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 «Онлайн кинотеатр»

Выполнил студент Зворыкин Дмитрий Александрович

(Ф.И.О.)

Руководитель проекта ст. препод. Дубовик М.В.

(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 ст. препод. Дубовик М.В.

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

Нормоконтролер ст. препод. Дубовик М.В.

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 3](#_Toc135379377)

[1 Постановка задачи 4](#_Toc135379378)

[1.1 Обзор аналогов 4](#_Toc135379379)

[2 Проектирование web-приложения 7](#_Toc135379380)

[2.1 Диаграмма вариантов использования 7](#_Toc135379381)

[2.2 Диаграмма развертывания 7](#_Toc135379382)

[2.3 Проектирование базы данных 9](#_Toc135379383)

[3 Разработка web-приложения 13](#_Toc135379384)

[3.1 Разработка серверной части 13](#_Toc135379385)

[3.2 Разработка клиентской части 17](#_Toc135379386)

[4 Тестирование web-приложения 19](#_Toc135379387)

[5 Руководство пользователя 24](#_Toc135379388)

[5.1 Регистрация пользователя 24](#_Toc135379389)

[5.2 Аутентификация и авторизация пользователя 24](#_Toc135379390)

[5.3 Главная страница 26](#_Toc135379391)

[5.4 Страница фильма 26](#_Toc135379392)

[5.5 Администрирование 29](#_Toc135379393)

[Заключение 31](#_Toc135379394)

[Список используемых источников 32](#_Toc135379395)

[Приложение А 33](#_Toc135379396)

[Приложение Б 34](#_Toc135379397)

[Приложение В 35](#_Toc135379398)

# Введение

Web-приложения становятся неотъемлемой частью нашей жизни, обеспечивая нам широкий доступ к различным сервисам и функциональности через интернет. Один из таких сервисов, который набирает все большую популярность, это онлайн кинотеатры. В сети можно найти множество платформ, предлагающих просмотр фильмов и сериалов в удобное время и месте, не выходя из дома. Актуальность создания веб-приложений для онлайн кинотеатров обусловлена рядом факторов.

Во-первых, с развитием интернета и повсеместного доступа к высокоскоростным соединениям, пользователи все чаще предпочитают онлайн платформы для просмотра фильмов. Они получают возможность выбирать интересующие их контент из обширной библиотеки, без необходимости ограничиваться расписанием традиционных кинотеатров. Это особенно актуально в условиях современного ритма жизни, когда у людей ограниченное количество свободного времени.

Во-вторых, создание веб-приложения для онлайн кинотеатра позволяет расширить аудиторию потенциальных зрителей. Традиционные кинотеатры имеют географические ограничения и могут обслуживать только местное население. В то же время, веб-приложение дает возможность привлечь зрителей из разных стран и регионов, преодолевая границы и делая просмотр фильмов более доступным и удобным для всех.

Результатом данной работы будет функциональное веб-приложение «Онлайн кинотеатр», способное удовлетворить потребности пользователей в доступе к разнообразному фильмовому контенту. Приложение будет иметь потенциал для дальнейшего развития и расширения.

Для успешной реализации приложения необходимо выполнить задачи:

* провести аналитический обзор литературы, аналогов приложения;
* спроектировать архитектуру приложения;
* спроектировать структуру базы данных;
* разработать приложение;
* протестировать приложение.

В последующих разделах работы будет подробно рассмотрена архитектура и функциональность веб-приложения, описаны основные этапы его разработки и примененные технологии.

1 Постановка задачи

Главная задача курсового проекта является разработка функционального web-приложения, которое позволяет просматривать фильмы и информацию о них.

## 1.1 Обзор аналогов

Одним из ключевых моментов в разработке программного обеспечения является просмотр и изучения различных аналогов, и поиск в них недостатков и достоинств.

Перед тем как приступать к разработке своей системы можно изучить уже готовые решения и посмотреть, что по мнению других разработчиков является наиболее необходимым для пользователей, чтобы приложение действительно было полезным. Первым аналогом будет рассмотрено web-приложение «IVI».

На рисунке 1.1 представлен внешний вид web-приложения «IVI».

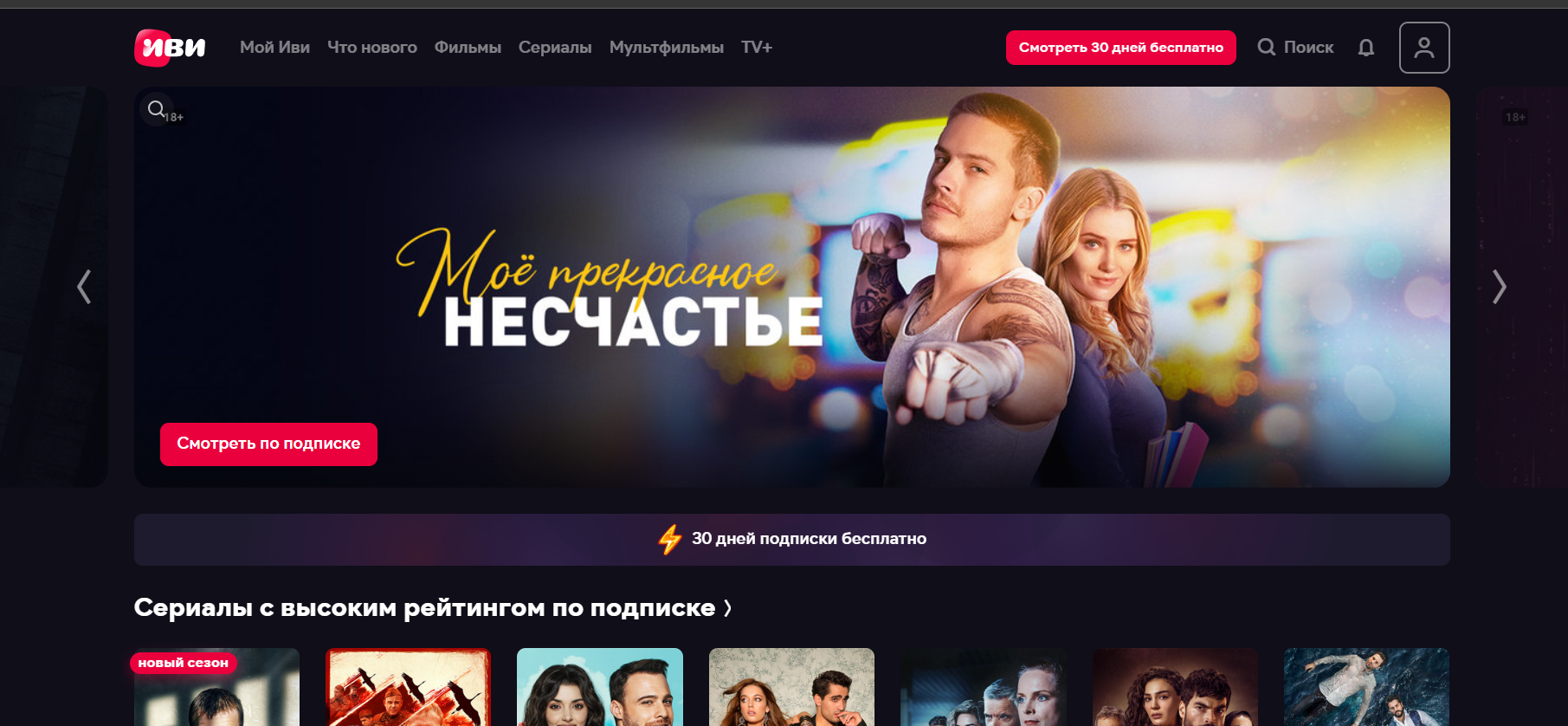


Рисунок 1.1 – Главная страница приложения «IVI»

Данный сайт, как и все из выбранных сайтов, хорошо оптимизирован под все размеры устройств так что можно смотреть фильмы с любого девайса. Имеет следующие достоинства:

– Вся коллекция IVI доступна в 1080p;

– Для смартфонов есть по два приложения: основное для всех и отдельное приложение IVI kids. Есть и приложение для Windows 10;

– Фильмы можно смотреть оффлайн;

**– Родительский контроль в аккаунте;**

**– Наличие бесплатного контента.**

Недостатки:

– Платная подписка;

– Наличие рекламы;

– Малое количество сериалов.

Следующий аналог представлен на рисунке 1.2 – web-приложение «Kinopoisk»

Это онлайн-сервис, предоставляющий пользователю возможность найти информацию о фильмах, сериалах, актерах и других кинематографических произведениях. Сервис является одним из крупнейших и наиболее популярных ресурсов в России и странах СНГ в сфере киноиндустрии.

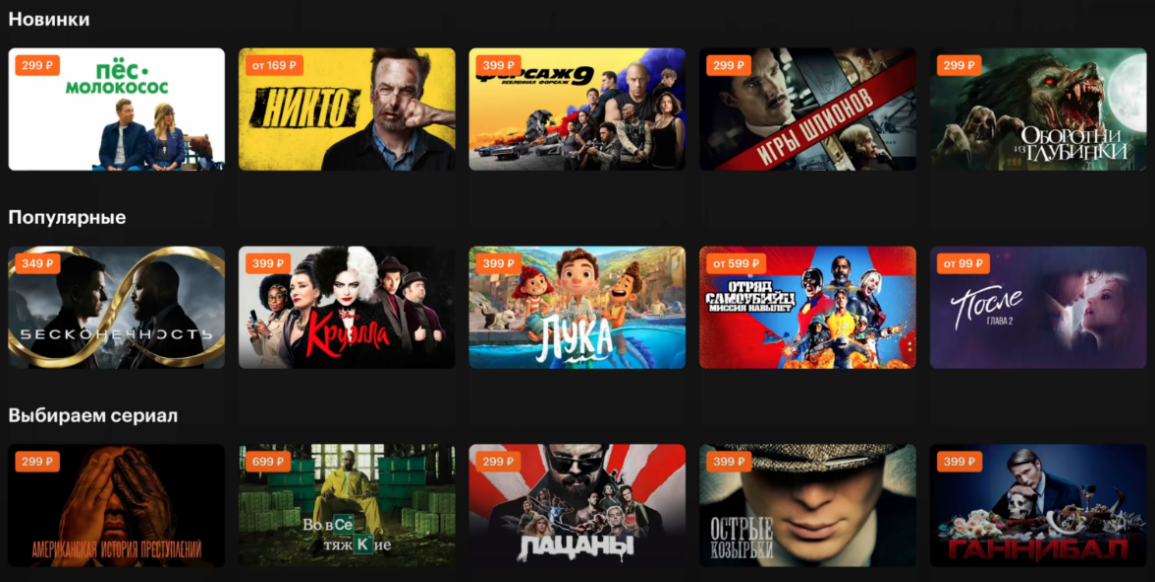


Рисунок 1.2 – Главная страница приложения «Kinopoisk»

Имеет следующие достоинства:

– Наличие фильмов в 4K;

– Можно выбрать аудиодорожку;

– Можно смотреть контент оффлайн.

Недостатки:

– Платная подписка;

– Не так разнообразно, как у аналогов.

Следующий аналог приложение «Voka». Отечественный сайт для просмотра фильмов и сериалов.

Имеет следующие достоинства:

– Возможность просмотра ТВ;

– Подписка в комплекте с некоторыми тарифными планами мобильной связи.

Главное окно программы представлено на рисунке 1.3.

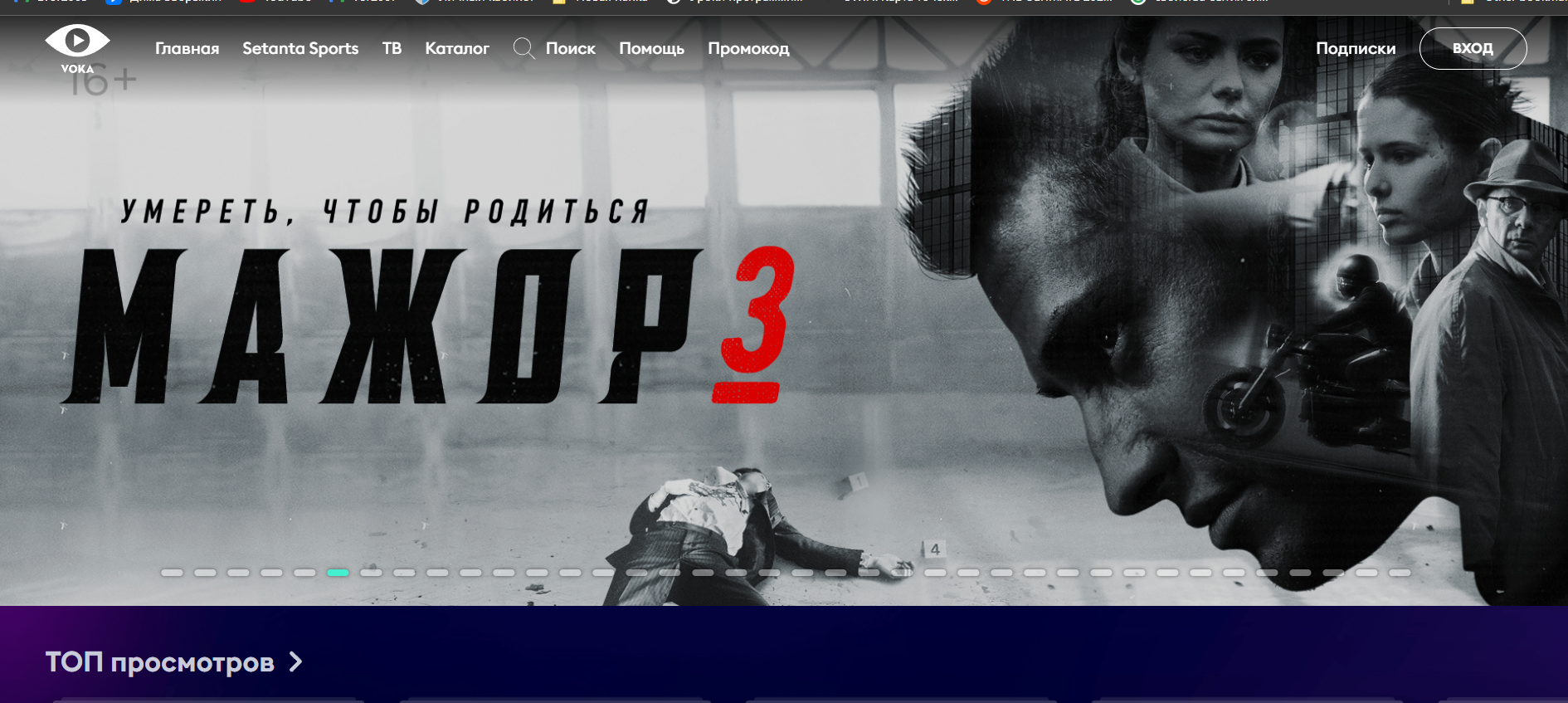


Рисунок 1.3 – Интерфейс приложения «VOKA»

К недостаткам данного приложения можно отнести:

– Малое количество фильмов;

– Малое количество зарубежных фильмов и новинок.

Схожесть рассмотренных программных средств заключается в том, что все они за месячную подписку дают доступ к каталогу фильмов и сериалов.

Главные отличия программ кроются в дизайне интерфейса, некоторыми возможностями и размерам каталога. Так на Ivi и Кинопоиске можно оставлять комментарии, читать рецензии критиков. В ivi также можно создать детский аккаунт с родительским контролем.

Обзор вышеперечисленных известных аналогов решений поставленных ранее задач позволяет проанализировать все преимущества и недостатки альтернативных реализаций и позволяет сформулировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте. Программное средство должно обеспечивать возможность выполнения перечисленных функций:

* сохранение информации в базе данных;
* организация регистрации и входа в систему;
* обмен данными между клиентом и сервером должен производиться по протоколу HTTPS;
* возможность просмотра, добавления, удаления, изменения фильмов;
* возможность просмотра, добавления, удаления, информации о фильмах;
* возможность поиска фильмов;
* возможность оставлять комментарии с оценками;
* отмечать понравившиеся;
* добавлять фильмы в «посмотреть позже»;
* просматривать историю.

2 Проектирование web-приложения

В рамках разработки данного приложения «Онлайн кинотеатр» была выбрана клиент-серверная архитектура, которая обеспечивает эффективное функционирование основных возможностей приложения.

Выбор клиент-серверной архитектуры обусловлен необходимостью эффективного управления данными и обеспечения масштабируемости приложения. Эта архитектура позволяет разделить ответственность между сервером и клиентом, что обеспечивает более гибкое взаимодействие между пользователем и системой.

## 2.1 Диаграмма вариантов использования

ДиаграммаUML *–* это графическое представление набора элементов,изображаемое в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Диаграмма вариантов использования представлена в приложении A.

Веб-приложение предоставляет различные функциональные возможности в зависимости от роли пользователя. На диаграмме вариантов использования показаны эти функции.

Исходя из диаграммы использования можно выделить 3 основные роли в приложении: пользователь, неавторизованный пользователь и администратор.

Для того, чтобы пользователь мог использовать возможности приложения, он должен быть зарегистрирован в нем. Для этого у него есть возможность регистрации и авторизации.

Администратор отвечает за данные, которые отображаются на сайте. Его обязанности включают поддержку актуальной информации о фильмах и обеспечение их корректного отображения, а также добавление новых фильмов и связанной информации о них.

## 2.2 Диаграмма развертывания

Диаграмма развертывания – это тип диаграммы UML (Unified Modeling Language), который показывает физическое размещение компонентов системы и связи между ними в рамках аппаратной инфраструктуры. Диаграмма развертывания может быть использована для представления как физических, так и логических элементов системы, таких как серверы, устройства хранения данных, сетевые устройства, приложения и т.д. Компоненты системы отображаются в виде узлов, а связи между ними – в виде линий. Диаграмма развертывания позволяет понять, как компоненты системы взаимодействуют между собой и как они связаны с аппаратным обеспечением. Диаграмма развертывания приложения представлена на рисунке 2.1.

Web-приложение развернуто на локальной машине, для общения между клиентом и сервером используются протоколы WebSocket и Https.На сервере запущен Nest сервер и СУБД PostgreSQL. В Nest сервере запущен веб сокет сервер. Все запросы на сервер обрабатываются контроллерами.



Рисунок 2.1 – диаграмма развертывания приложения

При разработке программного средства была выбрана архитектура REST.

REST (Representational State Transfer) – это стиль архитектуры программного обеспечения, который определяет принципы проектирования распределенных систем. Он основывается на использовании HTTPS-протокола и использует ресурсы в качестве основного понятия для обмена данными между клиентом и сервером.

Преимущества архитектуры REST:

– Гибкость: REST архитектура не зависит от языка программирования, что позволяет использовать любой язык программирования для реализации клиента и сервера.

– Простота: REST API обладает простым и понятным интерфейсом, что упрощает его использование и интеграцию в другие системы.

– Масштабируемость: REST API позволяет горизонтально масштабировать систему путем добавления новых серверов.

– Кэширование: REST API поддерживает кэширование ответов, что уменьшает нагрузку на сервер и ускоряет ответы на запросы.

– Безопасность: REST API обеспечивает высокий уровень безопасности при обмене данными между клиентом и сервером.

Недостатки архитектуры REST:

– Отсутствие строгих стандартов: REST не определяет четкие стандарты для реализации, поэтому реализации могут отличаться у разных разработчиков.

– Сложность маршрутизации: REST API может иметь сложные маршруты, что затрудняет его использование и интеграцию в другие системы.

– Недостаток управления состоянием: REST не предоставляет механизм для управления состоянием клиента и сервера, что может приводить к сложностям при отслеживании состояния системы.

Всего в приложении используется 4 основных сервиса:

– React.js – сервис, реализующий frontend-часть приложения;

– NEST.js – сервис, реализующий backend-часть приложения;

– PostgresSQL – БД;

– NGINX – сервер, отвечающий за стрим видео.

Каждый сервис программного средства максимально изолирован от других сервисов, что позволяет легко расширять функционал приложения. Для общения между сервисами программного средства используется JSON. JSON – текстовый формат обмена данными, основанный на JavaScript. Он представляет собой набор пар ключ - значение.

## 2.3 Проектирование базы данных

При создании базы данных для данного приложения проведение глубокого анализа предметной области является неотъемлемой частью процесса. Этот анализ позволяет точно определить структуру данных, которая отражает требования и потребности пользователей в отношении хранения и управления информацией. Он также способствует созданию модели данных, которая обеспечивает полный набор функциональных возможностей для эффективной работы с данными.

Одним из используемых подходов к проектированию базы данных является подход Code-first, который был применен в разработке данного приложения. Он основан на использовании файла "schema.prisma", содержащего описание структуры базы данных. При использовании данного подхода таблицы базы данных синхронизируются автоматически с помощью механизма миграции. Это позволяет гибко изменять структуру базы данных и обеспечивает удобство в развитии и поддержке приложения.

В контексте данной базы данных, основной таблицей является таблица "Movies", содержащая информацию о фильмах. Создание этой таблицы, а также других таблиц, происходит автоматически при запуске приложения. Все данные сохраняются в соответствии с моделью данных, обеспечивая точность и целостность хранения информации. Это важно для обеспечения надежности и корректности работы приложения.

Модель базы данных, разработанная для этого приложения, представлена в Приложении Б. Она отражает все необходимые сущности и связи между ними, обеспечивая эффективное управление данными и выполнение требуемых функций приложения. При необходимости модель базы данных может быть доработана и расширена в будущем, с учетом развития и расширения функциональности приложения.

Далее будет описана каждая таблица базы данных.

Таблица «Genre» содержит информацию о жанрах фильмов. Каждый жанр имеет уникальный идентификатор «id» и название «name». Также имеется связь с таблицей «Movie», так как каждый жанр может быть связан с несколькими фильмами.

Таблица «Video» содержит информацию о видеофайлах. Каждое видео имеет уникальный идентификатор «id», название файла «filename» и тип «type». Тип может быть «trailer» или «movie».

Таблица «Image» – содержит информацию о изображениях. Каждое изображение имеет уникальный идентификатор «id» и название файла «filename».

Таблица «People» содержит информацию о людях, связанных с фильмами. Каждый человек имеет уникальный идентификатор «id», имя «name», профессию «profession» и ссылку на изображение «imageId».

Таблица «Movie» содержит информацию о фильмах. Каждый фильм имеет уникальный идентификатор «id», название «title», описание «description», дату выхода «releaseDate», продолжительность «duration», рейтинг «rating», страну «country» и ссылку на главный постер «mainPosterId».

Таблица «PeopleOnMovies» содержит информацию о связи людей с фильмами. Каждая запись в таблице содержит уникальный идентификатор «id», идентификатор фильма «movieId», идентификатор человека «peopleId» и роль «Role».

Таблица «Role» содержит информацию о ролях. Каждая роль имеет уникальный идентификатор «id» и название «name». В приложении доступно только 2 роли, юзер и админ.

Таблица «User» содержит информацию о пользователях. Каждый пользователь имеет уникальный идентификатор «id», имя пользователя «username», пароль «password» он хранится в виде хэша в таблице, электронную почту «email», дату создания «createdOn», дату обновления «updatedOn» и идентификатор роли «RoleId».

Таблица «Favorite» содержит информацию о избранных фильмах пользователей. Каждая запись в таблице содержит идентификатор пользователя «userId» и идентификатор фильма «movieId».

Таблица «History» содержит информацию о истории просмотров пользователей. Каждая запись в таблице содержит идентификатор пользователя «userId», идентификатор фильма «movieId», время «time» и прогресс просмотра «progress».

Таблица «Comment» содержит информацию о комментариях пользователей к фильмам. Каждая запись в таблице содержит уникальный идентификатор «id», рейтинг «rating», содержание комментария «content», дату создания «createdOn», идентификатор пользователя «userId» и идентификатор фильма «movieId». Установлен лимит в один комментарий с оценкой на один фильм.

Таблица «GenreToMovie» является связующей таблицей между жанрами и фильмами. Каждая запись в таблице содержит идентификатор жанра «A» и идентификатор фильма «B».

Таблица «ImageToMovie» является связующей таблицей между изображениями и фильмами. Каждая запись в таблице содержит идентификатор изображения «A» и идентификатор фильма «B».

Таблица «MovieToVideo» является связующей таблицей между фильмами и видеофайлами. Каждая запись в таблице содержит идентификатор фильма «A» и идентификатор видеофайла «B».

Таблица «WatchLater» содержит информацию о фильмах, добавленных в список «Посмотреть позже» пользователей. Каждая запись в таблице содержит идентификатор пользователя «userId» и идентификатор фильма «movieId». Полный файл «schema.prisma» представлен в приложении B.

Таблица «Genre» содержит следующие поля:

* id - тип данных serial, содержит уникальный идентификатор жанра;
* name - тип данных text, содержит название жанра.

Таблица «Video» содержит следующие поля:

* id - тип данных serial, содержит уникальный идентификатор видеофайла;
* filename - тип данных text, содержит название файла;
* type - тип данных text, содержит тип видеофайла.

Таблица «Image» содержит следующие поля:

* id - тип данных serial, содержит уникальный идентификатор изображения;
* filename - тип данных text, содержит название файла изображения.

Таблица «People» содержит следующие поля:

* id - тип данных serial, содержит уникальный идентификатор человека;
* name - тип данных text, содержит имя человека;
* profession - тип данных text, содержит профессию человека;
* imageId - тип данных integer, содержит ссылку на изображение человека.

Таблица «Movie» содержит следующие поля:

* id - тип данных serial, содержит уникальный идентификатор фильма;
* title - тип данных text, содержит название фильма;
* description - тип данных text, содержит описание фильма;
* releaseDate - тип данных timestamp(3), содержит дату выхода фильма;
* duration - тип данных integer, содержит продолжительность фильма;
* numberOfViews - тип данных integer, содержит количество просмотров фильма;
* rating - тип данных integer, содержит рейтинг фильма;
* country - тип данных text, содержит страну производства фильма;
* mainPosterId - тип данных integer, содержит ссылку на главный постер фильма.

Таблица «PeopleOnMovies» содержит следующие поля:

* id - тип данных serial, содержит уникальный идентификатор связи между человеком и фильмом;
* movieId - тип данных integer, содержит ссылку на фильм;
* peopleId - тип данных integer, содержит ссылку на человека;
* Role - тип данных text, содержит роль человека в фильме.
* Таблица «Role» содержит следующие поля:
* id - тип данных serial, содержит уникальный идентификатор роли;
* name - тип данных text, содержит название роли.

Таблица «User» содержит следующие поля:

* id - тип данных text, содержит уникальный идентификатор пользователя;
* username - тип данных text, содержит имя пользователя;
* password - тип данных text, содержит пароль пользователя;
* email - тип данных text, содержит электронную почту пользователя;
* createdOn - тип данных timestamp(3), содержит дату создания записи о пользователе;
* updatedOn - тип данных timestamp(3), содержит дату обновления записи о пользователе;
* RoleId - тип данных integer, содержит ссылку на роль пользователя.

Таблица «Favorite» содержит следующие поля:

* userId - тип данных text, содержит ссылку на пользователя;
* movieId - тип данных integer, содержит ссылку на фильм.

Таблица «History» содержит следующие поля:

* userId - тип данных text, содержит ссылку на пользователя;
* movieId - тип данных integer, содержит ссылку на фильм;
* time - тип данных timestamp(3), содержит время просмотра фильма;
* progress - тип данных double precision, содержит прогресс просмотра фильма.

Таблица «Comment» содержит следующие поля:

* id - тип данных serial, содержит уникальный идентификатор комментария;
* rating - тип данных integer, содержит рейтинг комментария;
* content - тип данных text, содержит содержание комментария;
* createdOn - тип данных timestamp(3), содержит дату создания комментария;
* userId - тип данных text, содержит ссылку на пользователя;
* movieId - тип данных integer, содержит ссылку на фильм.

Таблица «GenreToMovie» содержит следующие поля:

* A - тип данных integer, содержит ссылку на жанр;
* B - тип данных integer, содержит ссылку на фильм.

Таблица «ImageToMovie» содержит следующие поля:

* A - тип данных integer, содержит ссылку на изображение;
* B - тип данных integer, содержит ссылку на фильм.

Таблица «MovieToVideo» содержит следующие поля:

* A - тип данных integer, содержит ссылку на фильм;
* B - тип данных integer, содержит ссылку на видеофайл.

Таблица «WatchLater» содержит следующие поля:

* userId - тип данных text, содержит ссылку на пользователя;
* movieId - тип данных integer, содержит ссылку на фильм.

3 Разработка web-приложения

В процессе выполнения курсового проекта было разработано веб-приложение, имеющее архитектурный стиль REST API. Таким образом было разработано два отдельных компонента backend, отвечающий за серверную часть программного средства, и frontend, отвечающий за клиентское представление. А также был сконфигурирован NGINX сервер.

## 3.1 Разработка серверной части

Nest [2] – это фреймворк для создания масштабируемых веб-приложений на Node.js. Он позволяет разработчикам быстро создавать серверные приложения, используя модульную архитектуру и инструменты для управления зависимостями.

На рисунке 3.1 отображена структура серверной части разрабатываемого программного средства, включающая в себя директории, конфигурационные файлы и файлы скриптов.

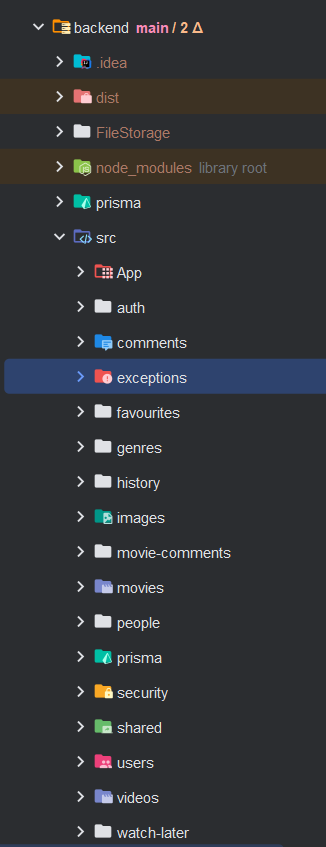


Рисунок 3.1 – Структура проекта сервера

Входной точкой в приложение является файл main.ts в нем выполняется метод bootstrap, в котором выполняется конфигурация сервера.

Nest – это модульный фреймворк, поэтому приложения могут быть организованы как набор независимых модулей, каждый из которых имеет свою структуру.

Каждый модуль состоит из сервиса и контроллера. Также модуль может содержать dto файлы.

В почти каждом модуле использовался сервис PrismaService для взаимодействия с базой данных. Он экспортируется в модуле Prisma. Сервис призмы представлен в листинге 3.1.

import { INestApplication, Injectable, OnModuleInit } from '@nestjs/common';  
import { PrismaClient } from '@prisma/client';  
  
@Injectable()  
export class PrismaService extends PrismaClient implements OnModuleInit {  
 async onModuleInit() {  
 await this.$connect();  
 }  
  
 async enableShutdownHooks(app: INestApplication) {  
 this.$on('beforeExit', async () => {  
 await app.close();  
 });  
 }  
}

Листинг 3.1 – Файл PrismaService.ts

Для примера возьмём модуль genres. У него есть сервис GenresService, контроллер GenresController а также dto CreateGenreDto и GenreDto.

GenresController. Данный файл является контроллером Nest приложения, который обрабатывает запросы, связанные с жанрами. В нем определен класс GenresController, который содержит различные методы для обработки запросов HTTP:

– create() – метод, обрабатывающий POST запросы на создание нового жанра.

– findAll() – метод, обрабатывающий GET запросы на получение списка всех жанров.

– findOne() – метод, обрабатывающий GET запросы на получение одного жанра по его идентификатору.

– update() – метод, обрабатывающий PUT запросы на обновление данных жанра.

– remove() – метод, обрабатывающий DELETE запросы на удаление жанра.

Контроллер использует сервис GenresService для выполнения бизнес-логики, связанной с жанрами. В контроллере определены декораторы, такие как @Controller(), @Post(), @Get(), @Param(), @Body() и @Delete(), которые указывают, какой HTTP метод должен быть использован для обработки запроса.

Также в контроллере импортирован класс CreateGenreDto из файла create-genre.dto.ts, который представляет собой объект для создания нового жанра. Он используется в методе create() для получения данных, введенных пользователем, и передачи их в сервис для сохранения в базе данных .Контроллер GenresController представлен в листинге 3.2.

@Controller('genres')  
export class GenresController {  
 constructor(private readonly genresService: GenresService) {}  
 @Post()  
 create(@Body() createGenreDto: CreateGenreDto) {  
 return this.genresService.create(createGenreDto);  
 }  
 @Get()  
 findAll() {  
 return this.genresService.findAll();  
 }  
 @Get(':id')  
 findOne(@Param('id') id: string) {  
 return this.genresService.findOne(+id);  
 }  
 @Put(':id')  
 update(@Param('id') id: string, @Body() updateGenreDto: CreateGenreDto) {  
 return this.genresService.update(+id, updateGenreDto);  
 }  
 @Delete(':id')  
 remove(@Param('id') id: string) {  
 return this.genresService.remove(+id);  
 }  
}

Листинг 3.2 – GenresController

GenresServic это сервис, отвечающий за взаимодействие с данными жанров из БД. В нём определены методы для создания, получения, обновления и удаления записей жанров.

В файле импортируются необходимые зависимости и используется декоратор @Injectable(), который говорит Nest о том, что данный класс является сервисом и может использоваться в других частях приложения. Конструктор класса принимает в качестве параметра экземпляр класса PrismaService, который используется для работы с базой данных.

Методы класса GenresService определены с использованием ключевых слов async и await. Они позволяют делать асинхронные запросы к базе данных и возвращать промисы, которые могут быть обработаны в других частях приложения.

Метод create создаёт новую запись жанра в базе данных на основе переданного объекта CreateGenreDto.

Метод findAll возвращает все записи жанров из базы данных в порядке возрастания их идентификаторов.

Метод findOne возвращает запись жанра с заданным идентификатором из базы данных. Если запись не найдена, выбрасывается исключение HttpException с кодом NOT\_FOUND.

Метод update обновляет запись жанра с заданным идентификатором на основе переданного объекта CreateGenreDto.

Метод remove удаляет запись жанра с заданным идентификатором из базы данных. Сервис GenresService представлен в листинге 3.3

@Injectable()  
export class GenresService {  
 constructor(private readonly prisma: PrismaService) {}  
 async create(createGenreDto: CreateGenreDto): Promise<Genre> {  
 const Genre = await this.prisma.genre.create({  
 data: {  
 name: createGenreDto.name,  
 },  
 });  
 return Genre;  
 }  
 async findAll(): Promise<Genre[]> {  
 const Genres = await this.prisma.genre.findMany({  
 orderBy: {  
 id: 'asc',  
 },  
 });  
 return Genres;  
 }

Листинг 3.3 – GenresService

Модуль movie-comments является компонентом нашего веб-приложения, предоставляющим пользователю возможность оставлять комментарии под фильмами в режиме реального времени. Реализация этой функциональности основана на технологии WebSocket, которая позволяет устанавливать постоянное двустороннее соединение между клиентом и сервером. Пример использования технологии web-sockets приведён в листинге 3.4.

@SubscribeMessage('comments:delete')  
async deleteComment(  
 @MessageBody() data: any,  
 @ConnectedSocket() client: Socket,  
) {  
 try {  
 const userID = await this.usersService.findIDbyLogin(data.username);  
 await this.commentsService.deleteComment(userID, data.commentId);  
 this.server.emit('comment:deleted', data.commentId);  
 } catch (error) {  
 this.logger.error(error);  
 }  
}

Листинг 3.4 – Пример использования технологии web-sockets

Протокол WebSocket («веб-сокет»), описанный в спецификации [RFC 6455](http://tools.ietf.org/html/rfc6455), обеспечивает возможность обмена данными между браузером и сервером через постоянное соединение. Данные передаются по нему в обоих направлениях в виде «пакетов», без разрыва соединения и дополнительных HTTP-запросов.

## 3.2 Разработка клиентской части

На рисунке 3.2 отображена структура клиентской части разрабатываемого программного средства.

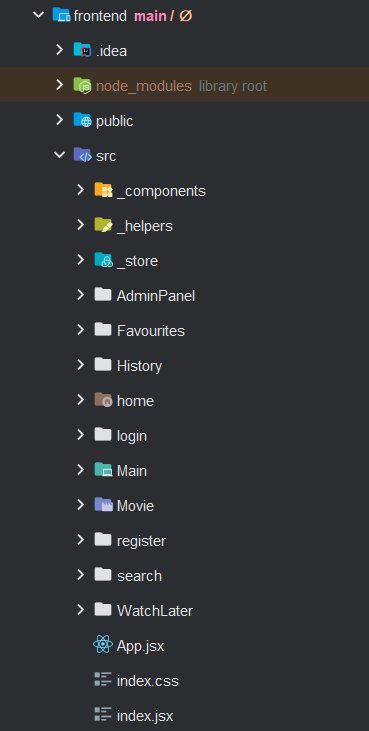


Рисунок 3.2 – Структура проекта клиента

Клиентская часть приложения разработана с использованием React - популярной JavaScript-библиотеки для создания пользовательских интерфейсов (UI). React предоставляет мощные инструменты для разработки многокомпонентных приложений, которые могут обновляться в режиме реального времени без необходимости перезагрузки страницы. Он применяет декларативный подход, позволяющий описывать состояние компонентов и их поведение.

Использование React и организация кода в виде компонентов позволяет разработчикам создавать модульные, гибкие и легко поддерживаемые приложения с отзывчивым пользовательским интерфейсом. Разделение кода на компоненты также способствует повторному использованию кода и улучшению производительности разработки.

Директория «src» содержит исходный код клиентской части приложения. Она содержит следующие файлы и папки:

– index.jsx: Это основной файл, который является точкой входа в приложение React. Он обычно импортирует компонент App и отображает его в корневом элементе DOM с помощью метода ReactDOM.render().

– App.js: Это компонент, который является основным компонентом приложения и содержит все другие компоненты.

– index.css: Это файл стилей для приложения. Он содержит общие стили для всех компонентов.

– \_components: Это папка, которая содержит все компоненты приложения. Каждый компонент обычно находится в отдельном файле и импортируется в основной файл .

– \_helpers: Это директория, которая содержит вспомогательные функции и утилиты, которые используются в приложении.

– images: Папка, которая содержит все изображения, используемые в приложении.

– \_store: Содержит код для реализации Redux-стора в приложении. Redux – это библиотека управления состоянием приложения, которая позволяет хранить данные в централизованном хранилище и управлять ими с помощью действий (actions) и редюсеры (reducers).

В остальных папках находятся страницы приложения. Размещение компонентов в отдельных папках позволяет легко находить и изменять нужные компоненты, а также упрощает их повторное использование в различных частях приложения.

Дополнительно, использование библиотеки Redux обеспечивает возможность управления состоянием приложения в централизованном хранилище. Это позволяет компонентам взаимодействовать с состоянием приложения без прямого обмена данных между ними. Централизованное хранение состояния также способствует уменьшению связности между компонентами, упрощает отладку и обеспечивает единообразный подход к управлению данными в приложении.

4 Тестирование web-приложения

Тестирование программного обеспечения – это процесс проверки, соответствует ли реальное поведение программы ожидаемому. Для этого используется конечный набор тестов, выбранный определенным образом. Чтобы избежать возникновения исключительных ситуаций, которые могут привести к нарушению нормальной работы пользователя, необходимо обработать такие ситуации. Функциональное тестирование проводится для проверки качества программного продукта, а именно, чтобы убедиться, что все функции реализованы правильно. Другими словами, это позволяет определить, сможет ли ПО выполнять необходимые задачи при заданных условиях. В программном средстве, которое я разрабатываю, предусмотрены механизмы, направленные на предотвращение исключительных ситуаций.

Рассмотрим вход пользователя в систему. Приложение проверяет корректно ли введены данные и выводит соответствующее сообщение, представлено на рисунке 4.1.



Рисунок 4.1 – Ошибка некорректных данных

Все поля в окнах авторизации и регистрации являются обязательными к заполнению. При вводе корректных данных пользователь переходит на главную страницу приложения. Также на странице логина пользователь может сменить пароль и отправить новый пароль на почту.

Пример ошибки заполнения представлен на рисунке 4.2.

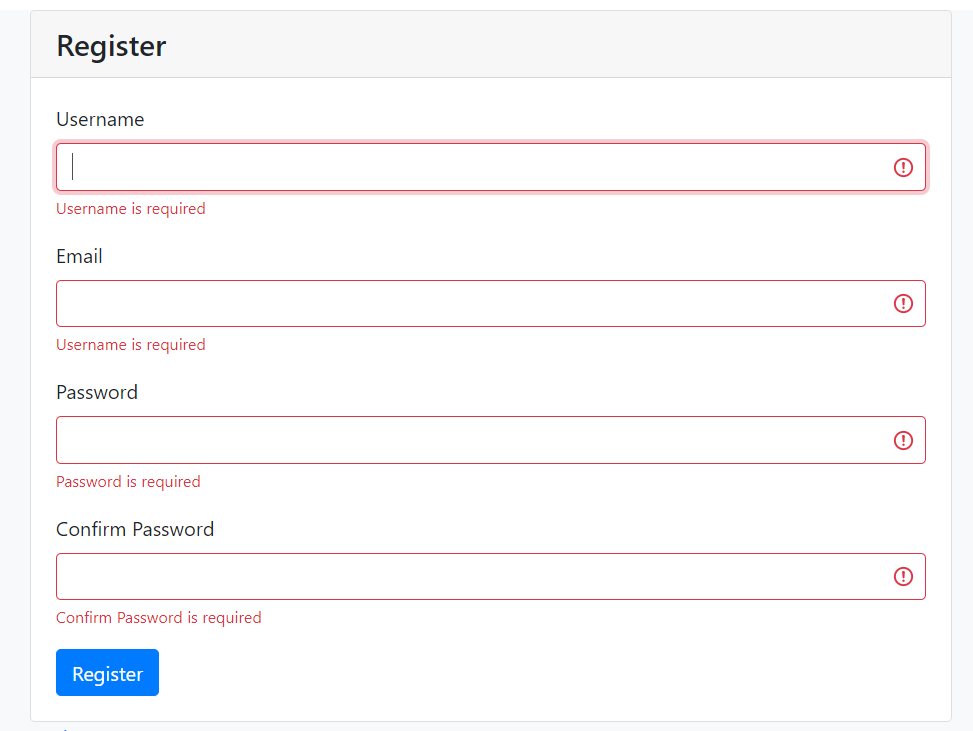


Рисунок 4.2 – Ошибка заполнения полей

Кроме этого поля также проверяются на корректное заполнение, пример на рисунке 4.3. Примером такого валидируемого поля может являться поле пароля в окне регистрации.

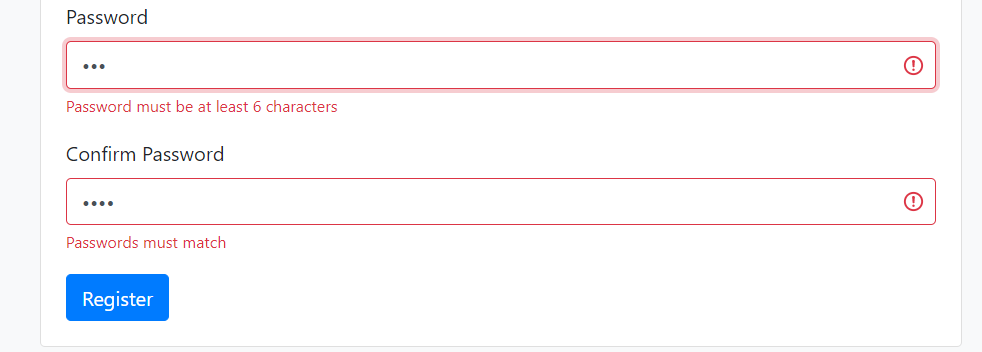


Рисунок 4.3 – Ошибка заполнения паролей

Поле username является уникальным, следовательно, при регистрации уже существующего пользователя, сервер ответит ошибкой. Пример ошибки на рисунке 4.4.



Рисунок 4.4 – Регистрация существующего пользователя

После успешной авторизации пользователь попадает на главную страницу.

На странице фильма пользователь может добавлять фильм в любимые или посмотреть позже. Также пользователь может добавить комментарий с оценкой под фильмом. Пользователь может оставить только один комментарий под фильмом. Если он попытается оставить больше одного комментария или оценку больше 10 или меньше 0, будет ошибка. Ошибка представлена на рисунке 4.5.

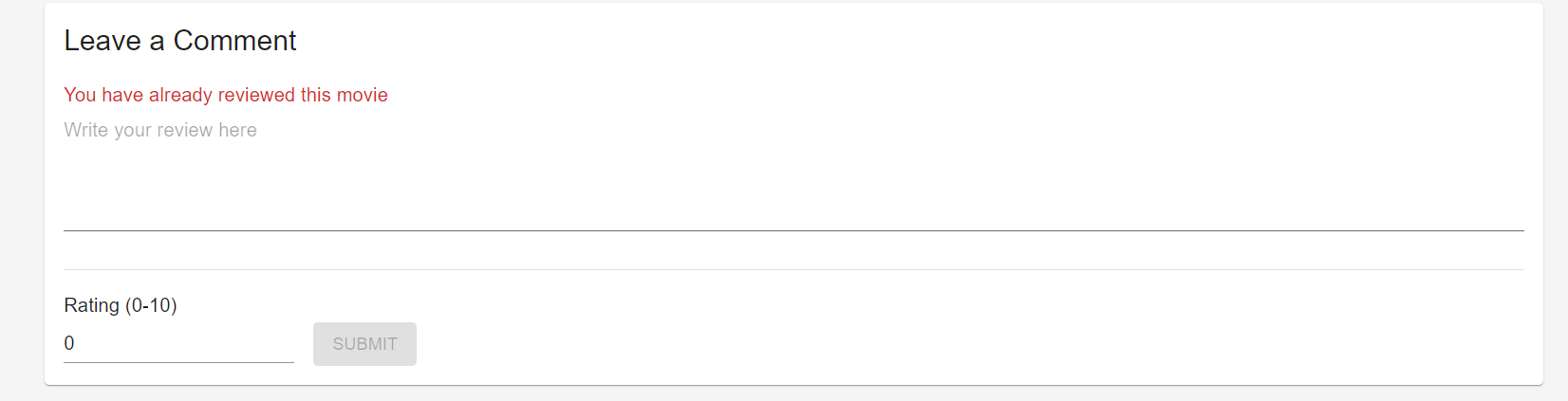


Рисунок 4.5 – Ошибка публикации комментария

Администратор также может столкнуться с ошибками при использовании программного средства, как и любой другой пользователь. Например, при попытке добавления данных.

При попытке добавления жанра, который уже есть в базе данных администратору будет показана ошибка. Ошибка представлена на рисунке 4.6.

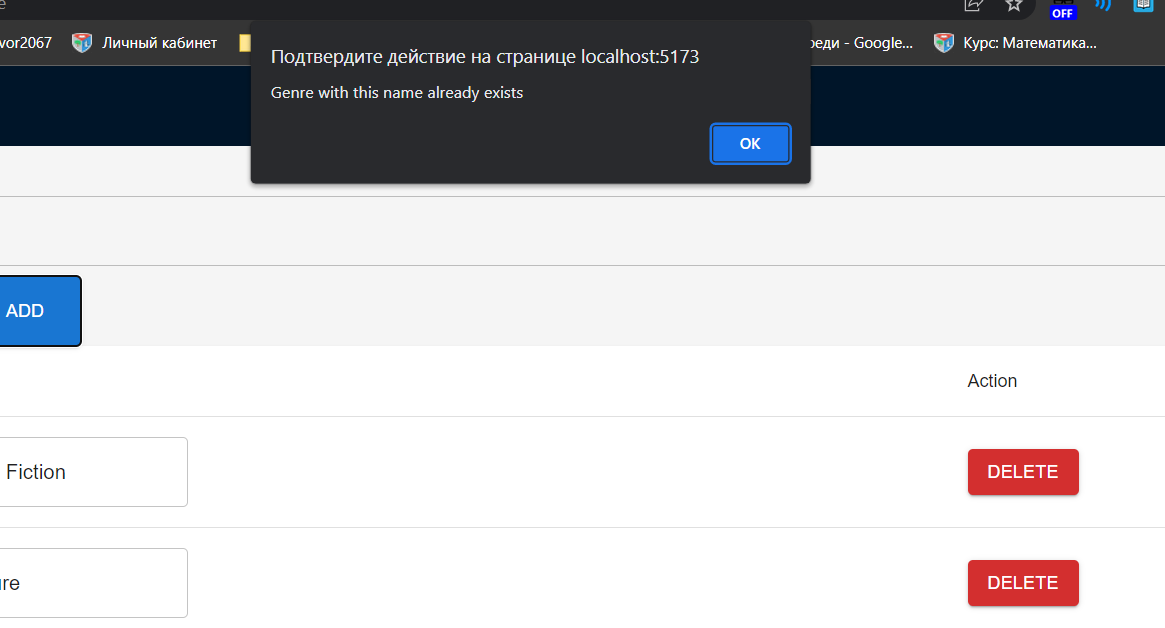


Рисунок 4.6 – Ошибка добавления жанра

Рассмотрим добавление и удаление комментария. Если все условия соблюдены, то при добавлении комментария на странице фильмов появится новый комментарий пользователя. Пример комментария представлен на рисунке 4.7

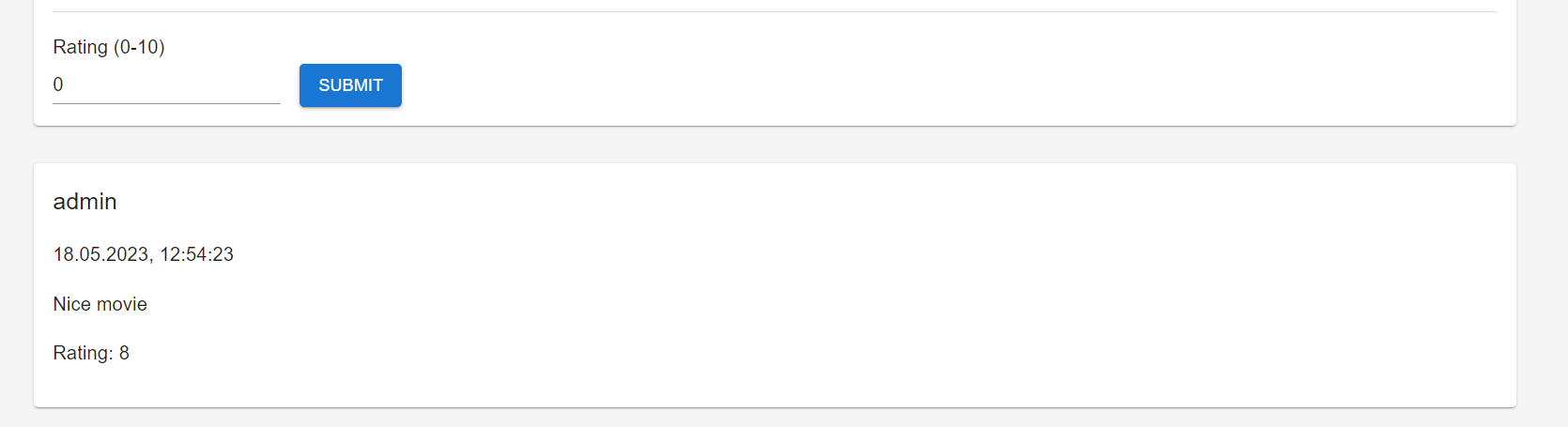


Рисунок 4.7 – Пример комментария

Он появится у всех пользователей одновременно без необходимости обновления страницы т. к. комментарии сделаны с применение веб сокетов.

Возможность удаления комментария есть только у того, кто является автором. Это показано на рисунке 4.8.

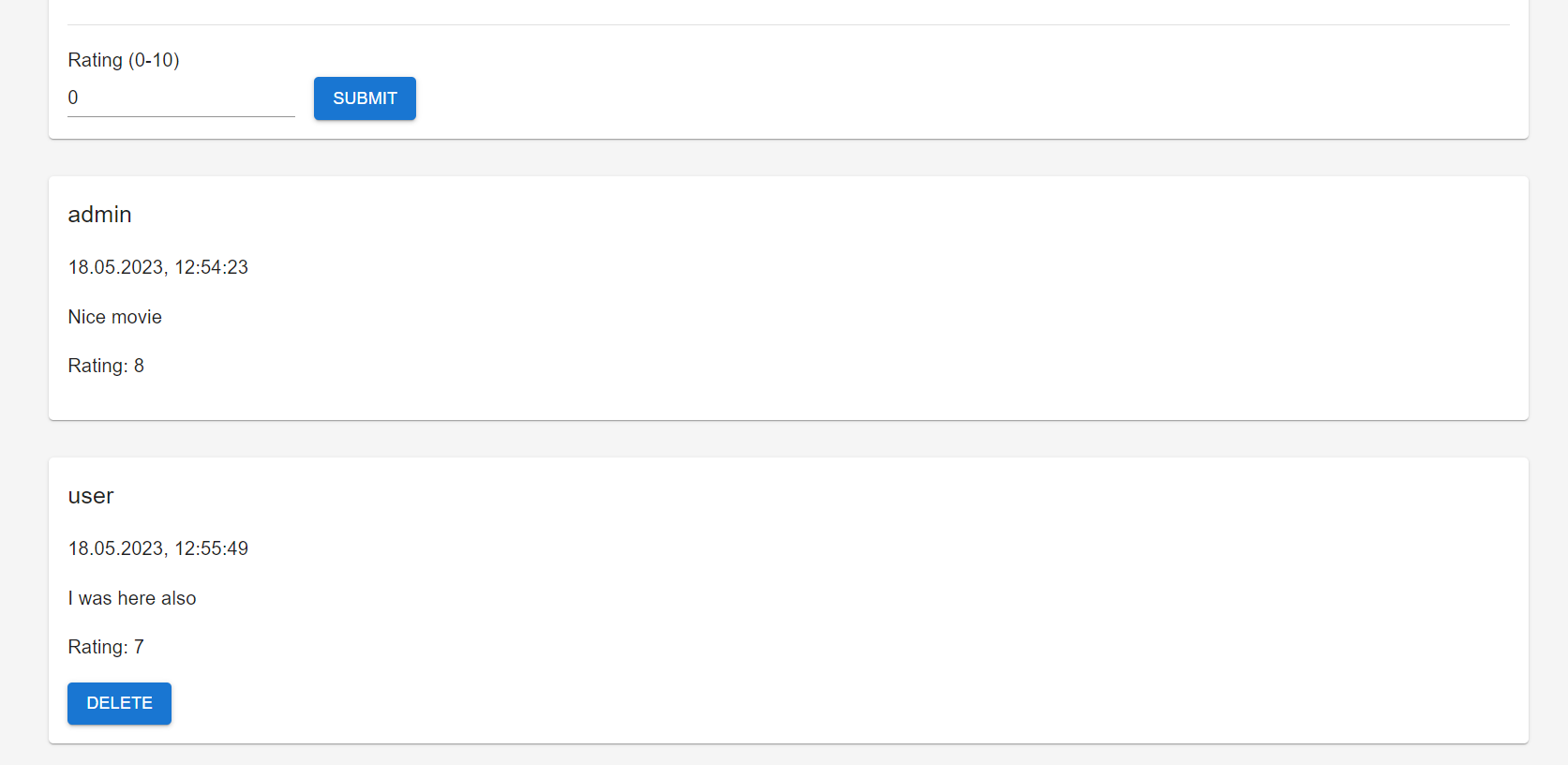


Рисунок 4.8 – Пример удаления комментария

Пользователь также может добавить любой фильм в понравившиеся или «посмотреть позже». Эти фильмы отображаются на странице понравившихся и при нажатии на них можно перейти на фильм. Пример страницы понравившихся представлен на рисунке 4.9.

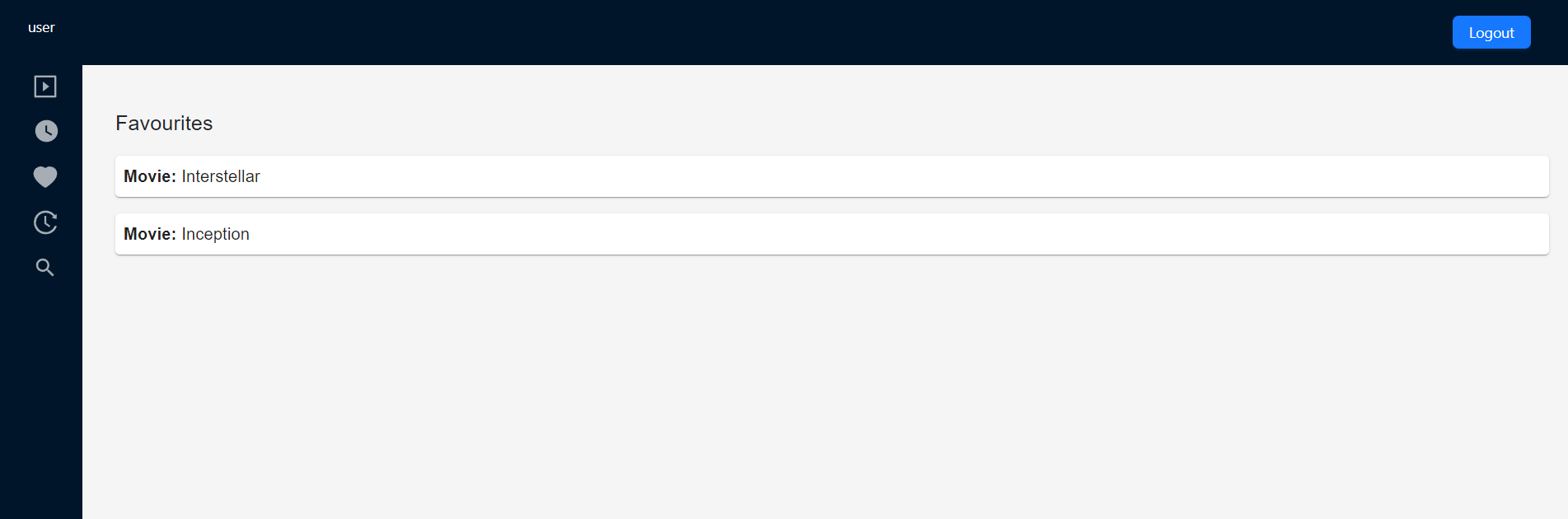


Рисунок 4.9– Пример страницы понравившихся

Таким образом, проведенное ручное тестирование в процессе разработки приложения позволяет уверенно заявить, что разработанное приложение функционирует корректно и соответствует своим требованиям и ожиданиям пользователей.

Тестирование позволило выявить и устранить возможные ошибки, проверить правильность обработки данных и пользовательских действий, а также убедиться в стабильной работе приложения.

5 Руководство пользователя

## 5.1 Регистрация пользователя

Для регистрации необходимо заполнить форму, которая продемонстрирована на рисунке 5.1.

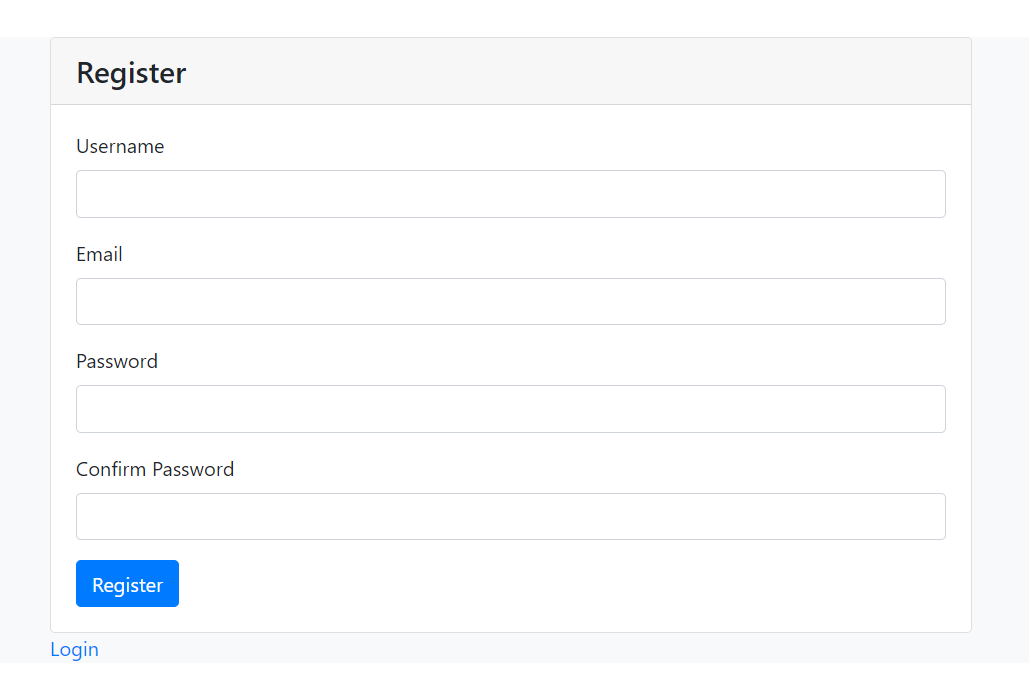


Рисунок 5.1 – Форма регистрации

После отправки пользователем запроса на регистрацию, происходит валидация всех полей формы. Если валидация будет неуспешной, пользователь увидит сообщения об ошибках в соответствующих полях для ввода информации. В случае успешной регистрации пользователь переадресовывается на страницу входа в приложение.

## 5.2 Аутентификация и авторизация пользователя

Для аутентификации необходимо ввести пароль, логин и нажать на кнопку «Login», что позволит проверить, существует ли пользователь с введенными данными в системе. Все поля являются обязательными для заполнения.

Если аутентификация прошла успешно и соблюдены все описанные правила, то происходит перенаправление на главную страницу, что означает, что пользователь был успешно авторизован. Скриншот формы авторизации представлен на рисунке 5.2.

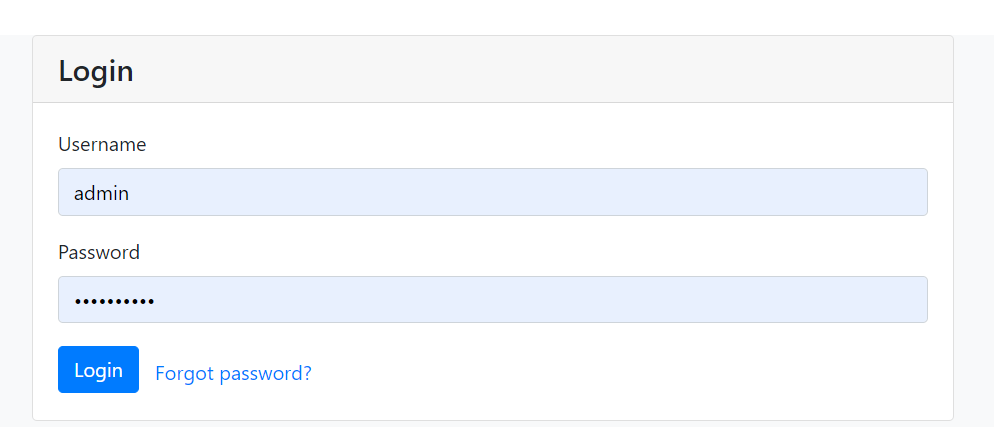


Рисунок 5.2 – Форма авторизации

При входе в систему авторизированному пользователю выдается токен, который записывается в local storage, тем самым позволяя разграничить возможности пользователей на сайте.

В случае потери и пароля пользователь может попросить смену пароля и отправку нового пароля на почту. При вводе некорректного адреса почты или почты к который нет привязанных пользователей пользователь увидит сообщения об ошибках. Пример формы восстановления пароля по почте приведена на рисунке 5.3

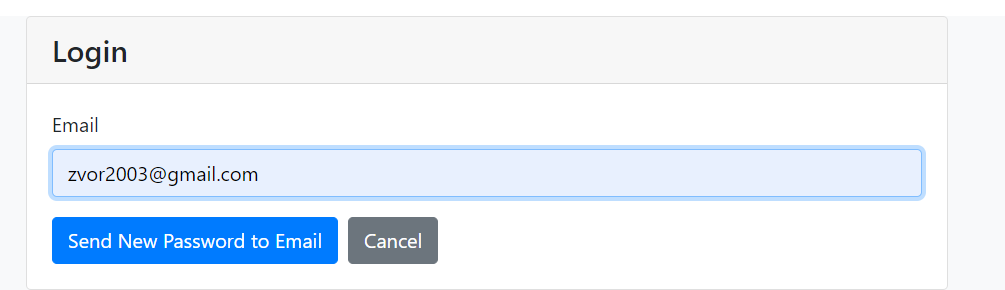


Рисунок 5.3 – Форма авторизации

В данном курсовом проекте логика приложения задумана так, что пользователь, который не вошел в систему имеет доступ только к страницам регистрации и авторизации. Если пользователь авторизовался, то у него появляется доступ к возможностям приложения.

## 5.3 Главная страница

Для того, чтобы пользователь смог увидеть фильмы, ему достаточно зайти на сайт. Пример главной страницы сайта представлен на рисунке 5.4

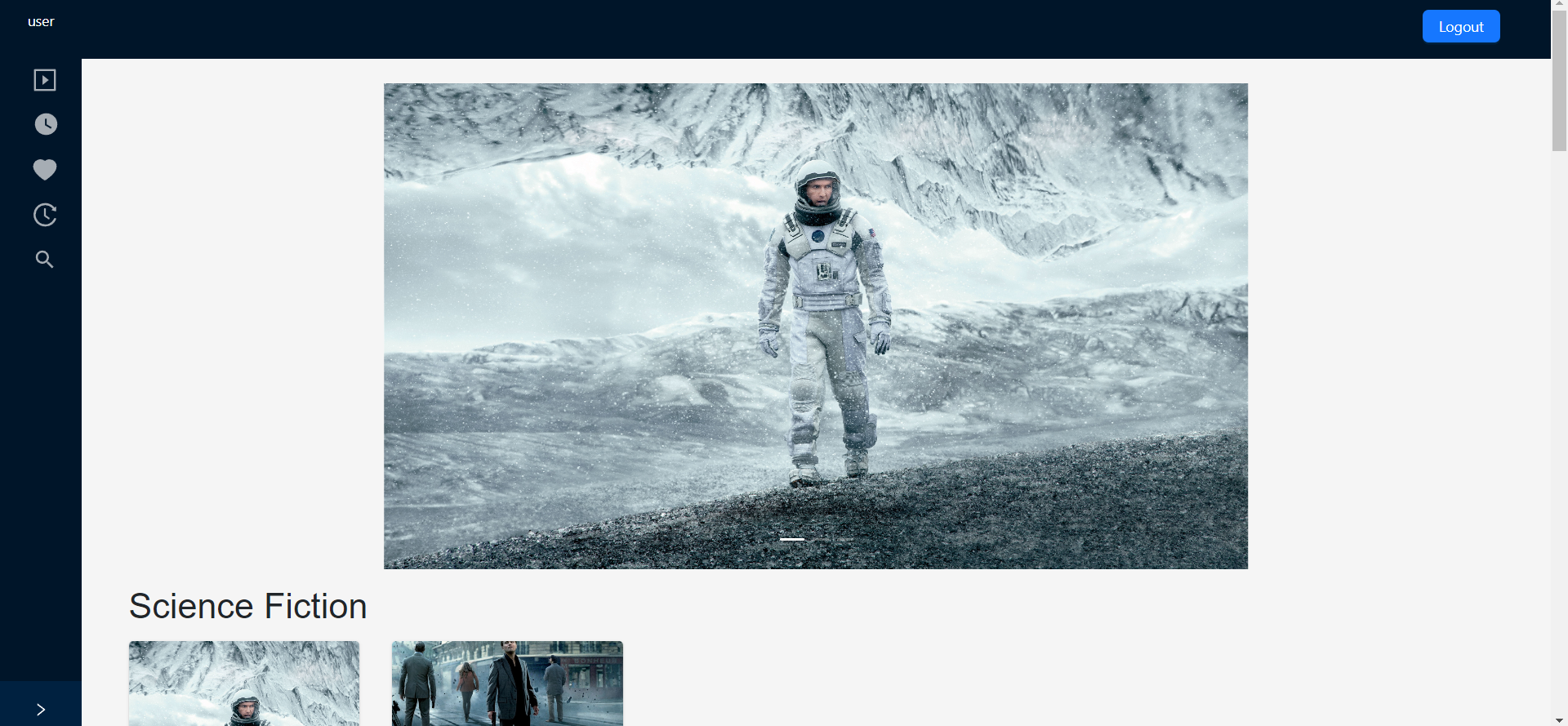


Рисунок 5.4 – Главная страница приложения

На данной странице можно просматривать последние 5 добавленных фильмов, а также все фильмы по жанрам. При нажатии на какой-либо из фильмов пользователь переадресуется на страницу фильма.

Также слева есть боковая панель с помощью, которой с любой страница можно перейти на:

– Главную страницу;

– Страницу фильмов, добавленных в “посмотреть позже”;

– Страницу фильмов, добавленных в “любимые”;

– Страницу истории просмотров;

– Страницу поиска.

## 5.4 Страница фильма

При переходе на страницу фильма пользователь обнаружит разнообразную информацию и функциональность. На верхней части страницы будут отображены название фильма, а также постер, трейлер и рейтинг фильма.

Рядом с названием фильма будут располагаться кнопки, позволяющие пользователю добавить фильм в разделы «Любимые» или «Посмотреть позже». Это предоставит пользователю удобный способ сохранить фильмы, которые его заинтересовали, и вернуться к ним позже.

Этот макет представлен на рисунке 5.7.

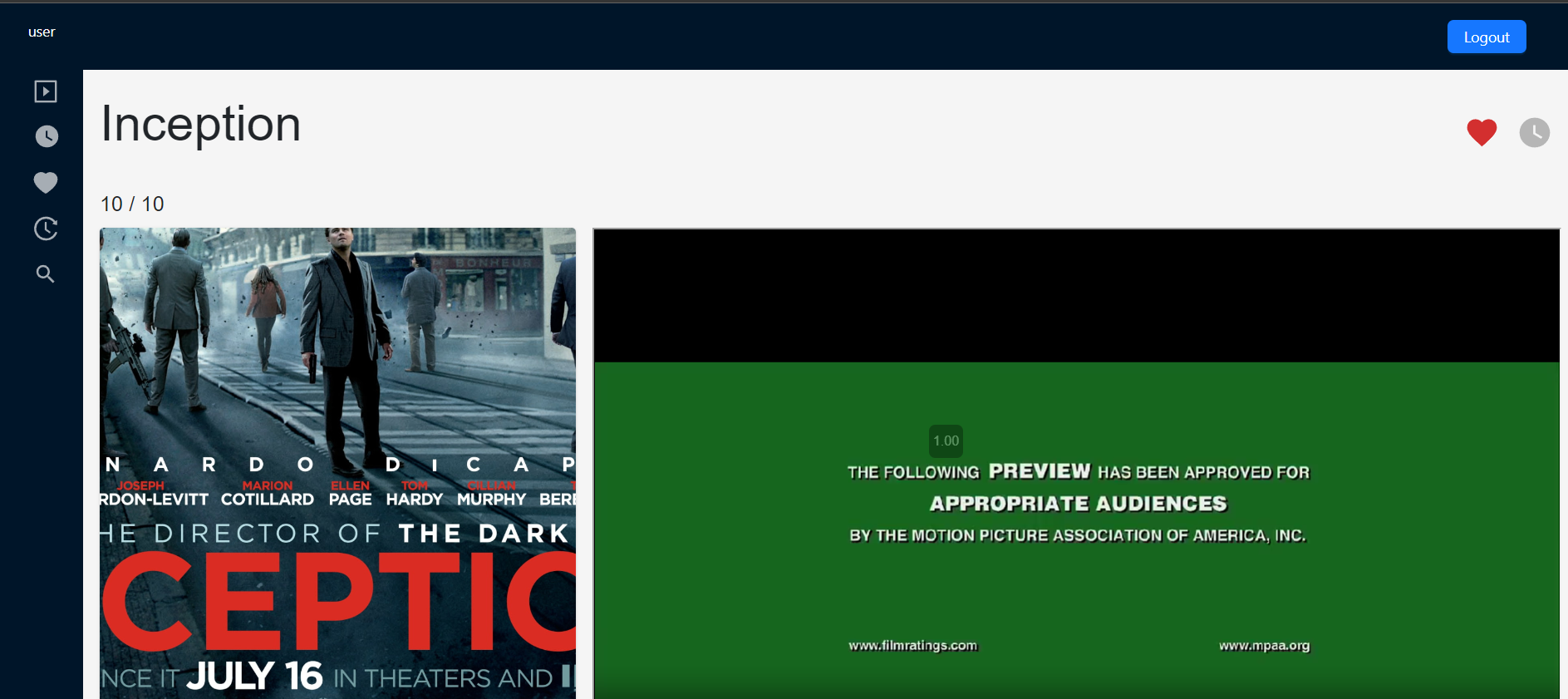


Рисунок 5.7 – Пример страницы фильма

Ниже на странице будут указаны год выпуска фильма, его продолжительность и жанры, а также краткое описание фильма. Пример такого представления можно увидеть на рисунке 5.8.

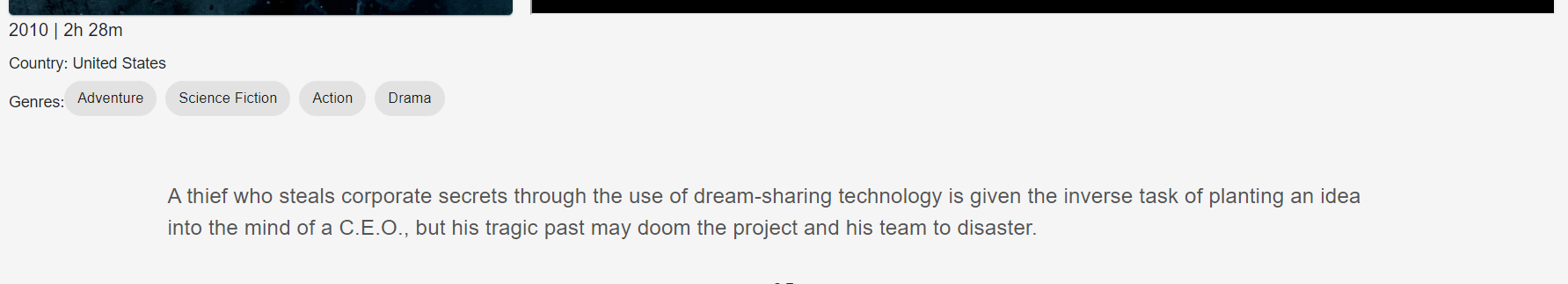


Рисунок 5.8 – Пример страницы фильма

Далее на странице, после информации о фильме, будут отображены все доступные трейлеры фильма, а также связанные с ним изображения. Это позволяет пользователям ознакомиться с превью фильма и получить дополнительную информацию о его содержании и визуальном оформлении. Пользователь имеет возможность выбрать конкретные трейлеры, которые он желает просмотреть.

На странице также будет присутствовать кнопка, позволяющая пользователю продолжить просмотр фильма с последней сессии. Эта функциональность особенно полезна для пользователей, которые прервут просмотр фильма и хотят вернуться к нему позже. Пользователи смогут удобно возобновлять просмотр и продолжать фильм с того момента, на котором они остановились, без необходимости искать нужное место в фильме вручную.

Пример страницы с описанным функционалом представлен на рисунке 5.9.

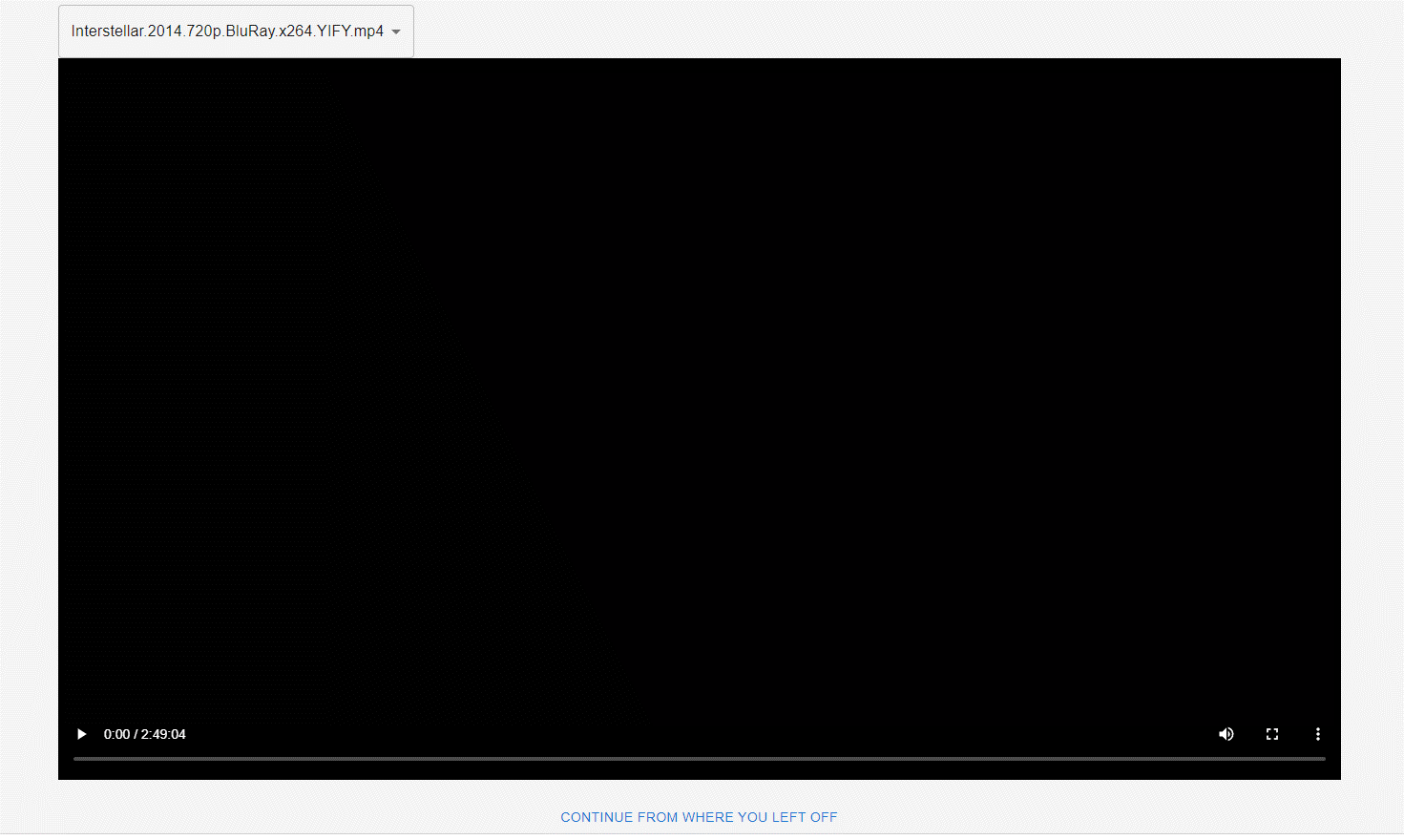


Рисунок 5.9 – Пример страницы фильма

На сайте также есть информация о команде фильма. А также есть возможность оставить комментарий с оценкой на фильм. Пользователь может просматривать комментарии всех пользователей. Пример приведён на рисунке 5.10.

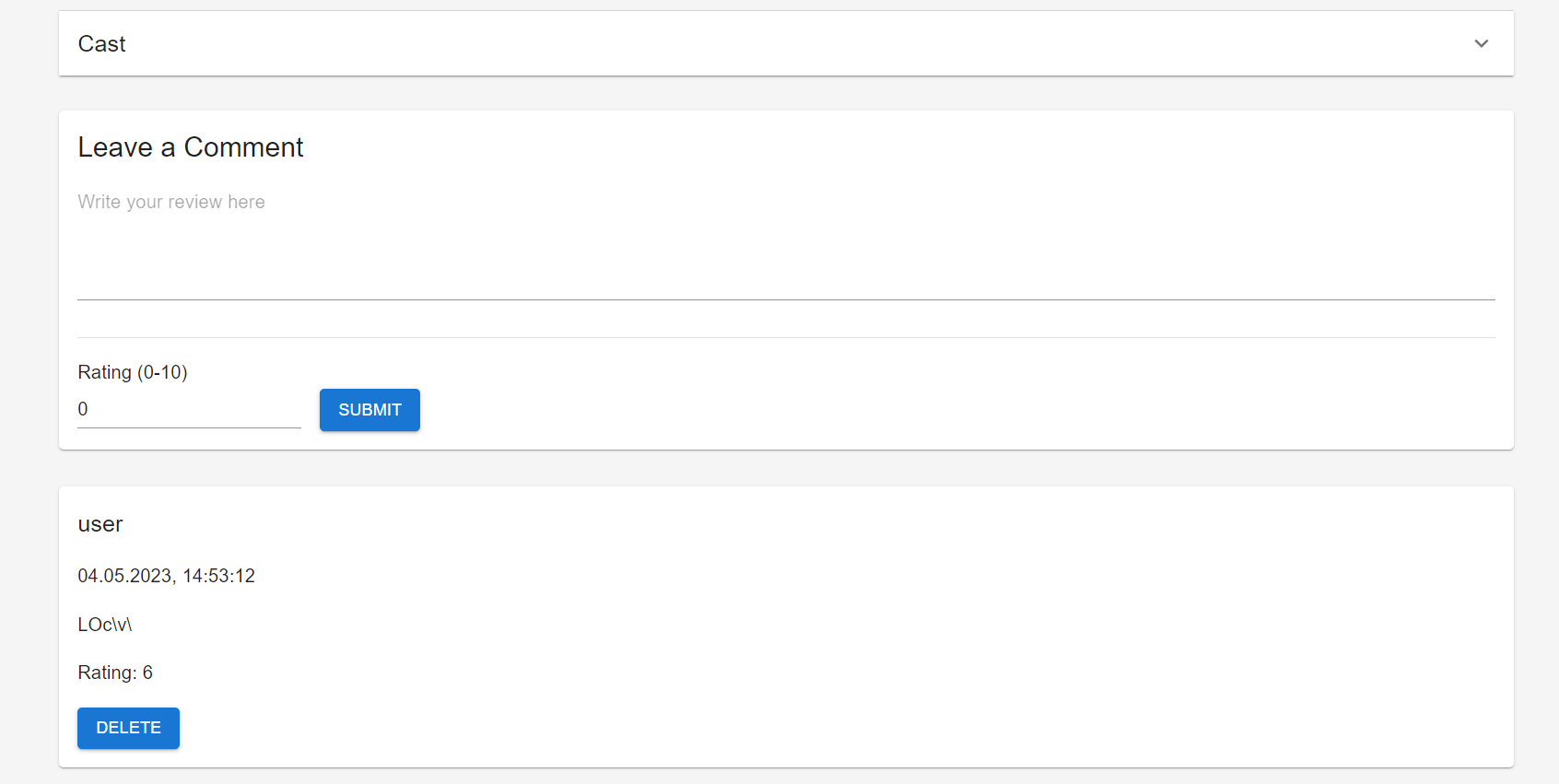


Рисунок 5.10 – Пример страницы фильма

Таким образом была рассмотрена страница фильма и её функционал.

## 5.5 Администрирование

В данном курсовом проекте была разработана возможность администрирования. Если пользователем зашёл на сайт с ролью админа, то в панели слева у него появится доступ к админ панели. Админ панель приведена на рисунке 5.11

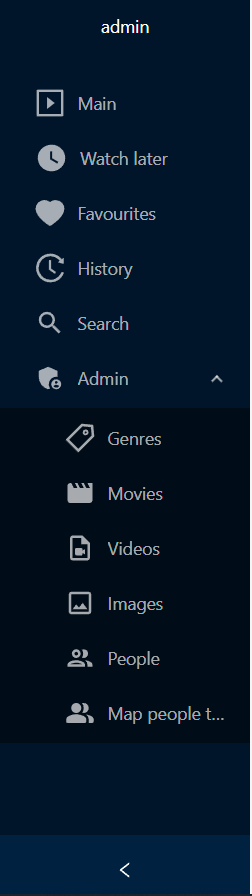


Рисунок 5.11 – Админ панель

Администратор приложения онлайн кинотеатра отвечает за управление и поддержку функциональности приложения. Администратор будет заниматься следующими задачами:

– Управление жанрами фильмов: Администратор сможет добавлять, редактировать и удалять жанры фильмов через административную панель. Это включает создание новых жанров и обновление существующих записей жанров.

– Управление изображениями: Администратор сможет загружать изображения, связанные с фильмами, и управлять ими. Он сможет добавлять новые изображения, изменять существующие и удалять их по необходимости.

– Управление людьми: Администратор сможет добавлять информацию о людях, связанных с фильмами, таких как актеры, режиссеры и другие участники съемочной группы. Он сможет добавлять новых людей, и удалять записи о них.

– Управление видео: Администратор сможет добавлять видео, такие как трейлеры фильмов, и управлять ими. Он сможет загружать новые видео, изменять существующие и удалять их по необходимости.

– Управление информацией о фильмах: Администратор сможет добавлять новые фильмы, редактировать существующую информацию о фильмах, такую как название, описание, дата выхода, страна производства и рейтинг. Он также сможет управлять жанрами, видео, изображениями и людьми, связанными с каждым фильмом.

Админ панели для создания/изменения фильмов приведен на рисунке 5.12.

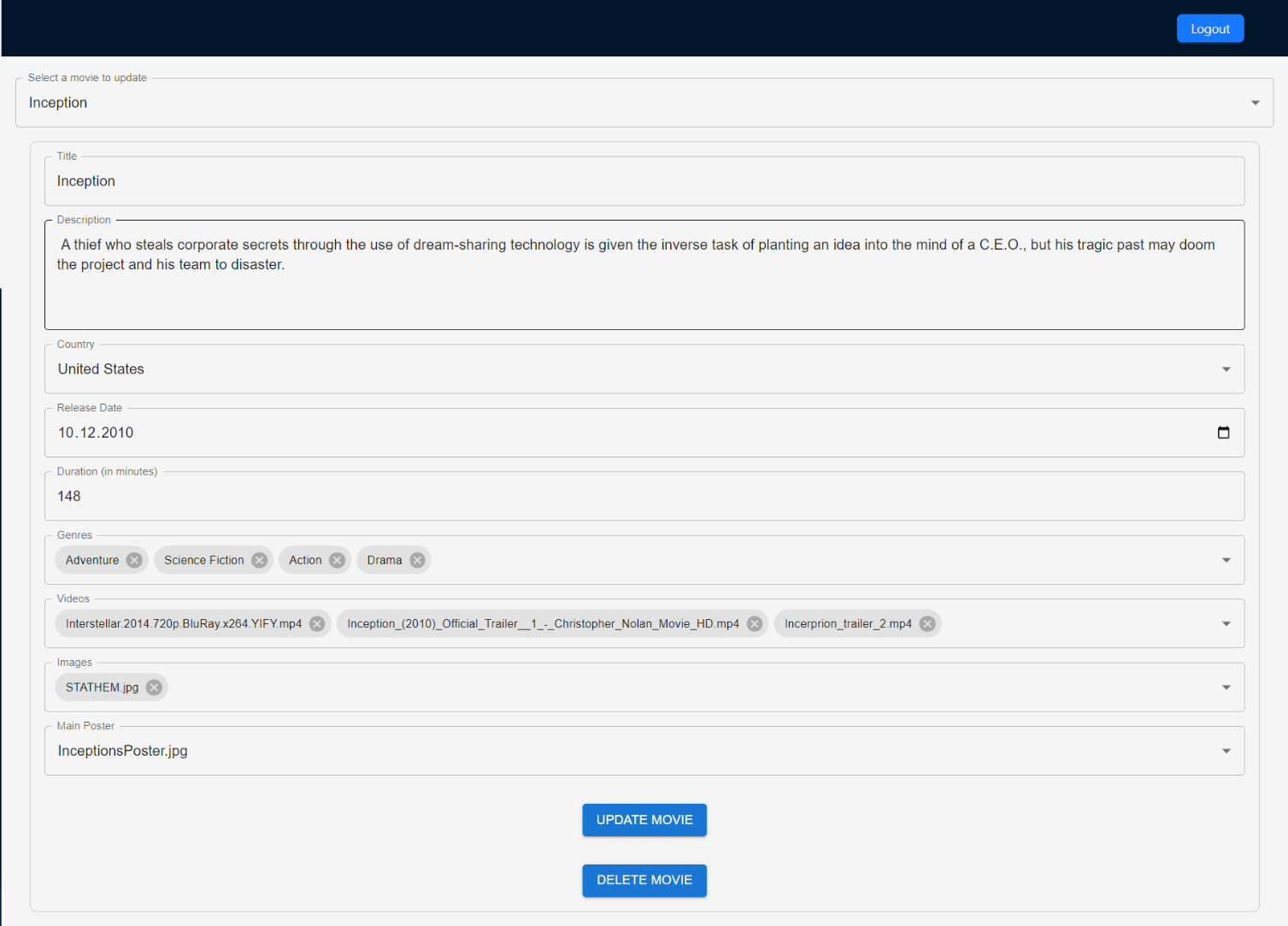


Рисунок 5.12 – Админ панели для создания/изменения фильмов

Таким образом был рассмотрен процесс администрирования.

# Заключение

В заключение данного курсового проекта можно отметить успешное выполнение поставленных задач и достижение требуемого функционала. В процессе разработки были исследованы основные аналоги веб-приложений, анализированы их преимущества и недостатки, что позволило лучше понять и определить возможности рассматриваемого веб-приложения.

Веб-приложение выполнено с использованием асинхронного программирования. Оно реализовано на языке TypeScript на платформе Node.js, что делает его кроссплатформенным. В качестве системы управления базой данных использована postgresql. Для разработки клиентской части использовалась библиотека React совместно с Redux.

В процессе выполнения курсовой работы была спроектирована база данных для хранения в ней информации. База данных была разработана с помощью схемы модели Prisma. Также был создан сервер и клиент с пользовательским интерфейсом.

Также следует отметить, что для данного проекта был создан сертификат для использования HTTPS протокола.

При разработке выполнены следующие пункты:

– сохранение информации в базе данных;

– организация регистрации и входа в систему;

– обмен данными между клиентом и сервером производиться по протоколу HTTPS;

– возможность просмотра, добавления, удаления, изменения фильмов;

– возможность просмотра, добавления, удаления, информации о фильмах;

– возможность поиска фильмов;

– возможность оставлять комментарии с оценками;

– отмечать понравившиеся;

– добавлять фильмы в «посмотреть позже»;

– просматривать историю.

При составлении руководства пользователя была описана работа с веб-приложением. Тестирование приложения подтвердило его надежность и устойчивость к возможным ошибкам.

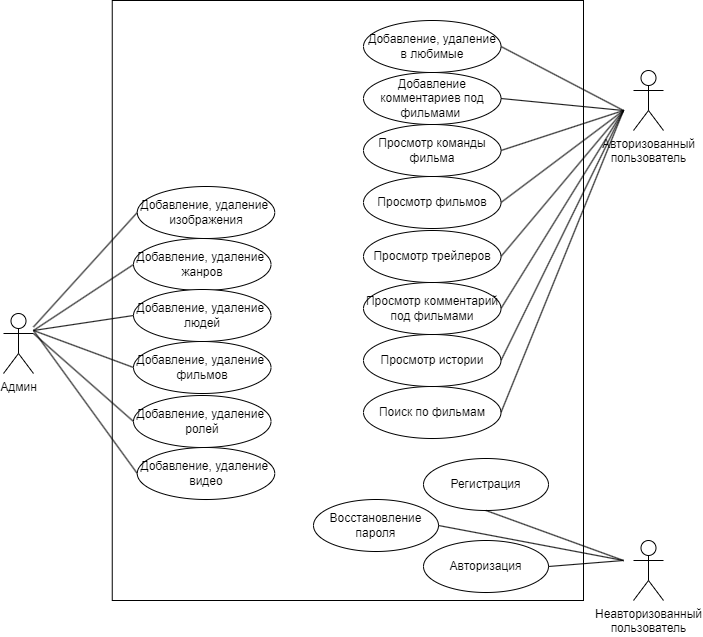
В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объёме.

# Список используемых источников

1. Документация React [Электронный ресурс] / react.dev– Режим доступа: <https://react.dev/learn/choosing-the-state-structure> – Дата доступа: 17.05.2023.
2. Nest документация [Электронный ресурс] / docs.nestjs – Режим доступа: https://docs.nestjs.com/ – Дата доступа: 17.05.2023.

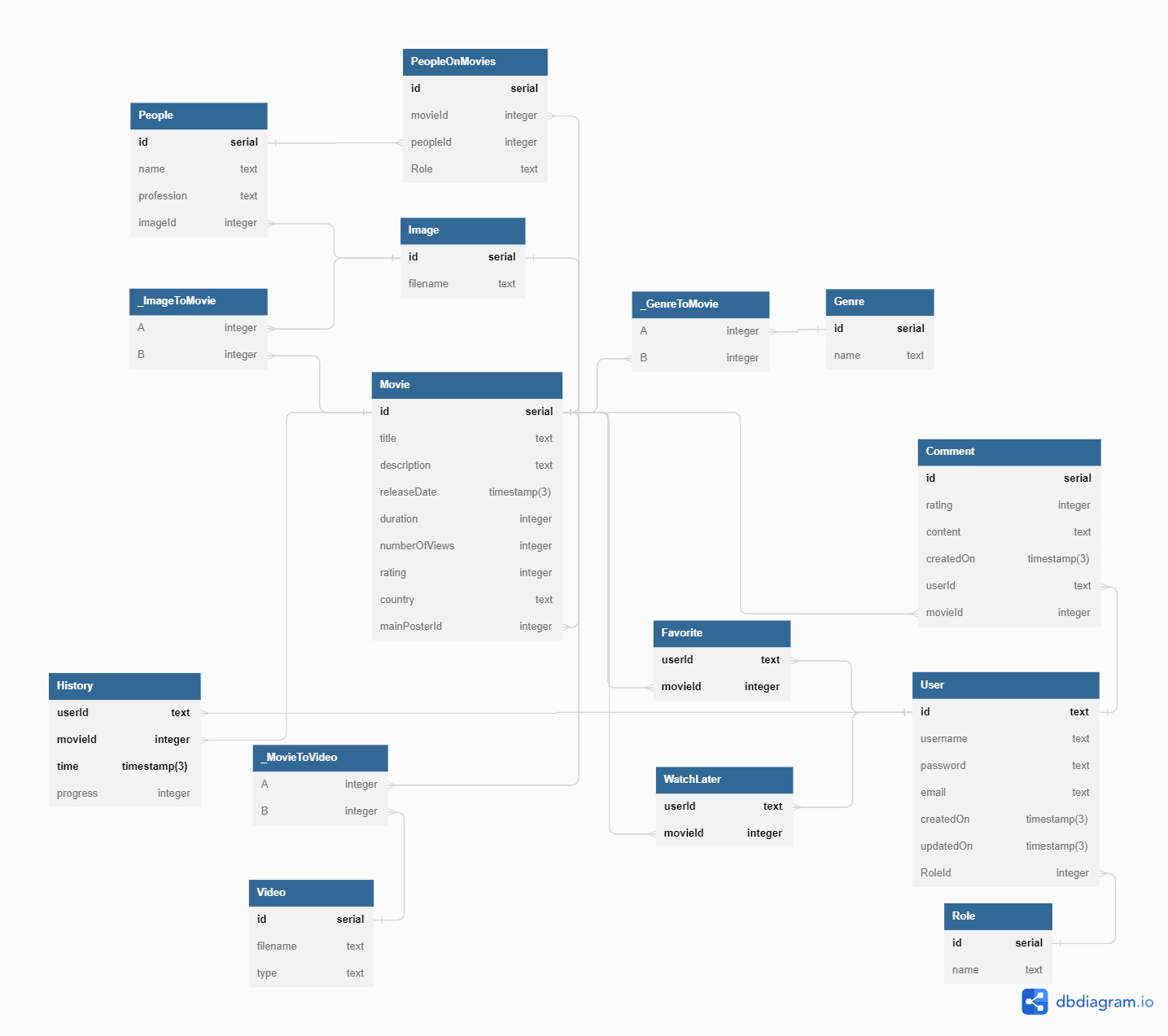
# Приложение А

Диаграмма вариантов использования



# Приложение Б

Модель базы данных



# Приложение В

Файл «schema.prisma»

model Genre {

id Int @id @default(autoincrement())

name String @unique

movies Movie[]

}

model Video {

id Int @id @default(autoincrement())

filename String @unique

type String

movies Movie[]

}

model Image {

id Int @id @default(autoincrement())

filename String @unique

People People[]

movies Movie[]

movie Movie[] @relation(«MainPoster»)

}

model People {

id Int @id @default(autoincrement())

name String @unique

profession String

imageId Int

Image Image @relation(fields: [imageId], references: [id])

movies PeopleOnMovies[]

}

model Movie {

id Int @id @default(autoincrement())

title String

description String

releaseDate DateTime

duration Int

numberOfViews Int @default(0)

rating Int?

country String?

Genre Genre[]

Video Video[]

Image Image[]

People PeopleOnMovies[]

Favorite Favorite[]

History History[]

Comment Comment[]

mainPosterId Int?

mainPoster Image? @relation(«MainPoster», fields: [mainPosterId], references: [id])

WatchLater WatchLater[]

}

model PeopleOnMovies {

id Int @id @default(autoincrement())

movieId Int

peopleId Int

Role String

Movie Movie @relation(fields: [movieId], references: [id], onDelete: Cascade)

People People @relation(fields: [peopleId], references: [id])

@@unique([movieId, peopleId, Role])

}

model Favorite {

userId String

movieId Int

User User @relation(fields: [userId], references: [id], onDelete: Cascade)

Movie Movie @relation(fields: [movieId], references: [id], onDelete: Cascade)

@@id([userId, movieId])

}

model WatchLater {

userId String

movieId Int

User User @relation(fields: [userId], references: [id], onDelete: Cascade)

Movie Movie @relation(fields: [movieId], references: [id], onDelete: Cascade)

@@id([userId, movieId])

}

model History {

userId String

movieId Int

time DateTime @default(now())

progress Float @default(0)

User User @relation(fields: [userId], references: [id], onDelete: Cascade)

Movie Movie @relation(fields: [movieId], references: [id], onDelete: Cascade)

@@id([userId, movieId, time])

}

model Comment {

id Int @id @default(autoincrement())

rating Int

content String?

createdOn DateTime @default(now())

userId String

movieId Int

User User @relation(fields: [userId], references: [id], onDelete: Cascade)

Movie Movie @relation(fields: [movieId], references: [id], onDelete: Cascade)

@@unique([userId, movieId])

}

model Role {

id Int @id @default(autoincrement())

name String @unique

User User[]

}

model User {

id String @id @default(uuid())

username String @unique

password String

email String @unique

createdOn DateTime @default(now())

updatedOn DateTime @default(now())

RoleId Int

Role Role @relation(fields: [RoleId], references: [id])

Favorite Favorite[]

History History[]

Comment Comment[]

WatchLater WatchLater[]

}