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 «Магазин компьютерных комплектующих»

Выполнил студент Гришин Иван Сергеевич

(Ф.И.О.)

Руководитель проекта асс. Нистюк О. А.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В.

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

Консультанты асс. Нистюк О. А.

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

Нормоконтролер асс. Нистюк О. А.

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

Курсовой проект защищен с оценкой

Минск 2022

Содержание

[Введение 3](#_Toc104122824)

[1 Постановка задачи 4](#_Toc104122825)

[2 Обзор аналогов 5](#_Toc104122826)

[3 Проектирование приложения 7](#_Toc104122827)

[4 Разработка приложения 10](#_Toc104122828)

[4.1 Разработка серверной части 10](#_Toc104122829)

[4.2 Разработка клиентской части 13](#_Toc104122830)

[4.3 Реализация основного функционала приложения 14](#_Toc104122831)

[5 Тестирование 15](#_Toc104122832)

[6 Руководство пользователя 19](#_Toc104122833)

[5.1 Руководство для неавторизованного пользователя 19](#_Toc104122834)

[5.2 Руководство для клиента 19](#_Toc104122835)

[**5.3** **Руководство администратора** 21](#_Toc104122836)

[Заключение 22](#_Toc104122837)

[Список литературы 23](#_Toc104122838)

[Приложение А 24](#_Toc104122839)

[Приложение Б 25](#_Toc104122840)

[Приложение В 27](#_Toc104122841)

# Введение

Web-приложение «Магазин компьютерных комплектующих» – это приложение, позволяющее приобрести запчасти для компьютера. Программный продукт позволяет ознакомится с техническими характеристиками компьютерных комплектующих, производить поиск необходимых деталей, оформлять заказ на покупку деталей.

Данное Web-приложение реализовано на платформе Node JS [1]. Серверная часть приложения написана с помощью фреймворка Express JS [2]. Для пользователя доступно React-приложение с асинхронным UI [3]. Управление сайтом интуитивно, понятно и удобно.

В качестве СУБД для базы данных была выбрана MySql[4], в связи с ее простотой, производительностью и надежностью.

Node.js – программная платформа, основанная на движке V8. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, подключать другие внешние библиотеки. Node.js применяется преимущественно на сервере. В основе Node.js лежит событийно-ориентированное и асинхронное программирование.

Npm – менеджер пакетов, входящий в состав Node.js, позволяющий расширить возможности приложения.

# 1 Постановка задачи

Для выполнения данного программного средства были поставлены следующие задачи:

1. Обзор литературы по теме, примеры похожих программных средств, анализ их с критической стороны.
2. Проектирование, создание логических схем работы программного средства.
3. Программирование, отладка модулей проекта, сборка и комплексная отладка. Описание классов, атрибутов и методов.
4. Проведение тестирования программного средства.
5. Разработка руководства по установке и использованию приложения.

Решение каждой из задач отображено соответственно в главах:

1. Обзор аналогов.
2. Проектирование приложения.
3. Разработка приложения.
4. Тестирование
5. Руководство пользователя.

Помимо этого, программное средство должно иметь HTTPS-сервер и WebSocket-сервер. Иметь реляционную базу данных с достаточным количеством таблиц, чтобы полностью реализовать функционал интернет-магазина. Также должны быть регистрация, аутентификация, авторизация.

**2 Обзор аналогов**

«5 Элемент» — белорусская сеть магазинов электроники и бытовой техники. Основана в 1994 году. По состоянию на ноябрь 2020 года сеть насчитывает 70 розничных магазинов. Первый магазин открылся в Минске и был известен под названием BOSCH. После открытия четвертого магазина в Могилеве сеть приобретает известное всем название «5 Элемент».

«5 Элемент» предоставляет возможность оформления заказов, покупки электроники и бытовой техники.

Для зарегистрированных пользователей существует опция добавления товаров в избранное, в корзину или же в список для сравнения товаров. Сайт имеет понятный интерфейс, однако огромное число кнопок может существенно повлиять на скорость поиска необходимых товаров у неопытных пользователей сети интернет.

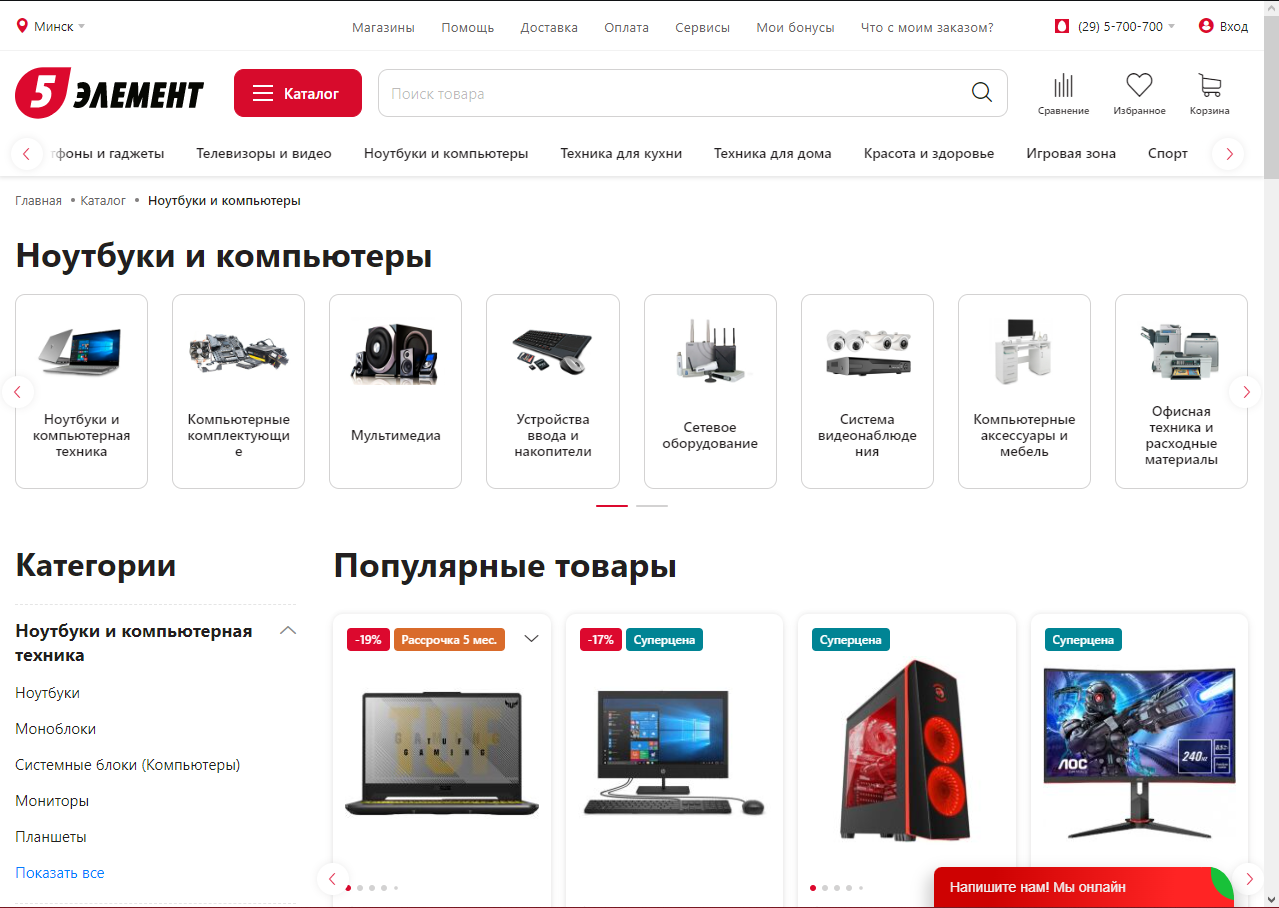


Рисунок 2.1 – Интерфейс сайта «5 элемент»

«21vek» — интернет-гипермаркет, специализирующийся на продаже электроники и бытовой техники, товаров для дома и прочего. Он является одним из лидеров интернет-порталов в Республике Беларусь. Бизнес-модель 21vek характеризуется как интернет-гипермаркет с более чем 450 тысячами позиций в каталоге.

Для авторизированных пользователей сайт позволяет выбирать и покупать товары. Ставить оценку и оставлять отзывы для конкретных продуктов. Есть корзина, списки для сравнения, личный кабинет пользователя, а так же есть онлайн-чат с консультантами.

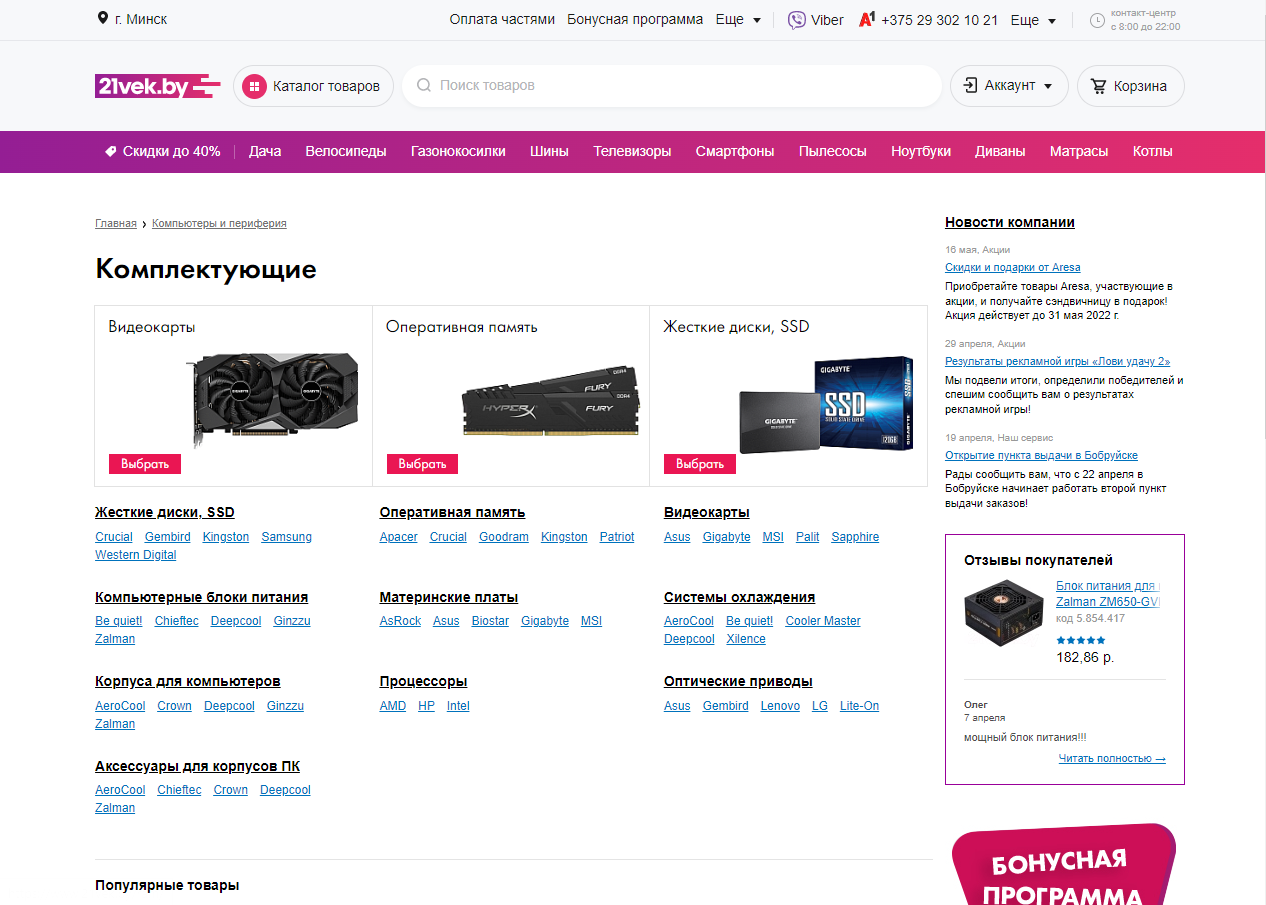


Рисунок 2.2 – Интерфейс сайта «21vek»

Проанализировав программы на схожую тематику, были разработаны технические требования к разработке веб-приложения. Приложение должно:

* Иметь 2 типа пользователей (администратор и клиент);
* иметь интуитивно понятное и простое управление;
* обращаться к базе данных, успешно извлекать, изменять и дополнять данные, осуществлять поиск по базе и автоматическую запись некоторых полей;
* иметь различный функционал в зависимости от роли;
* выглядеть аккуратно, не иметь лишней информации;
* быстро отвечать на запросы.

# 3 Проектирование приложения

Для реализации разрабатываемого проекта была выбрана технология Node JS, платформа Express и систему управления базами данных MySql.

Взаимодействие серверной части приложения с реляционной базой данных осуществляется с помощью ORM-библиотеки Sequelize[5]. Благодаря данному подходу появилась возможность сопоставлять таблицы в базе данных и отношения между ними с классами.

Схема базы данных для разрабатываемого программного средства отображена в приложении А. Рассмотрим её подробнее.

Схема отображает таблицы базы данных и связи между ними.

Как видно из схемы, в приложении используются 13 таблиц:

Таблица Users содержит данные пользователя для его аутентификации, авторизации и идентификации. Она включает в себя следующие поля:

* ID – уникальный идентификатор записи;
* TYPE – тип пользователя;
* LOGIN – логин пользователя в приложении;
* PHONE – номер телефона пользователя;
* PASSWORD – пароль пользователя;
* FULL\_NAME – имя пользователя.

Таблица TOKENS содержит в себе токены для входа:

* ID – уникальный идентификатор токена;
* REFRESH\_TOKEN – обновленный токен;
* USER\_ID – идентификатор пользователя.

Таблица ATTRIBUTES содержит в себе все данные об атрибутах продуктов:

* ID – уникальный идентификатор атрибута;
* NAME – имя атрибута;
* ATRIBUTE\_ID – идентификатор атрибута из таблиц с атрибутами.

Таблица ATTRIBUTE\_OPTIONS является связующей для продуктов и атрибутов:

* ID – уникальный идентификатор связи;
* PRODUCT\_ID – идентификатор продукта из таблицы PRODUCTS;
* ATTRIBUTE\_ID – идентификатор атрибута из таблицы ATTRIBUTES.

Таблица ATTRIBUTE\_SELECTS содержит данные конкретного типа атрибутов:

* ID – уникальный идентификатор атрибута;
* PRODUCT\_ID – идентификатор продукта из таблицы PRODUCTS;
* OPTION\_ID – идентификатор связи из таблицы ATTRIBUTE\_OPTIONS.

Таблица ATTRIBUTE\_CHECKBOXES содержит данные конкретного типа атрибута:

* ID – уникальный идентификатор связи;
* VALUE – значение атрибута;
* PRODUCT\_ID – идентификатор продукта из таблицы PRODUCTS;
* ATTRIBUTED\_ID – идентификатор атрибута из таблицы ATTRIBUTES.

Таблица ATTRIBUTE\_NUMBERS содержит данные конкретного типа атрибута:

* ID – уникальный идентификатор связи;
* VALUE – значение атрибута;
* PRODUCT\_ID – идентификатор продукта из таблицы PRODUCTS;
* ATTRIBUTE\_ID – идентификатор атрибута из таблицы ATTRIBUTES.

Таблица ATTRIBUTE\_TEXTS содержит данные конкретного типа атрибута:

* ID – уникальный идентификатор связи;
* VALUE – значение атрибута;
* PRODUCT\_ID – идентификатор продукта из таблицы PRODUCTS;
* ATTRIBUTE\_ID – идентификатор продукта из таблицы ATTRIBUTES.

Таблица BRANDS содержит в себе все данные о брендах, а именно:

* ID – уникальный идентификатор бренда;
* NAME – название бренда;
* DELETED – проверка, удалён ли бренд.

Таблица CATEGORIES содержит в себе все данные о категориях, а именно:

* ID – уникальный идентификатор категории;
* NAME – название категории;
* DELETED – проверка, удалена ли категория.

Таблица PRODUCTS содержит в себе все данные о продуктах, а именно:

* ID – уникальный идентификатор продукта;
* NAME – название продукта;
* PRICE – цена продукта;
* IMAGE – изображение продукта;
* DELETED – проверка, удалён ли продукт.
* CATEGORY\_ID – идентификатор категории из таблицы CATEGORIES.
* BRAND\_ID – идентификатор бренд из таблицы BRANDS.

Таблица PRODUCT\_ORDERS содержит в себе все данные товарах в заказе, а именно:

* ID – уникальный идентификатор заказа;
* PRICE – цена;
* QUANTITY – количество товаров в заказе;
* PRODUCT\_ID – идентификатор продукта из таблицы PRODUCTS.
* ORDER\_ID – идентификатор заказа из таблицы ORDERS.

Таблица ORDERS содержит в себе все данные о заказах, а именно:

* ID – уникальный идентификатор заказа;
* NUMBER – номер заказа;
* STATUS – статус заказа;
* USER\_ID – идентификатор пользователя из таблицы USERS.

Помимо этого, в каждой таблице есть столбца CREATED\_AT и UPDATED\_AT для отслеживания даты и времени изменение информации. Также, в таблицах PRODUCTS, CATEGORIES и BRANDS используется столбец DELETED для того, чтобы при удалении соответствующих брендов, категории или продукта из продажи, не пропадали уже имеющиеся заказы, находящиеся в статусе ожидания.

В разрабатываемом программном средстве существует возможность добавления, изменения и удаления записей в базе данных. Разработано REST API благодаря которому можно взаимодействовать с базой данных не только через web-приложение, но и посредством обычных HTTP/HTTPS запросов.

База данных к приложению подключается c помощью параметров, приведенных в листинге 3.1. Const sequelize вызывает метод Sequelize с параметрами, содержащими имя базы данных, имя пользователя и пароль.

const sequlize = new Sequelize(‘online\_store’,’root’,’dfyef2001’,{

host: ’localhost’,

dialect: ’MySQL’,

pool: {

max:10,

min:0,

acquire:30000,

idle:10000,

},

logging: false,

});

Листинг 3.3 – Подключение к базе данных

# 4 Разработка приложения

## 4.1 Разработка серверной части

На рисунке 4.1 отображена структура серверной части приложения

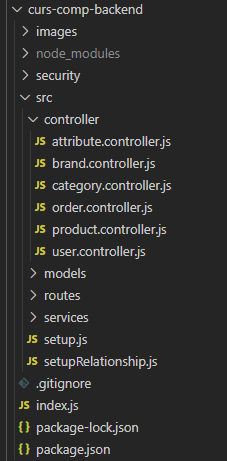


Рисунок 4.1 – Структура проекта сервера

Проект поделен на 4 раздела: «controller», «models», «routes», «services».

В директории «controllers» содержатся контроллеры. Контроллеры обеспечивают взаимодействие пользователя и системы, обрабатывая и проверяя запросы. Для формирования ответов они используют обработанные в блоке «services» данные из БД.

Пример одного из методов контроллера, а именно регистрация пользователя, приведен на листинге 4.2.

async registerUser(req, res) {

try {

const { login, phone, password, fullName } = req.body;

const findUser = await User.findOne({ where: { login } });

if (findUser) {

throw new Error('Такой пользователь уже существует');

}

const hashPassword = await bcrypt.hash(password, 3);

const user = await User.create({ login, phone, fullName, password: hashPassword, type: 'client' });

const accessToken = jwt.sign({ id: user.id, login: user.login, phone: user.phone, fullName: user.fullName, type: user.type }, 'access-secret', { expiresIn: '24h' });

return res.json({ token: accessToken });

} catch (error) {

res.status(500).send({

message: error.message || 'Непредвиденная ошибка',

});

}

Листинг 4.2 –Регистрация пользователя

Директория «models» содержит файлы с описанием всех таблиц и связей между ними в базе данных. Пример на листинге 4.3

module.exports = (sequelize, Sequelize) => {

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

name: {

type: Sequelize.STRING,

unique: true,

allowNull: false,

},

price: {

type: Sequelize.FLOAT,

allowNull: false,

defaultValue: 0,

},

image: {

type: Sequelize.STRING,

},

deleted: {

type: Sequelize.BOOLEAN,

defaultValue: false,

},

});

return Product;

};

Листинг 4.3 – Описание таблицы «PRODUCTS»

Директория «routes» представляет собой файлы с маршрутизацией запросов. Все запросы разделены по соответствующим таблицам, над которыми выполняются операции. Пример маршрутизатора приведен на листинге 4.4

const Router = require('express');

const router = new Router();

const brandController = require('../controller/brand.controller');

router.post('/brand/create', brandController.createBrand);

router.post('/brand/update', brandController.updateBrand);

router.post('/brand/delete', brandController.deleteBrand);

router.get('/brand/list', brandController.getBrands);

module.exports = router;

Листинг 4.4 – Маршрутизатор запросов к таблице «BRANDS»

Директорий «security» содержит ключ и сертификат для поддержки HTTPS-запросов.

## Разработка клиентской части

Клиентская часть проекта написана с использованием фреймворка «React». Фреймворк позволяем удобно сочетать язык JavaScript и декларативный подход при написании разметки и стилей. На рисунке 4.5 отображена структура проекта клиентской части.

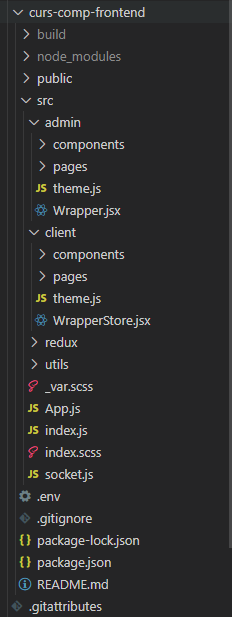


Рисунок 4.5 – Структура клиентской части приложения

Проект поделен на 2 раздела: «client», «admin», которые предоставляют разный функционал в зависимости от тип пользователя. Каждый из разделов содержит 2 основных директория: «components», «pages», содержащие react-компоненты и страницы приложения соответственно.

Клиентская часть реализует асинхронный подход к реализации пользовательского интерфейса.

На листинге 4.7 представлена структура раздела администратора.

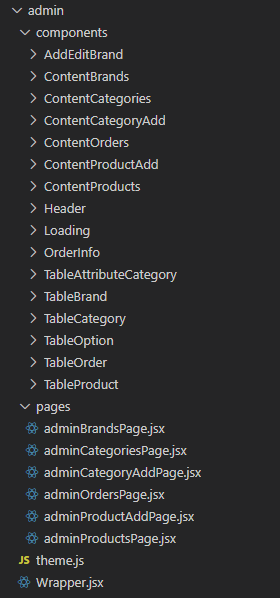


Рисунок 4.6 – Структура раздела администратора

## Реализация основного функционала приложения

Каждый слой приложения реализуют асинхронную систему вызовов. Слой работы с базой данных вызывает асинхронные методы для выполнения sql-команд, а верхние слои дожидаются их исполнения. Данная модель позволяет параллельную работу нескольких пользователей без ожидания завершения операций других пользователей.

Для обеспечения защиты данных во время процесса пользования приложением был использован протокол https. Для этого был создан и зарегистрирован сертификат, который впоследствии станет основой для https сервера.

В ходе реализации приложения было разработано REST API для предоставления гибкого интерфейса взаимодействия с приложением. Таким образом работа сервера не привязана к клиентской части. Клиентская часть может быть реализована на любом языке, любой платформе и любым желающим. Благодаря такой архитектуре можно реализовать свое собственное приложение и через него полноценно получать информацию о компьютерных комплектующих.

# 5 Тестирование

В разработанном программном средстве предусматриваются многие негативные сценарии.

Создавая новую учетную запись, предназначенную для помещения в базу данных, при незаполненных обязательных полях кнопка регистрации будет заблокирована, пример на рисунке 5.1.

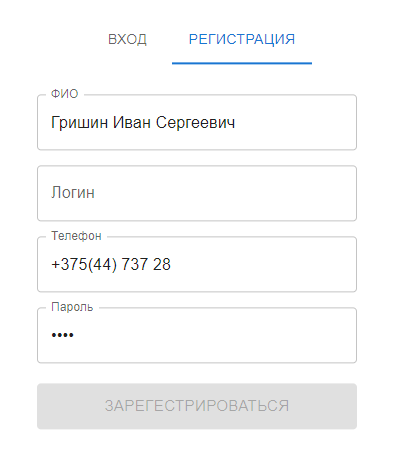


Рисунок 5.1 – Блокировка кнопки регистрации

При попытке зарегистрировать пользователя с логином, который уже имеется в базе данных, выводится сообщение, отображенное на рисунке 5.2.

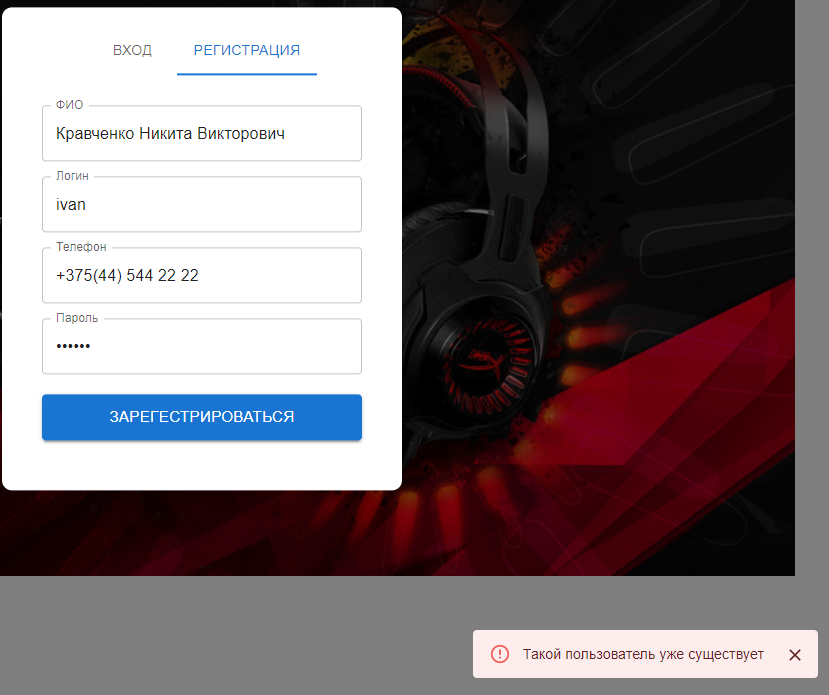


Рисунок 5.2 – Сообщение о существующем пользователе

Если пользователь уже зарегистрирован, он может нажать на кнопку «вход». После нажатия откроется форма авторизации в приложение, показанная на рисунке 5.3, где он может ввести свой логин и пароль для получения доступа к дополнительным возможностям приложения.

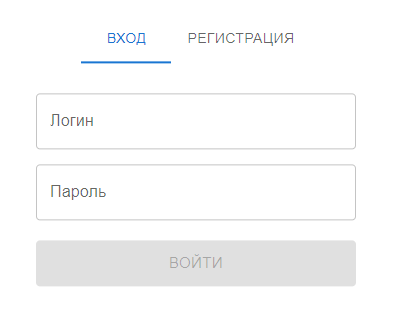


Рисунок 5.3 – Форма авторизации в приложение

Если при входе в систему указываются неверные данные, выводится сообщение об ошибке. Пример сообщения продемонстрирован на рисунке 5.4.

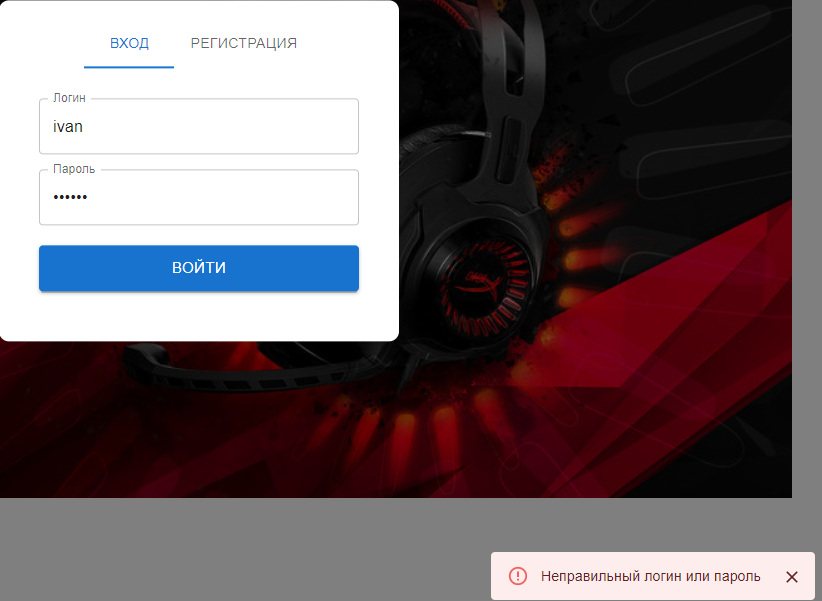


Рисунок 5.4 – Сообщение о неправильно введенных данных пользователя

При попытке неавторизованного пользователя поставить рейтинг фильму, произойдет перенаправление на страницу авторизации.

Клиентская часть приложения предусматривает роль пользователя и ограничивает некоторый функционал. Например, у неавторизованного пользователя отсутствует текстовое поля для написания комментария и кнопки отправки комментария, а вместо меню пользователя кнопки входа и регистрации. Демонстрация представлена на рисунке 5.7. Кроме того, если пользователь не авторизован, ему недоступно оценивание фильмов, редактирование и удаление комментариев, добавление личного аватара.

Несмотря на ограничения, неавторизованный пользователь всё же имеет доступ к основному функционалу приложения – получения информации о фильмах. Пользователь может просмотреть список фильмов, прочитать информацию о каждом из списка, просмотреть трейлер, прочитать комментарии.

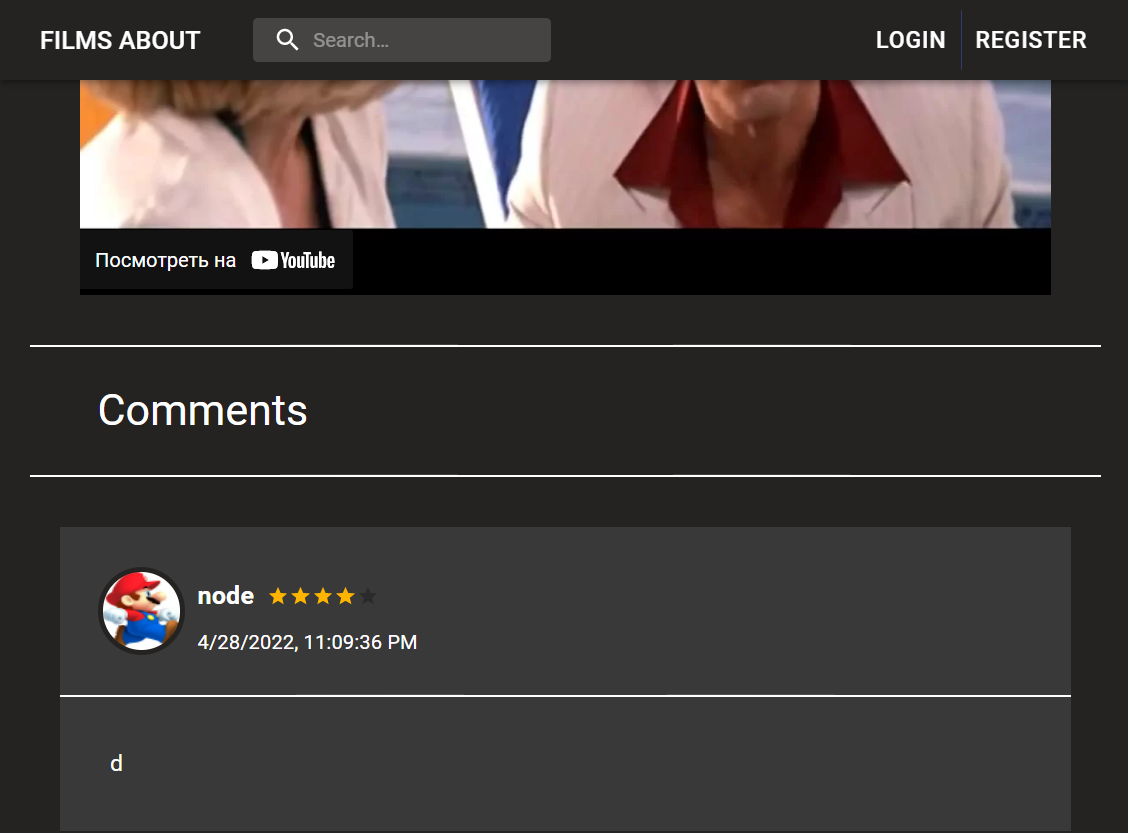


Рисунок 5.7 – Интерфейс неавторизованного пользователя

# 6 Руководство пользователя

Для корректной работы приложения необходимо наличие компьютера со всеми необходимыми для его работы компонентами (процессором, видеокартой, оперативной памятью, монитором и т.д.), браузер-клиент, а также подключение к сети Internet.

Фактически, установка для разработанного в данном курсовом проекте приложения не требуется. Если необходимо запустить приложение в локальной сети, нужно установить конфигурацию приложения, в частности скачать все используемые пакеты из npm. Также необходимо установить MySQL и создавать пользователя с базой данных. Все данные, необходимые для работы базы данных создаются автоматически при запуске серверной части приложения.

Чтобы иметь возможность пользоваться основным функционалом приложения, необходимо создать аккаунт администратора вручную используя скрипты в MySQL.

При входе в аккаунт, необходимо ввести данные, указанные при регистрации.

## Руководство для неавторизованного пользователя

В разработанном веб-приложении пользователь является неавторизованным, если токен, хранящийся в localstorage браузера, не является действующим, или же если токена вообще нет.

Функционал неавторизованного пользователя представляет собой авторизацию, регистрацию, выбор товаров, добавление в корзину.

Для создания аккаунта необходимо ввести адрес электронной почты и пароль, а также его подтверждение, при этом логин и адрес почты должны быть уникальными, а пароль должен быть не короче 6 символов.

Пользователь также может войти в созданный аккаунт, заполнив поля на форме аутентификации. Чтобы аутентификация прошла успешно, пользователю необходимо указать логин и пароль, которые были указаны при регистрации.

Если неавторизованный пользователь совершит попытку выполнить действия или открыть страницы, которые доступны только авторизованным пользователям, запрос будет перенаправлен на страницу авторизации.

## Руководство для клиента

После аутентификации, у пользователя появляется возможность комментировать и оценивать фильмы.

Комментарии служат как средство высказывания впечатлений о фильме. Они очень важны, чтобы пользователь, который только ищет фильм для просмотра мог сложить первоначальное мнение о том, стоит ли ему смотреть тот или иной фильм. Чтобы оставить комментарий необходимо зайти на страницу фильма, написать текст комментария и нажать на кнопку «Submit». Демонстрация процесса комментирования представлена на рисунке 6.1.

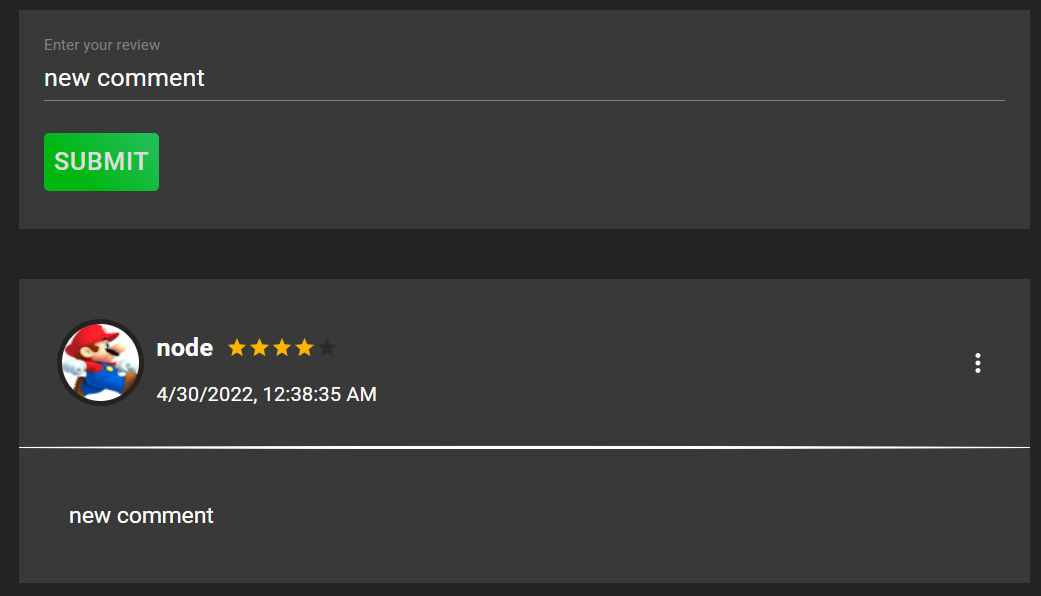


Рисунок 6.1 – Комментирование фильма

Первое, что бросается в глаза пользователя после постера фильма ­­– его рейтинг. Рейтинг фильма считается как среднее арифметическое всех пользовательских оценок. Для оценки фильма необходимо перейти на страницу фильма и нажать на одну из звезд. Демонстрация на рисунке 6.2.



Рисунок 6.2 – Оценка фильма

Авторизованный пользователь может редактировать информацию о своём аккаунте. Ему доступно изменение пароля, а также изменение аватара. Аватар – изображение, которое показывается рядом с комментарием и которое видят все пользователи. Страница пользовательских настроек приведена на рисунке 6.3.

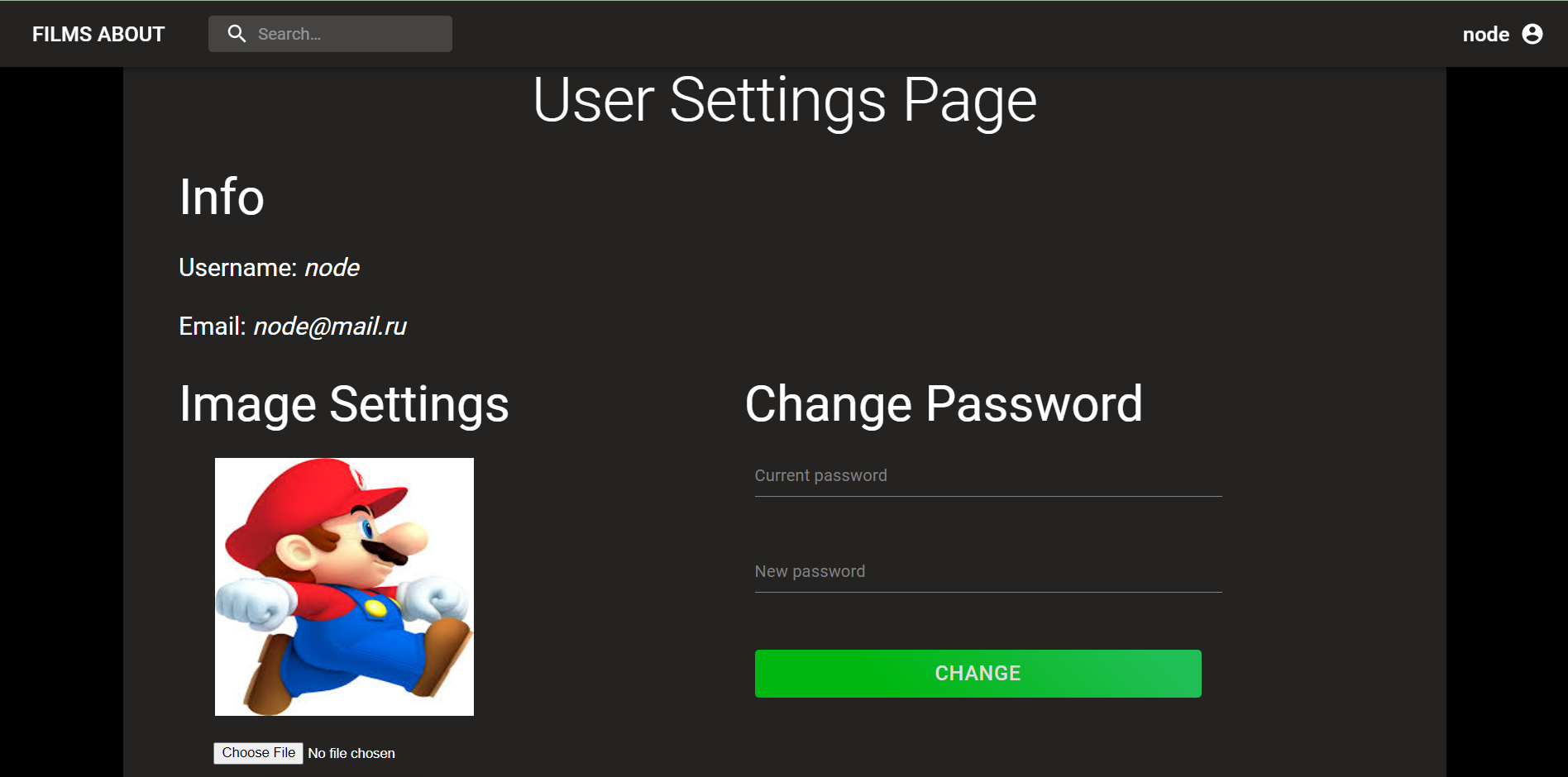


Рисунок 6.3 – Страница пользовательских настроек

Оставленные комментарии можно редактировать и удалять. Для этого необходимо нажать на иконку «вертикальное многоточие», после чего появится меню с доступными действиями. Меню приведено на рисунке 6.4

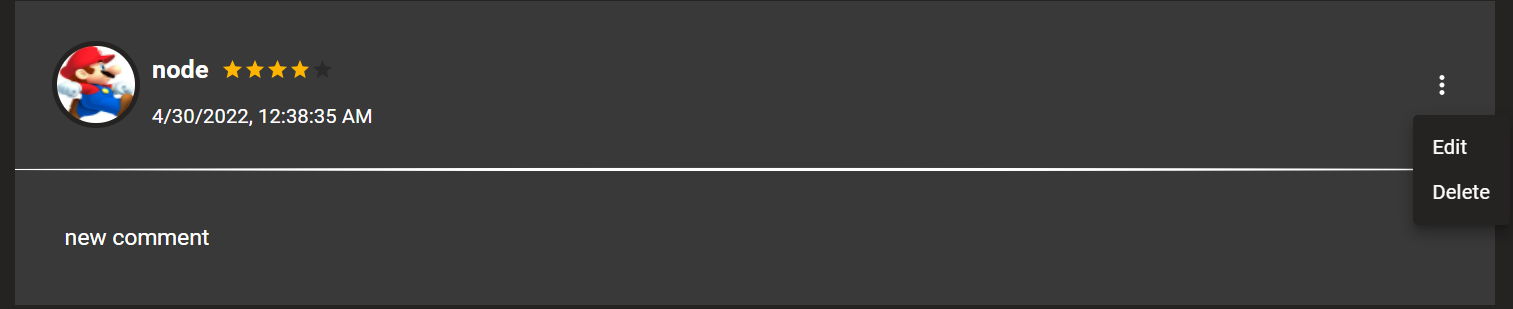


Рисунок 6.4 – Меню действий комментария

Нажатие на кнопку «Edit» открывает модальное окно в котором можно написать новый текст комментария, а кнопка «Delete» удаляет комментарий для всех пользователей.

* 1. **Руководство администратора**

После аутентификации, у администратора появляется возможность войти в режим администрирования. Для этого необходимо зайти в личный кабинет, аналогичным способ как и для клиента.

После чего появляется возможность добавления новых категорий, брендов и товаров в магазине. Демонстрация процесса создания новой категории представлен на рисунке 6.1.

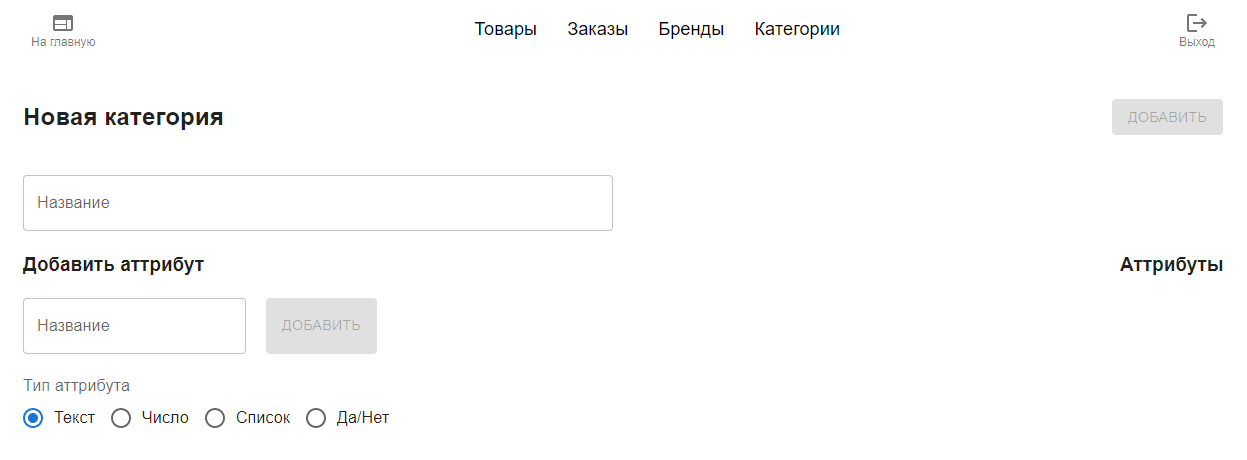


Рисунок 6.1 – Создание новой категории

Здесь администратор выбирает название новой категории, а также необходимые для неё атрибуты. После создание категории, её также можно изменить. Демонстрация на рисунке 6.2.

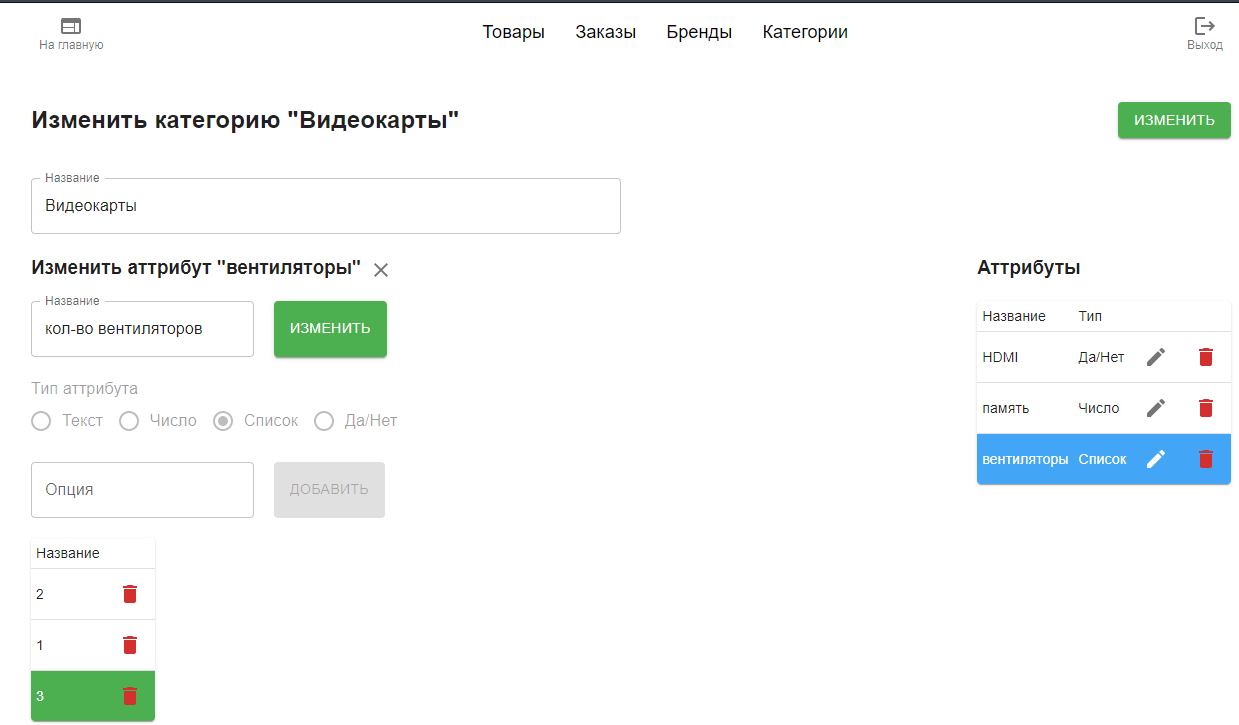


Рисунок 6.2 – Редактирование категории

Далее администратор можешь создавать новые бренды. Единственным параметром для бренда является его имя, пример представлен на рисунке 6.3.

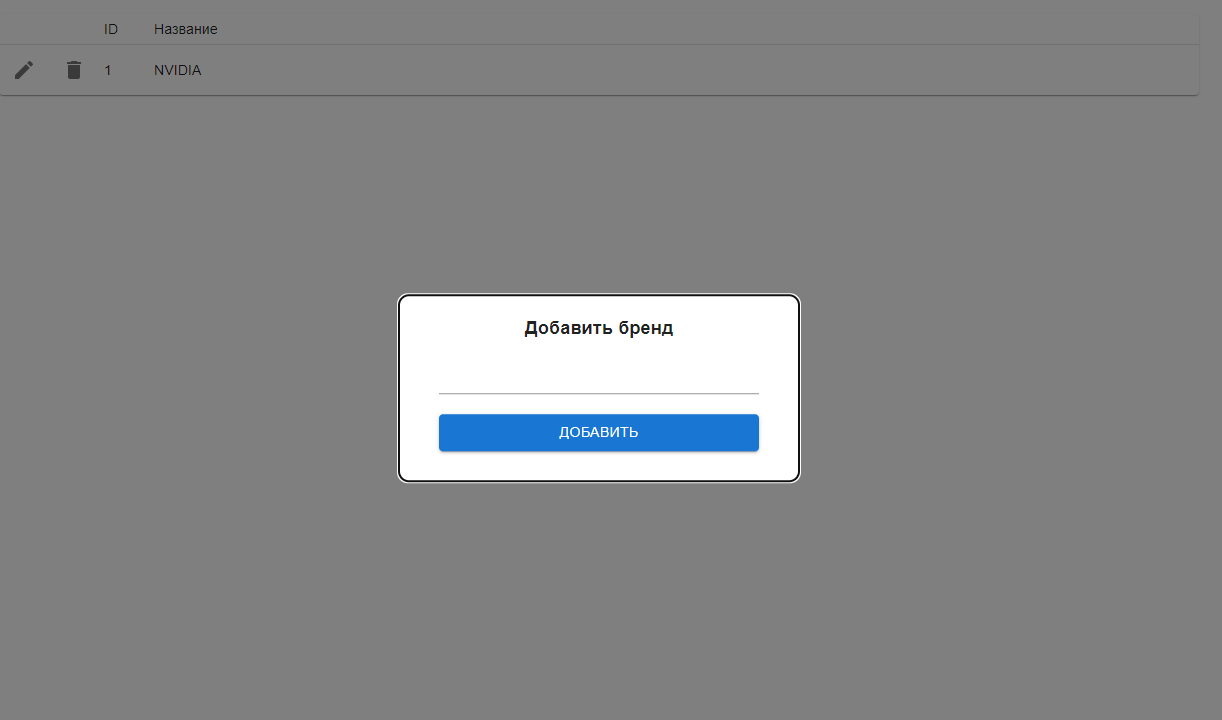


Рисунок 6.3 – Добавление нового бренда

Имя категорию и бренд, администратор можешь добавить новые товары для магазина. Добавление товаров приведено на рисунке 6.4

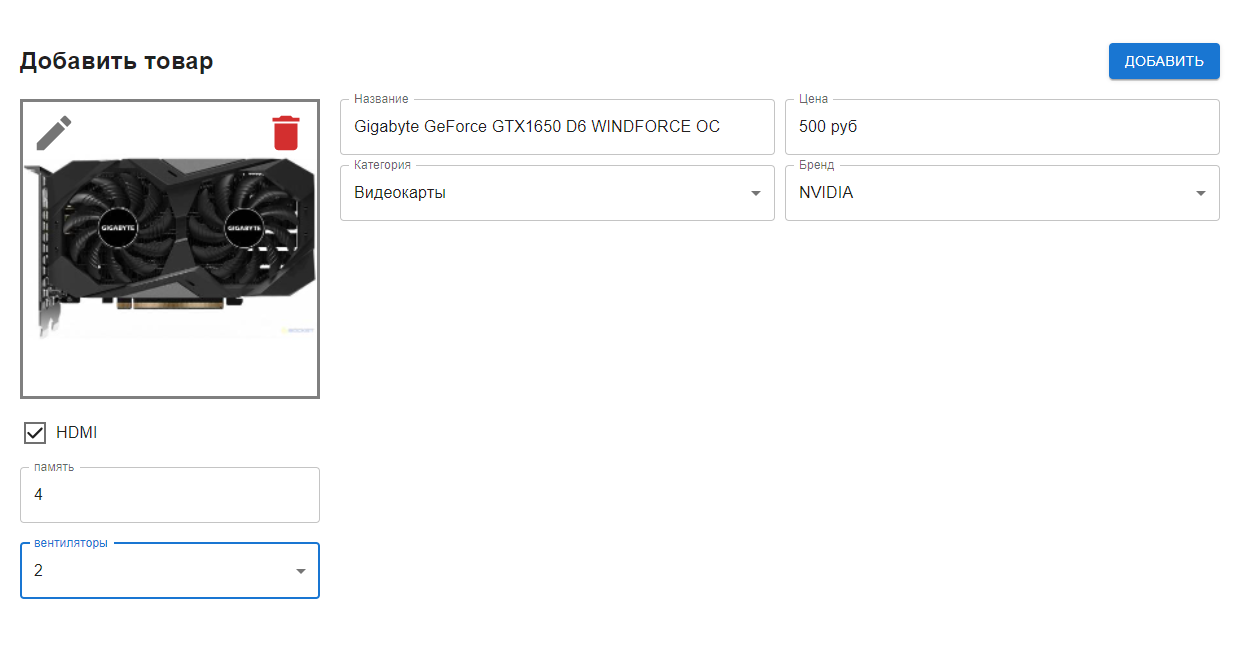


Рисунок 6.4 – Меню добавления товара

Здесь происходит выбор изображения товара, его название, цена в рублях, категория, бренд и атрибуты, созданные для конкретной категории.

# Заключение

В результате курсовой работы было разработано Web-приложение «Магазин компьютерных комплектующих». Основой приложения стал язык JavaScript. Серверная часть написана с помощью фреймворка Express, а клиентская с помощью фреймворка React. Обе технологии поддерживают асинхронное выполнение операций, что позволяет добиться высокой производительности приложения. В качестве базы данных выступила база MySQL от компании Oracle.

Программное средство выполняет следующие функции:

* имеет интуитивно понятное и простое управление;
* обращается к базе данных, успешно извлекает и вносит новые данные, осуществляет поиск по базе и автоматическую запись некоторых полей;
* имеет различный функционал в зависимости от типа пользователя;
* быстро обрабатывает запросы.
* функционирует по протоколам https и wss;
* выглядит аккуратно, не содержит лишнюю информацию.

Разработанное программное средство удовлетворяет всем требованиям технического задания.

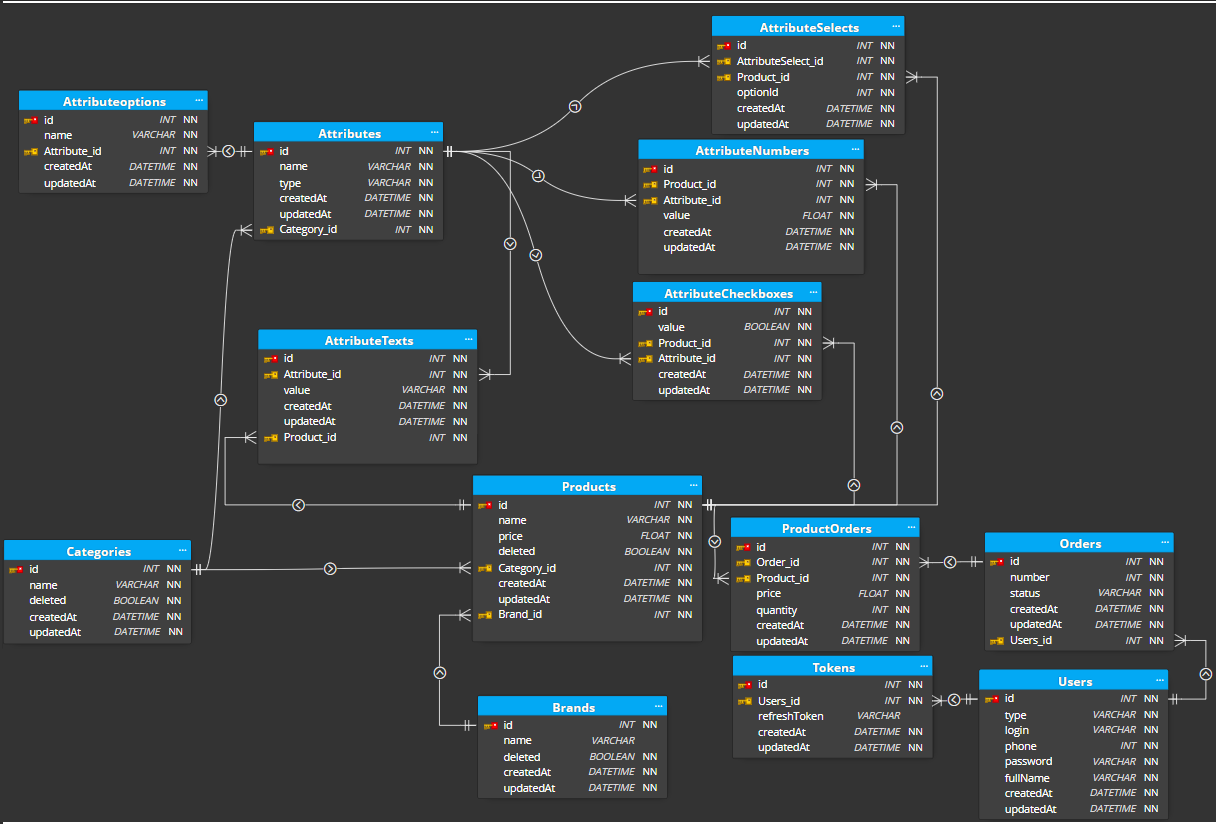
При разработке было уделено внимание производительности программного средства. Результатом проведённых работ стало то, что приложением могут с комфортом пользоваться большое количество пользователей параллельно. Приложение поддерживает асинхронный интерфейс и обработку пользовательских событий в режиме реального времени.

Приложение было успешно протестировано. В результате проведённых тестов ошибок обнаружено не было. Данный программный продукт является завершённым и готов к использованию.

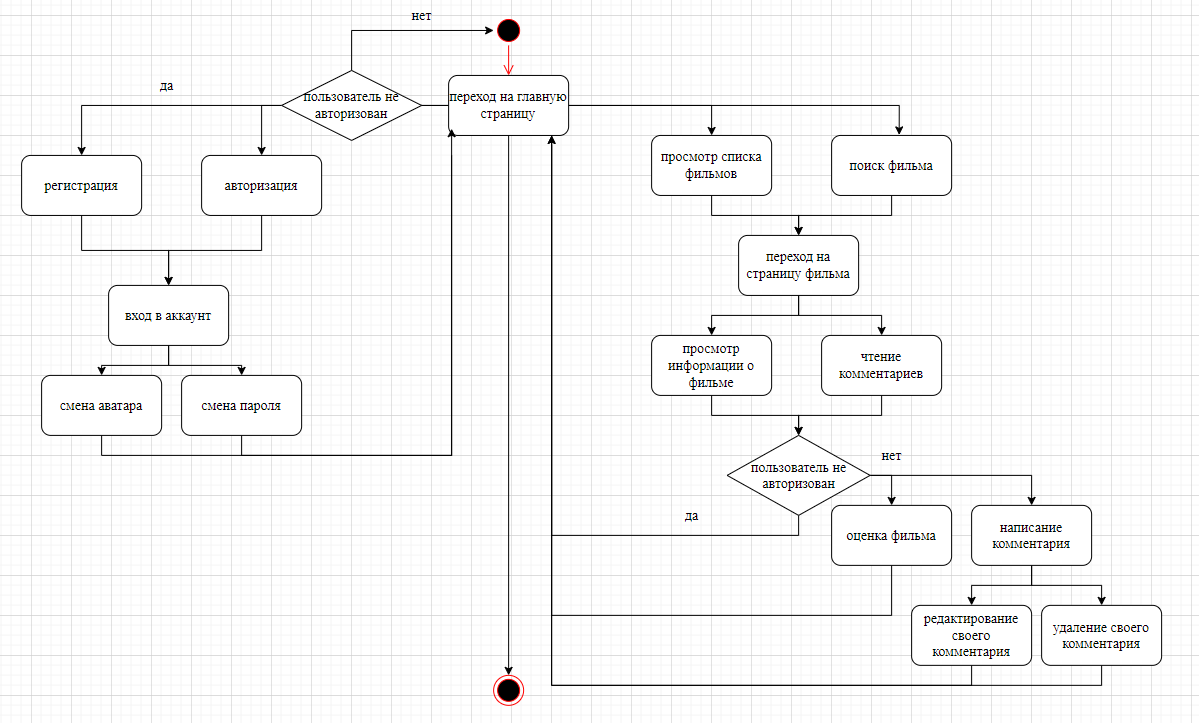
# Список литературы

1. Node.js v11.15.0 Documentation [Электронный ресурс] / OpenJS Foundation. – Режим доступа: https://nodejs.org/docs/latest-v11.x/api. – Дата доступа:10.05.2022;
2. Express [Электронный ресурс] – Режим доступа: <https://expressjs.com/ru/> – Дата доступа: 10.05.2022;
3. React [Электронный ресурс] – Режим доступа: https://ru.reactjs.org/docs/getting-started.html – Дата доступа: 10.05.2022.
4. MySQL [Электронный ресурс] – Режим доступа: https://dev.mysql.com/doc/– Дата доступа: 10.05.2022.
5. Sequelize [Электронный ресурс] – Режим доступа: <https://sequelize.org>/ – Дата доступа: 10.05.2022.

# Приложение А



# Приложение Б



# Приложение В

import express from "express";

import https from "https";

import fs from "fs";

import cors from "cors";

import cookieParser from "cookie-parser";

import bodyParser from "body-parser";

import { Server } from "rpc-websockets";

import { errorHandler } from "./api/helpers/errorHandler.js";

import caslConfig from "./api/helpers/caslConfig.js";

import corsConfig from "./api/helpers/corsConfig.js";

import wsEvents from "./api/helpers/wsEvents.js";

import filmController from "./api/controllers/filmController.js";

import userController from "./api/controllers/userController.js";

import commentController from "./api/controllers/commentController.js";

import ratingController from "./api/controllers/ratingController.js";

import tokenDecoder from "./api/helpers/tokenDecoder.js";

const port = 4000;

const wsPort = 5000;

const app = express();

app.use(cors(corsConfig));

app.use(tokenDecoder);

app.use(caslConfig);

app.use(cookieParser());

app.use(bodyParser.json());

app.use("/api/film", filmController);

app.use("/api/user", userController);

app.use("/api/comment", commentController);

app.use("/api/rating", ratingController);

app.use(errorHandler);

https

  .createServer(

    {

      key: fs.readFileSync("localhost.decrypted.key"),

      cert: fs.readFileSync("localhost.crt"),

    },

    app

  )

  .listen(port, function () {

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

  });

export const eventSocket = new Server({ port: wsPort, host: "localhost" });

eventSocket.event(wsEvents.commentAdded);

eventSocket.event(wsEvents.commentEdited);

eventSocket.event(wsEvents.commentDeleted);

import \* as core from "@casl/ability";

import { CaslActions, CaslModels, UserRoles } from "./enums.js";

const caslConfig = async (req, res, next) => {

  const { rules, can } = new core.AbilityBuilder(core.Ability);

  const userClaims = req.user || {};

  console.log(req.user)

  switch (userClaims.userRole) {

    case UserRoles.admin:

      can(CaslActions.delete, CaslModels.comment);

    case UserRoles.user:

      can(CaslActions.create, [CaslModels.comment, CaslModels.rating]);

      can([CaslActions.delete, CaslActions.update], CaslModels.comment, {

        userId: userClaims.userId,

      });

      can(CaslActions.read, [CaslModels.user, CaslModels.rating]);

      can(CaslActions.update, CaslModels.user, { id: userClaims.userId });

    default:

      can(CaslActions.read, [

        CaslModels.film,

        CaslModels.comment,

      ]);

  }

  req.ability = new core.Ability(rules);

  next();

};

export default caslConfig;

const corsOptions = {

  origin: "http://localhost:3000",

  credentials: true,

};

export default corsOptions;

export const errorHandler = (err, req, res, next) => {

  const status = err.status || 500;

  const message = err.message || "Internal Server Error";

  res.status(status).json(message);

};

import jwtService from "../../bussiness/services/jwtService.js";

const tokenDecoder = async (req, res, next) => {

  const tokenPayload = await jwtService.getTokenPayload(req);

  req.user = tokenPayload;

  next();

};

export default tokenDecoder;

import { Router } from "express";

import \* as core from "@casl/ability";

import { eventSocket } from "../../index.js";

import commentService from "../../bussiness/services/commentService.js";

import wsEvents from "../helpers/wsEvents.js";

import { CaslActions, CaslModels } from "../helpers/enums.js";

import {

  deleteSchema,

  getPageAmountByIdSchema,

  getPageByIdSchema,

  postSchema,

  putSchema,

} from "../validators/commentValidators.js";

const router = Router();

router.get("/:filmId/:pageNumber/:pageSize/:sort", async (req, res, next) => {

  try {

    await getPageByIdSchema.validateAsync(req.params);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const filmId = parseInt(req.params.filmId);

    const pageNumber = parseInt(req.params.pageNumber);

    const pageSize = parseInt(req.params.pageSize);

    const sort = req.params.sort || "Date";

    const comments = await commentService.getPage(filmId, pageNumber, pageSize, sort);

    res.status(200).json(comments);

  } catch (err) {

    next(err);

  }

});

router.get("/:filmId/:pageSize/", async (req, res, next) => {

  try {

    await getPageAmountByIdSchema.validateAsync(req.params);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const filmId = parseInt(req.params.filmId);

    const pageSize = parseInt(req.params.pageSize);

    const comments = await commentService.getPagesAmount(filmId, pageSize);

    res.status(200).json(comments);

  } catch (err) {

    next(err);

  }

});

router.post("/", async (req, res, next) => {

  try {

    await postSchema.validateAsync(req.body);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  const ability = req.ability;

  if (!ability.can(CaslActions.create, CaslModels.comment)) {

    return res.sendStatus(401);

  }

  try {

    const filmId = parseInt(req.body.filmId);

    const comment = await commentService.create(

      req.user.userId,

      filmId,

      req.body.text

    );

    eventSocket.emit(wsEvents.commentAdded, comment);

    res.status(200).json(comment);

  } catch (err) {

    next(err);

  }

});

router.delete("/:id", async (req, res, next) => {

  try {

    await deleteSchema.validateAsync(req.params.id);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  const ability = req.ability;

  const commentId = parseInt(req.params.id);

  try {

    const comment = await commentService.get(commentId);

    if (

      !ability.can(CaslActions.delete, core.subject(CaslModels.comment, comment))

    ) {

      return res.sendStatus(401);

    }

    await commentService.destroy(commentId);

    eventSocket.emit(wsEvents.commentDeleted, { commentId });

    res.status(200).json(commentId);

  } catch (err) {

    next(err);

  }

});

router.put("/", async (req, res, next) => {

  try {

    await putSchema.validateAsync(req.body);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  const ability = req.ability;

  const commentId = parseInt(req.body.commentId);

  try {

    const comment = await commentService.get(commentId);

    if (

      !ability.can(CaslActions.update, core.subject(CaslModels.comment, comment))

    ) {

      return res.sendStatus(401);

    }

    await commentService.update(commentId, req.body.text);

    eventSocket.emit(wsEvents.commentEdited, {

      commentId,

      text: req.body.text,

    });

    res.status(200).json(commentId);

  } catch (err) {

    next(err);

  }

});

export default router;

import { Router } from "express";

import filmService from "../../bussiness/services/filmService.js";

import { getIdByTitleSchema, getIdSchema } from "../validators/filmValidators.js";

const router = Router();

router.get("/all", async (req, res, next) => {

  try {

    const films = await filmService.getAll();

    res.status(200).json(films);

  } catch (err) {

    next(err);

  }

});

router.get("/:id", async (req, res, next) => {

  try {

    await getIdSchema.validateAsync(req.params.id);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const id = parseInt(req.params.id);

    const film = await filmService.get(id);

    res.status(200).json(film);

  } catch (err) {

    next(err);

  }

});

router.get("/id/bytitle/:title", async (req, res, next) => {

  try {

    await getIdByTitleSchema.validateAsync(req.params.title);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const title = req.params.title;

    const id = await filmService.getIdByTitle(title);

    res.status(200).json(id);

  } catch (err) {

    next(err);

  }

});

export default router;

import { Router } from "express";

import Joi from "joi";

import ratingService from "../../bussiness/services/ratingService.js";

import { CaslActions, CaslModels } from "../helpers/enums.js";

import { postSchema, getIdSchema } from "../validators/ratingValidators.js";

const router = Router();

router.get("/:id", async (req, res, next) => {

  const ability = req.ability;

  if (!ability.can(CaslActions.read, CaslModels.rating)) {

    return res.sendStatus(401);

  }

  try {

    await getIdSchema.validateAsync(req.params.id);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const filmId = parseInt(req.params.id);

    const rate = await ratingService.get(req.user.userId, filmId);

    res.status(200).json(rate);

  } catch (err) {

    next(err);

  }

});

router.post("/", async (req, res, next) => {

  const ability = req.ability;

  if (!ability.can(CaslActions.create, CaslModels.rating)) {

    return res.sendStatus(401);

  }

  try {

    await postSchema.validateAsync(req.body);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const newRating = await ratingService.set(

      req.user.userId,

      req.body.filmId,

      req.body.rate

    );

    res.status(200).json(newRating);

  } catch (err) {

    next(err);

  }

});

export default router;

import { Router } from "express";

import userService from "../../bussiness/services/userService.js";

import jwtService from "../../bussiness/services/jwtService.js";

import JwtConfig from "../../bussiness/helpers/tokenConfig.js";

import {

  changePasswordSchema,

  loginSchema,

  registerSchema,

  changeAvatarSchema,

} from "../validators/userValidators.js";

import { CaslActions, CaslModels } from "../helpers/enums.js";

const router = Router();

router.get("/", async (req, res) => {

  const ability = req.ability;

  if (!ability.can(CaslActions.read, CaslModels.user)) {

    return res.sendStatus(401);

  }

  const id = req.user.userId;

  const user = await userService.get(id);

  return res.status(200).json(user);

});

router.post("/login", async (req, res, next) => {

  try {

    await loginSchema.validateAsync(req.body);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const { accessToken, refreshToken } = await userService.login(

      req.body.username,

      req.body.password

    );

    res.cookie(

      JwtConfig.tokens.refresh.type,

      refreshToken,

      JwtConfig.refreshOptions

    );

    return res.status(200).json({ accessToken });

  } catch (err) {

    next(err);

  }

});

router.post("/register", async (req, res, next) => {

  try {

    await registerSchema.validateAsync(req.body);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const token = await userService.register(

      req.body.username,

      req.body.password,

      req.body.email

    );

    return res.status(200).json(token);

  } catch (err) {

    next(err);

  }

});

router.put("/password", async (req, res, next) => {

  const ability = req.ability;

  if (!ability.can(CaslActions.update, CaslModels.user)) {

    return res.sendStatus(401);

  }

  try {

    await changePasswordSchema.validateAsync(req.body);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    await userService.updatePassword(

      req.user.userId,

      req.body.oldPassword,

      req.body.newPassword

    );

    return res.sendStatus(204);

  } catch (err) {

    next(err);

  }

});

router.put("/avatar", async (req, res, next) => {

  const ability = req.ability;

  if (!ability.can(CaslActions.update, CaslModels.user)) {

    return res.sendStatus(401);

  }

  try {

    await changeAvatarSchema.validateAsync(req.body);

  } catch (err) {

    err.status = 400;

    return next(err);

  }

  try {

    const avatar = await userService.updateAvatar(

      req.user.userId,

      req.body.avatar

    );

    return res.status(200).json({ avatar });

  } catch (err) {

    next(err);

  }

});

router.put("/refresh", async (req, res, next) => {

  try {

    const { accessToken, refreshToken } = await jwtService.refreshAccessToken(

      req

    );

    if (!accessToken || !refreshToken) {

      return res.sendStatus(401);

    }

    res.cookie(

      JwtConfig.tokens.refresh.type,

      refreshToken,

      JwtConfig.refreshOptions

    );

    return res.status(200).json({ accessToken });

  } catch (err) {

    next(err);

  }

});

export default router;