МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Мобильное приложение для рецензирования и оценки фильмов

Выполнил студент Шатохин Данила Сергеевич

(Ф.И.О.)

Руководитель проекта ассистент Кантарович В.С

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

Курсовой проект защищен с оценкой

Минск 2024

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра информационных систем и технологий

Утверждаю

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ В.В. Смелов\_

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г.

**ЗАДАНИЕ**

**к курсовому проектированию**

**по дисциплине** "Современные технологии web-приложений"

|  |  |
| --- | --- |
| Специальность: 1-40 05 01 03 «Информационные системы и технологии» Группа: \_\_1\_\_  Студент: Шатохин Д.С. |  |
| **Тема: Мобильное приложение для рецензирования и оценки фильмов** | |

**1. Срок сдачи студентом законченной работы**: "16 декабря 2024 г."

**2. Исходные данные к проекту:**

**2.1.** Функционально приложение должно:

* обеспечивать реализацию 2 ролей: пользователь, модератор;
* обеспечивать возможность регистрация и авторизации;
* обеспечить функционал для пользователя: возможность поиска и сортировки фильмов по жанрам, возможность управления профилем (аватар, имя, дата рождения, пол), возможность добавления и удаления комментариев с оценкой к фильмам, возможность управления избранными: добавления и удаление фильмов, возможность просмотра списка посещённых пользователем страниц фильмов;
* обеспечить функционал для модератора: возможность управления фильмами (добавление, редактирование, удаление), возможность управления комментариями к фильму (удаление).

**2.2.** Программное средство должно быть выполнено с использованием асинхронного программирования, взаимодействовать с базой данных. Программное средство должно представлять собой мобильное приложение c асинхронным UI (React-Native). Отображение, бизнес логика и хранилище данных должны быть максимально независимы друг от друга для возможности расширения. Диаграммы вариантов использования, классов реализации задачи, последовательности разработать на основе UML. Язык разработки проекта JavaScript, платформа Node.js, фреймворк Nest.js. Мобильное приложение должно быть логически завершенным. Управление программой должно быть интуитивно понятным и удобным. Листинги проекта должны содержать комментарии. Проект должен быть контейнеризирован с использованием Docker для облегчения развертывания и поддержки различных платформ.

**3. Содержание расчетно-пояснительной записки**

* Введение
* Постановка задачи
* Проектирование мобильного приложения
* Разработка мобильного приложения
* Тестирование мобильного приложения
* Руководство пользователя
* Заключение
* Список используемых источников
* Приложения

**4. Форма представления выполненной курсовой работы:**

* + теоретическая часть курсового проекта должны быть представлены в формате MS Word. Оформление записки должно быть согласно выданным правилам;
  + необходимые схемы, диаграммы и рисунки допускается делать в MS Office Visio, Rational Rose, WS или копии экрана (интерфейс);
  + листинги программы представляются частично в приложении;
  + к записке необходимо приложить CD (DVD), который должен содержать: пояснительную записку, листинги и инсталляцию проекта.

**Календарный план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Введение | 08.10.2024 |  |
| 2 | Аналитический обзор литературы по теме проекта. Изучение требований, определение вариантов использования | 15.10.2024 |  |
| 3 | Анализ и проектирование архитектуры приложения (построение диаграмм, проектирование бизнес-слоя, представления и данных) | 22.10.2024 |  |
| 4 | Проектирование структуры базы данных. Разработка дизайна пользовательского интерфейса | 22.10.2024 |  |
| 5 | Кодирование программного средства | 19.11.2024 |  |
| 6 | Тестирования и отладка программного средства | 26.11.2024 |  |
| 7 | Оформление пояснительной записки | 03.12.2024 |  |
| 8 | Сдача проекта | 16.12.2024 |  |

**5. Дата выдачи задания** 16.09.2024

Руководитель**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** *В.С. Кантарович*

(подпись)

Задание принял к исполнению **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(дата и подпись студента)

Содержание

[Введение 6](#_Toc185322667)

[1 Постановка задачи и обзор аналогичных решений 8](#_Toc185322668)

[1.1 Обзор аналогов 8](#_Toc185322669)

[1.2 Постановка задач 11](#_Toc185322670)

[1.3 Вывод по разделу: 11](#_Toc185322671)

[2 Проектирование мобильного приложения 12](#_Toc185322672)

[2.1 Выбор средств реализации 12](#_Toc185322673)

[2.2 Описание работы программы 15](#_Toc185322674)

[2.3 Проектирование базы данных 17](#_Toc185322675)

[2.3.1 Логическая схема базы данных 17](#_Toc185322676)

[2.3.2 Описание базы данных 17](#_Toc185322677)

[2.4 Контейнеризация 19](#_Toc185322678)

[2.5 Проектирование основных алгоритмов 20](#_Toc185322679)

[2.6 Вывод по разделу 23](#_Toc185322680)

[3 Разработка мобильного приложения 24](#_Toc185322681)

[3.1 Разработка серверной части 24](#_Toc185322682)

[3.1.1 Cтруктура серверной части приложения 24](#_Toc185322683)

[3.1.2 Конкретные реализации 27](#_Toc185322684)

[3.2 Разработка клиентской части 28](#_Toc185322685)

[3.3 Контейнеризация приложения с использованием Docker 32](#_Toc185322686)

[3.4 Вывод по разделу 33](#_Toc185322687)

[4 Тестирование мобильного приложения 34](#_Toc185322688)

[4.1 Валидация форм ввода 34](#_Toc185322689)

[4.2 Вывод по разделу 36](#_Toc185322690)

[5 Руководство пользователя 37](#_Toc185322691)

[5.1 Руководство пользователя 37](#_Toc185322692)

[5.1.1 Регистрация пользователя 37](#_Toc185322693)

[5.1.2 Авторизация пользователя 37](#_Toc185322694)

[5.1.3 Главная страница приложения 38](#_Toc185322695)

[5.1.4 Страница пользователя 39](#_Toc185322696)

[5.1.5 Страница редактирования пользователя 40](#_Toc185322697)

[5.1.6 Страница избранных фильмов пользователя 41](#_Toc185322698)

[5.1.7 Страница жанра фильмов 41](#_Toc185322699)

[5.1.8 Страница фильма 42](#_Toc185322700)

[5.2 Руководство модератора 44](#_Toc185322701)

[5.2.1 Страница добавления и редактирования фильмов 44](#_Toc185322702)

[5.2.2 Удаление оценок 44](#_Toc185322703)

[5.2.3 Удаление фильма 45](#_Toc185322704)

[5.3 Руководство по развёртыванию приложения 46](#_Toc185322705)

[5.3.1 Подготовка к развёртыванию 46](#_Toc185322706)

[5.3.2 Структура развёртывания 46](#_Toc185322707)

[5.3.3 Запуск приложения 46](#_Toc185322708)

[5.4 Вывод по разделу 46](#_Toc185322709)

[Заключение 48](#_Toc185322710)

[Список используемых источников 49](#_Toc185322711)

[ПРИЛОЖЕНИЕ А 50](#_Toc185322712)

[ПРИЛОЖЕНИЕ Б 52](#_Toc185322713)

[ПРИЛОЖЕНИЕ В 53](#_Toc185322714)

[ПРИЛОЖЕНИЕ Г 54](#_Toc185322715)

[ПРИЛОЖЕНИЕ Д 55](#_Toc185322716)

[ПРИЛОЖЕНИЕ Е 57](#_Toc185322717)

[ПРИЛОЖЕНИЕ Ж 59](#_Toc185322718)

[ПРИЛОЖЕНИЕ З 60](#_Toc185322719)

[ПРИЛОЖЕНИЕ И 61](#_Toc185322720)

Введение

Современный рынок кинематографии и увлеченность людей киноискусством создают потребность в удобных инструментах для обмена отзывами и оценки фильмов. В условиях стремительного развития мобильных технологий и увеличения числа пользователей смартфонов актуальной становится разработка приложения, предоставляющего пользователям возможность оставлять рецензии, оценивать фильмы и находить интересующие их киноленты по заданным критериям. Такое приложение не только удовлетворяет потребности любителей кино, но и способствует созданию активного сообщества, где пользователи могут делиться своими мнениями и рекомендациями.

Целью данного курсового проекта является создание мобильного приложения для рецензирования и оценки фильмов, в котором пользователи смогут оставлять комментарии, ставить оценки и просматривать списки лучших фильмов. Архитектура приложения будет построена таким образом, чтобы обеспечить максимальную независимость между пользовательским интерфейсом, бизнес-логикой и хранилищем данных, что позволит расширять функционал без сложных изменений.

Для разработки данного проекта выбраны современные технологии, включая асинхронный UI на React Native для создания удобного и отзывчивого интерфейса, Node.js с фреймворком NestJS для реализации серверной части, а также базу данных PostgreSQL для хранения информации о фильмах, отзывах и пользователях. Приложение будет контейнеризировано с использованием Docker, что упростит его развертывание и поддержку на различных платформах.

Актуальность проекта обусловлена растущим интересом к мобильным приложениям и увеличением времени, проводимого пользователями за смартфонами. Создание специализированного приложения для рецензирования и оценки фильмов отвечает современным потребностям рынка, предоставляя пользователям быстрый доступ к информации о фильмах, возможность делиться своими впечатлениями и получать рекомендации на основе личных предпочтений. Кроме того, внедрение системы ролей, позволяющей различать возможности обычных пользователей и модераторов, способствует поддержанию высокого качества контента и предотвращению распространения нежелательных материалов.

Для достижения поставленной цели необходимо решить следующие задачи:

* Разработка системы регистрации и авторизации: создать безопасный функционал для регистрации и авторизации пользователей, обеспечивая защиту и безопасное хранение учетных данных;
* Реализация системы ролей: внедрить поддержку двух ролей – пользователь и модератор, каждая из которых обладает уникальными возможностями для эффективного взаимодействия с приложением;
* Функционал для пользователей: предоставить пользователям возможность осуществлять поиск и сортировку фильмов по жанрам, управлять профилем (изменение аватара, имени, даты рождения и пола), добавлять и удалять комментарии к фильмам с оценкой, управлять избранными фильмами (добавление и удаление), а также просматривать список ранее посещенных страниц фильмов.
* Функционал для модераторов: предоставить модераторам возможности по управлению фильмами (добавление, редактирование и удаление) и модерации комментариев (удаление), что позволит обеспечить порядок и соответствие контента стандартам платформы;
* Проектирование и разработка базы данных: создать эффективную и масштабируемую структуру базы данных, обеспечивающую хранение информации о пользователях, фильмах, рецензиях и оценках;
* Контейнеризация приложения: использовать Docker для контейнеризации серверной части и базы данных, обеспечивая изоляцию и упрощение процесса развертывания приложения на различных платформах;
* Тестирование и обеспечение безопасности: провести всестороннее тестирование всех компонентов приложения и внедрить меры по обеспечению безопасности данных пользователей и устойчивости системы к потенциальным угрозам.

Выполнение перечисленных задач позволит создать функциональное, безопасное и удобное мобильное приложение для рецензирования и оценки фильмов, удовлетворяющее современные требования пользователей и способствующее развитию активного сообщества любителей кино.

1 Постановка задачи и обзор аналогичных решений

В ходе исследования аналогичных мобильных приложений для рецензирования и оценки фильмов были проанализированы несколько платформ с похожими функциональными возможностями. Анализ показал, что большинство этих приложений предоставляет базовые функции для просмотра и оценивания фильмов, но часто не охватывает расширенные потребности пользователей, такие как удобная система поиска, гибкая структура комментариев, а также детализированные рейтинги и рецензии.

Также были изучены статьи и практические примеры, связанные с разработкой приложений для рецензирования и оценки фильмов. Эти материалы помогли сформировать основу для реализации ключевых функций проекта и углубили понимание подходов к организации пользовательского интерфейса и архитектуры приложения. Результаты анализа легли в основу требований и проектирования мобильного приложения.

1.1 Обзор аналогов

IMDb [7] — это крупнейшая в мире онлайн-база данных о фильмах, сериалах, актерах, режиссерах и других участниках киноиндустрии. Она предоставляет пользователям доступ к информации о миллионах фильмов и телешоу, включая рейтинги, рецензии, трейлеры, подробности съемок и данные о кассовых сборах. IMDb используется как профессионалами индустрии, так и любителями кино, чтобы искать, оценивать и обсуждать фильмы. Его особенности включают в себя:

* Огромная база данных фильмов, сериалов и актеров IMDb содержит информацию о миллионах фильмов, телешоу и их создателях, включая редкие и архивные работы.
* Каждый фильм или сериал имеет рейтинг, который формируется на основе оценок пользователей. Также можно прочитать текстовые рецензии.
* На странице фильма можно найти данные о режиссере, актерах, сюжете, саундтреке, трейлере, а также списке наград и номинаций.
* IMDb предлагает тематические подборки, такие как «Лучшие фильмы года», «Самые ожидаемые премьеры», «Культовые классики».
* IMDb доступен через удобное мобильное приложение и полноценную веб-версию, что позволяет использовать сервис на любом устройстве.

Недостатки IMDb:

* Страницы IMDb насыщены информацией, что делает их сложными для восприятия, особенно для новых пользователей.
* IMDb ориентирован на англоязычную аудиторию, и большинство информации представлено только на английском языке.
* IMDb не предоставляет возможностей для активного взаимодействия между пользователями (например, обсуждений или комментариев к чужим рецензиям).
* IMDb не использует алгоритмы, которые подстраиваются под вкусы пользователя, предлагая фильмы, подходящие именно ему.
* Большинство рецензий и обсуждений написаны на английском языке, что ограничивает их доступность для пользователей из других стран.

IMDb остается одной из наиболее популярных и широко используемых платформ для любителей кино и сериалов по всему миру, привлекая пользователей своей обширной базой данных, надежностью и удобным доступом к информации. Главная страница сервиса представлена на рисунке 1.1.

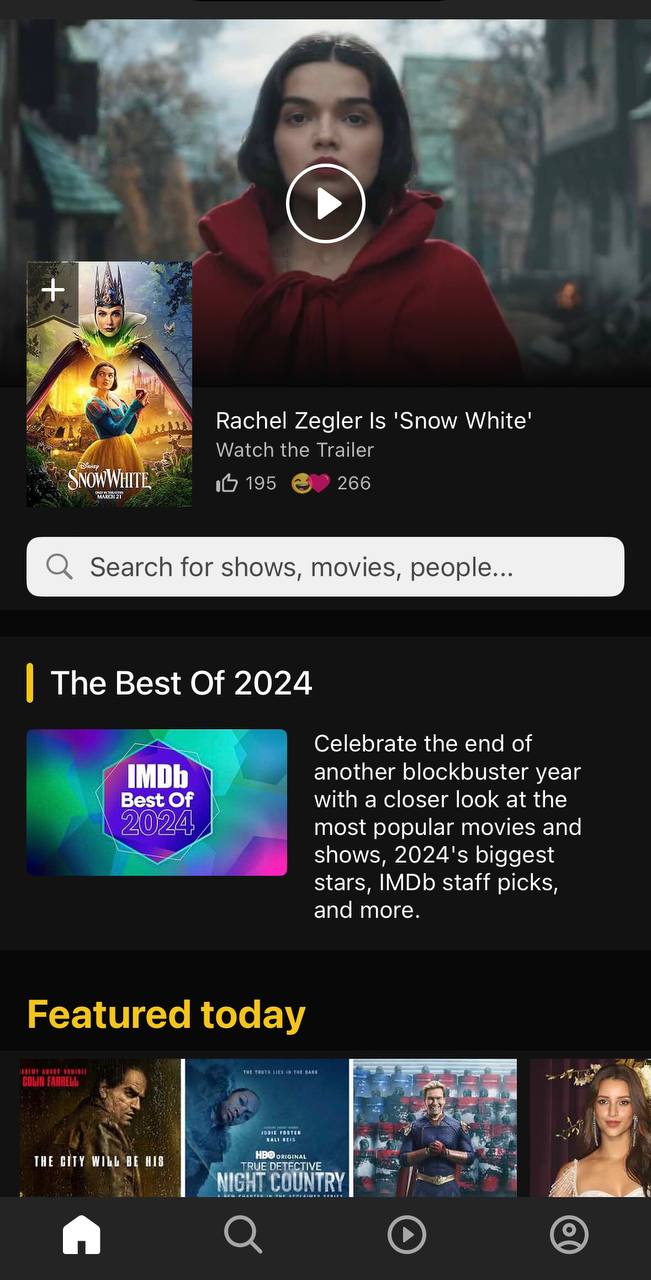
****

Рисунок 1.1 – Главная страница «IMDb»

КиноПоиск [6] — это крупнейшая русскоязычная платформа для поиска, оценки и рецензирования фильмов и сериалов. Сервис предоставляет пользователям доступ к огромной базе данных о фильмах, включая рецензии, рейтинги, трейлеры, а также возможность просмотра контента через встроенный потоковый сервис. КиноПоиск популярен среди зрителей благодаря локализованному контенту и удобству использования. Его особенности включают в себя:

* Сервис ориентирован на русскоязычную аудиторию, предоставляя информацию, адаптированную для локального рынка.
* Платформа позволяет не только искать фильмы, но и сразу их смотреть через подписку на КиноПоиск HD.
* КиноПоиск предоставляет расширенные возможности поиска по жанрам, годам, странам, актерам и другим параметрам.
* Платформа позволяет пользователям оставлять свои оценки и рецензии, а также изучать чужие мнения.
* КиноПоиск предлагает тематические подборки фильмов, а также персонализированные рекомендации на основе предпочтений пользователя.

Недостатки Кинопоиска:

* Для использования некоторых функций, включая просмотр фильмов через КиноПоиск HD, требуется оплаченная подписка.
* Некоторые разделы могут показаться перегруженными, особенно для тех, кто впервые использует платформу.
* Рейтинги и рецензии формируются в основном русскоязычными пользователями, что делает их менее объективными для фильмов, популярных за пределами СНГ.
* КиноПоиск плохо интегрируется с международными сервисами, такими как IMDb или Rotten Tomatoes.
* Иногда сервис продвигает фильмы или сериалы, что может влиять на объективность рейтингов.

Интерфейс главной страницы приложения представлен на рисунке 1.2.

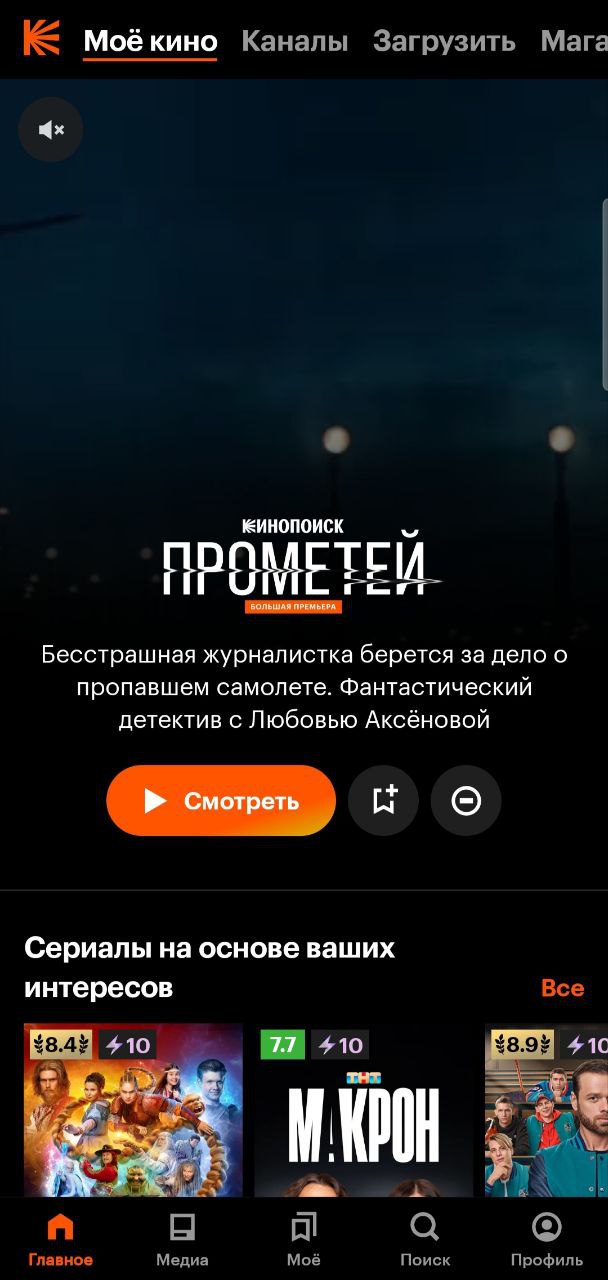


Рисунок 1.2 – Главная страница «Кинопоиск»

КиноПоиск продолжает оставаться ведущей платформой для любителей кино, предлагая пользователям удобный доступ к рецензиям, рейтингам и просмотру фильмов в рамках одного сервиса.

1.2 Постановка задач

На основе проведенного анализа и требований к разработке мобильного приложения для рецензирования и оценки фильмов были сформулированы следующие задачи:

* Разработка системы регистрации и авторизации: создать безопасный функционал для регистрации и авторизации пользователей, обеспечивая защиту и безопасное хранение учетных данных;
* Реализация системы ролей: внедрить поддержку двух ролей – пользователь и модератор, каждая из которых обладает уникальными возможностями для эффективного взаимодействия с приложением;
* Функционал для пользователей: предоставить пользователям возможность осуществлять поиск и сортировку фильмов по жанрам, управлять профилем (изменение аватара, имени, даты рождения и пола), добавлять и удалять комментарии к фильмам с оценкой, управлять избранными фильмами (добавление и удаление), а также просматривать список ранее посещенных страниц фильмов;
* Функционал для модераторов: предоставить модераторам возможности по управлению фильмами (добавление, редактирование и удаление) и модерации комментариев (удаление), что позволит обеспечить порядок и соответствие контента стандартам платформы.

Эти задачи обеспечат удобство, безопасность и функциональность приложения для пользователей, а также расширенные возможности для модерации и контроля качества контента.

1.3 Вывод по разделу:

Исходя из анализа КиноПоиска и IMDb, можно сделать вывод о ключевых характеристиках и функциональных особенностях успешного приложения для рецензирования и оценки фильмов. Идеальное приложение должно обладать широкой и актуальной базой данных, предоставляя пользователям подробную информацию о фильмах и сериалах. Оно также должно обеспечивать удобный интерфейс для поиска, сортировки и фильтрации контента, предлагать персонализированные рекомендации и позволять пользователям активно взаимодействовать через рецензии, рейтинги и обсуждения. Наличие локализованного контента и возможность потокового просмотра добавляют ценность и делают платформу более универсальной.

2 Проектирование мобильного приложения

Проект, разработанный в рамках данной работы, представляет собой мобильное приложение с клиент-серверной архитектурой, ориентированное на эффективное управление процессом рецензирования и оценки фильмов. Архитектура приложения включает две основные части: клиентскую и серверную.

Клиентская часть обеспечивает взаимодействие с пользователями двух ролей — пользователь и модератор, предоставляя интерфейс для поиска фильмов, управления профилем, добавления комментариев и оценок, а также работы с избранными фильмами и просмотром списка посещенных страниц. Серверная часть отвечает за обработку запросов, управление ролями и модерацию контента, а также взаимодействие с базой данных для хранения информации о фильмах, комментариях и профилях пользователей.

Такое разделение функциональности способствует повышению производительности приложения, обеспечивает его надежную работу и масштабируемость, делая использование приложения интуитивно понятным и удобным для всех пользователей.

2.1 Выбор средств реализации

При разработке мобильного приложения для рецензирования и оценки фильмов был проведён тщательный анализ доступных технологий и инструментов, позволяющих эффективно реализовать все заявленные функциональные требования. Выбор средств реализации основывался на критериях производительности, кроссплатформенности, удобства разработки, масштабируемости и безопасности. В результате был сформирован стек технологий, включающий React Native для клиентской части, Nest.js для серверной части, PostgreSQL для базы данных, а также дополнительные инструменты для управления состоянием и обработки запросов. Данный выбор обеспечивает высокую производительность приложения, удобство дальнейшего расширения функционала и поддерживает лучший пользовательский опыт.

2.1.1 Клиентская часть

Для разработки клиентской части мобильного приложения был выбран фреймворк React Native. Этот выбор обусловлен следующими преимуществами:

* Кроссплатформенность: React Native позволяет создавать приложения для iOS и Android одновременно, что значительно сокращает время и ресурсы на разработку и поддержку;
* Производительность: Благодаря использованию нативных компонентов, приложения на React Native демонстрируют высокую производительность и отзывчивость, сопоставимую с полностью нативными приложениями;
* Сообщество и поддержка: Большое сообщество разработчиков и обширная экосистема библиотек облегчают процесс разработки и позволяют быстро решать возникающие задачи.
* Интуитивно понятный дизайн: Использование современных библиотек и инструментов позволяет создать удобный и привлекательный интерфейс, который легко адаптируется под потребности пользователей.

Кроме того, для управления состоянием приложения и оптимизации работы с данными были интегрированы Redux и RTK Query, что обеспечивает эффективное управление данными и минимизирует количество избыточных запросов к серверу.

2.1.2 Серверная часть

Серверная часть приложения реализована на основе фреймворка Nest.js. Выбор этого фреймворка был обусловлен следующими факторами:

* Модульная архитектура: Nest.js поддерживает модульный подход, что облегчает организацию кода, его масштабирование и повторное использование компонентов;
* Совместимость с TypeScript: Использование TypeScript обеспечивает строгую типизацию, что снижает вероятность ошибок и упрощает процесс разработки и поддержки кода;
* Высокая производительность: Nest.js построен на базе Node.js и использует асинхронную обработку запросов, что позволяет обрабатывать большое количество одновременных запросов с минимальной задержкой;
* Интеграция с другими инструментами: Nest.js легко интегрируется с ORM Prisma и базой данных PostgreSQL, обеспечивая эффективное взаимодействие с данными и упрощая управление ими.

Серверная часть отвечает за обработку запросов от клиентского приложения, управление пользователями, фильмами, рецензиями и комментариями, а также за обеспечение безопасности и целостности данных.

2.1.3 Централизованная база данных

В качестве системы управления базами данных была выбрана PostgreSQL. Этот выбор обусловлен следующими преимуществами:

* Надёжность и безопасность: PostgreSQL обеспечивает высокий уровень надёжности данных и поддерживает различные механизмы безопасности, включая шифрование и управление доступом;
* Производительность: PostgreSQL оптимизирован для быстрого выполнения сложных запросов и поддерживает индексацию, что обеспечивает эффективный доступ к большим объёмам данных;
* Гибкость и расширяемость: Поддержка различных типов данных и расширяемость функционала с помощью плагинов позволяет адаптировать базу данных под специфические требования приложения;
* Сообщество и поддержка: Широкое сообщество пользователей и разработчиков обеспечивает доступ к обширной документации и инструментам для управления базой данных.

База данных PostgreSQL хранит информацию о пользователях, фильмах, рецензиях и комментариях, обеспечивая целостность и консистентность данных через использование транзакций и ограничений.

2.1.4 Архитектура мобильного приложения

Проект разделен на клиентскую и серверную части. Клиентская часть представляет собой мобильный интерфейс, разработанный на React Native с использованием Redux и RTK Query. Она позволяет пользователю отправлять запросы на сервер и отображать полученные данные, предоставляя удобные инструменты для поиска, оценки и рецензирования фильмов. В рисунке 2.1 представлена схема развертывания приложения.

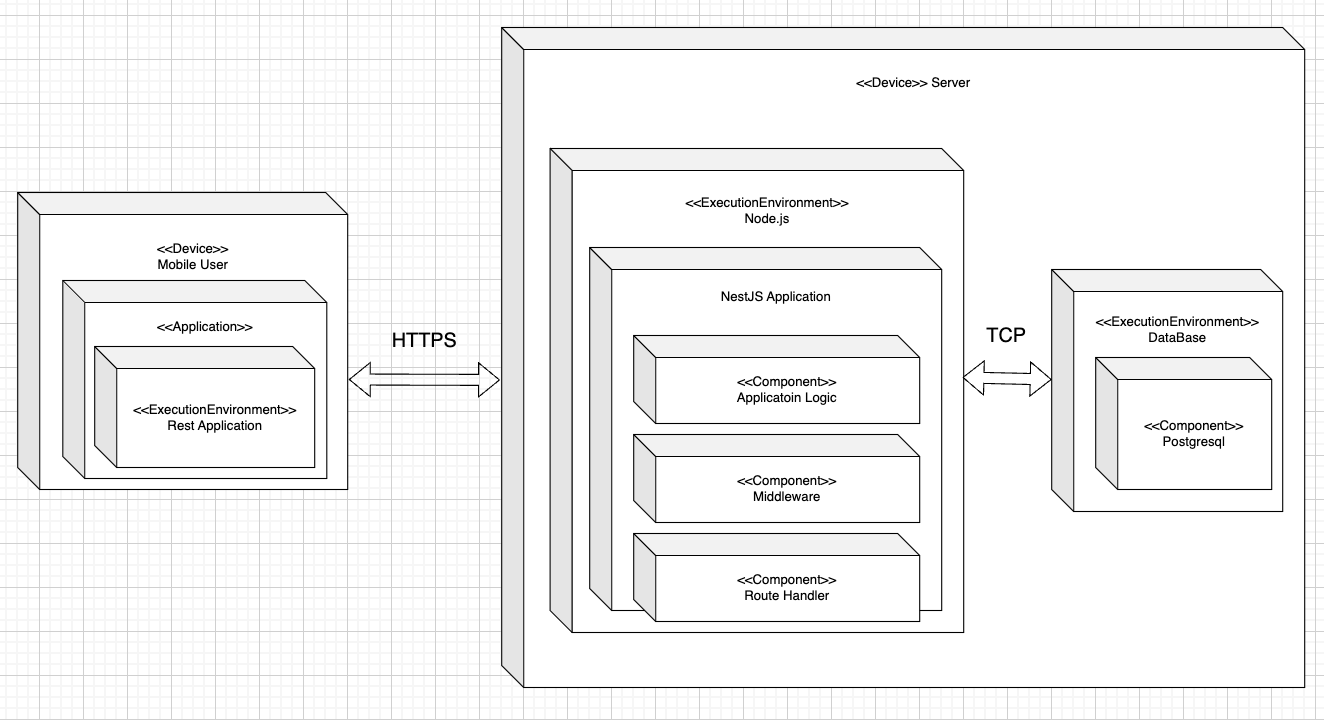


Рисунок 2.1 – Структурная схема приложения

Серверная часть реализована на фреймворке Nest.js, который обрабатывает запросы от клиента и взаимодействует с базой данных через Prisma и PostgreSQL. Обмен данными между клиентом и сервером осуществляется по протоколу HTTPS. Клиент отправляет запросы по заданным маршрутам, сервер обрабатывает их и возвращает соответствующие ответы.

Для создания интуитивно понятного и современного интерфейса используются Redux для централизованного управления состоянием и RTK Query для оптимизации работы с запросами. В серверной части применяются промежуточные модули, такие как CORS для обработки запросов с различных источников, а также middleware для обработки ошибок. Разделение приложения на клиентскую и серверную части обеспечивает высокую производительность и отзывчивость мобильного приложения для рецензирования и оценки фильмов.

2.2 Описание работы программы

В данном разделе подробно описывается функционирование мобильного приложения для рецензирования и оценки фильмов, а также представлены основные диаграммы, иллюстрирующие его архитектуру и взаимодействие компонентов.

2.2.1 Диаграмма вариантов использования

После запуска приложения сервер настраивается для обработки входящих запросов, открывая определённый порт, что позволяет клиентской части приложения обращаться к нему для выполнения различных операций. Мобильное приложение, разработанное на **React Native,** загружается на устройстве пользователя, обеспечивая удобный и отзывчивый интерфейс с помощью встроенных стилей React Native. Пользователь может взаимодействовать с ключевыми функциями приложения, такими как поиск фильмов по жанрам, оценка и написание рецензий, управление своим профилем и списком избранных фильмов. Вся функциональность организована для обеспечения интуитивного и приятного пользовательского опыта. Основные сценарии использования и функции приложения представлены на **use case-диаграмме,** изображённой на рисунке 2.2.

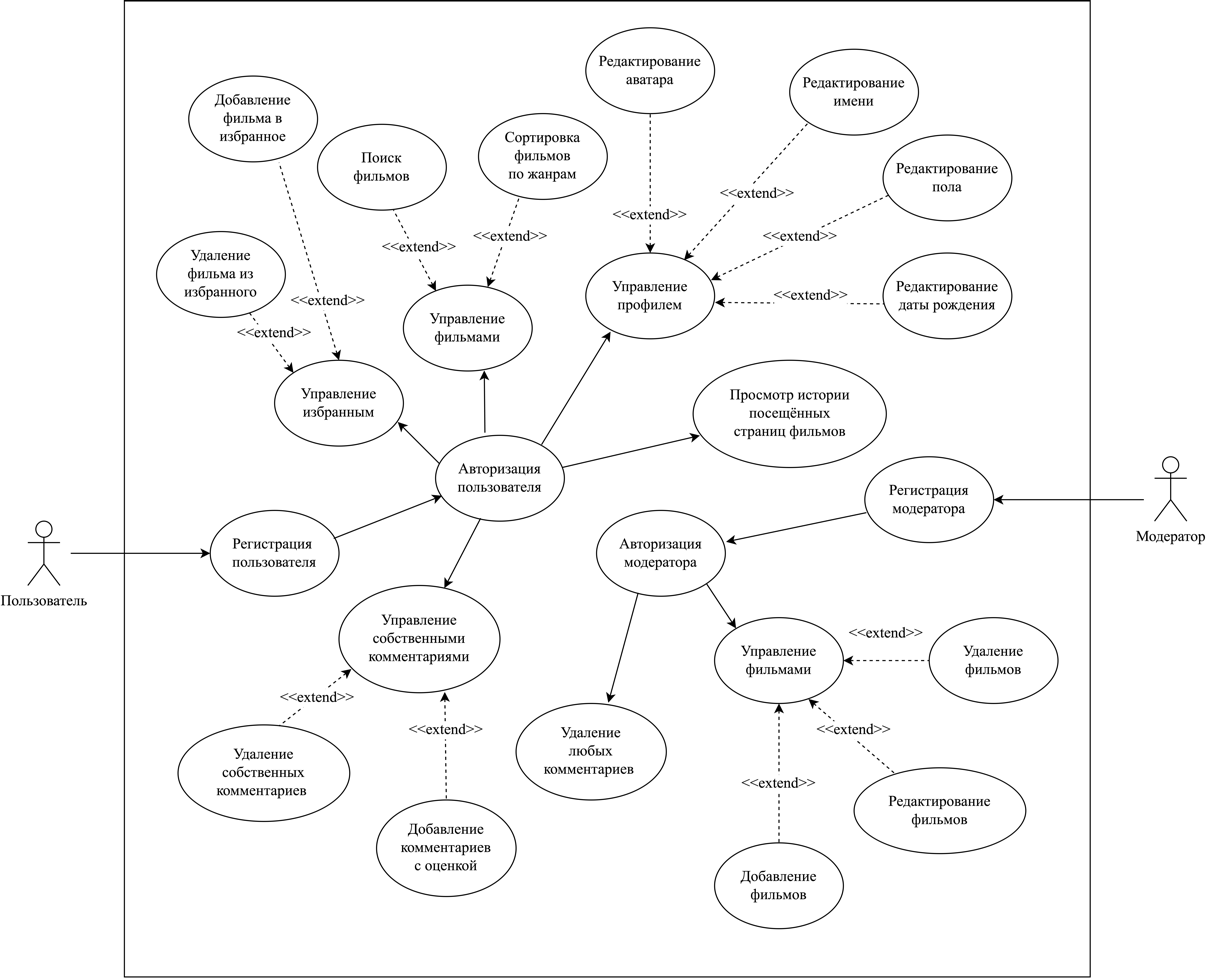


Рисунок 2.2 – Use case-диаграмма

Рисунок 2.2 демонстрирует основные сценарии взаимодействия пользователя с приложением, включая поиск фильмов, оценку, написание рецензий и управление профилем.

2.2.2 Структурная диаграмма

Когда пользователь выполняет определённые действия, такие как отправка запроса на добавление комментария, клиентская часть формирует HTTPS-запрос и направляет его на сервер. Сервер обрабатывает поступивший запрос, выполняя соответствующие операции, такие как запись данных в базу данных PostgreSQL через Prisma или обработка других данных. Если операция завершается успешно, сервер возвращает ответ обратно клиентскому приложению, которое обновляет интерфейс для отображения актуальной информации. В случае возникновения ошибок сервер применяет промежуточные модули для отправки сообщений об ошибках клиенту, что позволяет пользователю получать уведомления о проблемах и принимать соответствующие меры.

На рисунке 2.3 представлена UML-диаграмма, иллюстрирующая основные компоненты приложения и их взаимодействие, показывая, как разные модули работают вместе для обеспечения функциональности приложения и поддержки удобного пользовательского опыта.



Рисунок 2.3 – Структурная схема приложения

Эта диаграмма иллюстрирует архитектурное взаимодействие между клиентским приложением и сервером, подчёркивая, как данные обмениваются через протокол HTTPS, а также как происходит взаимодействие с базой данных. Такой подход обеспечивает многогранные возможности для пользователей, позволяя им не только отправлять запросы на сервер, но и получать ответы в режиме реального времени. Благодаря такой структуре, приложение становится более интерактивным и адаптивным, что позволяет пользователям эффективно управлять задачами и обрабатывать информацию из базы данных без задержек. В результате пользователи могут легко взаимодействовать с интерфейсом, получая актуальную информацию и оперативно обновляя данные, что, в свою очередь, повышает общую продуктивность и комфортность работы с приложением.

Таким образом, представленные диаграммы демонстрируют как пользовательские сценарии взаимодействия с приложением, так и техническую архитектуру системы, обеспечивающую её стабильную и эффективную работу.

2.3 Проектирование базы данных

2.3.1 Логическая схема базы данных

База данных спроектирована для выполнения всех задач, связанных с управлением рецензиями и оценками фильмов. В качестве СУБД выбрана PostgreSQL из-за ее надежности, поддерживаемой структурированности данных и широких возможностей для работы с реляционными моделями. PostgreSQL позволяет эффективно хранить и извлекать данные, а также легко интегрируется с фреймворком Nest.js. Структура базы данных представлена на рисунке 2.4.

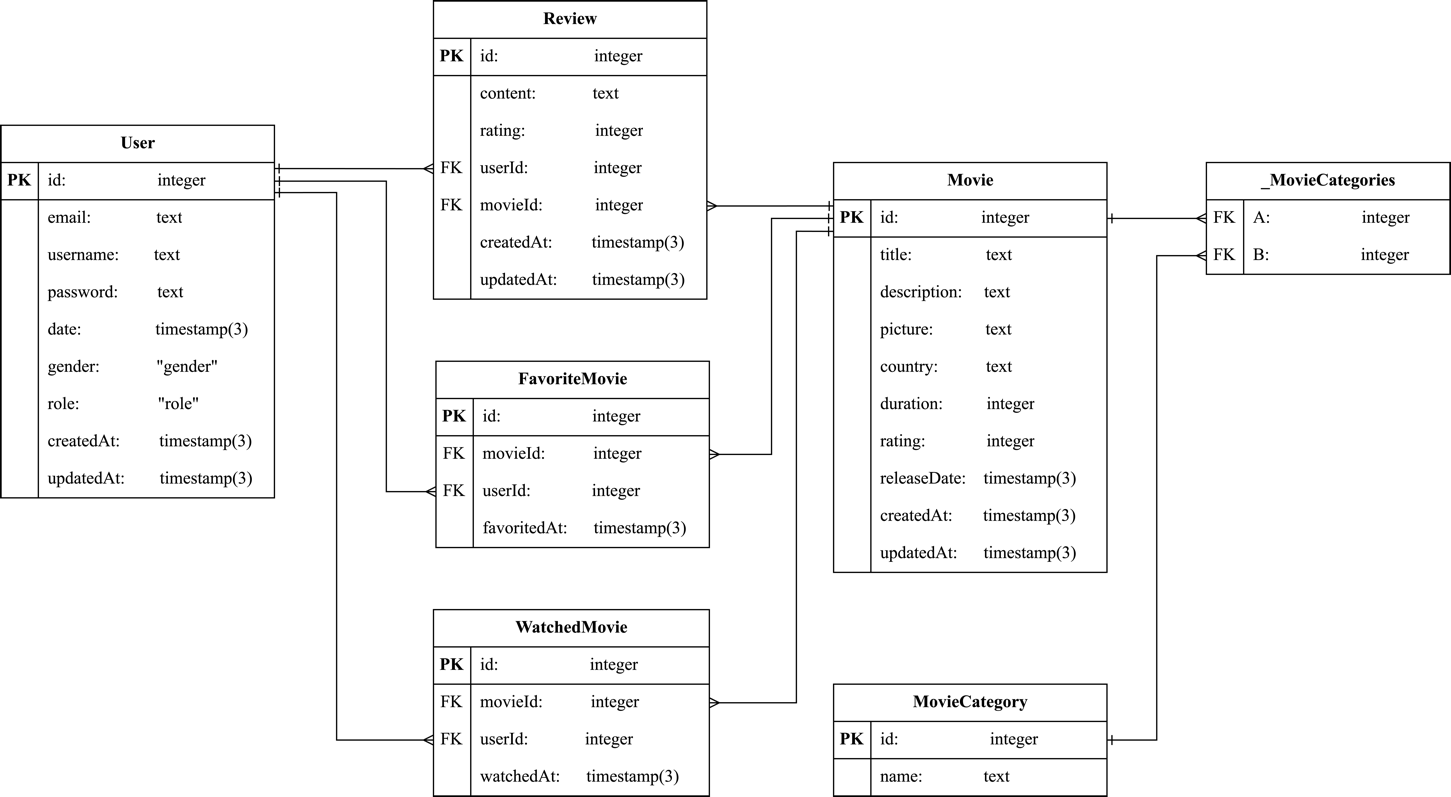


Рисунок 2.4 – Модель базы данных

Таким образом, проектирование базы данных обеспечивает хранение информации о пользователях, фильмах и других сущностях, необходимых для работы приложения.

2.3.2 Описание базы данных

В таблице «Users» хранятся все зарегистрированные пользователи. Описание структуры таблицы «Users» представлено в таблице 2.1.

Таблица 2.1 – Структура таблицы «Users»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | int, primary key | Идентификатор пользователя |
| email | text,unique | Электронная почта пользователя |
| username | text | Имя пользователя |
| password | text | Хеш пароля пользователя |
| date | timestamp | Дата рождения пользователя |
| gender | enum | Пол пользователя |
| watchedMovies | WatchedMovie[] | История просмотренных фильмов |
| favoriteMovies | FavoriteMovie[] | Избранные фильмы |
| reviews | Reviews[] | Комментарии к фильму |
| createAt | DateTime | Дата создания |
| updateAt | DateTime | Дата обновления |
| picture | text | Картинка пользователя |
| role | enum | Роль пользователя |

В таблице «Movie» хранится информация о фильмах. Описание структуры таблицы «Movie» представлено в таблице 2.2.

Таблица 2.2 – Структура таблицы «Movie»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | int, primary key | Идентификатор публикации |
| title | text | Название фильма |
| picture | text | Картинка фильма |
| description | text | Описание фильма |
| category | MovieCategory[] | Категории |
| reviews | Review[] | Комментарии |
| watchedBy | WatchedMovie[] | Просмотренные фильмы |
| favoritedBy | FavoriteMovie[] | Избранные фильмы |
| rating | int | Возрастной рейтинг |
| duration | int | Длительность фильма |
| country | text | Страна |
| createAt | DateTime | Дата создания |
| updateAt | DateTime | Дата обновления |

Таблица «MovieCategory» содержит категории фильмов. Её структура представлена в таблице 2.3.

Таблица 2.3 – Структура таблицы «MovieCategory»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | int, primary key | Идентификатор комментария |
| name | text | Название категории |
| movies | Movie[] | Список фильмов |

В таблице «WatchedMovie» содержится история просмотренных фильмов. Описание структуры таблицы «WatchedMovie» представлено в таблице 2.4.

Таблица 2.4 – Структура таблицы «WatchedMovie»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | int, primary key | Идентификатор просмотренного фильма |
| movieId | int, foreign key | Идентификатор фильма |
| userId | int, foreign key | Идентификатор пользователя |
| watchedAt | DateTime | Дата просмотра |

В таблице «FavoriteMovie» содержится избранные фильмы. Описание структуры таблицы «FavoriteMovie» представлено в таблице 2.5.

Таблица 2.5 – Структура таблицы «FavoriteMovie»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | int, primary key | Идентификатор просмотренного фильма |
| movieId | int, foreign key | Идентификатор фильма |
| userId | int, foreign key | Идентификатор пользователя |
| favoriteAt | DateTime | Дата просмотра |

В таблице «Review» комментарии к фильмам. Описание структуры таблицы представлено в таблице 2.6.

Таблица 2.6 – Структура таблицы «Review»

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип, ограничение целостности | Описание столбца |
| id | int, primary key | Идентификатор просмотренного фильма |
| movieId | int, foreign key | Идентификатор фильма |
| userId | int, foreign key | Идентификатор пользователя |
| content | text | Контент комментария |
| rating | int | Оценка комментария |
| createAt | DateTime | Дата создания |
| updateAt | DateTime | Дата обновления |

Таким образом при проектировании базы данных используется 6 таблиц, присутствует связь один-ко-многим и многие-ко-многим.

Такая архитектура базы данных предоставляет гибкость в управлении данными, высокую производительность и безопасность.

2.4 Контейнеризация

Контейнеризация мобильного приложения для рецензирования и оценки фильмов с использованием Docker обеспечивает изоляцию и консистентность среды выполнения для основных компонентов системы, включая серверную часть и базу данных. Применение Docker позволяет создать отдельные контейнеры для Nest.js сервера и PostgreSQL, что исключает конфликты зависимостей и упрощает управление версиями используемых технологий.

Использование Docker способствует унификации среды разработки и производственного развертывания, что снижает вероятность ошибок, связанных с различиями в конфигурациях. Это позволяет разработчикам быстро развертывать приложение на различных платформах без дополнительных настроек, обеспечивая стабильную работу в разных средах.

Контейнеризация также облегчает масштабирование приложения. В случае увеличения нагрузки, можно легко развернуть дополнительные экземпляры контейнеров сервера и базы данных, что обеспечивает гибкость и адаптивность системы к изменяющимся требованиям. Такой подход позволяет эффективно распределять ресурсы и поддерживать высокую производительность приложения.

Кроме того, Docker упрощает процесс обновления и поддержки приложения. Обновления серверной части или базы данных осуществляются путем замены соответствующих контейнеров на новые версии, что минимизирует время простоя и обеспечивает непрерывность работы сервиса. Контейнеризация также облегчает резервное копирование и восстановление данных, так как база данных находится в отдельном контейнере с управляемыми томами данных.

Использование Docker в данном проекте способствует созданию устойчивой и управляемой инфраструктуры, которая поддерживает эффективное взаимодействие между клиентской и серверной частями приложения. Это обеспечивает надежную работу системы, высокую производительность и удобство в управлении, что важно для обеспечения качественного пользовательского опыта и успешного функционирования мобильного приложения для рецензирования и оценки фильмов.

2.5 Проектирование основных алгоритмов

2.5.1 Алгоритм авторизации пользователей

Процесс авторизации пользователей в мобильном приложении для рецензирования и оценки фильмов включает несколько этапов, обеспечивающих безопасный и эффективный доступ к защищённым ресурсам приложения.

Первоначально пользователь вводит свои учетные данные, такие как электронная почта и пароль, через интерфейс мобильного приложения. Клиентская часть приложения отправляет запрос на серверную часть для аутентификации пользователя. Серверная часть, реализованная на Nest.js, получает запрос и извлекает информацию о пользователе из базы данных PostgreSQL с помощью сервиса UserService. Далее сервер сравнивает предоставленный пароль с сохранённым в базе данных. Если пароли совпадают, сервер генерирует JSON Web Token (JWT), который содержит идентификатор пользователя, его электронную почту и роль.

Сгенерированный JWT отправляется обратно клиенту, который сохраняет его для последующих взаимодействий с сервером. При каждом последующем запросе к защищённым маршрутам клиент включает этот токен в заголовок запроса.

На сервере механизм AuthGuard перехватывает входящие запросы к защищённым ресурсам. Он извлекает JWT из заголовков запроса и проверяет его подлинность и валидность с помощью секрета, указанного в конфигурации. Если токен действителен, AuthGuard добавляет информацию о пользователе в объект запроса и позволяет продолжить обработку запроса. В случае отсутствия токена или его недействительности сервер возвращает ошибку UnauthorizedException, что предотвращает несанкционированный доступ к защищённым ресурсам.

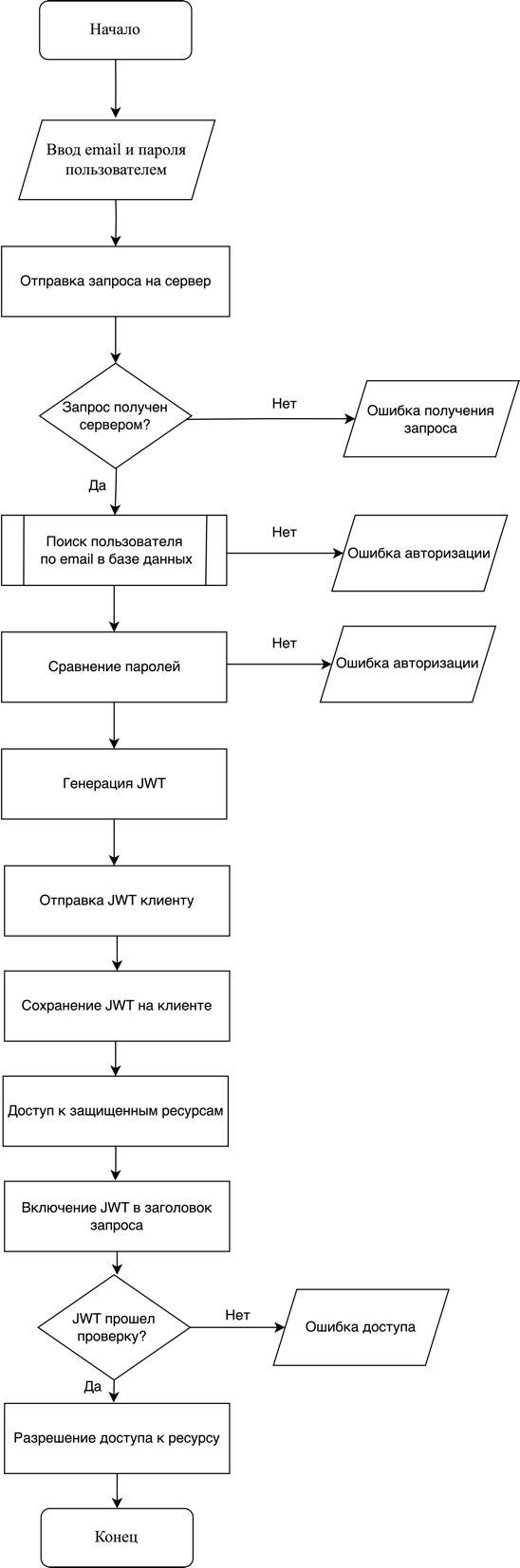


Рисунок 2.5 – Схема алгоритма авторизации пользователей

Рисунок 2.5 иллюстрирует последовательность шагов алгоритма авторизации пользователей, начиная с входа в систему и заканчивая проверкой JWT для доступа к защищённым ресурсам.

2.5.2 Алгоритм работы страницы просмотренного фильма

Процесс работы страницы просмотренного фильма в мобильном приложении начинается с того, что пользователь открывает страницу конкретного фильма. Приложение отправляет запрос на сервер с токеном аутентификации, чтобы отметить этот фильм как просмотренный. Сервер проверяет валидность токена и идентификатор пользователя, после чего обращается к базе данных, чтобы определить, есть ли данный фильм уже в списке просмотренных фильмов пользователя. Если фильм уже присутствует в списке, сервер удаляет существующую запись и добавляет новую запись с текущей датой просмотра, тем самым перемещая фильм в конец списка, что отражает его последнее посещение. Если записи о фильме нет, сервер просто добавляет новую запись в базу данных. После успешного обновления или добавления записи сервер отправляет подтверждение обратно в приложение, которое обновляет интерфейс страницы «Просмотренные фильмы», отображая актуальный список фильмов. В случае возникновения ошибок на любом этапе, приложение уведомляет пользователя о необходимости повторной попытки.

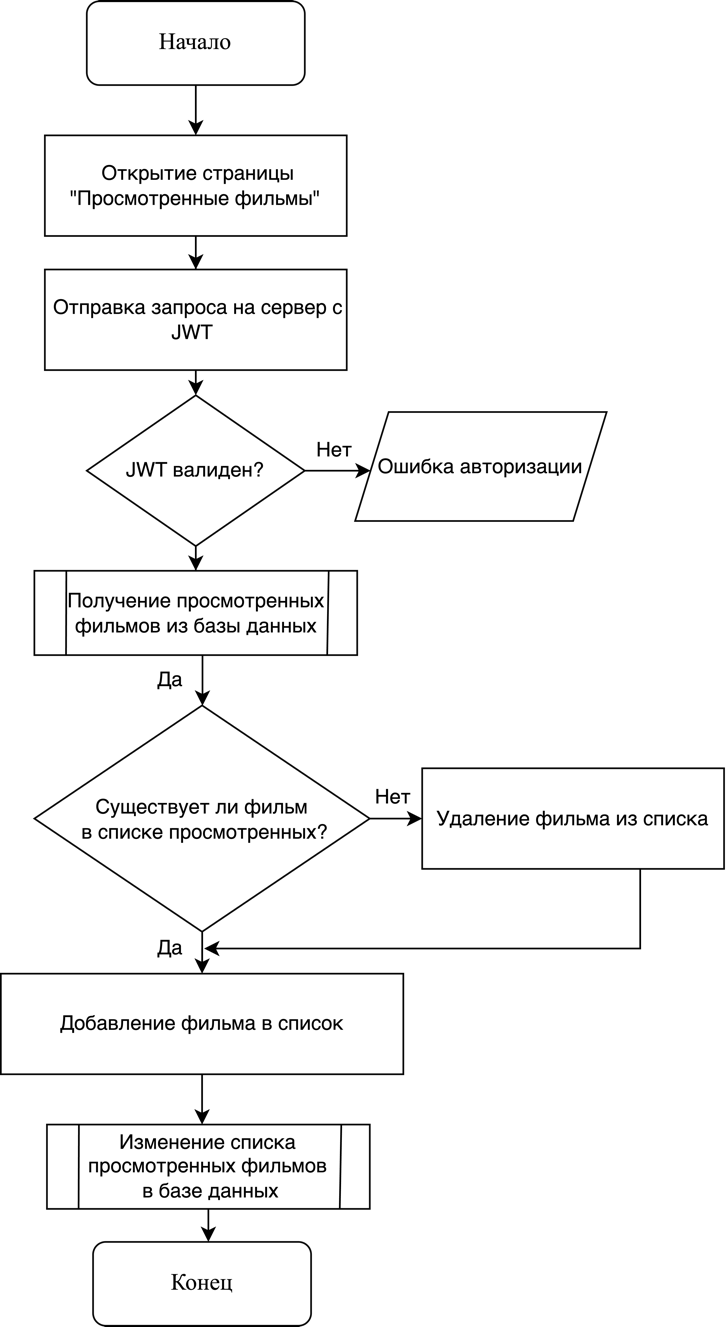


Рисунок 2.6 – Алгоритм работы страницы просмотренного фильма

Рисунок 2.6 иллюстрирует последовательность шагов алгоритма работы страницы просмотренного фильма, включая проверку наличия фильма в списке просмотренных и соответствующие действия по его удалению и добавлению.

2.6 Вывод по разделу

Таким образом, проект представляет собой интегрированную систему для рецензирования и оценки фильмов, предоставляющую пользователям удобный мобильный интерфейс на базе React Native. Клиентская часть обеспечивает кроссплатформенную доступность, интуитивно понятный дизайн и эффективное управление состоянием через Redux и RTK Query, что позволяет пользователям легко искать фильмы, оставлять оценки и писать рецензии. Серверная часть, реализованная с использованием Nest.js, обрабатывает запросы, обеспечивает безопасность и масштабируемость системы, взаимодействуя с централизованной базой данных PostgreSQL. PostgreSQL гарантирует надежное хранение данных, высокую производительность и быстрый доступ к информации о пользователях, фильмах и рецензиях. Дополнительные меры безопасности, такие как HTTPS и middleware для обработки ошибок, повышают защищенность платформы. Разделение на клиентскую и серверную части способствует гибкости и легкости в дальнейшем расширении функционала приложения.

3 Разработка мобильного приложения

Разработка мобильного приложения была осуществлена с использованием современных технологий и подходов, направленных на обеспечение эффективной работы и удобства пользователей. React Native для клиентской части приложения, чтобы создать интерактивный и отзывчивый пользовательский интерфейс. Для управления состоянием приложения мы использовали Redux Tool Kit. На серверной стороне используется Node.js с использованием фреймворка NestJs, благодаря которому мы создавали контроллеры, сервисы, middlewares, guards. Для работы с базой данных использовалась библиотека Prisma.

3.1 Разработка серверной части

3.1.1 Cтруктура серверной части приложения

На рисунке 3.1 представлена структура серверной части приложения, включающая в себя директории, конфигурационные файлы и файлы скриптов.

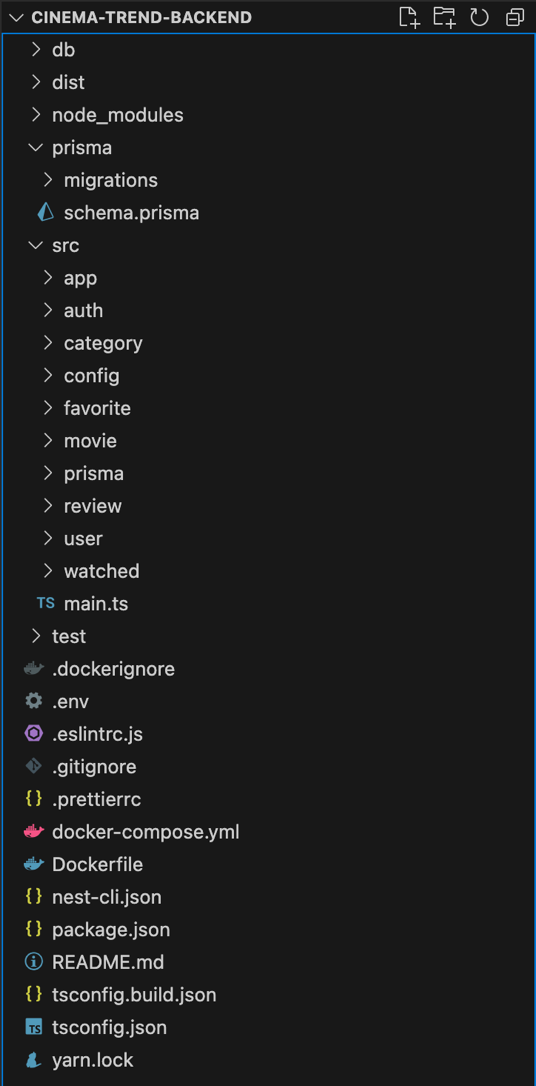
****

Рисунок 3.1 – Структура серверной части

В директории «prisma» содержатся модели базы данных, которые используются для доступа к данным и для создания новых данных. Листинг моделей представлен в приложении А.

В файле main.ts мы создаем экземпляр приложения NestJS, используя основной модуль AppModule, который содержит всю логику нашего приложения. Мы также включаем поддержку CORS для обеспечения возможности взаимодействия с фронтендом, работающим на другом источнике, что важно для корректной работы с различными веб-приложениями. После этого приложение запускается на порту 4200, что позволяет пользователям взаимодействовать с сервером через веб-браузер. Используя TypeScript, мы получаем типизацию данных, что повышает безопасность кода и помогает избежать ошибок на этапе разработки. Этот подход обеспечивает надежную настройку и запуск серверной части приложения.

Структура серверного приложения является фундаментальным элементом его организации и эффективного функционирования. Разбивка на папки помогает нам систематизировать код, делая его более читаемым, понятным и легко поддерживаемым. Давайте рассмотрим пример директории user из раздела src более подробно.

Директория user состоит из следующих файлов:

* + user.controller.ts
  + user.dto.ts
  + user.module.ts
  + user.service.ts

Содержимое представлено на рисунке 3.2.

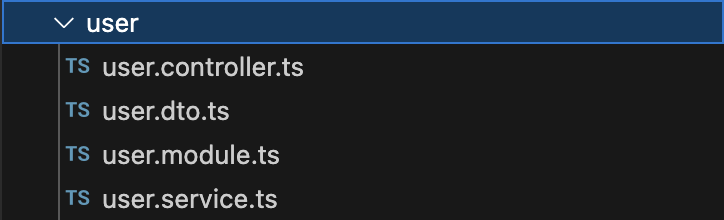


Рисунок 3.2 – Содержимое директория «user»

Файл user.controller.ts в проекте на NestJS отвечает за обработку HTTP-запросов, которые касаются пользователей. В этом файле реализованы различные маршруты для создания, получения и обновления информации о пользователях. Он использует декораторы и механизмы NestJS для обработки запросов и работы с данными. Листинг файла представлен в приложении Б.

В этом файле используются декораторы @Controller, которые являются основой для определения маршрутов в приложении NestJS. Контроллеры выполняют роль обработчиков входящих HTTP-запросов, связывая их с соответствующими методами. В NestJS контроллеры отвечают за управление логикой маршрутов, например, создание, получение или обновление данных, а также за вызов соответствующих сервисов, которые инкапсулируют бизнес-логику приложения.

Файл user.dto.ts содержит классы, которые определяют структуру данных для передачи между клиентом и сервером в приложении NestJS. DTO (Data Transfer Object) служат для валидации и типизации данных, которые приходят в запросах, а также для обеспечения правильной структуры данных, отправляемых в ответах. Листинг файла представлен в приложении В.

В данном файле используются декораторы из библиотеки class-validator, которая помогает с валидацией данных. Библиотека позволяет легко проверять, что данные, передаваемые в запросах или ответах, соответствуют необходимым требованиям. Это делает приложение более безопасным и защищенным от ошибок, связанных с неправильными данными.

Файл user.module.ts в проекте на NestJS представляет собой модуль, который инкапсулирует всю логику, связанную с пользователями, и организует взаимодействие между различными компонентами, такими как контроллеры, сервисы и другие модули. Листинг файла представлен в приложении Г.

В файле user.module.ts используется декоратор @Module(), который настраивает и определяет, какие компоненты будут входить в данный модуль.

UserModule объединяет все необходимые компоненты для работы с пользователями, такие как контроллеры и сервисы. Он инкапсулирует всю логику, связанную с пользователями, и предоставляет ее другим частям приложения через экспортируемый сервис UserService. Модуль также упрощает подключение всех зависимостей и их правильное связывание между собой.

Таким образом, файл user.module.ts организует структуру приложения, обеспечивая правильную связь между контроллером, сервисом и базой данных, а также облегчает масштабируемость и поддержку проекта.

Файл user.service.ts содержит логику, которая отвечает за обработку запросов и операций, связанных с пользователями, таких как создание, обновление и поиск пользователей в базе данных. Этот файл выполняет ключевую роль в бизнес-логике приложения и управляет взаимодействием с базой данных и внешними сервисами. Листинг файла представлен в приложении Д.

Основные компоненты файла:

* + Декоратор @Injectable(): Он используется для того, чтобы класс мог быть внедрен как зависимость в другие части приложения (например, в контроллеры). Это означает, что UserService будет доступен для других модулей и контроллеров.
  + Конструктор: В конструкторе UserService инициализируется PrismaService для работы с базой данных. Также настраивается Firebase Admin SDK для работы с хранилищем изображений пользователей. Firebase используется для загрузки и хранения аватаров пользователей.

user.service.ts отвечает за все операции, связанные с пользователями, и обеспечивает их надежную обработку, включая взаимодействие с базой данных и внешними сервисами, такими как Firebase для хранения изображений.

В целом, структура серверного приложения обеспечивает нам удобство, организацию и масштабируемость кода, что является ключевым для разработки надежного и эффективного приложения. Каждая директория хранит внутри себя файлы, которые выполняют определенную функцию и способствует улучшению архитектуры приложения в целом.

В файле package.json хранится список зависимостей (библиотек), необходимых приложению NodeJS для работы, содержимое файла представлено в приложении Е.

3.1.2 Конкретные реализации

Для понимания, как работает авторизация в приложении, разберем AuthService, расположенный в соответствующей директории. Этот сервис отвечает за регистрацию и авторизацию пользователей. Рассмотрим функцию signIn, которая предназначена для входа пользователя в систему. Исходный код функции представлен в листинге файла в приложении З.

В данной функции происходит следующее: из входящих данных берутся почта и пароль пользователя. Сначала система проверяет наличие пользователя в базе данных через метод findOne из UserService. Если пользователь найден, происходит сравнение введенного пароля с паролем из базы данных. В случае несоответствия паролей вызывается исключение UnauthorizedException. При успешной проверке формируется токен JWT (JSON Web Token) с данными пользователя, такими как id, email и роль. Этот токен возвращается клиенту для последующей аутентификации.

Таким образом, функция signIn обеспечивает безопасный процесс авторизации пользователя, включая проверку данных и генерацию токена. Логика, связанная с регистрацией пользователей, также реализована в этом сервисе.

Теперь рассмотрим WatchedService, который отвечает за управление просмотренными фильмами пользователей. Разберем функцию createWatched, которая добавляет фильм в список просмотренных или перемещает его в конец списка. Исходный код функции представлен в листинге файла в приложении И.

Функция принимает id пользователя и id фильма. Сначала система проверяет, существует ли запись о просмотренном фильме для данного пользователя с использованием метода findFirst. Если запись найдена, она удаляется из базы данных с помощью метода delete. После этого создается новая запись через функцию createWatchedMovie, что позволяет переместить фильм в конец списка просмотренных. Это гарантирует актуальность порядка и отсутствие дубликатов в коллекции.

Теперь рассмотрим функцию getWatched, которая получает список просмотренных фильмов пользователя. Исходный код представлен в листинге файла в приложении И.

Эта функция обращается к базе данных через Prisma для получения всех просмотренных фильмов, отсортированных по дате просмотра в порядке убывания. Затем из результатов извлекаются id фильмов, после чего сервис MovieService возвращает полную информацию о фильмах по их идентификаторам.

Таким образом, сервис WatchedService обеспечивает добавление фильмов в список просмотренных, обновление их порядка и получение актуальной информации о просмотренных фильмах пользователя.

Оба сервиса, AuthService и WatchedService, играют важную роль в серверной части приложения. Первый отвечает за аутентификацию и безопасность пользователей, а второй позволяет эффективно управлять списком просмотренных фильмов, обеспечивая высокую производительность и стабильность работы приложения.

3.2 Разработка клиентской части

Разработка клиентской части приложения включает в себя создание интерфейса пользователя (UI) и реализацию взаимодействия с сервером для получения и отправки данных. На рисунке 3.6 изображена структура клиентской части разрабатываемого приложения.

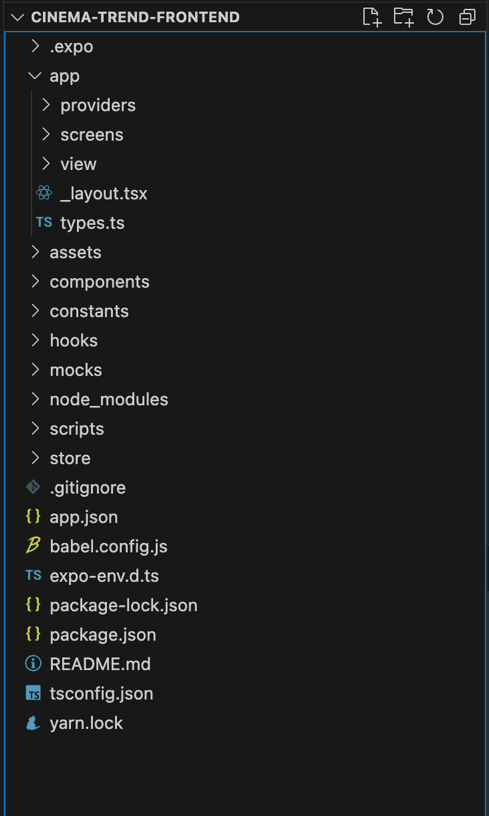


Рисунок 3.5 – Структура клиентской части

Клиентская часть приложения написана на React Native. Архитектура клиентской части нашего проекта представляет собой тщательно спроектированную структуру. Эта модульная организация не только обеспечивает удобство масштабирования, но и позволяет нам эффективно управлять разработкой и поддержкой приложения.

Директория store в вашем проекте использует RTK Query для работы с асинхронными запросами и управления состоянием API. Структура этой директории обеспечивает эффективную организацию кода, используя Redux Toolkit для управления состоянием и централизованного управления запросами. Содержимое представлено на рисунке 3.6.

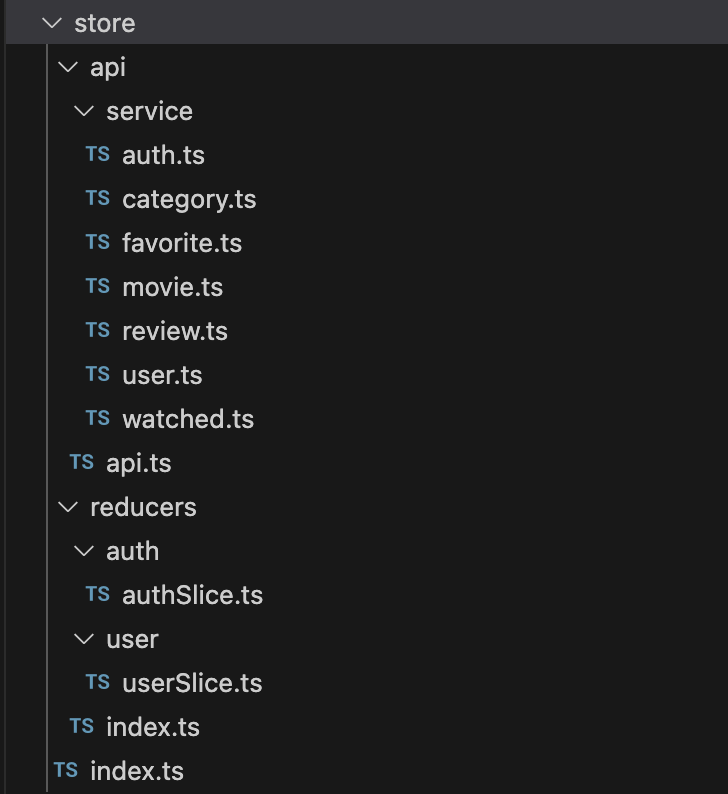


Рисунок 3.6 – Содержимое директории «store»

api — Папка, в которой реализована логика работы с API с использованием RTK Query. Это упрощает выполнение асинхронных запросов (GET, POST, PUT, DELETE) и автоматическое кэширование данных. Внутри есть:

* + service — Здесь хранится файл с конфигурацией RTK Query, который определяет базовые параметры запроса, такие как URL и обработка заголовков.
  + api.ts — В этом файле создается экземпляр API с помощью createApi из RTK Query. Здесь описывается baseQuery, который выполняет запросы к серверу, и prepareHeaders, который добавляет токен аутентификации в заголовки для авторизованных запросов. Также определяется список tagTypes (такие как "Review", "Watched", "Favorite", "User", "Movie"), которые используются для кэширования и обновления данных.

reducers — Здесь находятся редьюсеры для управления состоянием приложения. Например: auth — Редьюсер для работы с аутентификацией (например, хранение токена пользователя). user — Редьюсер для управления состоянием данных о пользователе.

index.ts — Этот файл отвечает за экспорт глобального состояния и конфигурацию Redux store.

Директория «components». Здесь мы храним кастомные компоненты, которые включают в себя стилевые компоненты. Это придает уникальный вид каждому элементу пользовательского интерфейса. Например, в этой папке мы создаем основные элементы, такие как карточки для фильмов, поиск, текст и т.д. Содержимое представлено на рисунке 3.7.

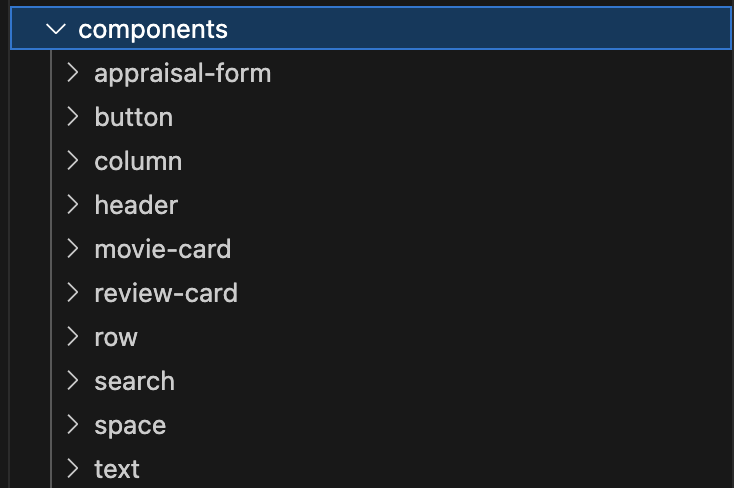


Рисунок 3.7 – Содержимое директории «components»

Директория «hooks». Эта папка содержит типизированные кастомные хуки, которые предоставляют нам возможность отделить бизнес-логику от дизайна. Это позволяет нам повторно использовать логику и легко поддерживать ее. Содержимое представлено на рисунке 3.8.

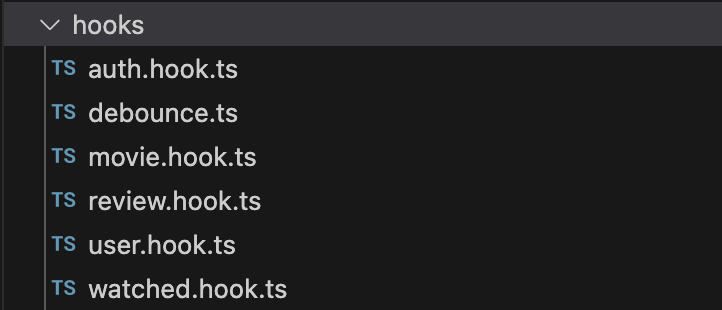


Рисунок 3.8 – Содержимое директории «hooks»

Директория «screens». Эта папка содержит все экраны приложения. Она позволяет нам разделить интерфейс на отдельные части для более легкого управления и разработки. Содержимое представлено на рисунке 3.9.

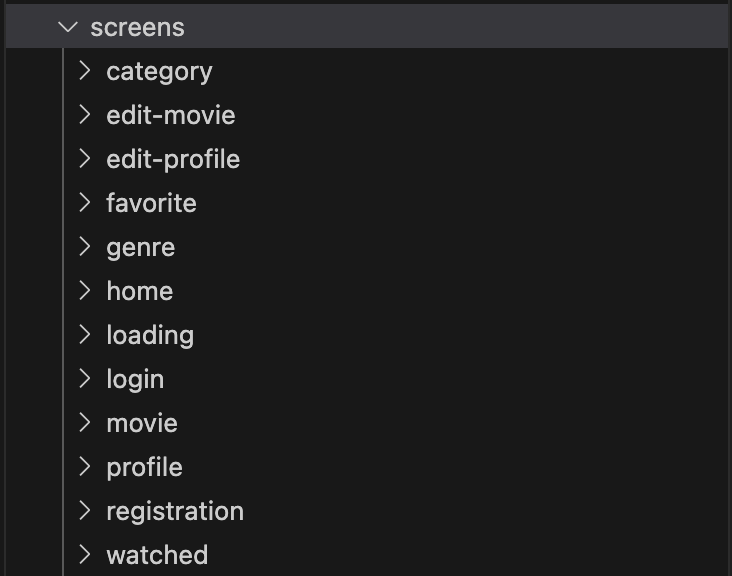


Рисунок 3.9 – Содержимое директории «screens»

Директория assets предназначена для хранения статических файлов, которые используются в приложении, таких как изображения, шрифты и другие ресурсы. Внутри этой директории находятся подкаталоги, каждый из которых отвечает за хранение определенных типов файлов:

* + fonts — Папка для хранения файлов шрифтов, которые используются в приложении.
  + images — Папка, предназначенная для хранения изображений, таких как фотографии, картинки и иконки, которые используются в интерфейсе приложения.
  + svg — Папка для хранения SVG-компонентов, которые могут быть использованы для векторных изображений, иконок или графики, поддерживающей масштабируемость без потери качества.

Содержимое представлено на рисунке 3.10.

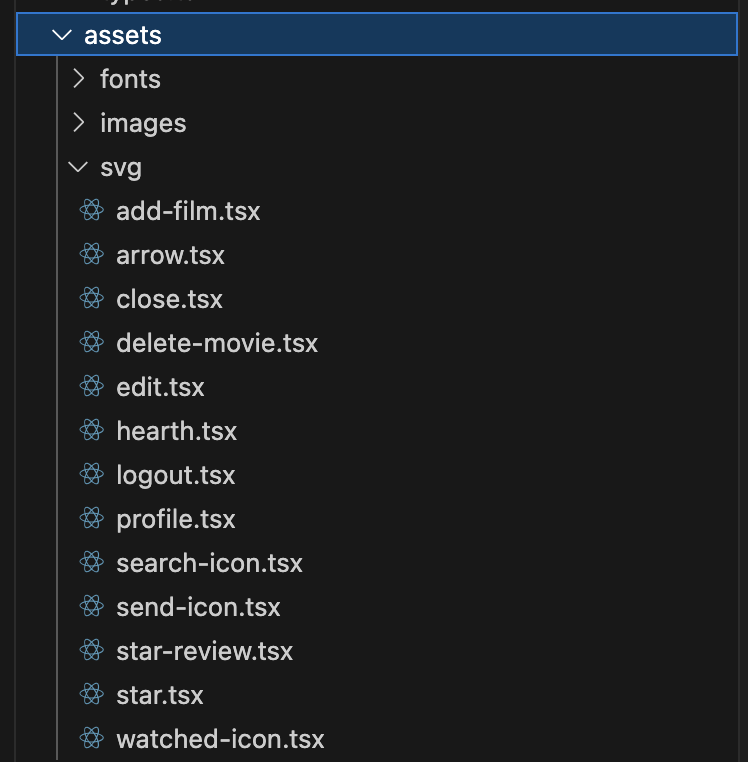


Рисунок 3.10 – Содержимое директории «assets»

Директория «views». Позволяет интегрировать готовые представления в наши экраны, что значительно упрощает разработку и поддержку кода.

Эта трехуровневая архитектура разделена на дизайн, бизнес-логику и данные, что обеспечивает четкое разделение ответственности и повышает эффективность разработки и поддержки приложения.

3.3 Контейнеризация приложения с использованием Docker

Для обеспечения надежного и удобного развертывания серверной части и базы данных мобильного приложения была применена технология контейнеризации с использованием Docker. Контейнеризация позволяет изолировать каждую компоненту системы в отдельных контейнерах, обеспечивая согласованность среды выполнения и упрощая управление зависимостями.

В рамках проекта были созданы два основных контейнера: один для серверной части приложения на NestJS, и другой для базы данных PostgreSQL. Каждый из этих контейнеров работает независимо, что предотвращает конфликты зависимостей и обеспечивает гибкость при обновлении отдельных компонентов системы. Серверный контейнер настроен для автоматического запуска приложения при старте контейнера, а контейнер с базой данных обеспечивает постоянное хранение данных через специальные тома (volumes), что гарантирует сохранность информации даже при перезапуске контейнеров.

Для управления взаимодействием между контейнерами и упрощения процесса развертывания был использован Docker Compose. С помощью Docker Compose все необходимые компоненты приложения можно запустить одной командой, что значительно ускоряет процесс развертывания и облегчает масштабирование системы при необходимости увеличения нагрузки. Контейнеры подключены к общей сети, что позволяет им эффективно взаимодействовать друг с другом без дополнительных настроек сети.

Кроме того, Docker обеспечивает простоту обновления и поддержки приложения. Обновления серверной части или базы данных выполняются путем замены соответствующих контейнеров на новые версии, что минимизирует время простоя и обеспечивает непрерывность работы сервиса. Контейнеризация также облегчает процесс резервного копирования и восстановления данных, поскольку база данных хранится в отдельном контейнере с управляемыми томами данных.

Использование Docker способствует созданию устойчивой и управляемой инфраструктуры, которая поддерживает эффективное взаимодействие между серверной частью и базой данных приложения. Это обеспечивает стабильную работу системы, высокую производительность и удобство в управлении, что важно для обеспечения качественного пользовательского опыта и успешного функционирования мобильного приложения для рецензирования и оценки фильмов.

Подробная конфигурация Docker, включая файл docker-compose.yml, представлена в приложении Ж.

3.4 Вывод по разделу

В данном разделе была подробно описана разработка мобильного приложения для рецензирования и оценки фильмов с использованием современных технологий и подходов. Клиентская часть приложения была создана на основе React Native, что обеспечило интерактивный и отзывчивый пользовательский интерфейс. Для управления состоянием приложения использовался Redux Tool Kit, позволяющий эффективно обрабатывать данные и обеспечивать плавную работу интерфейса. На серверной стороне была реализована архитектура с использованием Node.js и фреймворка NestJS, что позволило организовать контроллеры, сервисы, middlewares и guards, обеспечивая модульность и масштабируемость системы. Для взаимодействия с базой данных применялась библиотека Prisma, которая обеспечила удобное и типизированное взаимодействие с PostgreSQL, гарантируя надежное хранение и доступ к данным о пользователях, фильмах и их оценках. Внедрение контейнеризации с использованием Docker и Docker Compose позволило изолировать серверную часть и базу данных в отдельных контейнерах, обеспечивая согласованность среды выполнения и упрощая процесс развертывания. Это также способствовало легкому масштабированию приложения и повышению его надежности. Структура серверного и клиентского приложения была организована таким образом, чтобы обеспечить четкое разделение обязанностей, улучшить читаемость и поддержку кода, а также упростить дальнейшее развитие функционала. Подробная конфигурация Docker, включая файл docker-compose.yml, представлена в приложении Ж. В целом, разработка мобильного приложения была выполнена с учетом требований к удобству пользователей, безопасности данных и масштабируемости системы, что обеспечивает надежную и эффективную работу приложения, отвечающего современным стандартам и потребностям пользователей.

4 Тестирование мобильного приложения

4.1 Валидация форм ввода

В этом разделе подробно рассмотрим процесс тестирования завершенного приложения, сосредоточившись на клиентской части.

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

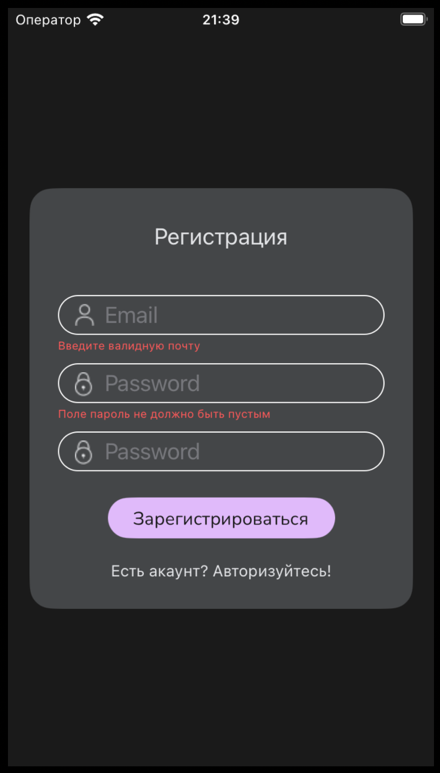


Рисунок 4.1 – Вывод ошибки некорректных данных при регистрации

При авторизации, если пользователя не существует, также будет выведено сообщение об ошибке. Вывод ошибки представлен на рисунке 4.2.

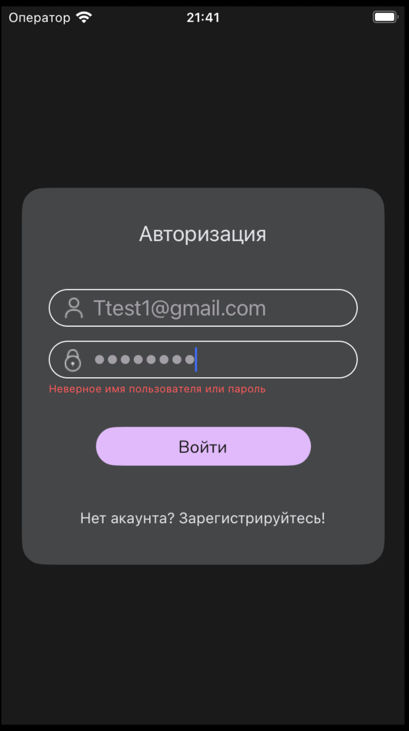


Рисунок 4.2 – Вывод ошибки некорректных данных при авторизации

Также, если пароли не совпадают, то будет выведена ошибка. Вывод ошибки представлен на рисунке 4.3.

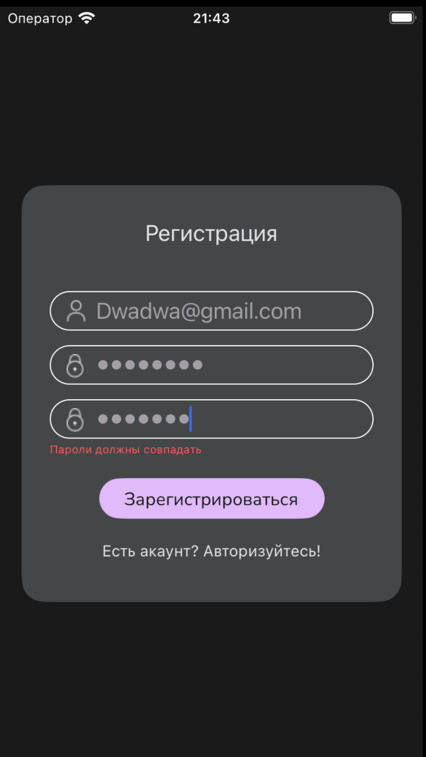


Рисунок 4.3 – Вывод ошибки паролей на экране регистрации

При указания в поле email значение, которое не имеет схожести со структурой email, также выдаст соответствующую ошибку. Вывод ошибки представлен на рисунке 4.4.

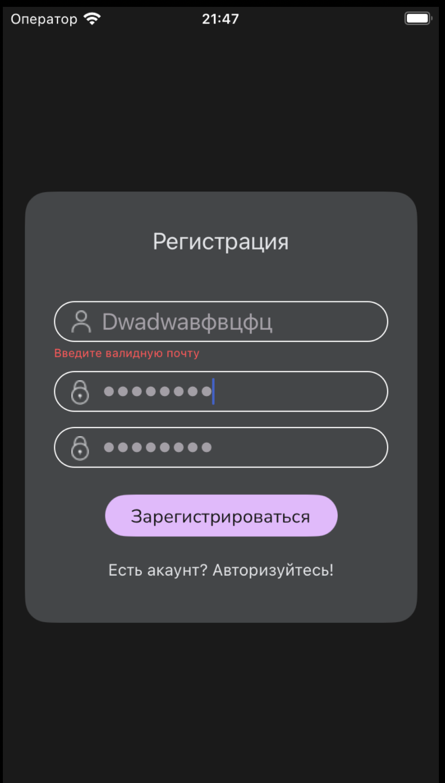


Рисунок 4.4 – Вывод ошибки почты на экране регистрации

Если пользователь является модератором и перейдет с главного меню в раздел добавления фильма и введет некорректные данные, или попытается отправить пустые поля, то ему выдаст соответствующие ошибки. Вывод ошибки представлен на рисунке 4.5.

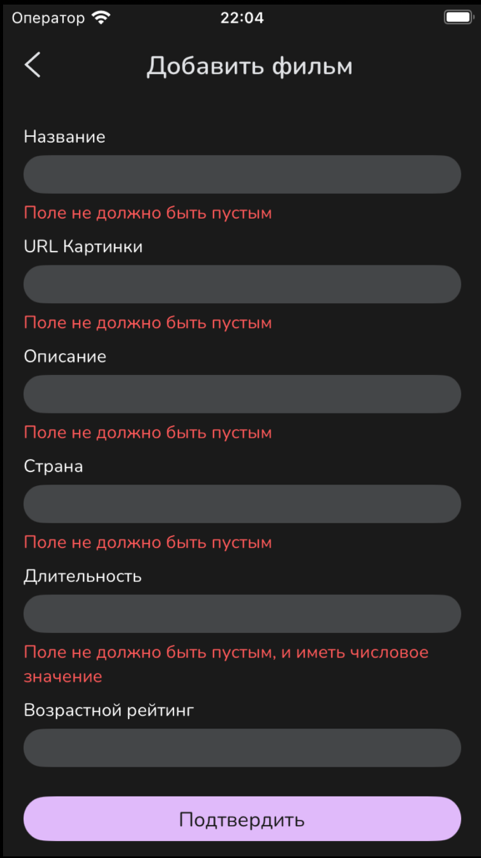


Рисунок 4.5 – Вывод ошибки о некорректных данных на экране добавления фильмов

Точно такая же ситуация произойдет на экране редактирования фильмов. Если модератор решить обновить данные, указав не валидное значение в поле, то получит ошибку.

4.2 Вывод по разделу

В результате тестирования клиентской части приложения было подтверждено его высокое качество и надежность. Все функциональные возможности, включая регистрацию, авторизацию, создания и редактирования фильмов, успешно прошли проверку. Обработка некорректных данных и пустых полей была реализована эффективно, что гарантирует удобство использования и безопасность приложения для всех пользователей.

5 Руководство пользователя

5.1 Руководство пользователя

В данном разделе представлены инструкции по использованию мобильного приложения для рецензирования и оценки фильмов для пользователя.

5.1.1 Регистрация пользователя

При регистрации пользователь должен ввести свою почту, пароль и подтвердить его. Форма регистрации представлена на рисунке 5.1.

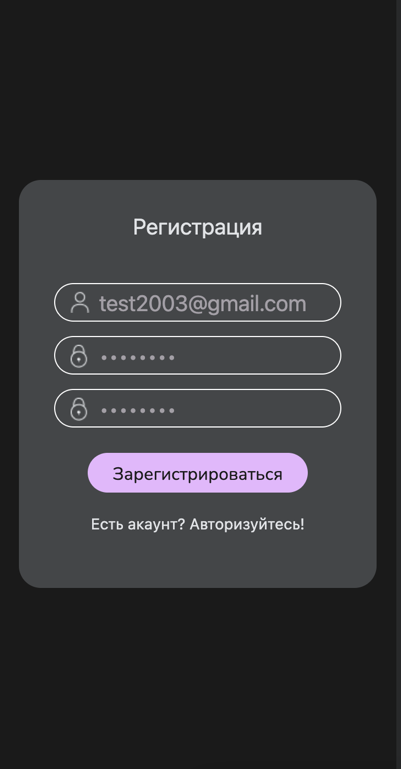


Рисунок 5.1 – Форма регистрации

После того, как пользователь нажмет на кнопку «Зарегистрироваться», будет отправлен запрос на регистрацию, на стороне сервера выполнится валидация всех полей. В случае ошибки, она будет выведена, а в случае успеха будет перенаправление на страницу авторизации.

5.1.2 Авторизация пользователя

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

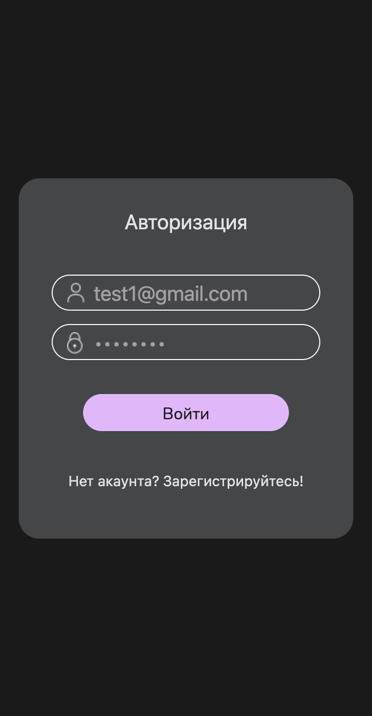


Рисунок 5.2 – Форма авторизации

В случае правильных учётных данных происходит перенаправление на главную страницу приложения.

5.1.3 Главная страница приложения

Попадая на главную страницу приложения, пользователь видит отображение несколько топов по фильмам, поле поиска, вариативность выбора жанра фильма, а также фильмы, страницы которых он посетил. В шапке справа находится кнопка профиля, слева кнопка понравившихся фильмов. Главная страница приложения представлена на рисунке 5.3.

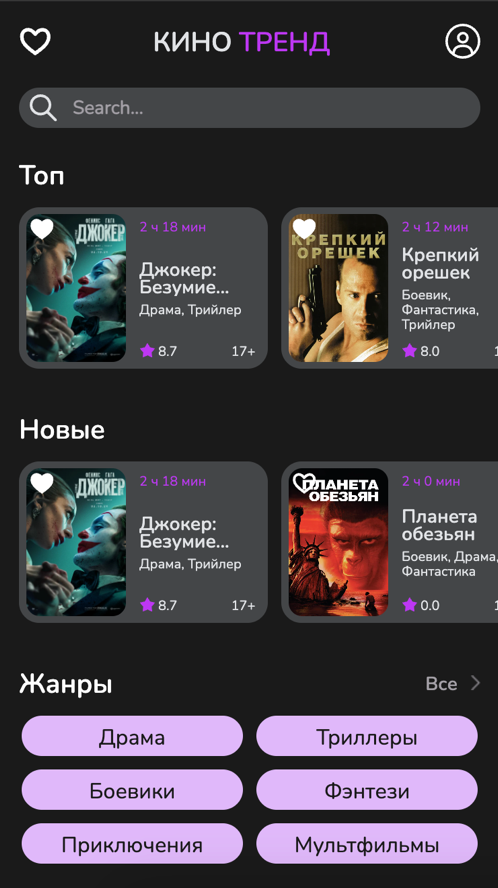


Рисунок 5.3 – Главная страница приложения

Поиск позволяет делать нам запрос к серверу для получения списка фильмов в названии которых содержится введенное значение. Демонстрация работоспособности поиска представлена на рисунке 5.4.

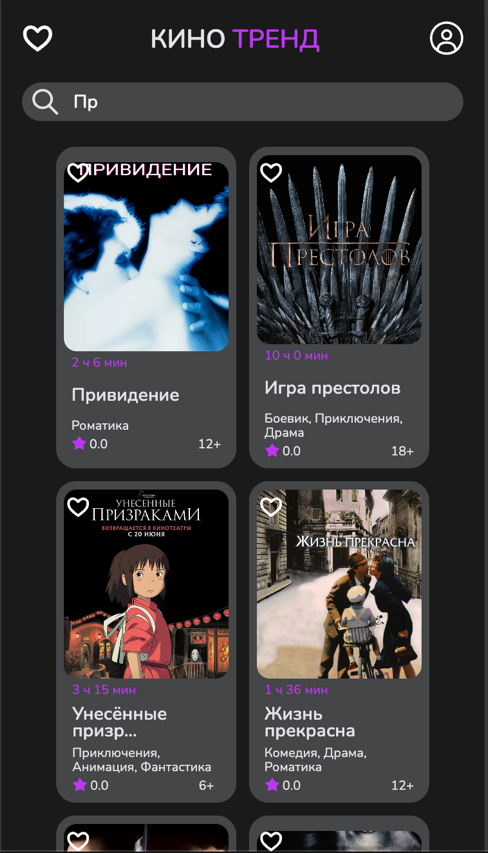


Рисунок 5.4 – Демонстрация работоспособности поиска

В правом верхнем углу, на шапке, находится иконка пользователя. Нажав на нее, попадем на страницу пользователя.

5.1.4 Страница пользователя

Попадая на страницу пользователя, можно увидеть иконку пользователя, имя пользователя, его почту при регистрации, кнопку выхода из аккаунта, кнопку редактирования пользователя, а также, ниже, два раздела:

* + - понравившийся фильмы
    - просмотренные страницы фильмов

Страница пользователя представлена на рисунке 5.5.

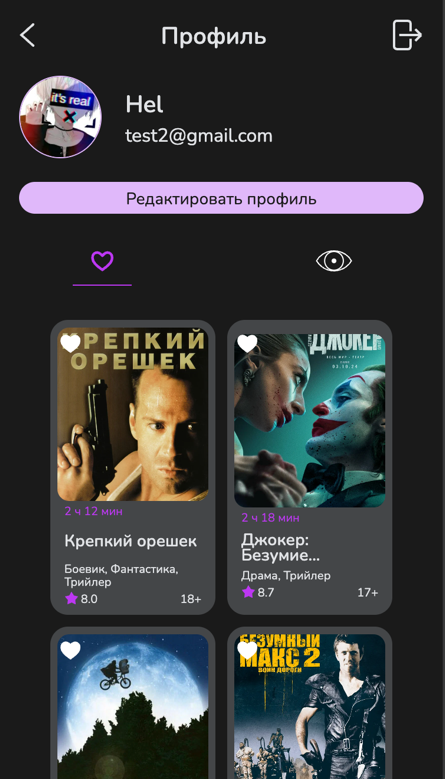


Рисунок 5.5 – Страница пользователя

Далее нажмем на кнопку «Редактировать профиль» и рассмотрим эту страницу.

5.1.5 Страница редактирования пользователя

На экране редактирования пользователя, можно изменить его имя, пол, аватар пользователя и год рождения. Страница редактирования пользователя представлена на рисунке 5.6.

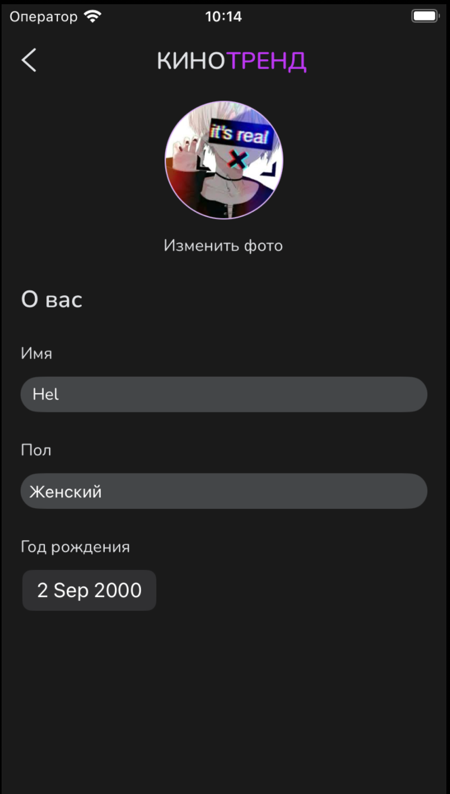


Рисунок 5.6 – Страница редактирования пользователя

Все изменения отправляются на backend после окончания изменения наших данных.

Вернемся на главную страницу нашего приложения, с помощью кнопки назад и рассмотрим другие страницы нашего приложения.

5.1.6 Страница избранных фильмов пользователя

На главной страницы приложения в левом верхнем углу можно увидеть кнопку форме сердца. Нажав на нее, откроется список фильмов, которые отметил пользователь, как понравившиеся. Такой же раздел можно увидеть на странице профиля пользователя. Страница избранных фильмов представлена на рисунке 5.7.

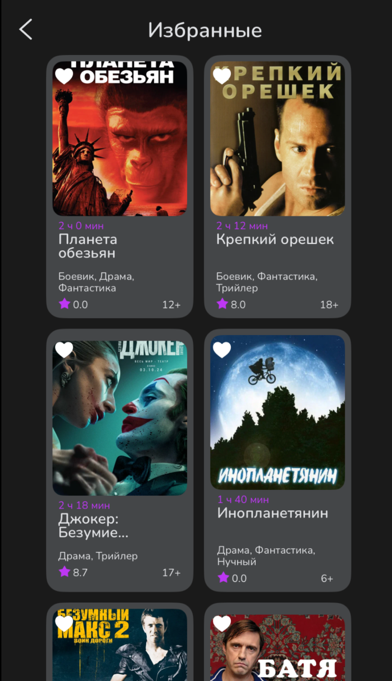


Рисунок 5.7 – Страница избранных фильмов

Добавить фильм в избранные можно нажав на иконку сердечка на карточке фильма, точно также можно и убрать фильм из избранных, нажав снова на иконку сердечка в карточке фильма.

Вернемся на главную страницу нашего приложения, с помощью кнопки назад и рассмотрим другие страницы нашего приложения.

5.1.7 Страница жанра фильмов

На главной странице приложения расположен раздел «Жанры» с кнопкой «Все» рядом. Пользователь может выбирать фильмы по жанрам как непосредственно на главной странице, так и через кнопку «Все», которая открывает полный список жанров. При выборе конкретного жанра отображаются все фильмы, относящиеся к этому жанру. Страница с выбранным жанром представлена на рисунке 5.8.

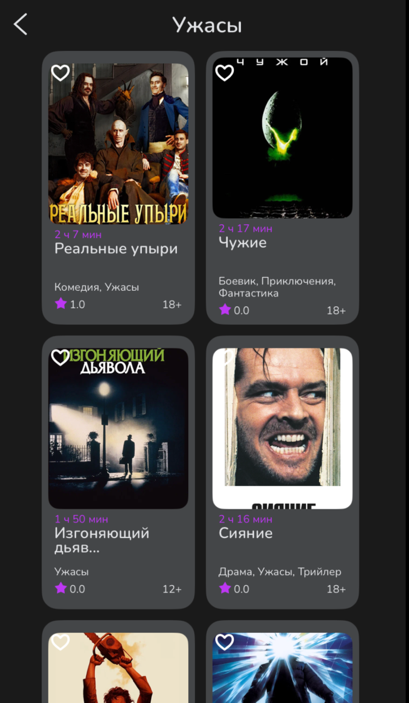


Рисунок 5.8 – Страница c выбранным жанром для фильмов

Далее перейдем на страницу самого фильма и рассмотрим эту ее поподробнее.

5.1.8 Страница фильма

На странице фильма можно наблюдать картинку фильма, оценку фильма, которая строится от среднего значения оставленных оценок других пользователей и их количество, название фильма, описание фильма, возрастной рейтинг, время длительности, страна производство, год производства, к каким жанрам он относится и описание фильма. Страница фильма представлена на рисунке 5.9.

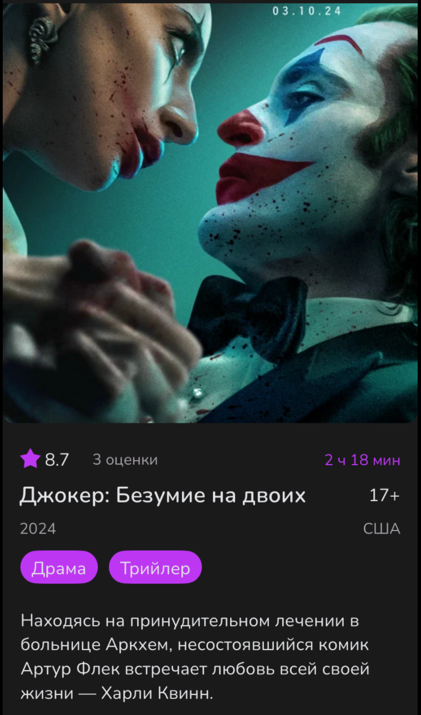


Рисунок 5.9 - Страница фильма

Ниже, можно увидеть раздел оценки фильма, где могут быть оценки, которые оставили другие пользователи, а также форма, для оставления оценки. Если пользователь уже оставил оценку, он больше не сможет оставлять оценку, пока не удалить старую. Раздел с формой оценки представлен на рисунке 5.10

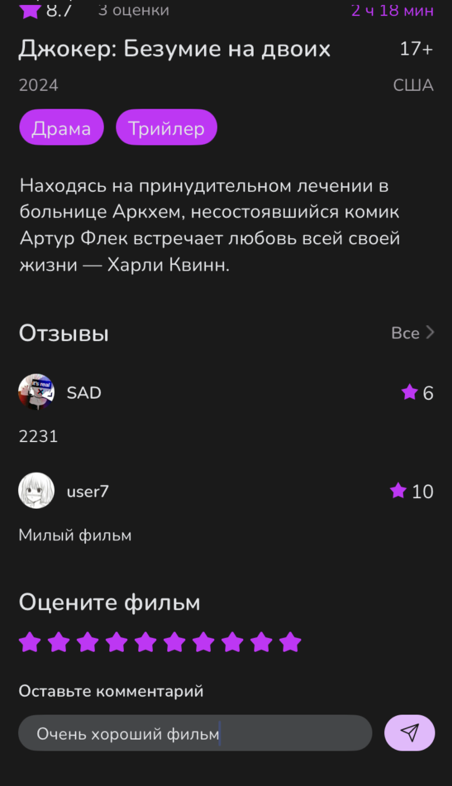


Рисунок 5.10 – Раздел с формой оценки

В этом разделе находится кнопка «все», нажав на которую, пользователь перейдет на страницу со всеми оценками, которые были оставлены на этот фильм. Страница всех оценок представлена на рисунке 5.11.

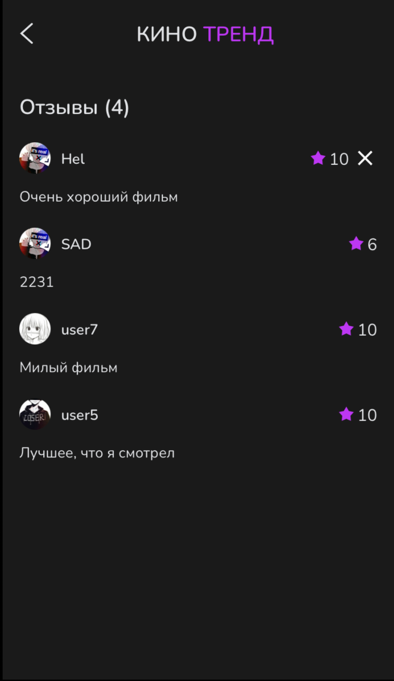


Рисунок 5.11 – Страница всех оценок

Так как пользователь уже оставили оценку, то он не может оставлять больше, но у него есть возможность удалить свою оценку, чтобы написать новую.

5.2 Руководство модератора

В этом разделе описаны функции и инструменты, доступные модераторам для управления контентом и поддержания качества информации в приложении.

5.2.1 Страница добавления и редактирования фильмов

Если пользователь является модератором, то он может добавлять новые фильмы и редактировать существующие. На главной странице приложения, в правом верхнем углу, на шапке, модератор может увидеть появившеюся иконку в виде плюса, если он на нее нажмет, то откроется страница добавления фильма. Страница добавления фильма представлена на рисунке 5.12.

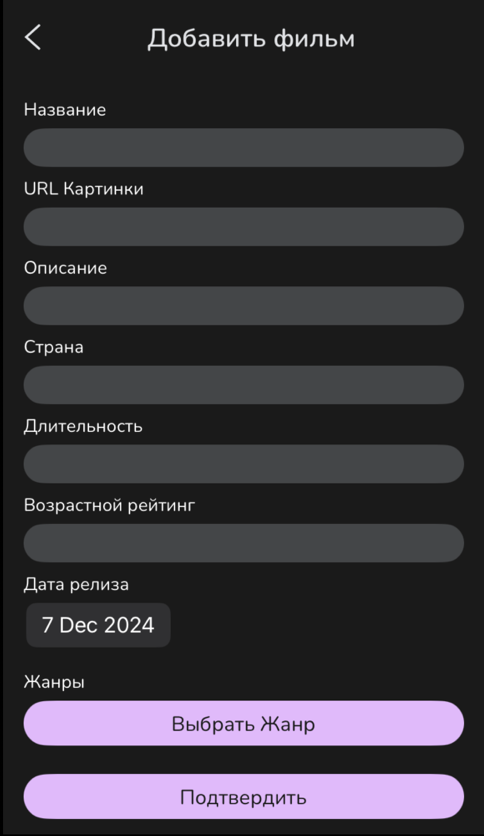


Рисунок 5.12 – Страница добавления фильма

Это дает возможность модератору контролировать содержание рецензий и удалять неподобающие из них.

5.2.2 Удаление оценок

Модератор на странице фильма может также очищать оценки и даже те, которые он не оставлял. Демонстрация возможности удаления оценок всех пользователей представлена на рисунке 5.13.

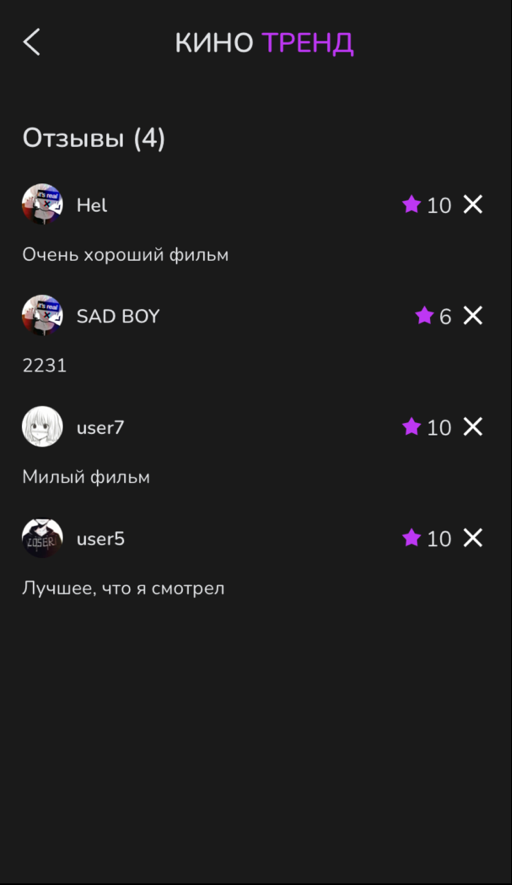


Рисунок 5.13 – Демонстрация возможности удаления оценок всех пользователей

Это позволяет модератору контролировать содержание рецензий и удалять те, которые являются неподобающимися.

5.2.3 Удаление фильма

Удаление фильма также доступно модератору на странице фильма. Кнопка удаления представлена на странице фильма в рисунке 5.14.

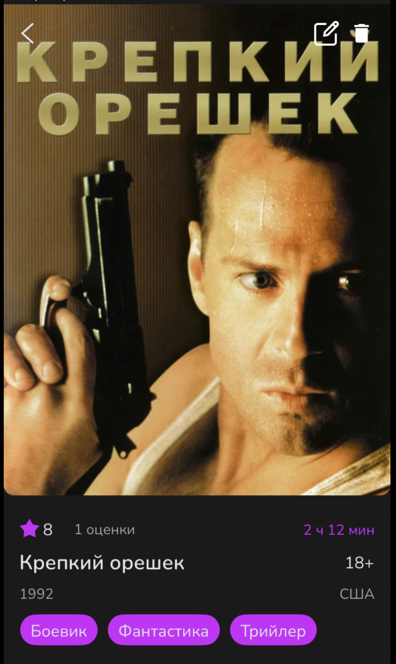


Рисунок 5.14 – Кнопка удаления фильма на странице фильма

Таким образом, модераторы обладают необходимыми инструментами для управления контентом и поддержания высокого качества информации в приложении, что обеспечивает безопасное и комфортное взаимодействие пользователей с платформой.

5.3 Руководство по развёртыванию приложения

В данном разделе приведены инструкции по развертыванию мобильного приложения для рецензирования и оценки фильмов с использованием Docker и файла конфигурации docker-compose.yml.

5.3.1 Подготовка к развёртыванию

Перед началом развёртывания необходимо выполнить следующие шаги:

* Установить Docker и Docker Compose на целевом устройстве;
* Убедиться, что файл docker-compose.yml и все необходимые директории проекта (включая сервер и базу данных) находятся в одном рабочем каталоге;
* Проверить наличие файла .env с необходимыми переменными окружения для работы приложения (например, данные для подключения к базе данных, порты и секреты).

5.3.2 Структура развёртывания

Конфигурация развертывания включает два основных сервиса:

База данных (PostgreSQL):

* Запускается в отдельном контейнере.
* Для инициализации используется резервная копия данных backup.sql.
* Данные хранятся в томе db\_data, что гарантирует их сохранность при перезапуске контейнера.
* Порт 5433 на хосте перенаправляется на порт 5432 контейнера.
* Серверная часть (NestJS):
* Запускается в отдельном контейнере с использованием Dockerfile.
* Взаимодействует с базой данных через внутреннюю сеть Docker.
* Порт 4200 контейнера перенаправляется на порт 4200 хост-системы.

5.3.3 Запуск приложения

Для развертывания приложения необходимо открыть терминал в корневом каталоге проекта и выполнить команду docker-compose up --build. Эта команда соберет образы и запустит контейнеры для всех сервисов приложения. После успешного запуска необходимо убедиться, что база данных PostgreSQL доступна на порту 5433, а серверная часть работает на порту 4200 и корректно взаимодействует с базой данных.

5.4 Вывод по разделу

Раздел «Руководство пользователя» предоставил детальные инструкции по использованию мобильного приложения как для обычных пользователей, так и для модераторов. Пользователи ознакомились с процессами регистрации, авторизации, навигации по основным страницам приложения, а также с функционалом управления своими профилями, избранными и просмотренными фильмами. Модераторы получили чёткое представление о возможностях добавления и редактирования фильмов, а также о способах модерации оценок и комментариев пользователей.

Дополнительно был рассмотрен процесс развёртывания приложения, который позволяет эффективно запустить систему на сервере с использованием **Docker** и **Docker Compose**. Пошаговая инструкция по развёртыванию обеспечивает сборку контейнеров для серверной части и базы данных PostgreSQL, настройку портов и проверку работы всех сервисов. Раздел контейнеризации подробно описывает использование Docker для изоляции компонентов приложения и хранения данных в томах, что гарантирует сохранность информации и стабильность работы при перезапусках.

Заключение

В результате выполнения курсового проекта было создано мобильное приложение для рецензирования и оценки фильмов, которое отвечает актуальным требованиям пользователей, предоставляя удобные инструменты для обмена отзывами, оценки фильмов и поиска интересующих кинолент. Архитектура приложения была спроектирована таким образом, чтобы обеспечить гибкость и возможность расширения функционала без значительных изменений, что делает приложение удобным и перспективным для дальнейшего развития.

В процессе разработки были успешно выполнены следующие задачи:

* + Разработка системы регистрации и авторизации, обеспечивающая безопасное хранение и защиту учетных данных пользователей.
  + Реализация системы ролей, поддерживающая две роли — пользователь и модератор, с уникальными возможностями для взаимодействия с приложением.
  + Функционал для пользователей, включая возможность поиска и сортировки фильмов, управление профилем, добавление и удаление комментариев, управление избранными фильмами и просмотр списка ранее посещенных страниц.
  + Функционал для модераторов, включающий возможность управления фильмами (добавление, редактирование, удаление) и модерации комментариев для поддержания порядка на платформе.

Анализ существующих приложений показал, что большинство из них предоставляет базовые функции, но не учитывает все потребности пользователей, такие как удобная система поиска и гибкая структура комментариев.

Результаты анализа стали основой для проектирования ключевых функций приложения. Приложение использует современные технологии, включая React Native для создания асинхронного и отзывчивого интерфейса, Node.js с NestJS для серверной части и PostgreSQL для хранения данных. Использование Docker для контейнеризации приложения обеспечит его легкое развертывание и поддержку на различных платформах.

Список используемых источников

1. Node.js документация. [Электронный ресурс]. Режим доступа: https://nodejs.org. Дата доступа: 25.09.2024.
2. Documentation | NestJS - A progressive Node.js framework. [Электронный ресурс]. Режим доступа: https://docs.nestjs.com. Дата доступа: 10.10.2024.
3. Prisma Documentation. [Электронный ресурс]. Режим доступа: https://www.prisma.io/docs. Дата доступа: 12.10.2024.
4. Introduction - React Native. [Электронный ресурс]. Режим доступа: https://reactnative.dev/. Дата доступа: 15.10.2024.
5. Redux Toolkit | Redux Toolkit. [Электронный ресурс]. Режим доступа: https://redux-toolkit.js.org/. Дата доступа: 16.10.2024.
6. Кинопоиск. [Электронный ресурс]. Режим доступа: https://www.kinopoisk.ru/. Дата доступа: 11.10.2024.
7. IMDb. [Электронный ресурс]. Режим доступа: https://www.imdb.com. Дата доступа: 11.10.2024.
8. GitHub. [Электронный ресурс]. Режим доступа: https://github.com/SAD-BOY-CAT/curs-work-cinema-trend. Дата доступа: 12.12.2024.

ПРИЛОЖЕНИЕ А

Листинг «Модели базы данных»

|  |
| --- |
| datasource db {  provider = "postgresql"  url = env("DATABASE\_URL")  }  generator client {  provider = "prisma-client-js"  }  model User {  id Int @id @default(autoincrement())  email String @unique  username String?  password String  role Role @default(USER)  picture String?  date DateTime?  gender Gender?  watchedMovies WatchedMovie[] @relation("UserWatchedMovies")  favoriteMovies FavoriteMovie[] @relation("UserFavoriteMovies")  reviews Review[] @relation("UserReviews")  createdAt DateTime @default(now())  updatedAt DateTime @updatedAt  }  model Movie {  id Int @id @default(autoincrement())  title String  picture String  description String?  releaseDate DateTime  categories MovieCategory[] @relation("MovieCategories")  reviews Review[] @relation("MovieReviews")  watchedBy WatchedMovie[] @relation("MovieWatchedBy")  favoritedBy FavoriteMovie[] @relation("MovieFavoritedBy")  rating Int?  duration Int?  country String?  createdAt DateTime @default(now())  updatedAt DateTime @updatedAt  }  model MovieCategory {  id Int @id @default(autoincrement())  name String  movies Movie[] @relation("MovieCategories")  }  model WatchedMovie {  id Int @id @default(autoincrement())  movieId Int  user User @relation("UserWatchedMovies", fields: [userId], references: [id])  userId Int  movie Movie @relation("MovieWatchedBy", fields: [movieId], references: [id])  watchedAt DateTime @default(now())  }  model FavoriteMovie {  id Int @id @default(autoincrement())  userId Int  movieId Int  user User @relation("UserFavoriteMovies", fields: [userId], references: [id])  movie Movie @relation("MovieFavoritedBy", fields: [movieId], references: [id])  favoritedAt DateTime @default(now())  }  model Review {  id Int @id @default(autoincrement())  content String  rating Int @default(0)  userId Int  movieId Int  user User @relation("UserReviews", fields: [userId], references: [id])  movie Movie @relation("MovieReviews", fields: [movieId], references: [id])  createdAt DateTime @default(now())  updatedAt DateTime @updatedAt  }  enum Gender {  MALE  FEMALE  OTHER  }  enum Role {  USER  MODERATOR  } |

ПРИЛОЖЕНИЕ Б

Листинг файла «user.controller.ts»

|  |
| --- |
| import {  Body,  Controller,  Post,  UseGuards,  Request,  Get,  Put,  } from '@nestjs/common';  import { UserService } from './user.service';  import { CreateUserDto, UpdateUserDto } from './user.dto';  import { AuthGuard } from 'src/auth/auth.guard';  @Controller('user')  export class UserController {  constructor(private readonly userService: UserService) {}  @Post()  async create(@Body() data: CreateUserDto) {  return this.userService.create(data);  }  @UseGuards(AuthGuard)  @Get()  async getUser(@Request() req) {  return await this.userService.findOne(req.user.username);  }  @UseGuards(AuthGuard)  @Put()  async updateUser(@Body() data: UpdateUserDto, @Request() req) {  const userId = Number(req.user.sub);  const { picture, ...userData } = data;  let fileBuffer: Buffer | undefined;  if (picture && picture !== '') {  const base64Data = data.picture.replace(/^data:image\/\w+;base64,/, '');  fileBuffer = Buffer.from(base64Data, 'base64');  }  return await this.userService.update(userData, userId, fileBuffer);  }  } |

ПРИЛОЖЕНИЕ В

Листинг файла «user.dto.ts»

|  |
| --- |
| import { Gender, User } from '@prisma/client';  import { IsDateString, IsString } from 'class-validator';  export class CreateUserDto {  @IsString()  email: string;  @IsString()  password: string;  }  export class UpdateUserDto {  @IsString()  picture: string;  @IsString()  username: string;  @IsString()  gender: Gender;  @IsDateString()  date: string;  }  export type UserDTO = Omit<User, 'password'>; |

ПРИЛОЖЕНИЕ Г

Листинг файла «user.module.ts»

|  |
| --- |
| import { Module } from '@nestjs/common';  import { UserService } from './user.service';  import { UserController } from './user.controller';  import { PrismaService } from 'src/prisma/prisma.service';  @Module({  controllers: [UserController],  providers: [UserService, PrismaService],  exports: [UserService],  })  export class UserModule {} |

ПРИЛОЖЕНИЕ Д

Листинг файла «user.service.ts»

|  |
| --- |
| import { BadRequestException, Injectable } from '@nestjs/common';  import { PrismaService } from 'src/prisma/prisma.service';  import { CreateUserDto, UpdateUserDto } from './user.dto';  import \* as admin from 'firebase-admin';  import { v4 as uuidv4 } from 'uuid';  import \* as serviceAccount from 'src/config/cinema-trend-firebase-adminsdk-neitk-a1670b9956.json';  @Injectable()  export class UserService {  private readonly bucket;  constructor(private readonly prismaService: PrismaService) {  admin.initializeApp({  credential: admin.credential.cert(serviceAccount as admin.ServiceAccount),  storageBucket: 'cinema-trend.appspot.com',  });  this.bucket = admin.storage().bucket();  }  async findOne(email: string) {  return await this.prismaService.user.findFirst({  where: { email },  });  }  async create(data: CreateUserDto) {  const { email, password } = data;  return await this.prismaService.user.create({  data: { email, password },  });  }  async update(  data: Omit<UpdateUserDto, 'picture'>,  userId: number,  fileBuffer?: Buffer,  ) {  if (fileBuffer) {  // Если есть файл, загружаем его  const pictureUrl = await this.uploadAvatar(fileBuffer);  const user = await this.prismaService.user.update({  where: { id: userId },  data: { picture: pictureUrl, ...data },  });  if (!user) throw new BadRequestException();  return {  username: user.username,  gender: user.gender,  date: user.date,  picture: user.picture,  };  } else {  const { username, gender, date } = data;  const user = await this.prismaService.user.update({  where: { id: userId },  data: { username, gender, date },  });  if (!user) throw new BadRequestException();  return {  username: user.username,  gender: user.gender,  date: user.date,  picture: user.picture,  };  }  }  private async uploadAvatar(fileBuffer: Buffer): Promise<string> {  const fileName = `users/${uuidv4()}.jpg`;  const firebaseFile = this.bucket.file(fileName);  await firebaseFile.save(fileBuffer, {  metadata: {  contentType: 'image/jpeg',  },  });  await firebaseFile.makePublic();  return `https://storage.googleapis.com/${this.bucket.name}/${fileName}`;  }  } |

ПРИЛОЖЕНИЕ Е

Листинг файла «package.json»

|  |
| --- |
| {  "name": "cinema-trend-backend",  "version": "0.0.1",  "description": "",  "author": "",  "private": true,  "license": "UNLICENSED",  "scripts": {  "build": "nest build",  "format": "prettier --write \"src/\*\*/\*.ts\" \"test/\*\*/\*.ts\"",  "start": "nest start",  "start:dev": "nest start --watch",  "start:debug": "nest start --debug --watch",  "start:prod": "node dist/main",  "lint": "eslint \"{src,apps,libs,test}/\*\*/\*.ts\" --fix",  "test": "jest",  "test:watch": "jest --watch",  "test:cov": "jest --coverage",  "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-node/register node\_modules/.bin/jest --runInBand",  "test:e2e": "jest --config ./test/jest-e2e.json"  },  "dependencies": {  "@nestjs/common": "^10.0.0",  "@nestjs/core": "^10.0.0",  "@nestjs/jwt": "^10.2.0",  "@nestjs/mapped-types": "\*",  "@nestjs/platform-express": "^10.0.0",  "@prisma/client": "^5.19.1",  "@types/multer": "^1.4.12",  "class-validator": "^0.14.1",  "firebase-admin": "^12.7.0",  "prisma": "^5.19.1",  "reflect-metadata": "^0.2.0",  "rxjs": "^7.8.1",  "uuid": "^11.0.2"  },  "devDependencies": {  "@nestjs/cli": "^10.0.0",  "@nestjs/schematics": "^10.0.0",  "@nestjs/testing": "^10.0.0",  "@types/express": "^4.17.17",  "@types/jest": "^29.5.2",  "@types/node": "^20.3.1",  "@types/supertest": "^6.0.0",  "@typescript-eslint/eslint-plugin": "^7.0.0",  "@typescript-eslint/parser": "^7.0.0",  "eslint": "^8.42.0",  "eslint-config-prettier": "^9.0.0",  "eslint-plugin-prettier": "^5.0.0",  "jest": "^29.5.0",  "prettier": "^3.0.0",  "source-map-support": "^0.5.21",  "supertest": "^7.0.0",  "ts-jest": "^29.1.0",  "ts-loader": "^9.4.3",  "ts-node": "^10.9.1",  "tsconfig-paths": "^4.2.0",  "typescript": "^5.1.3"  },  "jest": {  "moduleFileExtensions": [  "js",  "json",  "ts"  ],  "rootDir": "src",  "testRegex": ".\*\\.spec\\.ts$",  "transform": {  "^.+\\.(t|j)s$": "ts-jest"  },  "collectCoverageFrom": [  "\*\*/\*.(t|j)s"  ],  "coverageDirectory": "../coverage",  "testEnvironment": "node"  }  } |

ПРИЛОЖЕНИЕ Ж

Листинг файла «docker-compose.yml»

|  |
| --- |
| version: '3.8'  services:  db:  build:  context: ./db  container\_name: cinema\_db  env\_file:  - .env  volumes:  - db\_data:/var/lib/postgresql/data  - ./db/backup.sql:/docker-entrypoint-initdb.d/backup.sql  ports:  - "5433:5432"  networks:  - app-network  nest-app:  build:  context: .  dockerfile: Dockerfile  container\_name: my\_nest\_app  env\_file:  - .env  depends\_on:  - db  ports:  - "4200:4200"  networks:  - app-network  networks:  app-network:  driver: bridge  volumes:  db\_data: |

ПРИЛОЖЕНИЕ З

Листинг файла «auth.service.ts»

|  |
| --- |
| import {  ConflictException,  Injectable,  UnauthorizedException,  } from '@nestjs/common';  import { UserService } from 'src/user/user.service';  import { LoginDto, RegisterDto } from './auth.dto';  import { JwtService } from '@nestjs/jwt';  @Injectable()  export class AuthService {  constructor(  private readonly userService: UserService,  private readonly jwtService: JwtService,  ) {}  async signIn(data: LoginDto) {  const { email, password } = data;  const user = await this.userService.findOne(email);  if (user.password !== password) {  throw new UnauthorizedException();  }  const payload = { sub: user.id, username: user.email, role: user.role };  return {  token: await this.jwtService.signAsync(payload),  };  }  async signUp(data: RegisterDto) {  const user = await this.userService.findOne(data.email);  if (user) throw new ConflictException('User already exists');  await this.userService.create(data);  return true;  }  } |

ПРИЛОЖЕНИЕ И

Листинг файла «watched.service.ts»

|  |
| --- |
| import { Injectable } from '@nestjs/common';  import { MovieService } from 'src/movie/movie.service';  import { PrismaService } from 'src/prisma/prisma.service';  @Injectable()  export class WatchedService {  constructor(  private readonly prismaServise: PrismaService,  private readonly movieService: MovieService,  ) {}  async getWatched(userId: number) {  const watched = await this.prismaServise.watchedMovie.findMany({  where: { userId: userId },  orderBy: {  watchedAt: 'desc',  },  });  const moviesId = watched.map((watchedMovie) => watchedMovie.movieId);  const movies = await this.movieService.findMoviesById(moviesId, userId);  return movies;  }  async createWatched(userId: number, movieId: number) {  const watchedMovie = await this.prismaServise.watchedMovie.findFirst({  where: {  userId: userId,  movieId: movieId,  },  });  if (watchedMovie) {  await this.prismaServise.watchedMovie.delete({  where: {  id: watchedMovie.id,  },  });  }  return await this.createWatchedMovie(userId, movieId);  }  private async createWatchedMovie(userId: number, movieId: number) {  return await this.prismaServise.watchedMovie.create({  data: {  userId: userId,  movieId: movieId,  },  });  }  } |