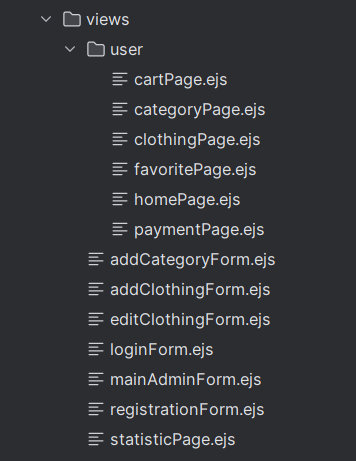


Рисунок 3.3 – Содержимое директория «view»

Файл app.js является основным компонентом приложения. Помимо инициализации всех остальных компонентов приложения он также инициализирует websoket-клиента.

4 Тестирование web-приложения

В данном разделе мы сфокусируемся на процессе тестирования завершенного приложения, с основным вниманием на клиентской части. Тестирование является важным этапом разработки, поскольку позволяет выявить и исправить ошибки и недочеты перед выпуском приложения.

При регистрации пользователя в случае ввода некорректных данных, будет выведено классическое сообщение об ошибке. Вывод ошибки представлен на рисунке 4.1.

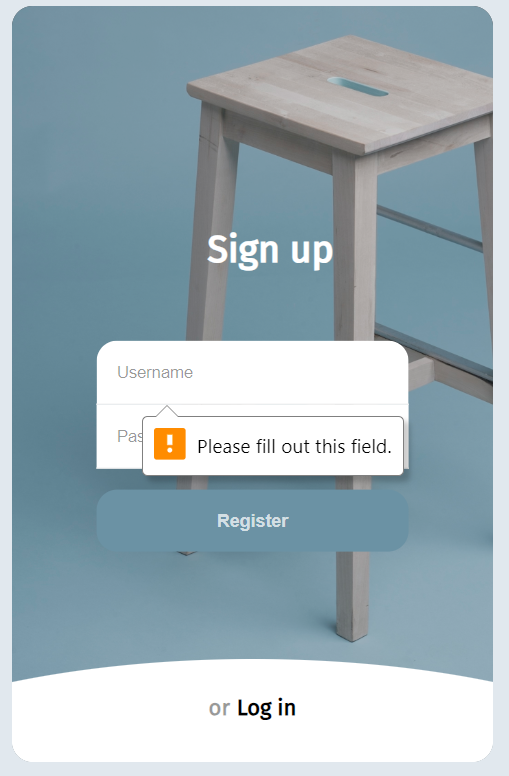


Рисунок 4.1 – Обработка некорректных данных при регистрации

При авторизации в приложении также могут возникать ошибки. Пример обработки ошибки, возникающей если пользователя не существует в системе, представлен на рисунке 4.2.

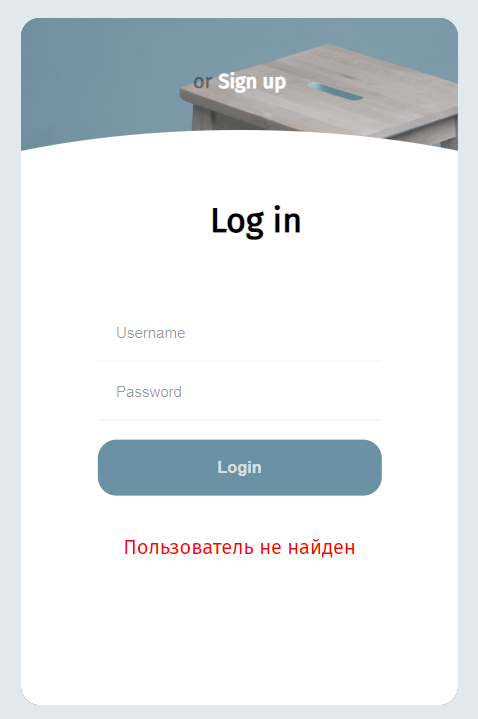


Рисунок 4.2 – Вход под несуществующим пользователем

Пример обработки ошибки, возникающей если пользователь ввел некорректный пароль, представлен на рисунке 4.3.

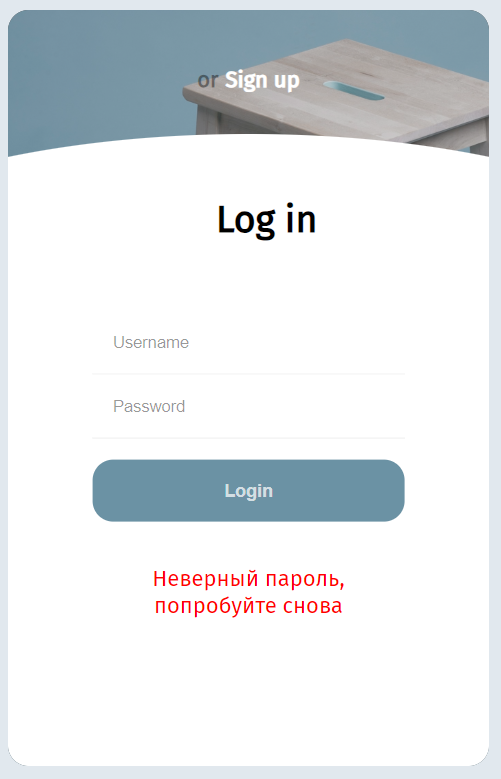


Рисунок 4.3 – Вход с неправильным паролем

Ошибки также обрабатываются при добавлении или изменении данных различных сущностей. Пример обработки ошибок числовых данных при добавлении товара представлен на рисунке 4.4.

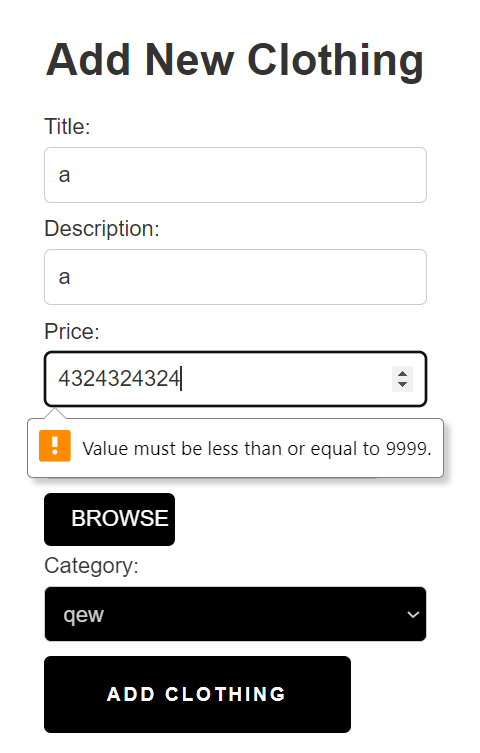


Рисунок 4.4 – Вход числовых данных неверного формата

Пример обработки ошибок при отправке пустых данных представлен на рисунке 4.5.

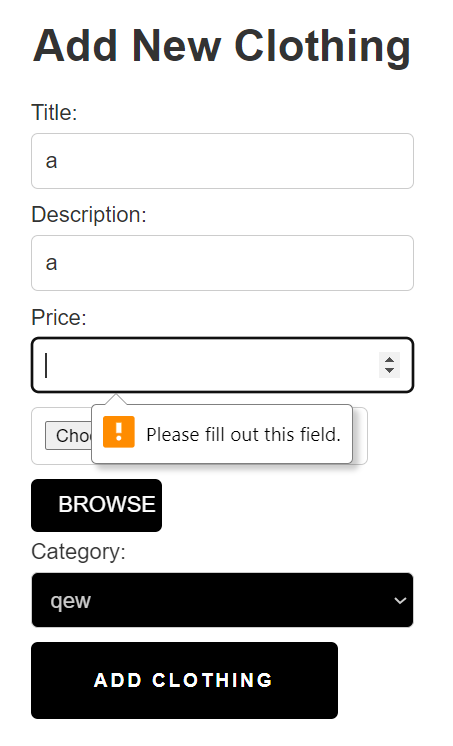


Рисунок 4.5 – Ошибка отправки пустых данных

Приложение также имеет обработку ошибки на добавлении категории с таким же именем. Обработка ошибок на ошибки на добавлении категории с таким же именем представлена на рисунке 4.6.

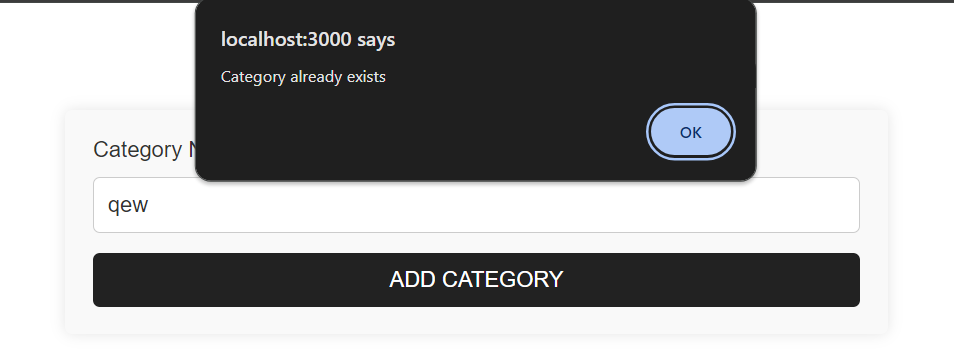


Рисунок 4.6 – Ошибка превышения количества товаров в корзине

Таким образом было проведено тестирование приложения. Были продемонстрированы типичные ситуации, вызывающие ошибки и реакция на них. Результаты тестирования позволили выявить и исправить множество ошибок и проблем, связанных с функциональностью и производительностью приложения.

5 Руководство пользователя

При регистрации пользователь должен ввести уникальное имя, и пароль. Форма регистрации представлена на рисунке 5.1.

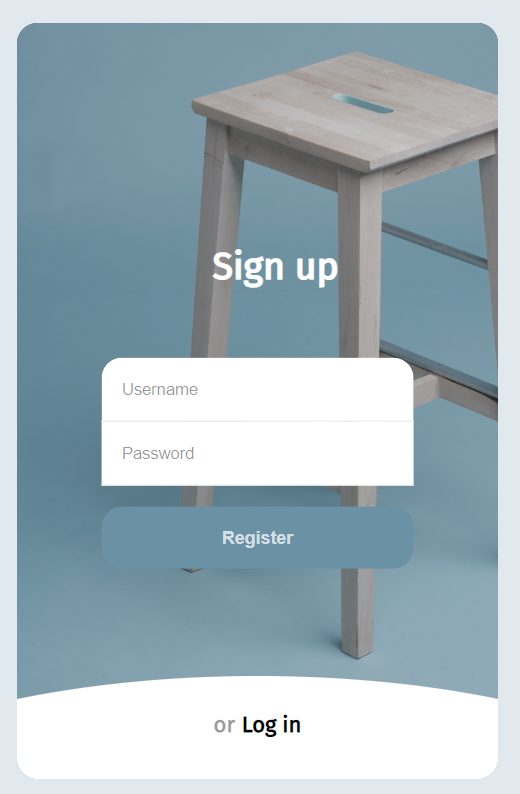


Рисунок 5.1 – Форма регистрации

После того, как пользователь нажмет на кнопку «Registration», будет отправлен запрос на регистрацию, на стороне сервера выполнится валидация и, если всё валидно, данные нового пользователя добавится в базу данных, а сам пользователь будет перенаправлен на главную страницу.

На странице авторизации пользователь должен ввести уникальное имя и пароль, введённые при регистрации. Форма авторизации представлена на рисунке 5.2.

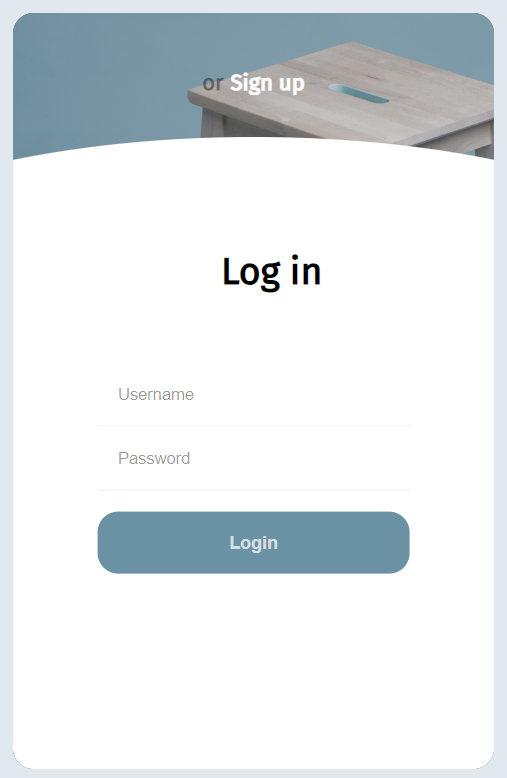


Рисунок 5.2 – Страница «Авторизации»

В случае правильных учётных данных происходит перенаправление на главную страницу.

Попадая на главную страницу, пользователь видит карточки товаров, а также, у него есть возможность взаимодействовать с меню, добавлять товар в избранное и корзину. Главная страница представлена на рисунке 5.3.

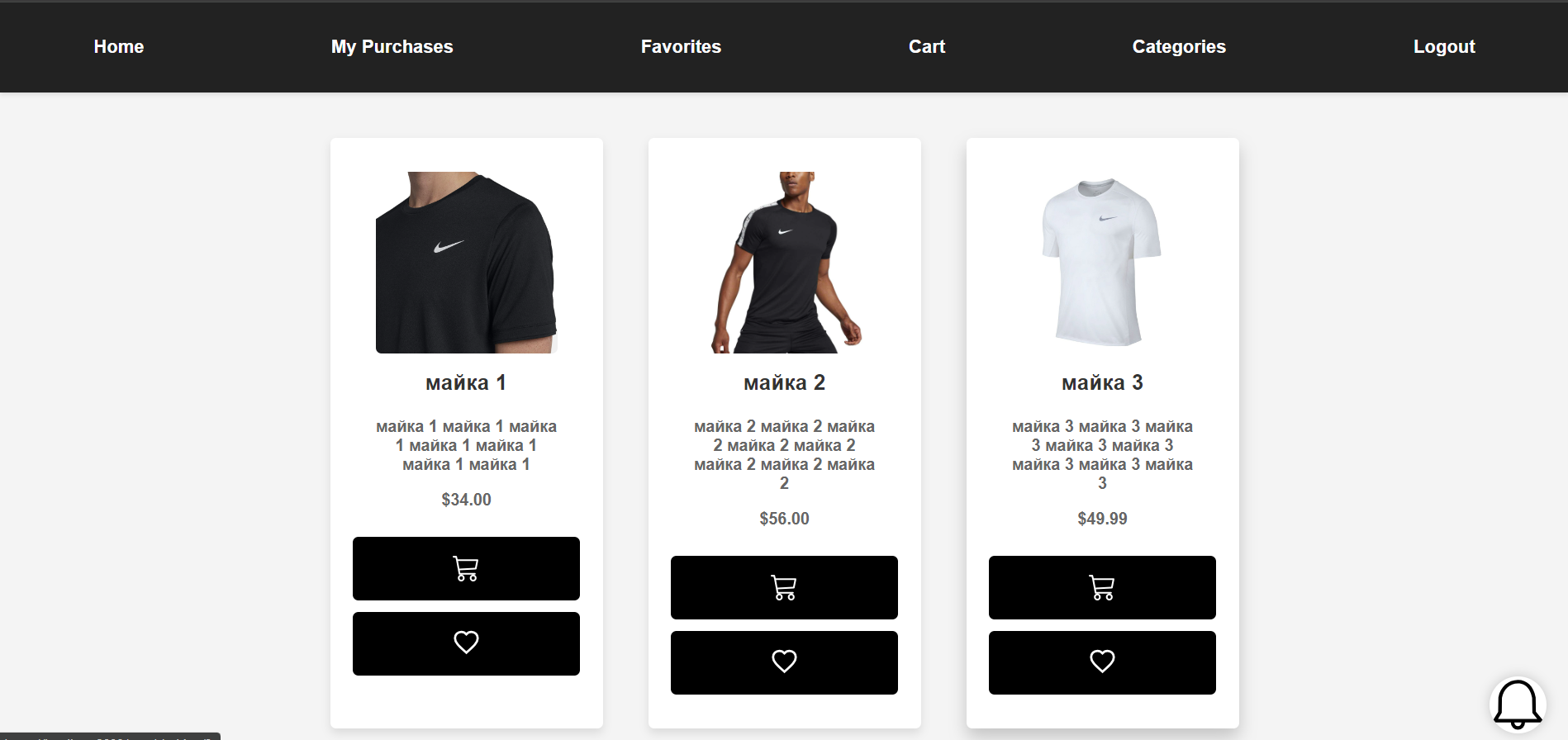


Рисунок 5.3 – Главная страница

У пользователя есть возможность отфильтровать товары по категориям. При нажатии на любу.ю категорию в которой есть товары, пользоветелю отображется отсортированный список товаров по выбранной категории. Фильтр представлен рисунке 5.4.

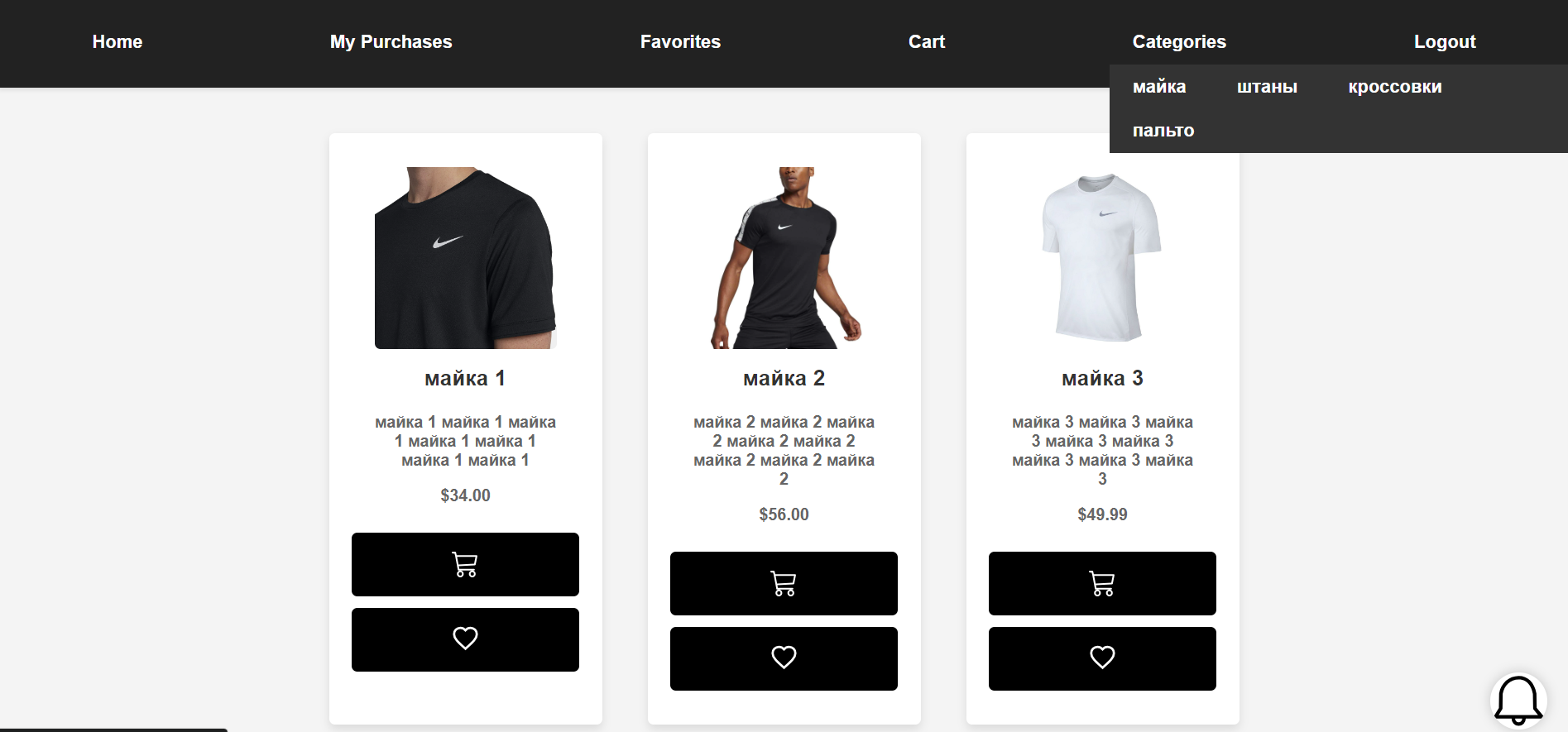


Рисунок 5.4 – Фильтрация карточек по категориям

В главном меню пользователь может нажать на фотографию и тогда он сможет увидеть карточку товара. Карточка представлена на рисунке 5.5.

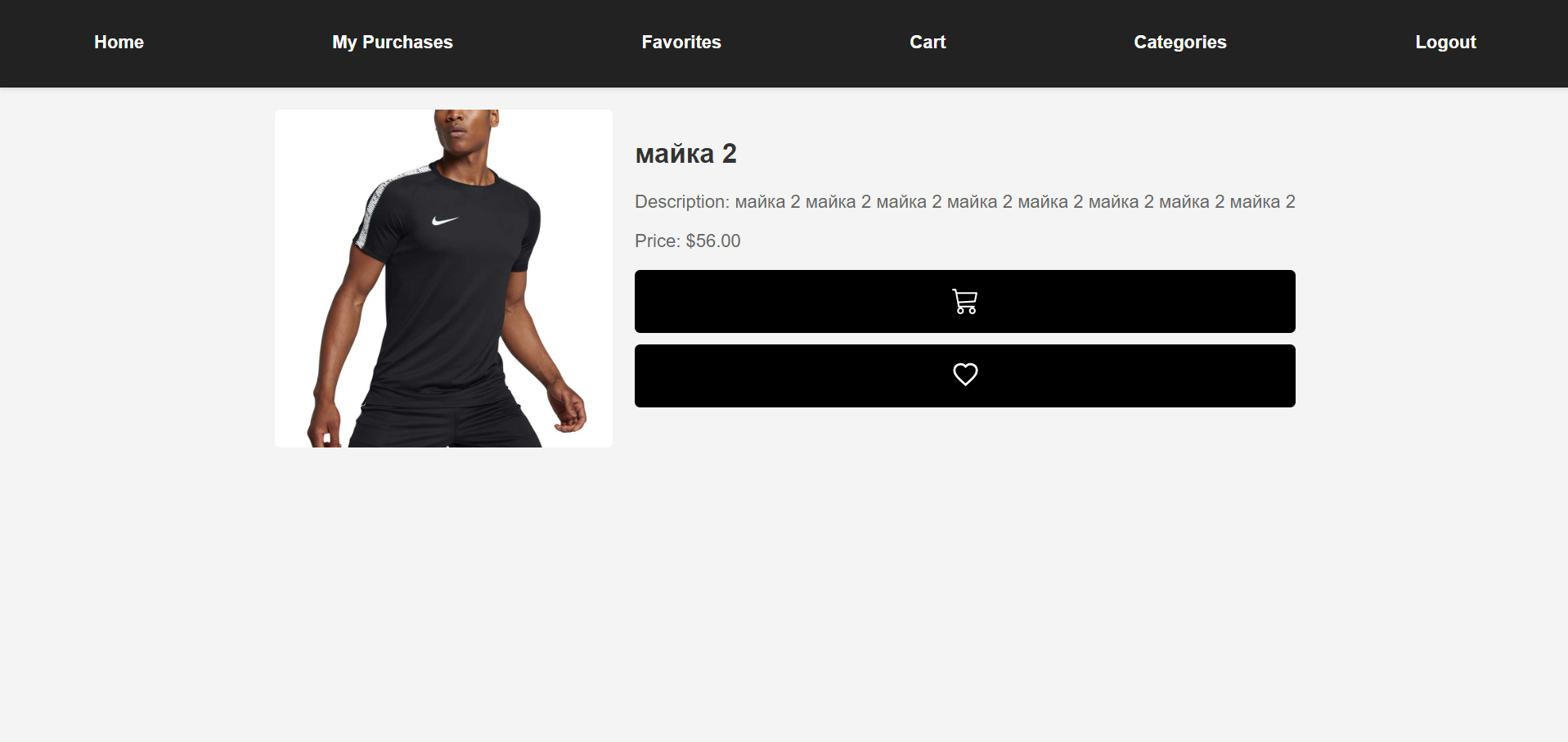


Рисунок 5.5 – Карточка товара

Для того, чтобы добавить товар в корзину, нужно на главной странице выбрать товар. Затем нужно нажать на кнопку «Добавления товар в корзину». Пример представлен на рисунке 5.6.



Рисунок 5.6 – Добавление товара в корзину

Далее, нажав на кнопку корзины в навигационном меню, пользователь может перейти в раздел «Корзина». Интерфейс раздела представлен на рисунке 5.7.

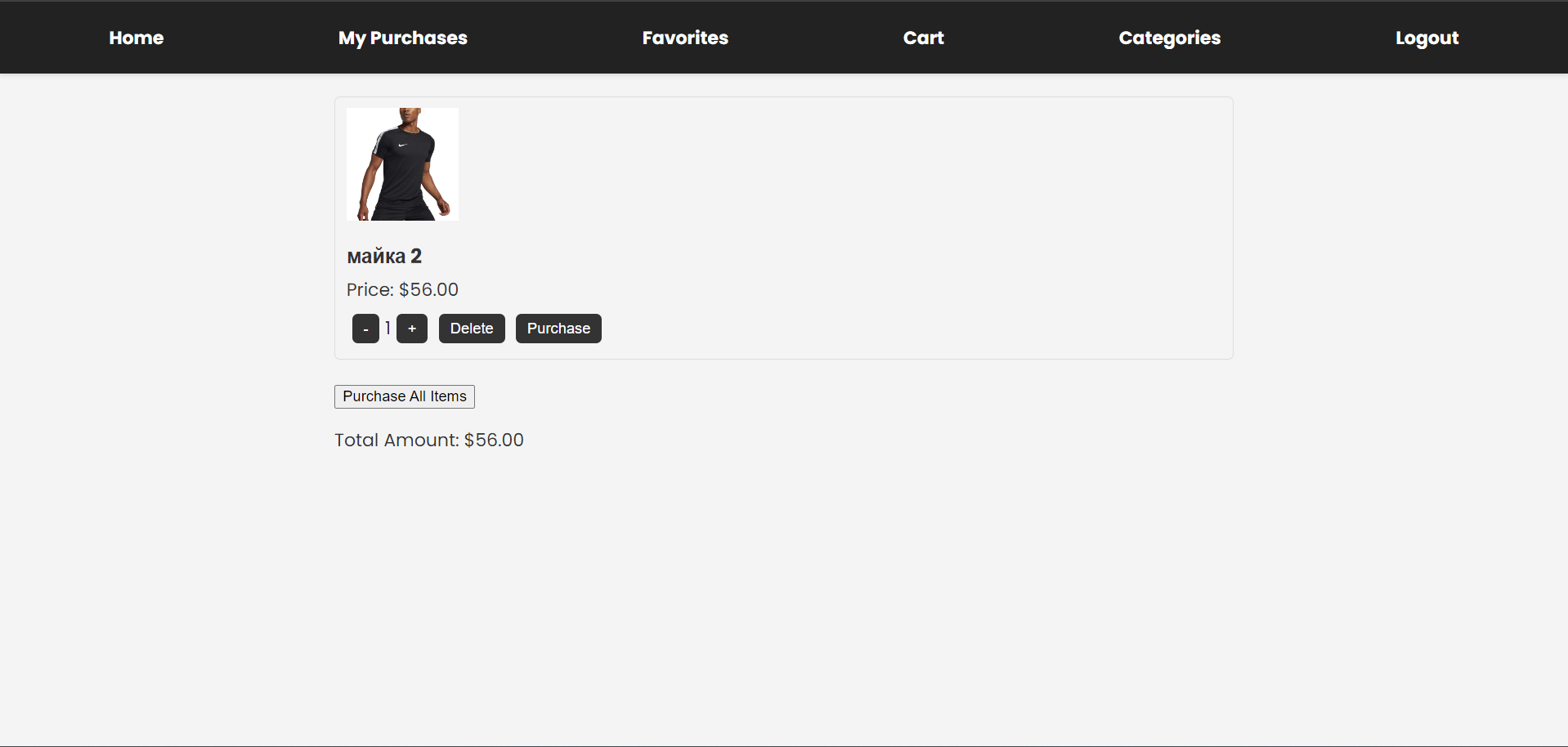


Рисунок 5.7 – Интерфейс раздела «Корзина»

В данном разделе, пользователь может изменить количество товара нажав на кнопки «-» или «+». Если количество товара достигает нуля, то товар удаляется из корзины. При изменении количества товара также меняется цена в соответствии с количеством. Нажав на кнопку «удалить», пользователь может удалить товар из корзины. Так же нажав на кнопку «Заказать» и заполнив поля формы пользоваетль оформляет заказ и получает сообщени о заказе на почту. Пример заполнения представлен на рисунке 5.8.

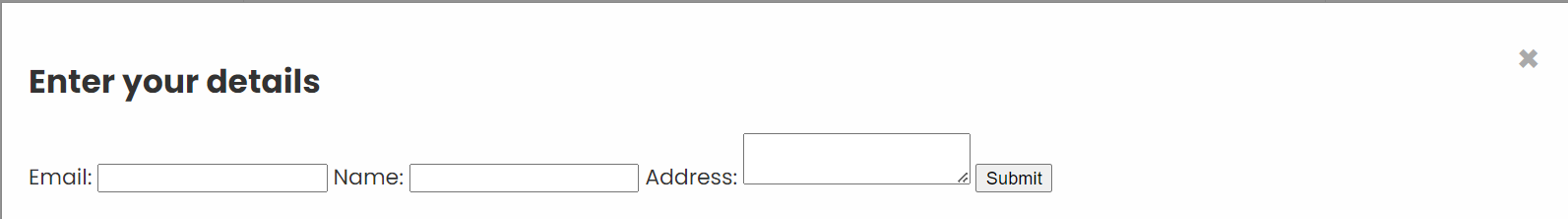


Рисунок 5.8 – Совершение заказа

Данные о заказе будут записаны в странице заказов, где он сможет увидеть свой заказ. Пример представлен на рисунке 5.9.

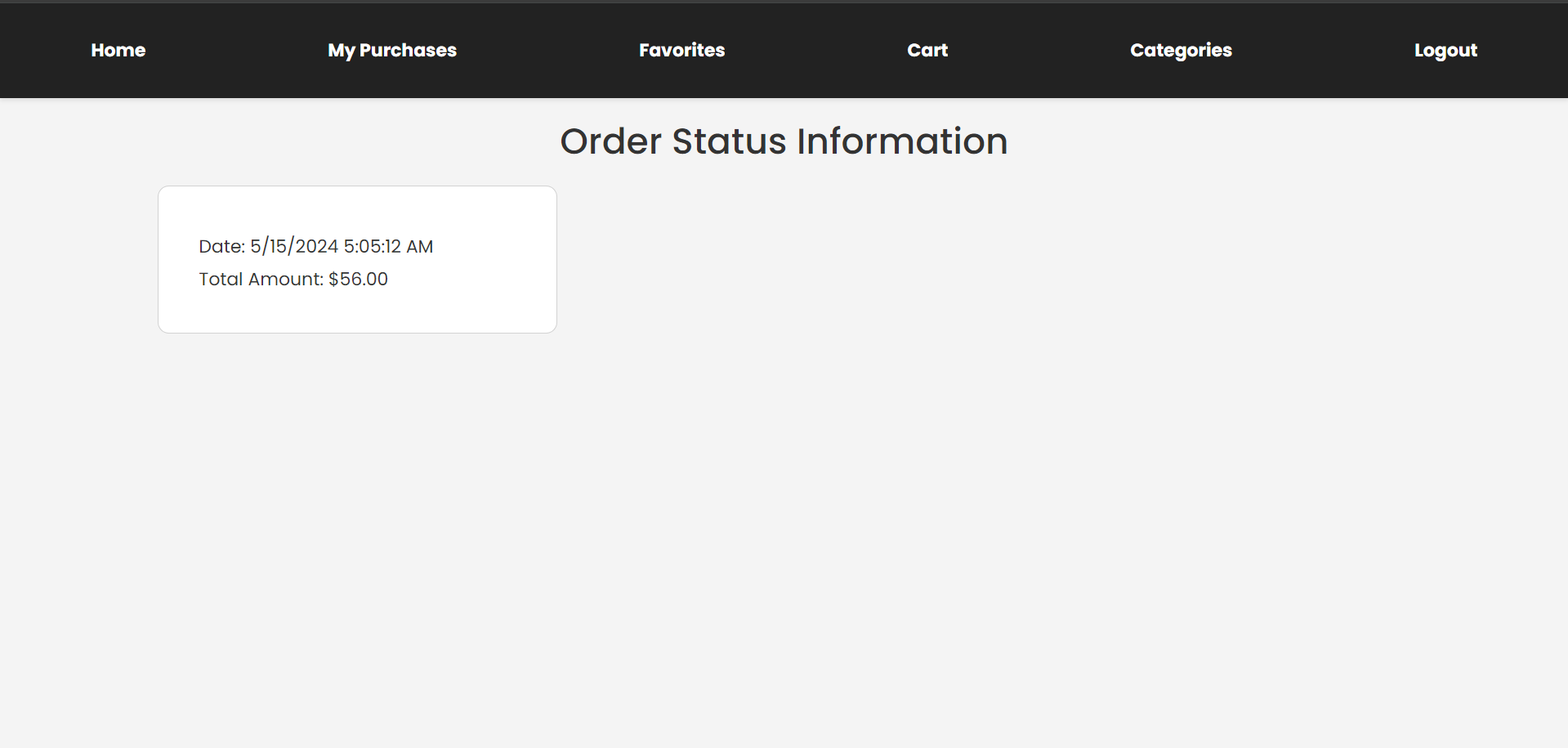


Рисунок 5.9 – Заказы

Также заказы можно отследить на странице администратора. При получении заказа, данные записыватся и отображаются на странице статистики продажю. Интерфейс панели представлен на рисунке 5.10.

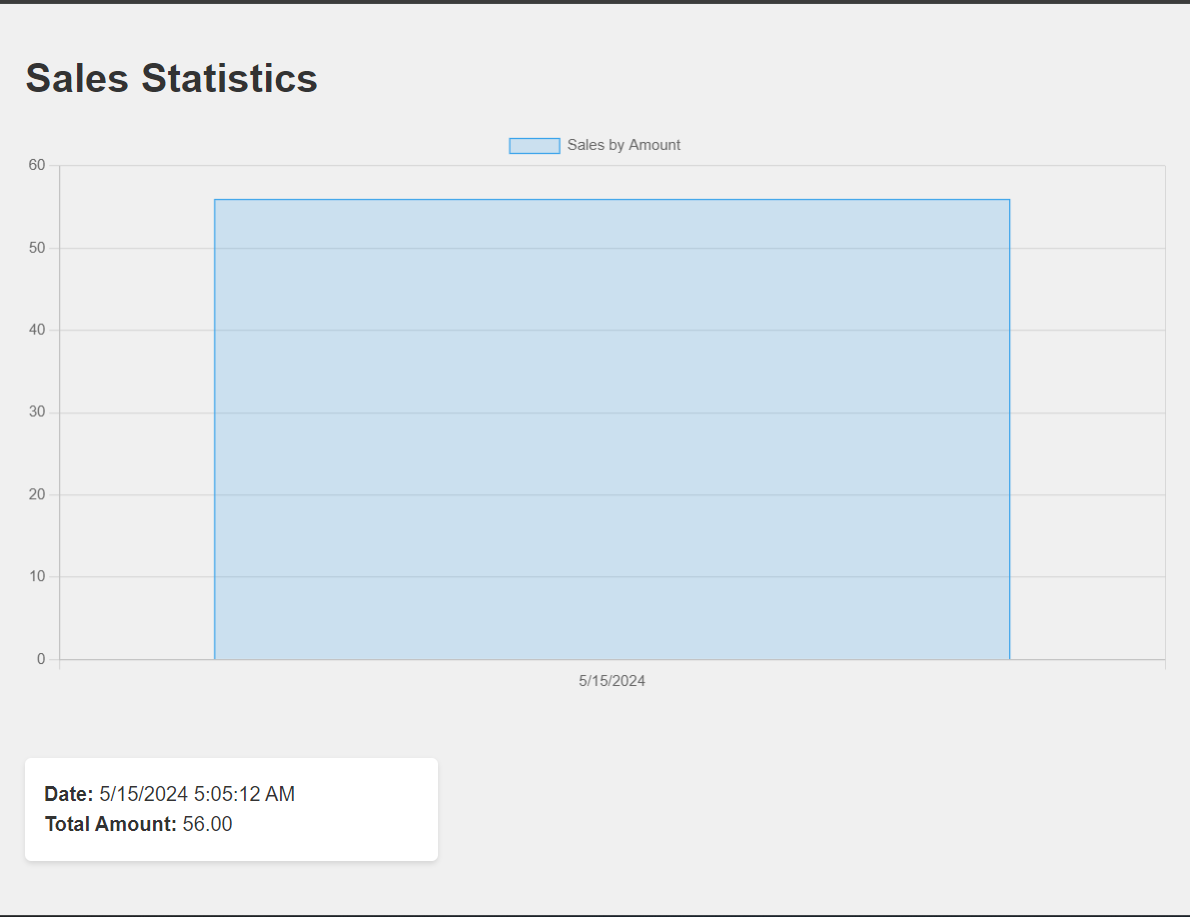


Рисунок 5.10 – Статистика продаж

Кроме того, с панели администратора добавлять товары. Примеры форм добавления карточки товара представлены на рисунке 5.11.

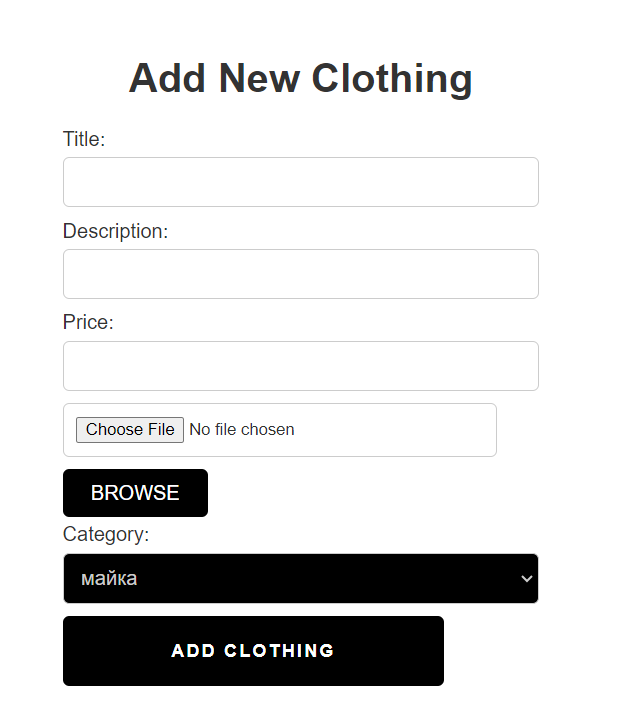


Рисунок 5.11 – Формы добавления товара

В панели администратора есть возможность изменять товары. Примеры форм изменения карточки товара представлены на рисунке 5.12.

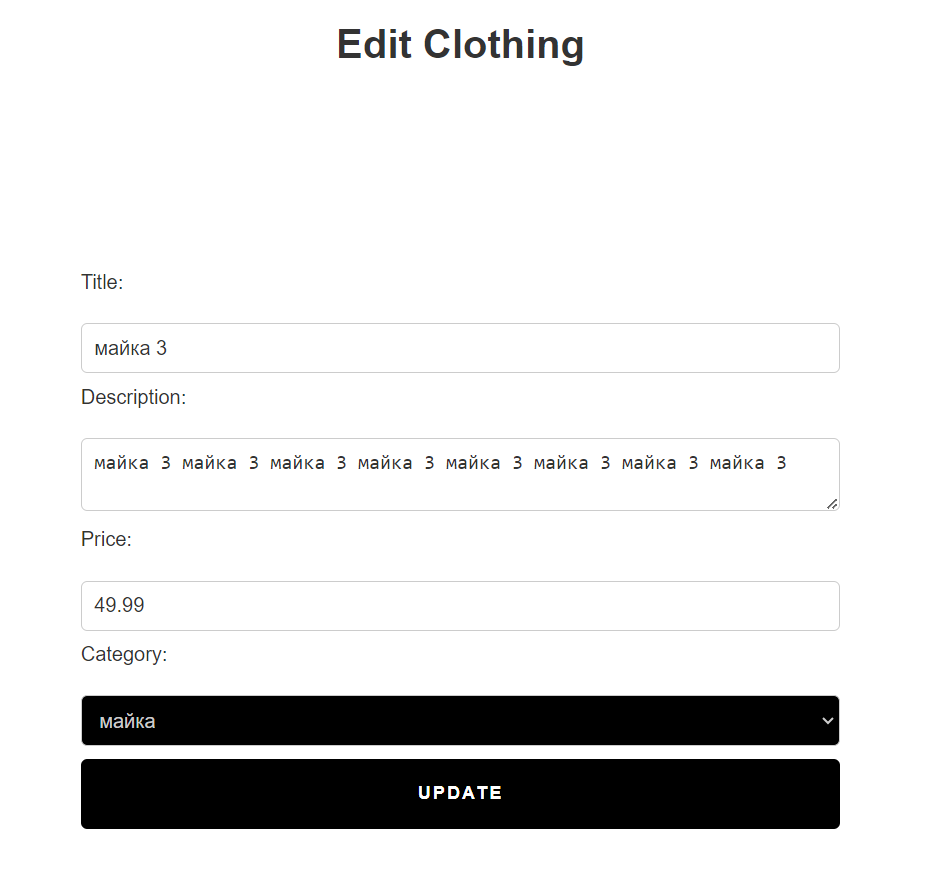


Рисунок 5.12 – Форма изменения товара

На главной странице администратора можно удаялть товар из каталога.  
Кнопка удаления представлена на рисунке 5.13.

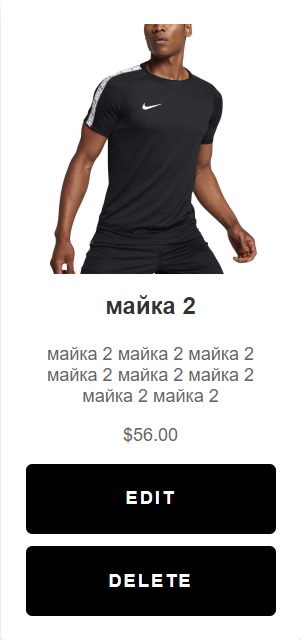


Рисунок 5.13 – Кнопка удаления карточки товара

В панели администратора есть возможность добавлть категории товаров. Примеры форм добавления категории товаров представлены на рисунке 5.14.

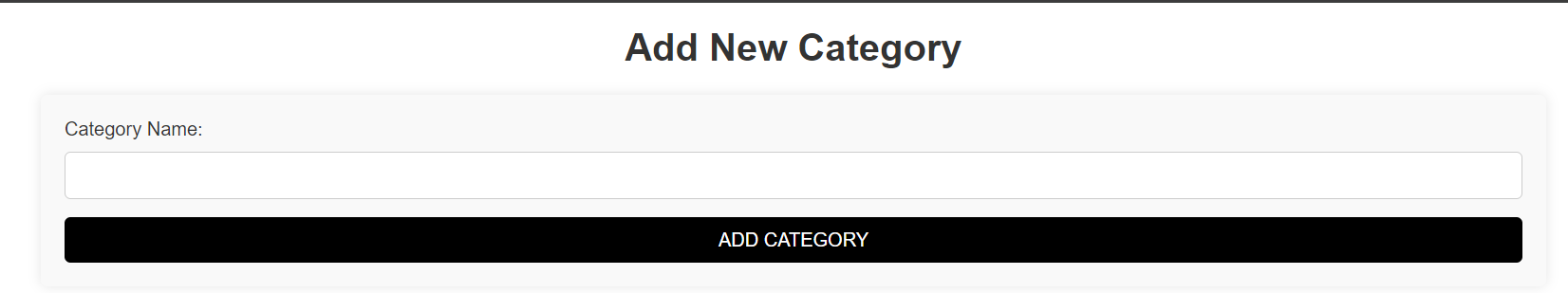


Рисунок 5.14 – Форма добавления категории

Кроме того, главный администратор также может отправлять уведомления всем пользователям. Интерфейс для работы с меню представлен на рисунке 5.15.



Рисунок 5.15 – Интерфейс для отправки уведомлений

При отправке уведомления пользоваетлям приходит сообщение, которые они могут удалить после прочнеия. Формы для просмотра уведомлений представлены на рисунке 5.16.

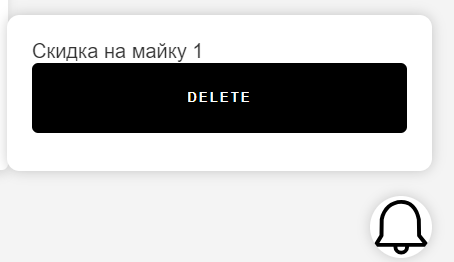


Рисунок 5.16 – Форм для просмотра уведомлений

Далее, нажав на кнопку корзины в навигационном меню, пользователь может перейти в раздел «Избранное». Интерфейс раздела избранное представлен на рисунке 5.17.

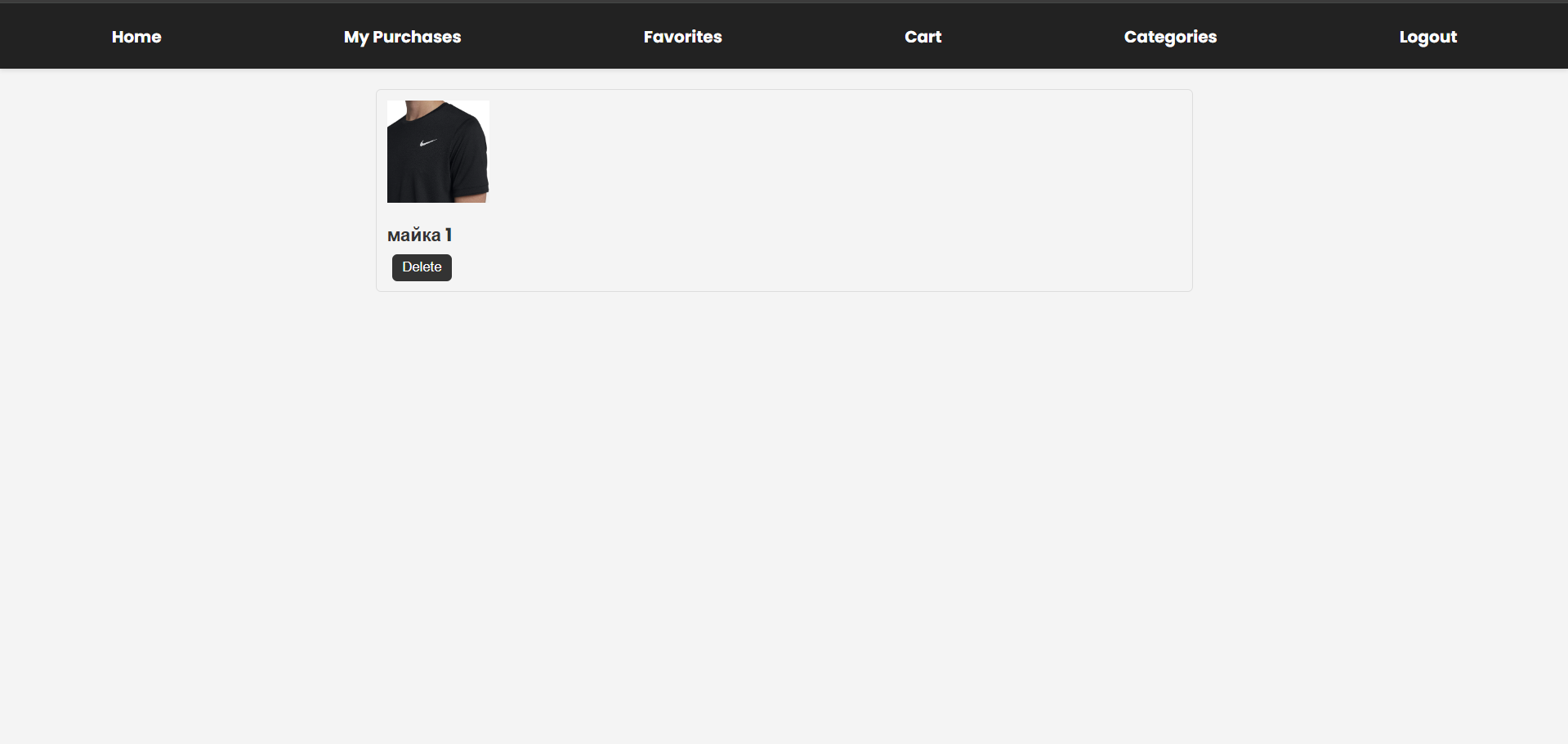


Рисунок 5.17 – Интерфейс раздела избранное

На странице корзины есть кнопка для покупки всех товаров корзины, при нажатии оформляется заказ, включающий все товары в корзине. Корзина очищается, а данные о покупке записываются в заказы. Кнопка покупки всех товаров представлена на рисунке 5.18.

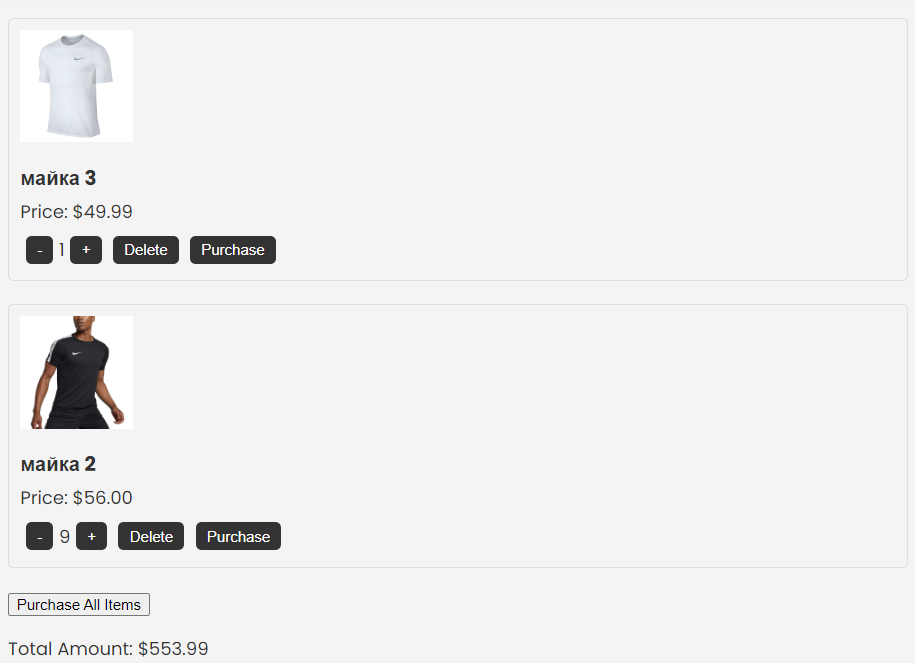


Рисунок 5.18 – Кнопка покупки всех товаров корзины

Таким образом, в данной главе было разработано руководство по использованию, которое облегчает пользователю освоение функционала приложения для интернет-магазина одежды. Пользователь сможет ознакомиться с процессом выбора и заказа товаров, а также с функционалом своей учетной записи, включая возможности отслеживания заказов и управления личной информацией.

Заключение

Результатом выполнения курсового проекта стало web-приложение, созданное с использованием серверного фреймворка Express.js, ejs-шаблонов и базы данных PostgreSQL.

Перед началом разработки был произведен аналитический обзор прототипов приложений подобной тематики, а также определение функциональных возможностей разрабатываемого приложения.

Был разработан удобный и понятный интерфейс, в котором могут разобраться даже неопытные пользователи.

В процессе разработки серверной части было реализовано API для работы со всеми необходимыми сущностями, также была реализована стратегия JWT для обеспечения безопасности.

Были реализованы следующие требования:

* обеспечивать возможность регистрации и авторизации;
* поддерживать роли администратора и пользователя;
* позволять изменять информацию о товаре;
* предоставлять возможность добавить товар в избранное;
* предоставлять возможность поиска товара по критериям;
* позволять просматривать карточки товаров;
* просмотр статистики продаж;
* предоставлять возможность добавлять товар в корзину;
* предоставлять возможность удаления товара;
* предоставлять возможность оформления заказа.

Для обеспечения безопасного обмена информации между клиентом и сервером был сгенерирован ssl-сертификат.

Важным этапом разработки было тестирование программного продукта, которое позволило выявить и исправить ошибки и недостатки в работе приложения. Тестирование позволило убедиться, что программа работает верно и соответствует требованиям.

В целом, разработанное веб-приложение представляет собой важный инструмент для современного ресторанного бизнеса, способный значительно улучшить процессы управления и обслуживания, а также повысить удовлетворенность клиентов.

Список используемых источников

1. Express 4.x API [Электронный ресурс] / StrongLoop, Inc. – Режим доступа: https://expressjs.com/en/4x/api.html. – Дата доступа: 19.03.2024.

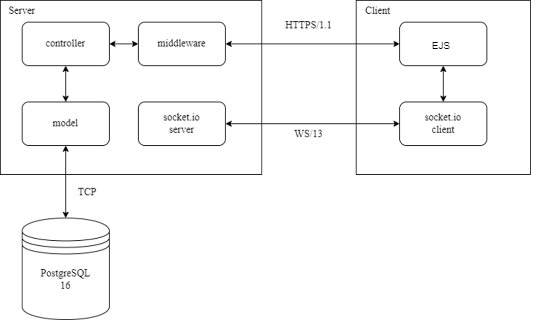
2. Socket.io документация [Электронный ресурс] / Режим доступа: https://socket.io/docs/v4/. – Дата доступа: 25.03.2024.

3. PostgreSQL документация [Электронный ресурс] / Режим доступа: https://www.postgresql.org/docs/. – Дата доступа: 15.03.2024.

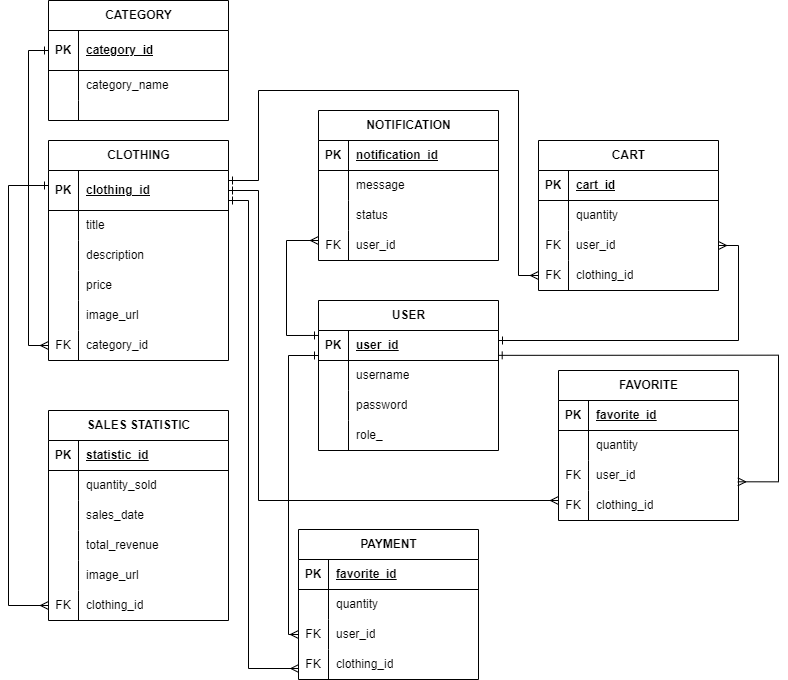
ПРИЛОЖЕНИЕ А



ПРИЛОЖЕНИЕ Б



ПРИЛОЖЕНИЕ В



ПРИЛОЖЕНИЕ Г

|  |
| --- |
| const User = sequelize.define('User', {  user\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  username: { type: DataTypes.STRING(255), allowNull: false },  password: { type: DataTypes.STRING(255), allowNull: false },  role\_: { type: DataTypes.BOOLEAN, allowNull: false }  });  const ClothingCategory = sequelize.define('ClothingCategory', {  category\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  category\_name: { type: DataTypes.STRING(50), allowNull: false }  });  const Clothing = sequelize.define('Clothing', {  clothing\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  title: { type: DataTypes.STRING(100), allowNull: false },  description: { type: DataTypes.STRING(500), allowNull: false },  price: { type: DataTypes.NUMERIC(10, 2), allowNull: false },  image\_url: { type: DataTypes.STRING(500), allowNull: true } // Ссылка на изображение товара  });  const CartItem = sequelize.define('CartItem', {  item\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  quantity: { type: DataTypes.INTEGER, allowNull: false },  image\_url: { type: DataTypes.STRING(500), allowNull: true },  title: { type: DataTypes.STRING(100), allowNull: false }  });  const FavoriteItem = sequelize.define('FavoriteItem', {  favorite\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  image\_url: { type: DataTypes.STRING(500), allowNull: true },  title: { type: DataTypes.STRING(100), allowNull: false }  });  const Notification = sequelize.define('Notification', {  notification\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  message: { type: DataTypes.STRING(255), allowNull: false },  status: {type: DataTypes.BOOLEAN, allowNull: false}  })  const Payment = sequelize.define('Payment', {  payment\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  payment\_date: { type: DataTypes.DATE, allowNull: false },  amount: { type: DataTypes.NUMERIC(10, 2), allowNull: false },  });  const SalesStatistic = sequelize.define('SalesStatistic', {  statistic\_id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },  quantity\_sold: { type: DataTypes.INTEGER, allowNull: false },  sales\_date: { type: DataTypes.DATE, allowNull: false },  total\_revenue: { type: DataTypes.NUMERIC(10, 2), allowNull: false },  image\_url: { type: DataTypes.STRING(500), allowNull: true } // Ссылка на изображение товара в статистике  });  Payment.belongsTo(User);  Payment.belongsTo(Clothing);  User.hasMany(CartItem);  CartItem.belongsTo(User);  User.hasMany(Notification);  Notification.belongsTo(User);  ClothingCategory.hasMany(Clothing);  Clothing.belongsTo(ClothingCategory);  Clothing.hasMany(CartItem);  CartItem.belongsTo(Clothing);  User.belongsToMany(Clothing, { through: FavoriteItem });  Clothing.belongsToMany(User, { through: FavoriteItem });  Clothing.hasMany(SalesStatistic);  SalesStatistic.belongsTo(Clothing); |