Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

Кафедра информационных систем и программной инженерии

**КУРСОВОЙ ПРОЕКТ**

Проектирование и разработка программной системы информационной системы «Онлайн фильмотека»

Выполнил: студент гр. ПРИ-123

Парфенов Р.С.

Принял: доц. каф. ИСПИ

Вершинин В.В.

Владимир, 2025

**Аннотация**

В рамках курсового проекта была проанализирована предметная область «Онлайн фильмотека», а также разработано приложение для данной информационной системы. Данный проект включает в себя этапы анализа, проектирования, реализации и тестирования приложения. Реализация программной системы производилась с использованием технологии .NET.

Созданное веб приложение предназначено для различного множества пользователей, которые заинтересованы подбором фильмов, описаний к ним, личными списками и хорошей системе рекомендаций.

**Annotation**

As part of the course project, the subject area "Online Movie Library" was analyzed, and an application for this information system was developed. The project includes the stages of analysis, design, implementation, and testing of the application. The software system was implemented using .NET technology.

The developed web application is intended for a diverse range of users interested in movie selection, film descriptions, personal watchlists, and a robust recommendation system.

**Оглавление**

[Введение 4](#_Toc216214965)

[1. Постановка задачи 5](#_Toc216214966)

[2. Описание предметной области 6](#_Toc216214967)

[2.1 Общее описание предметной области 6](#_Toc216214968)

[2.2 Пользователи разрабатываемой подсистемы 6](#_Toc216214969)

[2.3 Начальная оценка и выделение сущностей 6](#_Toc216214970)

[2.4 Словарь предметной области 7](#_Toc216214971)

[2.5 Требования к системе 7](#_Toc216214972)

[2.5.1 Функциональные требования 7](#_Toc216214973)

[2.5.2 Нефункциональные требования 10](#_Toc216214974)

[3. Проектирование приложения 11](#_Toc216214975)

[3.1 Диаграммы UML 11](#_Toc216214976)

[3.2 Расширенное описание прецедентов 11](#_Toc216214977)

[4. Реализация системы 15](#_Toc216214978)

[4.1 Разработка базы данных. 15](#_Toc216214979)

[4.2 Веб-приложение. 19](#_Toc216214980)

[4.2.1 Серверная часть. 19](#_Toc216214981)

[4.2.2 Клиентская часть. 22](#_Toc216214982)

[4.2.3 Результат реализации. 22](#_Toc216214983)

[5. Тестирование приложения 27](#_Toc216214984)

[Заключение 39](#_Toc216214985)

[Список использованных источников 40](#_Toc216214986)

[Приложение А 41](#_Toc216214987)

[Приложение Б 42](#_Toc216214988)

[Приложение В 44](#_Toc216214989)

[Приложение Г 45](#_Toc216214990)

[Приложение Д 46](#_Toc216214991)

[Приложение Е 47](#_Toc216214992)

# **Введение**

В современном цифровом мире онлайн-фильмотеки становятся всё более востребованными, поскольку пользователи стремятся быстро находить интересующие их фильмы, получать персонализированные предложения и управлять собственными коллекциями контента. Однако многие существующие платформы либо не предоставляют гибких инструментов для создания и редактирования личных списков фильмов, либо предлагают рекомендации, основанные лишь на примитивных алгоритмах, не учитывающих индивидуальные предпочтения пользователя.

В рамках данного курсового проекта разрабатывается веб-приложение — онлайн-фильмотека, ориентированная на удовлетворение именно этих потребностей. Система позволяет пользователям не только просматривать подробную информацию о фильмах (жанр, режиссёр, актёры, сюжет, рейтинг и т.д.), но и формировать собственные списки: «К просмотру», «Любимые», «Просмотренные».

Особое внимание уделено реализации интеллектуальной системы рекомендаций. Алгоритм учитывает явные действия пользователя (оценки, добавление в списки, частота просмотра информации о фильме). На основе этих данных формируются персонализированные предложения, которые со временем адаптируются под меняющиеся вкусы пользователя.

# **Постановка задачи**

Цель работы: разработать веб-приложение "Онлайн-фильмотека", которое предоставит пользователям платформу для поиска информации о фильмах, ведения персональных списков просмотра, обмена оценками и получения рекомендаций.

Для достижения поставленной цели необходимо решить следующие задачи:

* Поддержка регистрации и авторизации пользователей: реализовать возможность создания учетной записи и входа в систему для посетителей сайта.
* Реализация каталога фильмов: обеспечить доступ к каталогу фильмов с функциями поиска, сортировки и фильтрации для удобной навигации.
* Управление персональными списками фильмов: предоставить пользователям инструменты для добавления фильмов в личные списки, их удаления и изменения статуса просмотра.
* Реализация системы оценок и рекомендаций: позволить пользователям ставить оценки фильмам.
* Поддержка профилей пользователей: создать публичные профили, где пользователи смогут просматривать информацию и списки фильмов друг друга.
* Администрирование контента и пользователей: разработать для администраторов функционал по управлению каталогом фильмов (добавление, изменение, удаление) и учетными записями пользователей.

# **Описание предметной области**

## Общее описание предметной области

Программная система "Онлайн-фильмотека" представляет собой веб-приложение, предназначенное для любителей кино.

Система позволяет обычным посетителям сайта просматривать каталог фильмов, используя функции сортировки и фильтрации, осуществлять поиск фильмов по названию, а также регистрироваться и входить в систему.

Зарегистрированные пользователи имеют возможность управлять личными списками фильмов, а именно добавлять и удалять из них фильмы, а также изменять их статус. Они могут ставить оценки фильмам, на основе которых система будет формировать список рекомендованных фильмов. Также пользователи могут просматривать профили других участников и их списки фильмов.

Администраторы сайта могут управлять пользователями платформы (изменять их данные и роль, удалять учетные записи), а также управлять каталогом фильмов (добавлять, изменять и удалять фильмы).

## Пользователи разрабатываемой подсистемы

*Посетитель* – человек, не зарегистрированный в системе, являющийся гостем, который может просматривать каталог фильмов, а также публичные профиля пользователей.

*Пользователь* – человек, у которого могут быть личные списки фильмов и настроенная система рекомендаций.

*Администратор* – человек, занимающийся администрированием сайта приложения.

## Начальная оценка и выделение сущностей

Для предметной области «Онлайн фильмотека» были выделены следующие сущности: посетитель сайта (гость), пользователь сайта (зарегистрированный), администратор, фильм, каталог, список фильмов (личный), оценка, рекомендация, профиль пользователя.

## Словарь предметной области

*Посетитель сайта -* Неавторизованный пользователь, который может просматривать каталог, искать фильмы, а также регистрироваться или входить в систему.

*Пользователь -* Авторизованный пользователь, который имеет доступ к расширенным функциям: управление личным списком фильмов, выставление оценок, получение рекомендаций и просмотр профилей других пользователей.

*Администратор* - Пользователь с расширенными правами. Управляет каталогом фильмов (добавление, изменение, удаление) и пользователями системы (изменение данных, роли, удаление).

*Фильм* - Основная сущность контента в системе. Имеет атрибуты, такие как название, описание, год выпуска, жанр и т.д.

*Каталог* - Общий список всех фильмов, доступных в системе. Поддерживает функции поиска, сортировки и фильтрации.

*Список фильмов* - Персональный список, который создает и ведет зарегистрированный пользователь. Позволяет добавлять, удалять фильмы и менять их статус (например, "Просмотрено", "Запланировано").

*Оценка* - Численный или иной рейтинг, который пользователь выставляет фильму. Используется для формирования рекомендаций.

*Рекомендация* - Фильм, предложенный системой пользователю на основе его предыдущих оценок и предпочтений.

*Профиль пользователя -* Страница с публичной информацией о пользователе, включая его личные списки фильмов.

## Требования к системе

* + 1. **Функциональные требования**

1. Общие требования (для всех пользователей)

ФТ 1.1: Система должна предоставлять доступ к общему каталогу фильмов.

ФТ 1.2: Система должна обеспечивать возможность просмотра детальной страницы каждого фильма в каталоге.

2. Требования для роли "Посетитель сайта" (неавторизованный пользователь)

ФТ 2.1: Система должна предоставлять возможность регистрации новой учетной записи.

ФТ 2.2: Система должна предоставлять возможность входа в существующую учетную запись.

ФТ 2.3: Система должна позволять выполнять поиск фильма в каталоге по его названию.

ФТ 2.4: Система должна предоставлять инструменты для сортировки списка фильмов в каталоге (например, по названию, году выпуска, рейтингу).

ФТ 2.5: Система должна предоставлять инструменты для фильтрации списка фильмов в каталоге (например, по жанру, году).

ФТ 2.6: Система должна иметь функцию "Случайный фильм" для получения рекомендации произвольного фильма из каталога.

3. Требования для роли "Пользователь" (авторизованный пользователь)

ФТ 3.1: Все функции, доступные "Посетителю сайта", должны быть доступны и "Пользователю".

ФТ 3.2: Управление личным списком фильмов

ФТ 3.2.1: Система должна позволять пользователю добавлять любой фильм из каталога в свой личный список.

ФТ 3.2.2: Система должна позволять пользователю удалять фильм из своего личного списка.

ФТ 3.2.3: Система должна позволять пользователю изменять статус фильма в своем списке (например, "Планирую посмотреть", "Просмотрено", "Брошено").

ФТ 3.3: Оценки и рекомендации

ФТ 3.3.1: Система должна позволять пользователю ставить оценку просмотренному фильму.

ФТ 3.3.2: Система должна формировать и отображать для пользователя список рекомендованных фильмов на основе его оценок.

ФТ 3.3.3: Система должна предоставлять возможность получить одну конкретную рекомендацию фильма.

ФТ 3.4: Социальное взаимодействие

ФТ 3.4.1: Система должна позволять находить профили других пользователей.

ФТ 3.4.2: Система должна позволять просматривать профиль другого пользователя.

ФТ 3.4.3: Система должна позволять просматривать списки фильмов другого пользователя.

4. Требования для роли "Администратор"

ФТ 4.1: Все функции, доступные "Пользователю", должны быть доступны и "Администратору".

ФТ 4.2: Управление каталогом.

ФТ 4.2.1: Система должна позволять администратору добавлять новый фильм в общий каталог.

ФТ 4.2.2: Система должна позволять администратору изменять информацию о существующем фильме.

ФТ 4.2.3: Система должна позволять администратору удалять фильм из каталога.

ФТ 4.3: Управление пользователями

ФТ 4.3.1: Система должна предоставлять администратору доступ к списку всех зарегистрированных пользователей.

ФТ 4.3.2: Система должна позволять администратору изменять данные пользователя (имя, почта).

ФТ 4.3.3: Система должна позволять администратору изменять роль пользователя (например, с "Пользователь" на "Администратор").

ФТ 4.3.4: Система должна позволять администратору удалять учетную запись пользователя.

* + 1. **Нефункциональные требования**
* Реализация приложения с использованием платформы .NET;
* Использование базы данных MySQL;
* Использование Entity Framework для организации работы с базой данных.
* Разработка клиентской части приложения с использованием фреймворка React.js.

# **Проектирование приложения**

## Диаграммы UML

Диаграмма прецедентов, визуализирующая взаимодействие пользователей системы с ее функционалом, представлена в приложении A.

Диаграмма классов представлена в приложении Б.

Диаграмма состояний для сущности «Фильм пользователя» представлена в приложении В.

Диаграмма последовательностей для прецедента «Оценить фильм» представлена в приложении Г.

Диаграмма видов деятельности представлена в приложении Д.

## Расширенное описание прецедентов

**Расширенное описание прецедента «Оценка фильма»**

Предусловие: Пользователь авторизован в системе.

Действующее лицо: Пользователь

Основной поток:

1. Пользователь выбирает фильм для оценки (через каталог, свой список, случайный фильм или рекомендации).

2. Пользователь вызывает функцию оценки фильма, передавая идентификатор фильма, оценку (от 1 до 10) и опционально комментарий (до 2000 символов).

3. Система проверяет, что фильм существует в базе данных.

4. Система проверяет наличие статусов "Просмотрено" и "Оценен" в справочнике статусов.

5. Система проверяет, что фильм уже добавлен в личный список пользователя и имеет статус "Просмотрено" или "Оценен".

6. Система сохраняет оценку, устанавливает статус фильма в "Оценен" и сохраняет комментарий (если указан).

7. Система возвращает обновленную информацию о фильме в списке пользователя.

Альтернативный поток №1: Если фильм не найден в базе данных, система возвращает ошибку "Фильм не найден".

Альтернативный поток №2: Если обязательные статусы ("Просмотрено" или "Оценен") отсутствуют в справочнике, система возвращает ошибку с указанием отсутствующего статуса.

Альтернативный поток №3: Если фильм не добавлен в личный список пользователя или имеет статус, отличный от "Просмотрено" или "Оценен", система возвращает ошибку "Нельзя оценить фильм, пока он не добавлен в статус «Просмотрено»".

Альтернативный поток №4: Если переданы некорректные данные (оценка вне диапазона 1-10, превышена длина комментария), система возвращает ошибку валидации.

Альтернативный поток №5: Если пользователь уже оценил фильм ранее, система позволяет изменить оценку и комментарий, обновляя существующую запись.

Постусловие: Фильм получил оценку от пользователя, статус фильма в личном списке изменен на "Оценен", оценка и комментарий сохранены в базе данных.

**Расширенное описание прецедента «Получить список рекомендованных фильмов»**

Предусловие: Пользователь авторизован в системе.

Действующее лицо: Пользователь

Основной поток:

1. Пользователь открывает раздел рекомендаций.

2. Система определяет идентификатор пользователя из токена авторизации.

3. Система получает список фильмов, которые уже есть в личном списке пользователя (для исключения из рекомендаций).

4. Система получает жанровые предпочтения пользователя (жанры и их веса из таблицы UserGenreInterests).

5. Если у пользователя есть жанровые предпочтения:

- Система формирует словарь весов по жанрам.

- Система выбирает фильмы, которые имеют хотя бы один из предпочитаемых жанров и не находятся в личном списке пользователя.

- Для каждого фильма система вычисляет рейтинг рекомендации как сумму весов всех совпадающих жанров.

- Система сортирует фильмы по убыванию рейтинга, затем по названию.

- Система возвращает до 20 фильмов с наивысшим рейтингом.

6. Пользователь просматривает список рекомендованных фильмов с указанием рейтинга рекомендации для каждого.

Альтернативный поток №1: Если у пользователя нет жанровых предпочтений:

- Система выбирает случайный фильм из каталога, исключая фильмы из личного списка пользователя.

- Система возвращает список с одним случайным фильмом и сообщением "Предпочтения не заданы, показан случайный фильм".

Альтернативный поток №2: Если в каталоге не осталось фильмов вне личного списка пользователя:

- Система возвращает пустой список с сообщением "В каталоге не осталось фильмов вне вашего списка".

Альтернативный поток №3: Если не удалось определить пользователя из токена:

- Система возвращает ошибку авторизации "Не удалось определить пользователя".

Альтернативный поток №4: Если по текущим предпочтениям не удалось сформировать рекомендации (нет фильмов с подходящими жанрами):

- Система возвращает пустой список с сообщением "Не удалось сформировать рекомендации по текущим предпочтениям".

Постусловие: Пользователь получил список рекомендованных фильмов, отсортированный по релевантности на основе жанровых предпочтений, или случайный фильм, если предпочтения не заданы.

**Расширенное описание прецедента «Добавить новый фильм»**

Предусловие: Пользователь авторизован в системе и обладает правами администратора (роль "admin").

Действующее лицо: Администратор

Основной поток:

1. Администратор переходит в раздел добавления фильма.

2. Администратор заполняет форму с данными фильма:

- Обязательное поле: название (до 255 символов).

- Опциональные поля: описание, год выпуска, режиссер (до 255 символов), ссылка на постер (до 255 символов), продолжительность в минутах, список идентификаторов жанров, список идентификаторов тегов, список идентификаторов стран, список ссылок на трейлеры (каждая до 255 символов).

3. Администратор отправляет данные на сервер.

4. Система проверяет валидность модели данных.

5. Система выполняет валидацию данных:

- Проверяет, что название не пустое.

- Проверяет, что продолжительность (если указана) является положительным числом.

- Проверяет, что год выпуска (если указан) не меньше 1888.

- Проверяет длину строковых полей (режиссер, постер, трейлеры - не более 255 символов).

- Проверяет существование всех указанных идентификаторов жанров в базе данных.

- Проверяет существование всех указанных идентификаторов тегов в базе данных.

- Проверяет существование всех указанных идентификаторов стран в базе данных.

6. Система создает новый фильм в базе данных, связывая его с указанными жанрами, тегами, странами и трейлерами.

7. Система возвращает созданный фильм с полной информацией.

Альтернативный поток №1: Если обязательные поля не заполнены или содержат некорректные данные:

- Система возвращает ошибку валидации с описанием проблемы (например, "Название фильма обязательно для заполнения", "Продолжительность должна быть положительным числом", "Год релиза не может быть меньше 1888").

Альтернативный поток №2: Если указаны несуществующие идентификаторы жанров, тегов или стран:

- Система возвращает ошибку с указанием, какие идентификаторы не найдены (например, "Не найдены жанры с идентификаторами: 5, 12").

Альтернативный поток №3: Если длина строковых полей превышает допустимые значения:

- Система возвращает ошибку с указанием превышенного поля (например, "Ссылка на постер не может превышать 255 символов").

Альтернативный поток №4: Если пользователь не авторизован или не имеет прав администратора:

- Система возвращает ошибку авторизации.

Постусловие: Новый фильм добавлен в каталог системы со всеми указанными данными и связями (жанры, теги, страны, трейлеры).

# **Реализация системы**

## Разработка базы данных.

Скрипт создания разработанной базы данных, листинг sql кода:

-- Создание таблицы для хранения статусов фильмов в списках пользователей

CREATE TABLE statuses (

status\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL UNIQUE

);

-- Создание таблицы для хранения жанров

CREATE TABLE genres (

genre\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL UNIQUE

);

-- Создание таблицы для хранения тегов

CREATE TABLE tags (

tag\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL UNIQUE

);

-- Создание таблицы для хранения стран

CREATE TABLE countries (

country\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL UNIQUE

);

-- Создание таблицы для хранения информации о фильмах

CREATE TABLE movies (

movie\_id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

description TEXT,

release\_year INT,

director VARCHAR(255),

poster\_url VARCHAR(255),

duration\_minutes INT CHECK (duration\_minutes > 0)

);

-- Создание таблицы для трейлеров (один фильм может иметь несколько трейлеров)

CREATE TABLE trailers (

trailer\_id INT AUTO\_INCREMENT PRIMARY KEY,

movie\_id INT NOT NULL,

trailer\_url VARCHAR(255) NOT NULL,

FOREIGN KEY (movie\_id) REFERENCES movies(movie\_id) ON DELETE CASCADE

);

-- --- Связующие таблицы для фильмов ---

-- Связь фильмов и жанров (многие ко многим)

CREATE TABLE movie\_genres (

movie\_id INT NOT NULL,

genre\_id INT NOT NULL,

PRIMARY KEY (movie\_id, genre\_id),

FOREIGN KEY (movie\_id) REFERENCES movies(movie\_id) ON DELETE CASCADE,

FOREIGN KEY (genre\_id) REFERENCES genres(genre\_id) ON DELETE CASCADE

);

-- Связь фильмов и тегов (многие ко многим)

CREATE TABLE movie\_tags (

movie\_id INT NOT NULL,

tag\_id INT NOT NULL,

PRIMARY KEY (movie\_id, tag\_id),

FOREIGN KEY (movie\_id) REFERENCES movies(movie\_id) ON DELETE CASCADE,

FOREIGN KEY (tag\_id) REFERENCES tags(tag\_id) ON DELETE CASCADE

);

-- Связь фильмов и стран производства (многие ко многим)

CREATE TABLE movie\_countries (

movie\_id INT NOT NULL,

country\_id INT NOT NULL,

PRIMARY KEY (movie\_id, country\_id),

FOREIGN KEY (movie\_id) REFERENCES movies(movie\_id) ON DELETE CASCADE,

FOREIGN KEY (country\_id) REFERENCES countries(country\_id) ON DELETE CASCADE

);

-- --- Таблицы, связанные с пользователями ---

-- Создание таблицы пользователей

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(100) NOT NULL UNIQUE,

email VARCHAR(100) NOT NULL UNIQUE,

hashed\_password VARCHAR(255) NOT NULL,

`role` VARCHAR(50) NOT NULL DEFAULT 'user', -- 'role' - зарезервированное слово, используем кавычки

avatar\_url VARCHAR(255),

profile\_description TEXT

);

-- Предпочтения пользователей по жанрам с весом для рекомендаций

CREATE TABLE user\_genre\_interests (

user\_id INT NOT NULL,

genre\_id INT NOT NULL,

weight FLOAT NOT NULL DEFAULT 1.0 CHECK (weight > 0 AND weight <= 1.0),

PRIMARY KEY (user\_id, genre\_id),

FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE,

FOREIGN KEY (genre\_id) REFERENCES genres(genre\_id) ON DELETE CASCADE

);

-- Таблица для списков фильмов пользователя (статусы, оценки, комментарии)

CREATE TABLE user\_movies (

user\_movie\_id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT NOT NULL,

movie\_id INT NOT NULL,

status\_id INT,

score INT CHECK (score >= 1 AND score <= 10),

comment TEXT,

added\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

UNIQUE (user\_id, movie\_id),

FOREIGN KEY (user\_id) REFERENCES users(user\_id) ON DELETE CASCADE,

FOREIGN KEY (movie\_id) REFERENCES movies(movie\_id) ON DELETE CASCADE,

FOREIGN KEY (status\_id) REFERENCES statuses(status\_id) ON DELETE SET NULL

);

-- --- Наполнение справочников начальными данными ---

-- Добавление возможных статусов

INSERT INTO statuses (name) VALUES

('Буду смотреть'),

('Просмотрено'),

('Брошено'),

('Не буду смотреть'),

('Оценен');

-- Добавление 20 популярных жанров

INSERT INTO genres (name) VALUES

('Боевик'), ('Комедия'), ('Драма'), ('Фантастика'),

('Триллер'), ('Ужасы'), ('Детектив'), ('Фэнтези'),

('Мелодрама'), ('Анимация'), ('Приключения'), ('Исторический'),

('Биографический'), ('Мюзикл'), ('Документальный'), ('Военный'),

('Вестерн'), ('Семейный'), ('Криминал'), ('Мистика');

INSERT INTO countries (name) VALUES

('США'),

('Россия'),

('Канада'),

('Великобритания'),

('Франция'),

('Германия'),

('Испания'),

('Италия'),

('Австралия'),

('Япония'),

('Южная Корея'),

('Китай'),

('Индия'),

('Мексика'),

('Бразилия'),

('Аргентина'),

('Швеция'),

('Норвегия'),

('Дания'),

('Финляндия'),

('Польша'),

('Чехия'),

('Нидерланды'),

('Бельгия'),

('Швейцария'),

('Исландия');

INSERT INTO tags (name) VALUES

('Культовый'),

('Основано на реальных событиях'),

('Снято по книге'),

('Артхаус'),

('Фильм-нуар'),

('Киберпанк'),

('Постапокалипсис'),

('Супергерои'),

('Самураи'),

('Вампиры'),

('Зомби'),

('Путешествия во времени'),

('Космос'),

('Психологический'),

('Философский'),

('Тяжелый сюжет'),

('Легкий сюжет'),

('Семейный просмотр'),

('Темное фэнтези'),

('Роуд-муви'),

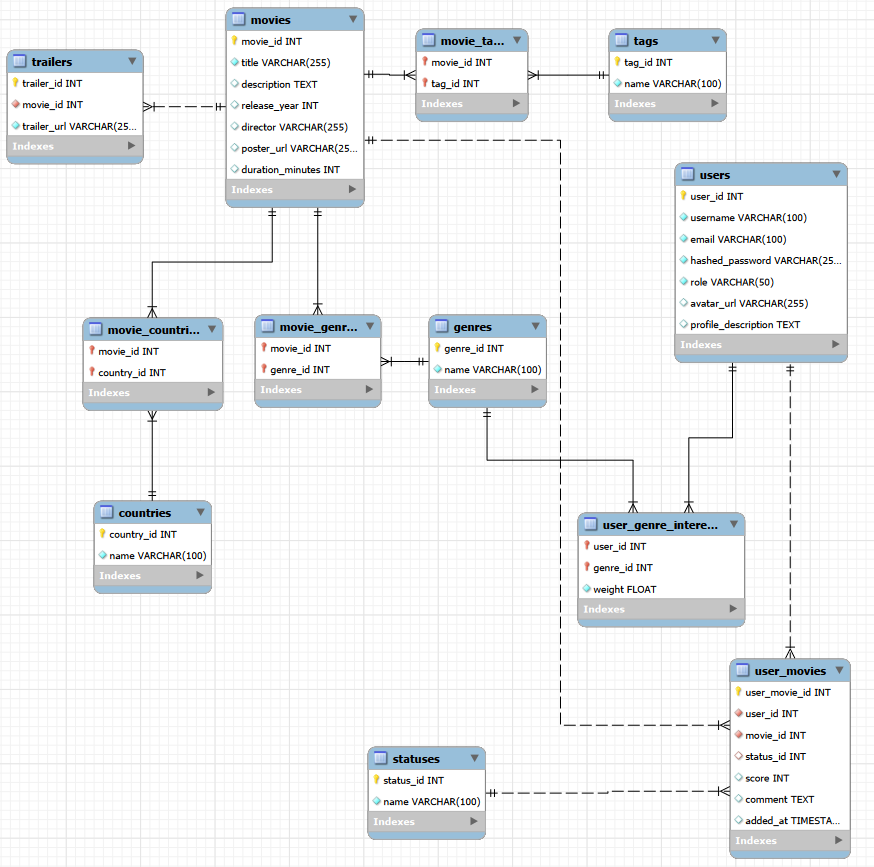
('Магия'),

('Оккультизм'),

('Антиутопия'),

('Музыка'),

('Криминальная драма');

  
Рисунок 1. EER Диаграмма физического уровня.

## Веб-приложение.

1. **Серверная часть.**

Весь исходный код можно просмотреть в гитхаб репозитории, в качестве описания работы, будет продемонстрирована работа эндпоинтов системы.

**Аутентификация**

Метод URI Описание Тело запроса

POST /api/auth/register Регистрация нового пользователя { "username": "", "email": "", "password": "" }

POST /api/auth/login Вход по логину и паролю { "username": "", "password": "" }

POST /api/auth/change-password (auth) Смена пароля. Требует oldPassword и newPassword { "oldPassword": "", "newPassword": "" }

GET /api/auth/check-auth (auth) Проверить валидность токена, вернуть id/username/role —

POST /api/auth/logout (auth) Очистить cookie с токеном —

**Фильмы**

Метод URI Описание Тело/параметры

GET /api/movies Список фильмов. Поддерживает query title, genreIds, tagIds, countryIds Query

GET /api/movies/{id} Получить фильм по id —

GET /api/movies/by-title/{title} Найти фильм по точному названию —

GET /api/movies/random Вернуть случайный фильм —

POST /api/movies (admin) Создать фильм MovieCreateUpdateDto

PUT /api/movies/{id} (admin) Обновить фильм MovieCreateUpdateDto

DELETE /api/movies/{id} (admin) Удалить фильм —

MovieCreateUpdateDto содержит поля: title, description, releaseYear, director, posterUrl, durationMinutes, genreIds, tagIds, countryIds, trailerUrls.

**Рекомендации**

Метод URI Описание

GET /api/recommendations/movies (auth) Топ рекомендаций пользователя по жанровым весам. Исключает фильмы из личного списка. Если предпочтений нет — возвращает случайный фильм.

**Жанровые предпочтения**

Метод URI Описание

GET /api/user/genre-interests (auth) Получить сохранённые веса жанров

PUT /api/user/genre-interests (auth) Обновить список предпочтений. Тело: { "interests": [ { "genreId": 1, "weight": 0.5 }, ... ] }

Веса находятся в диапазоне (0; 1]. Пустой список очищает предпочтения.

**Личный список и оценки**

Метод URI Описание

POST /api/user/movies (auth) Добавить/обновить фильм в личном списке. Тело: { "movieId": 1, "statusId": 2, "comment": "..." }

POST /api/user/movies/rate (auth) Выставить оценку (1–10). Тело: { "movieId": 1, "score": 8, "comment": "..." }. Фильм автоматически получает статус «Оценен».

DELETE /api/user/movies/{movieId} (auth) Удалить фильм из списка целиком

Таблица statuses должна содержать варианты из seed-скрипта: «Буду смотреть», «Просмотрено», «Брошено», «Не буду смотреть», «Оценен».

**Профиль пользователя**

Метод URI Описание

GET /api/user/profile (auth) Получить профиль (username, email, роль, описание, avatarUrl)

PUT /api/user/profile/description (auth) Обновить описание. Тело: { "profileDescription": "..." }

POST /api/user/profile/avatar (auth) Загрузить аватар (multipart/form-data, поле avatar). Допустимые форматы: jpg/jpeg/png/webp, размер ≤ 4 МБ.

Файлы сохраняются в wwwroot/avatars, доступны по URL /avatars/{file}.

**Справочники**

Для жанров, тегов и стран доступны одинаковые REST-операции.

Контроллер База URI Публичные методы Админ-методы

GenresController /api/genres GET /, GET /{id} POST /, PUT /{id}, DELETE /{id}

TagsController /api/tags GET /, GET /{id} POST /, PUT /{id}, DELETE /{id}

CountriesController /api/countries GET /, GET /{id} POST /, PUT /{id}, DELETE /{id}

Все операции изменения защищены [Authorize(Roles = "admin")].

**Админ: пользователи**

Метод URI Описание

GET /api/users (auth) Список пользователей (доступен всем авторизованным)

GET /api/users/{id} (auth) Получить пользователя по id

GET /api/users/by-username/{username} (auth) Найти по username

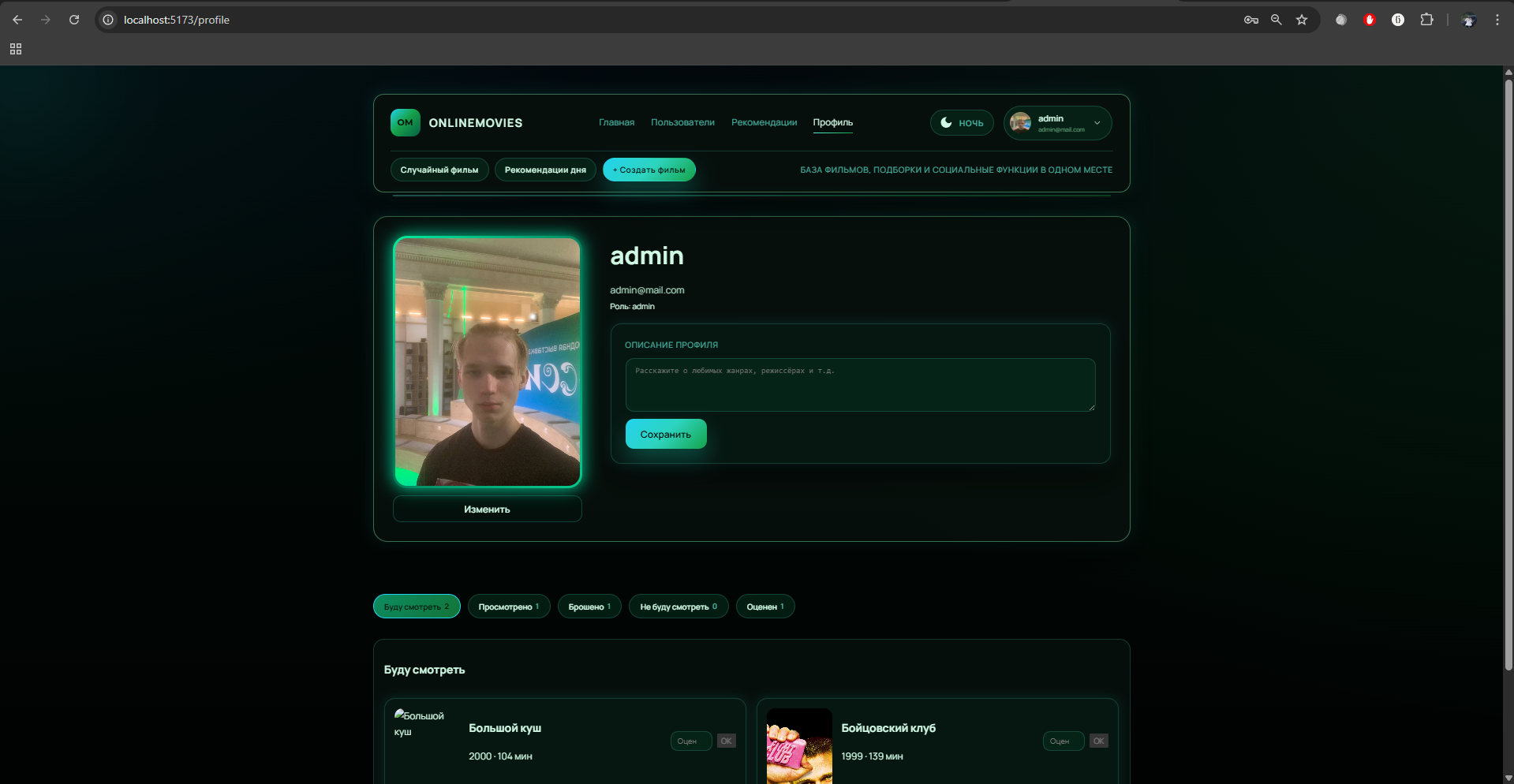
PUT /api/users/{id} (admin) Обновить пользователя: { "username": "", "email": "", "role": "admin/user" }

DELETE /api/users/{id} (admin) Удалить пользователя

1. **Клиентская часть.**

Весь исходный код можно просмотреть в гитхаб репозитории, использованный стек на фронтенд части был React + vite.

1. **Результат реализации.**

  
Рисунок 2. Профиль пользователя.

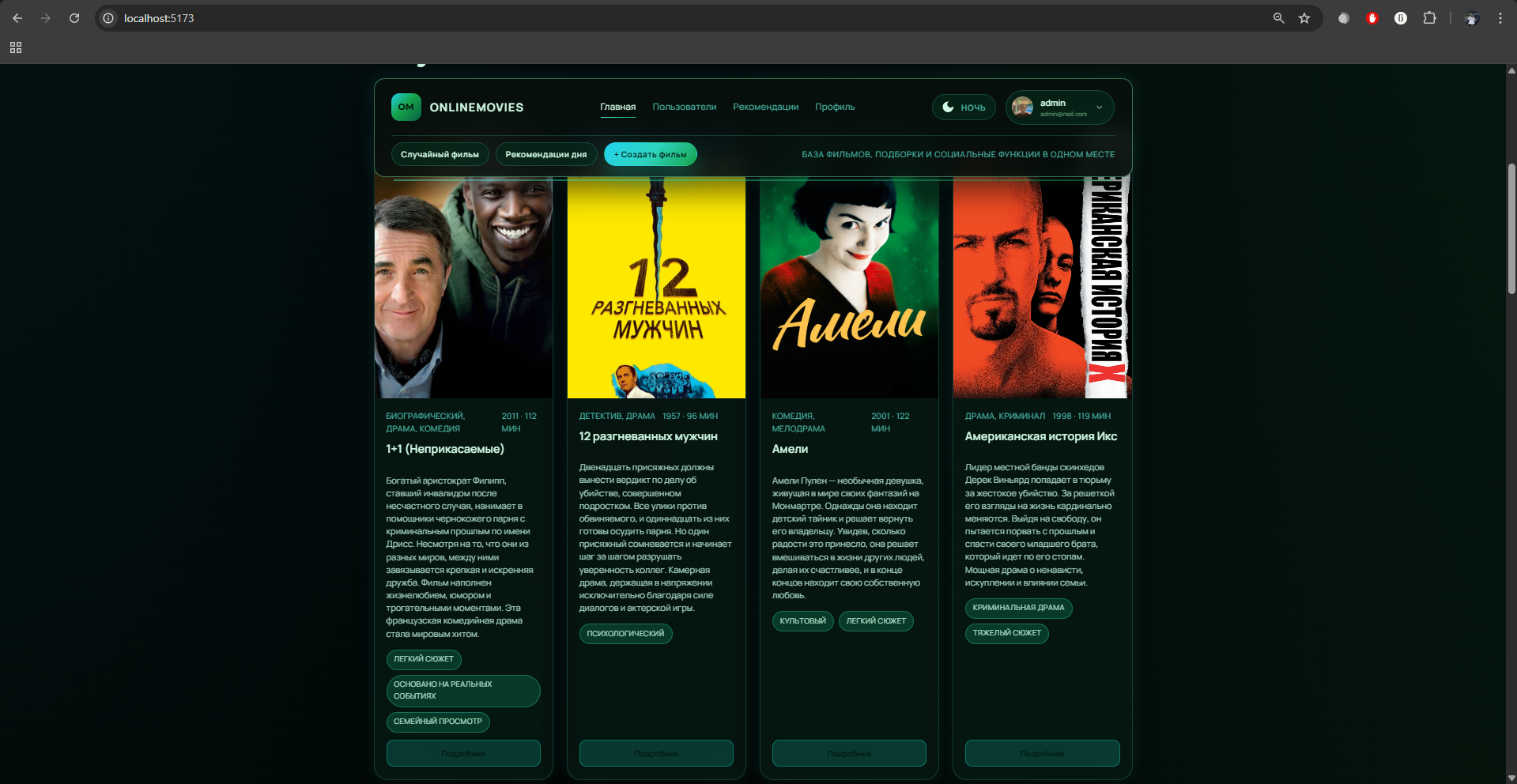


Рисунок 3. Каталог фильмов.

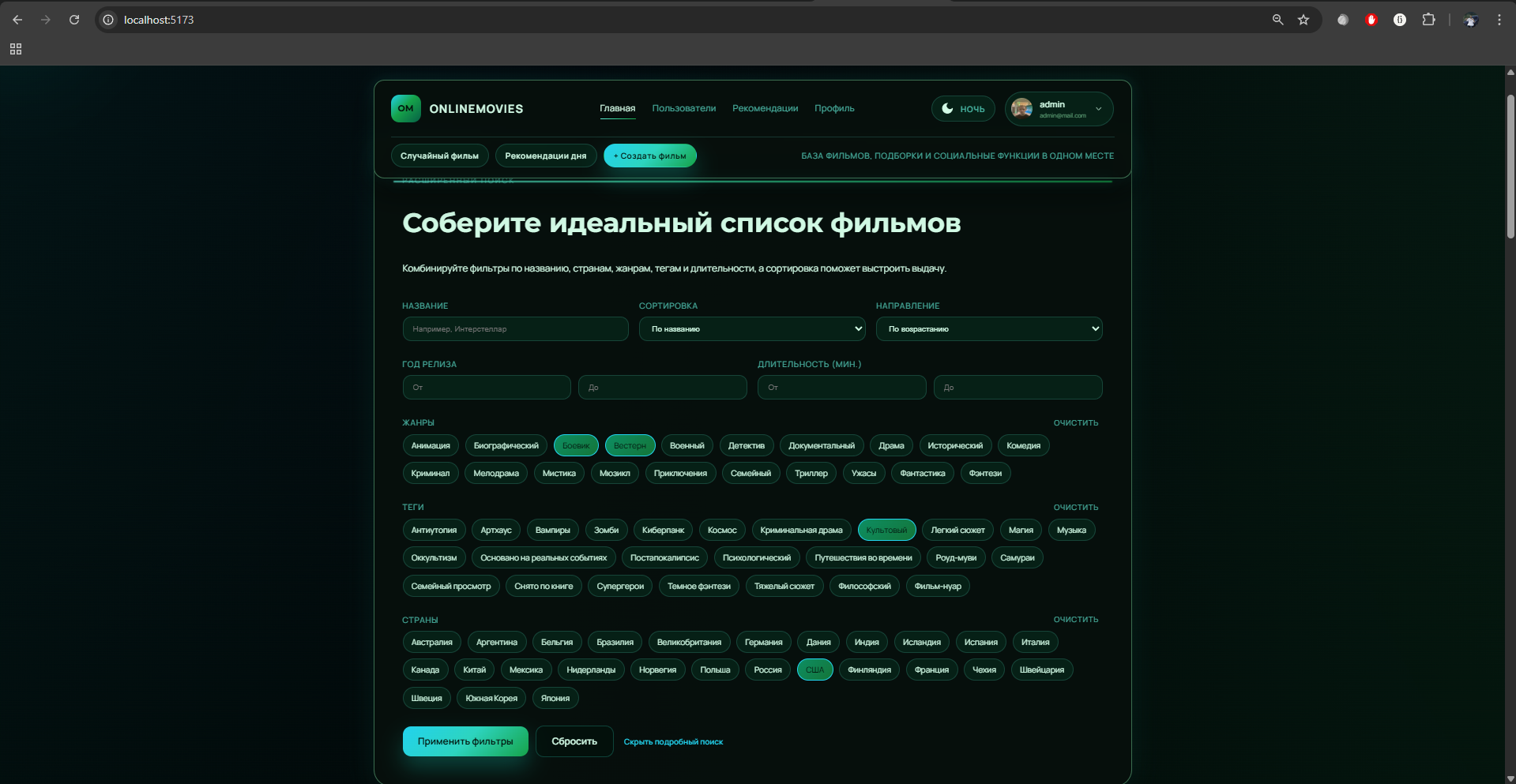


Рисунок 4. Расширенный поиск.

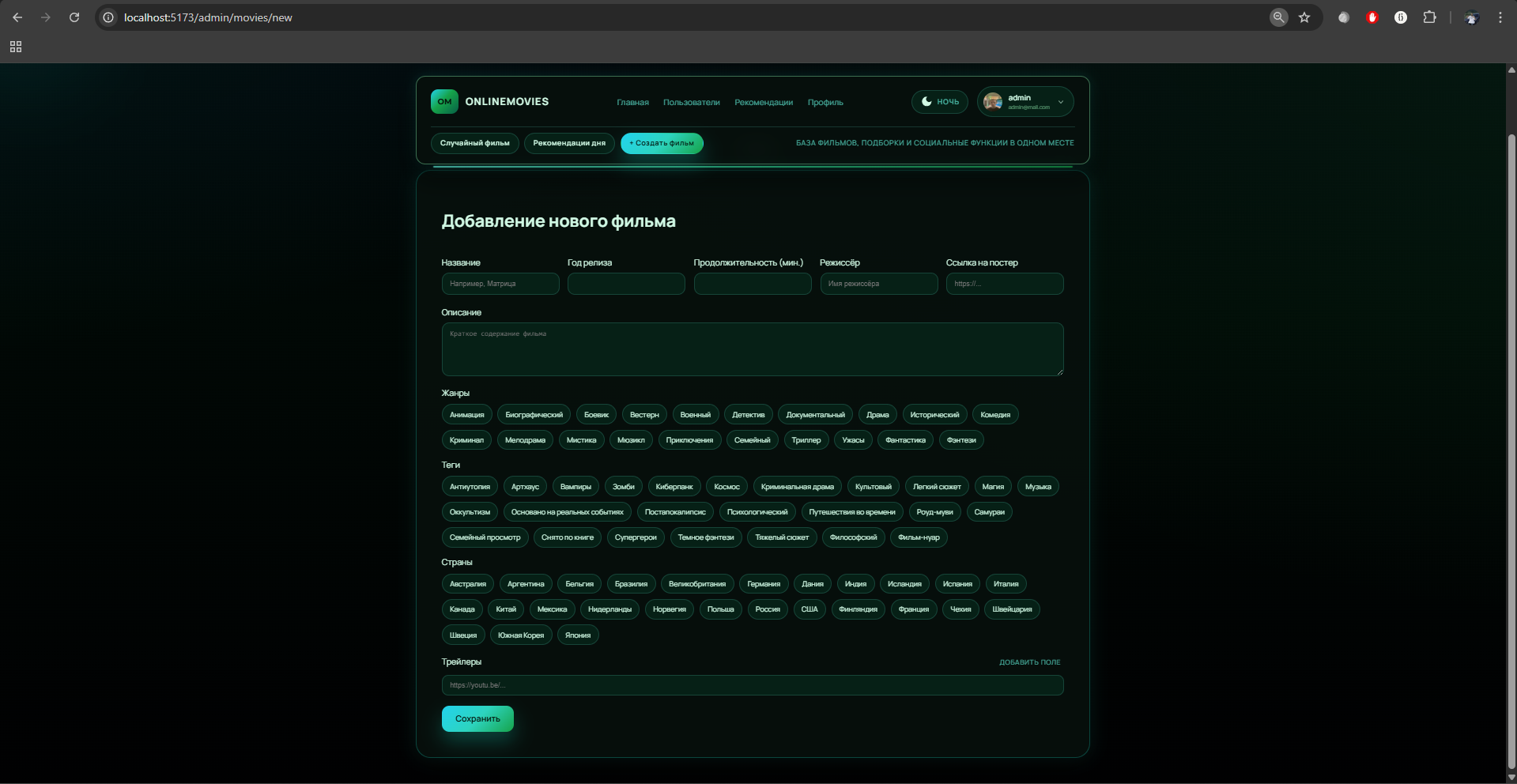


Рисунок 5. Создание фильма.

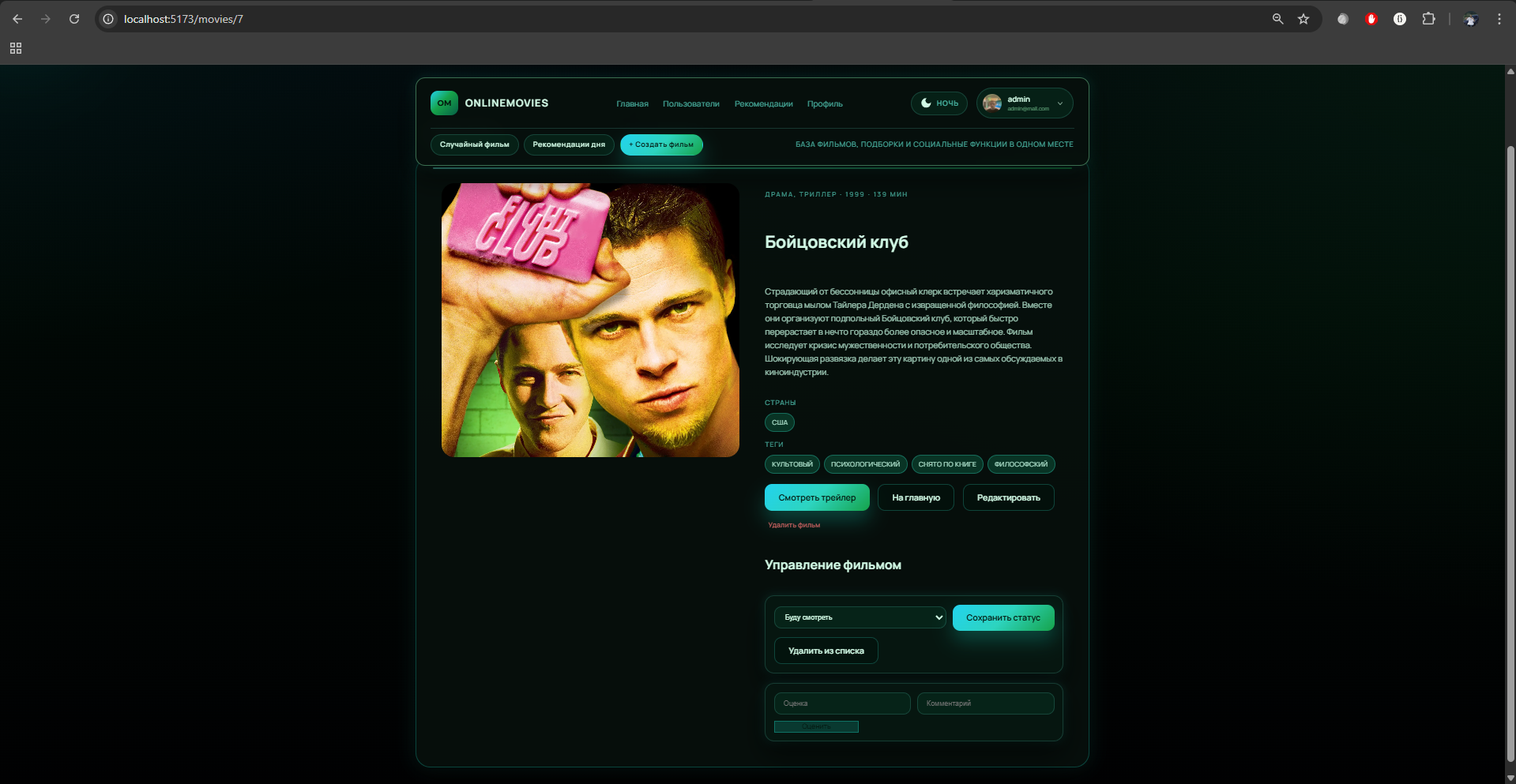


Рисунок 6. Страница фильма.

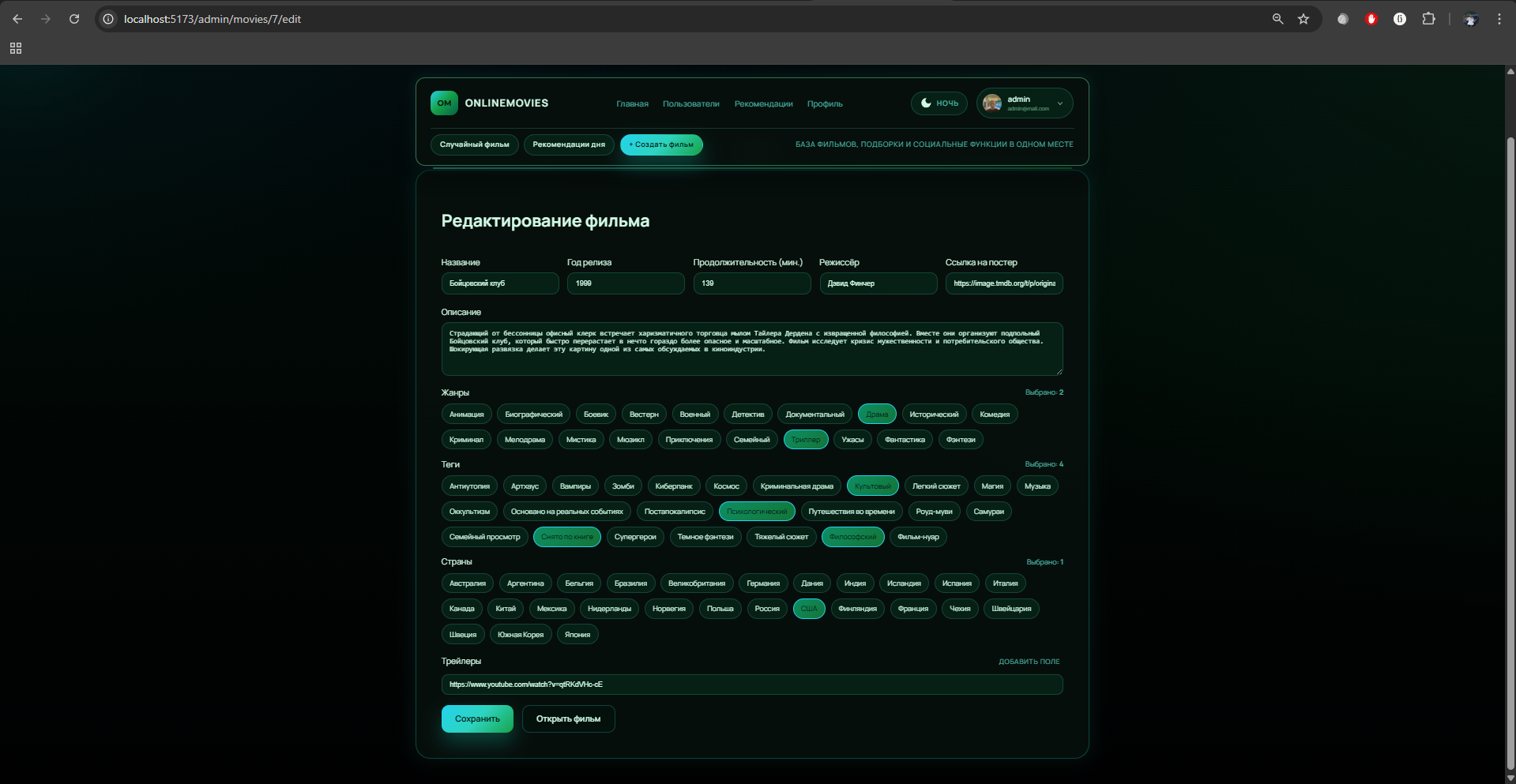


Рисунок 7. Страница редактирования фильма.

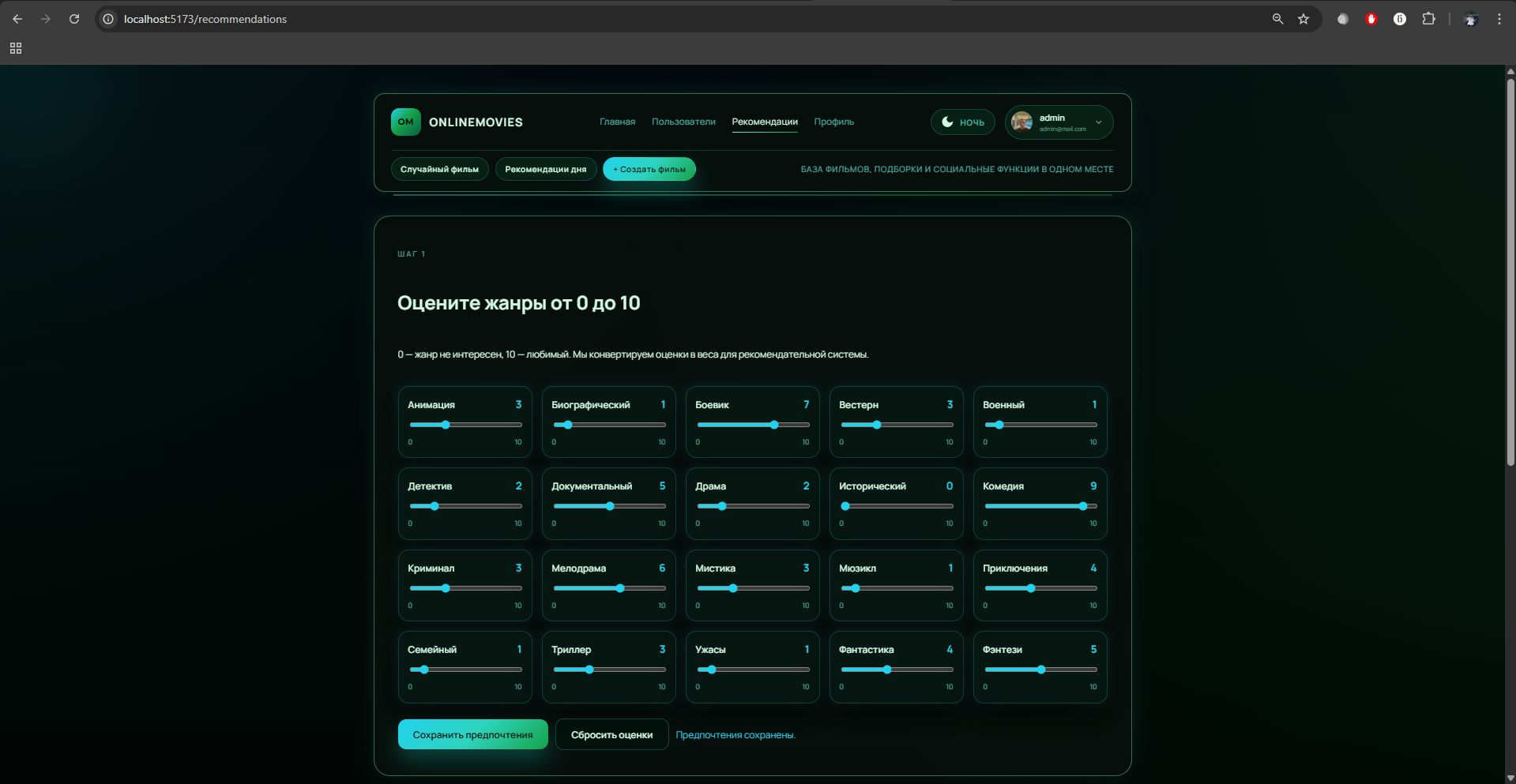


Рисунок 8. Страница предпочтений (Часть 1).

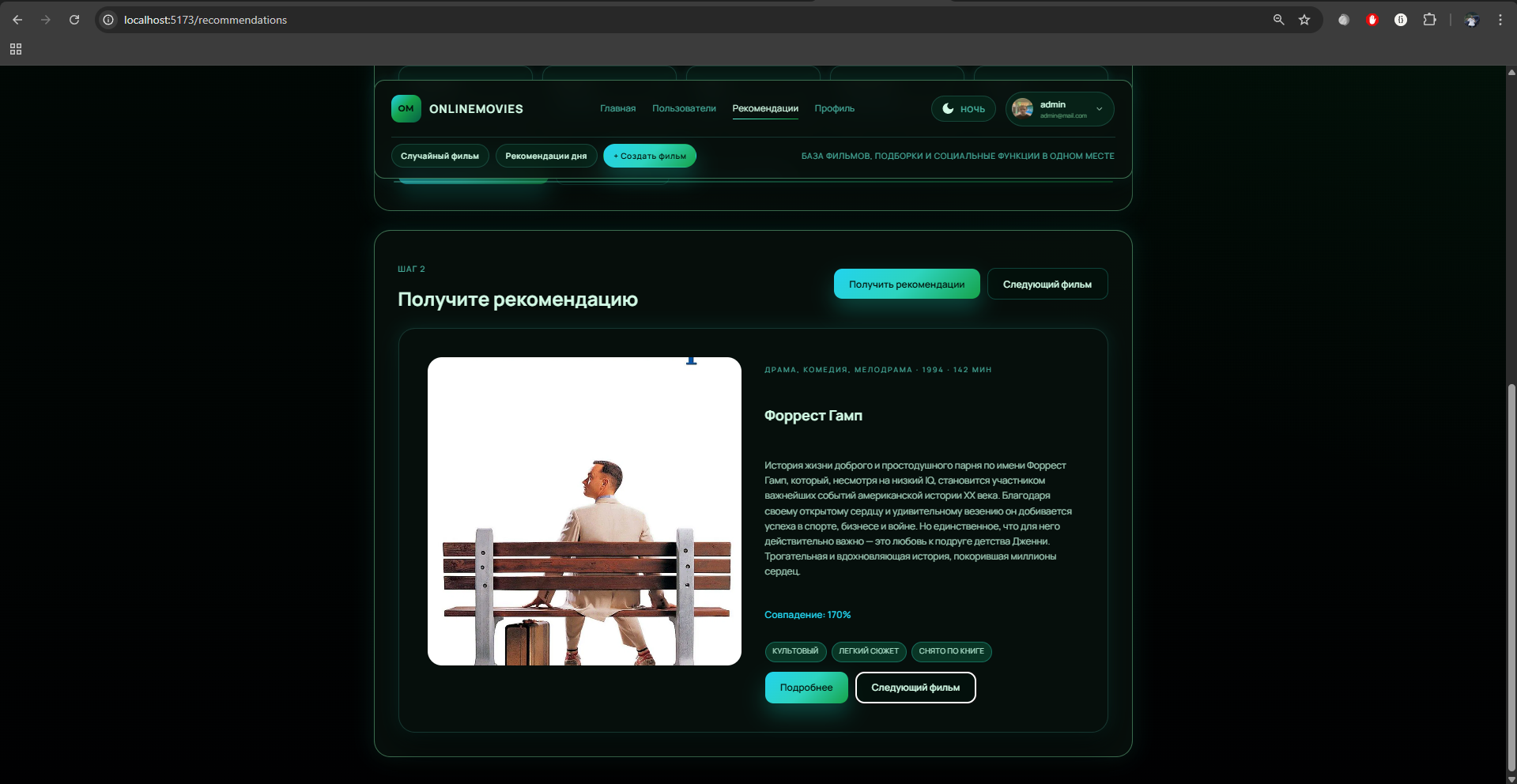


Рисунок 9. Страница предпочтений (Часть 2).

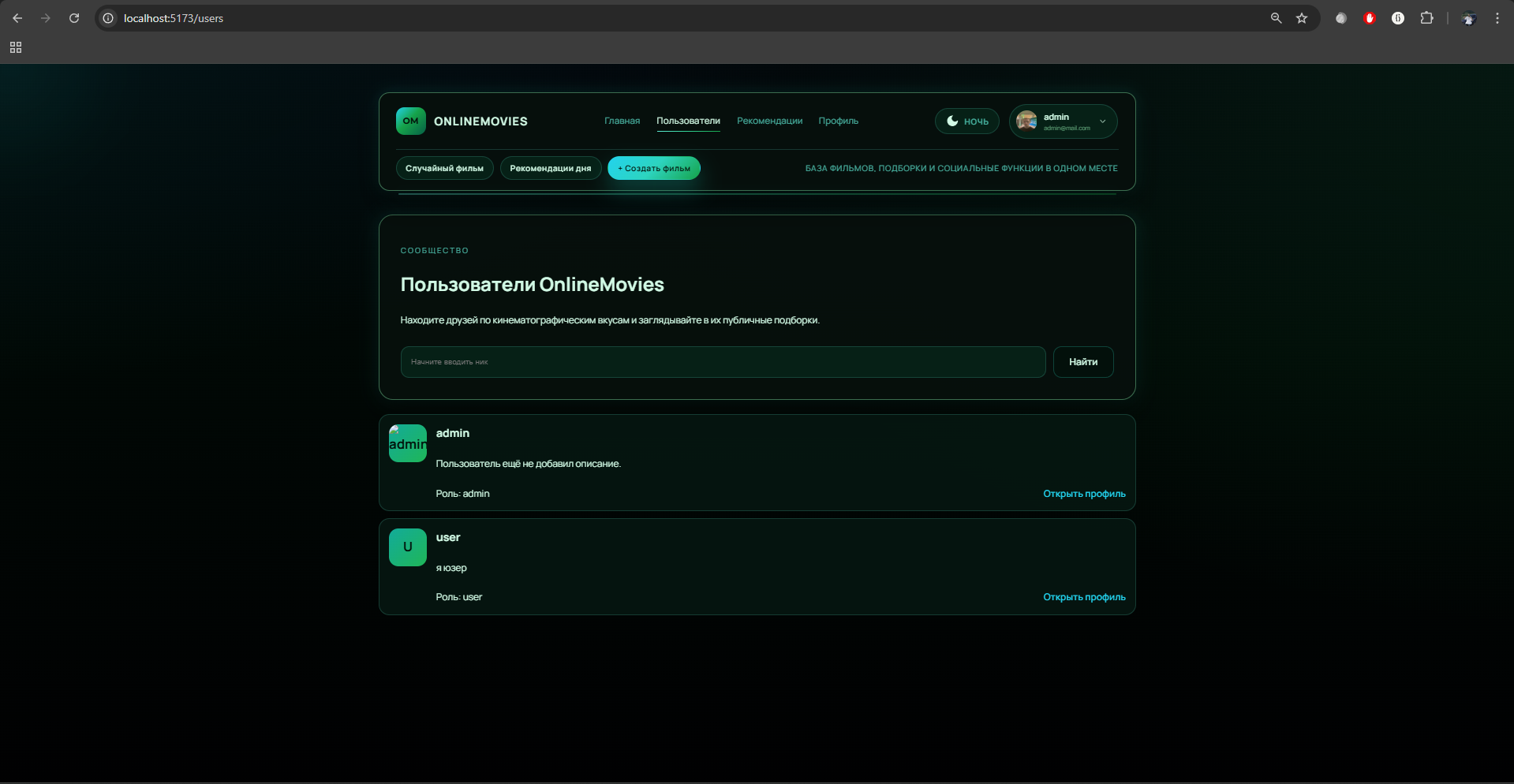


Рисунок 10. Поиск пользователей по сообществу.

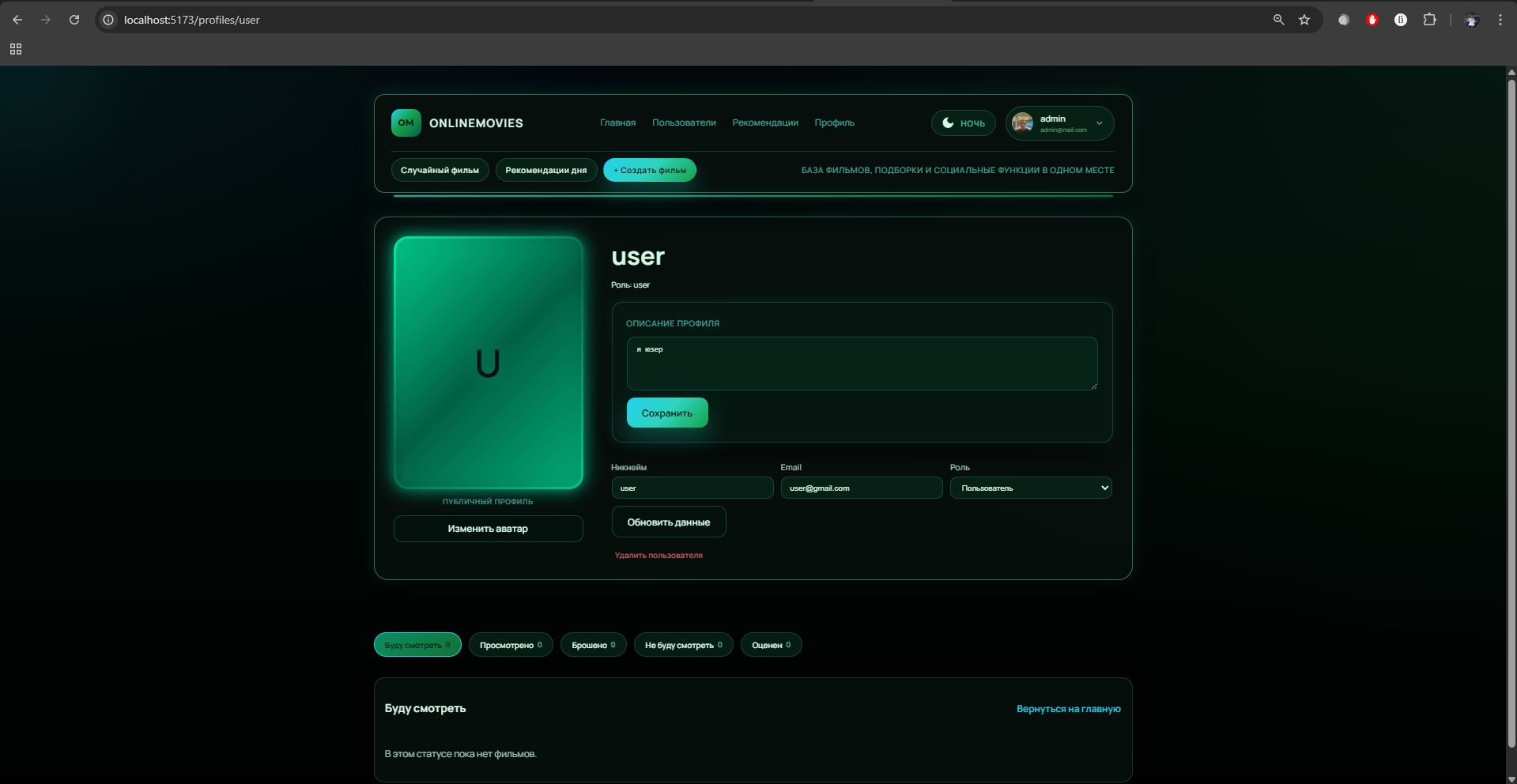


Рисунок 11. Профиль пользователя от лица администратора.

## Тестирование приложения

**Unit-тестирование**

Для модульного тестирования используется технологический стек xUnit, Entity Framework Core InMemory и Moq.

Unit-тесты покрывают следующие сценарии:

* Успешная оценка фильма проверяет корректную работу механизма оценки фильма, когда фильм находится в статусе Просмотрено.
* Валидация бизнес-правил проверяет, что нельзя оценить фильм, который не был просмотрен.
* Обработка граничных случаев проверяет поведение системы при попытке оценить фильм, отсутствующий в списке пользователя.
* Безопасность проверяет корректную обработку случая, когда пользователь был удален администратором, но его токен еще валиден.
* Изменение данных проверяет возможность изменения уже поставленной оценки.

Листинг кода:

namespace OnlineMovies.Tests;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using OnlineMovies.Controllers;

using OnlineMovies.Database;

using OnlineMovies.DTO;

using OnlineMovies.Models;

using OnlineMovies.Responses;

using System.Security.Claims;

using Xunit;

public class UserMoviesControllerTests

{

// Создает in-memory базу данных для тестирования.

private AppDbContext CreateDbContext()

{

var options = new DbContextOptionsBuilder<AppDbContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

return new AppDbContext(options);

}

// Создает контроллер с настроенным контекстом и пользователем.

private UserMoviesController CreateController(AppDbContext context, int userId, string username = "testuser", string role = "user")

{

var controller = new UserMoviesController(context)

{

ControllerContext = new ControllerContext

{

HttpContext = new DefaultHttpContext

{

User = new ClaimsPrincipal(new ClaimsIdentity(new[]

{

new Claim(ClaimTypes.NameIdentifier, userId.ToString()),

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, role)

}, "test"))

}

}

};

return controller;

}

// Тест: Нельзя оценить фильм, если он не находится в статусе "Просмотрено".

// Проверяет, что система корректно отклоняет попытку оценить фильм,

// который еще не был просмотрен пользователем.

[Fact]

public async Task RateMovie\_ShouldReject\_WhenMovieNotWatched()

{

// Arrange - подготовка данных

using var context = CreateDbContext();

// Создаем пользователя

var user = new User

{

UserId = 1,

Username = "testuser",

Email = "test@example.com",

HashedPassword = "hashed",

Role = "user"

};

context.Users.Add(user);

// Создаем фильм

var movie = new Movie

{

MovieId = 1,

Title = "Test Movie",

Description = "Test Description",

ReleaseYear = 2020

};

context.Movies.Add(movie);

// Создаем статусы

var watchedStatus = new Status { StatusId = 1, Name = "Просмотрено" };

var ratedStatus = new Status { StatusId = 2, Name = "Оценен" };

var plannedStatus = new Status { StatusId = 3, Name = "Буду смотреть" };

context.Statuses.AddRange(watchedStatus, ratedStatus, plannedStatus);

// Создаем запись о фильме со статусом "Буду смотреть" (не просмотрен)

var userMovie = new UserMovie

{

UserId = 1,

MovieId = 1,

StatusId = 3, // "Буду смотреть"

AddedAt = DateTime.UtcNow

};

context.UserMovies.Add(userMovie);

await context.SaveChangesAsync();

var controller = CreateController(context, 1);

var request = new UserMovieRateRequestDto

{

MovieId = 1,

Score = 8,

Comment = "Great movie"

};

// Act - выполнение действия

var result = await controller.RateMovie(request);

// Assert - проверка результата

var badRequestResult = Assert.IsType<BadRequestObjectResult>(result.Result);

var response = Assert.IsType<ApiResponse<UserMovieResponseDto>>(badRequestResult.Value);

Assert.Equal("Ошибка", response.Status);

Assert.Contains("Просмотрено", response.Message);

}

// Тест: Можно оценить фильм, если он находится в статусе "Просмотрено".

// Проверяет успешную оценку фильма после его просмотра.

[Fact]

public async Task RateMovie\_ShouldSucceed\_WhenMovieWatched()

{

// Arrange - подготовка данных

using var context = CreateDbContext();

// Создаем пользователя

var user = new User

{

UserId = 1,

Username = "testuser",

Email = "test@example.com",

HashedPassword = "hashed",

Role = "user"

};

context.Users.Add(user);

// Создаем фильм

var movie = new Movie

{

MovieId = 1,

Title = "Test Movie",

Description = "Test Description",

ReleaseYear = 2020

};

context.Movies.Add(movie);

// Создаем статусы

var watchedStatus = new Status { StatusId = 1, Name = "Просмотрено" };

var ratedStatus = new Status { StatusId = 2, Name = "Оценен" };

context.Statuses.AddRange(watchedStatus, ratedStatus);

// Создаем запись о фильме со статусом "Просмотрено"

var userMovie = new UserMovie

{

UserId = 1,

MovieId = 1,

StatusId = 1, // "Просмотрено"

AddedAt = DateTime.UtcNow

};

context.UserMovies.Add(userMovie);

await context.SaveChangesAsync();

var controller = CreateController(context, 1);

var request = new UserMovieRateRequestDto

{

MovieId = 1,

Score = 8,

Comment = "Great movie"

};

// Act - выполнение действия

var result = await controller.RateMovie(request);

// Assert - проверка результата

var okResult = Assert.IsType<OkObjectResult>(result.Result);

var response = Assert.IsType<ApiResponse<UserMovieResponseDto>>(okResult.Value);

Assert.Equal("Успешно", response.Status);

Assert.NotNull(response.Data);

Assert.Equal(8, response.Data.Score);

Assert.Equal("Оценен", response.Data.StatusName);

// Проверяем, что статус изменился на "Оценен" в базе данных

var updatedUserMovie = await context.UserMovies

.Include(um => um.Status)

.FirstOrDefaultAsync(um => um.UserId == 1 && um.MovieId == 1);

Assert.NotNull(updatedUserMovie);

Assert.Equal(2, updatedUserMovie.StatusId); // "Оценен"

Assert.Equal(8, updatedUserMovie.Score);

}

// Тест: Нельзя оценить фильм, если запись о фильме отсутствует.

// Проверяет, что система корректно отклоняет попытку оценить фильм,

// который вообще не добавлен в список пользователя.

[Fact]

public async Task RateMovie\_ShouldReject\_WhenMovieNotInList()

{

// Arrange - подготовка данных

using var context = CreateDbContext();

// Создаем пользователя

var user = new User

{

UserId = 1,

Username = "testuser",

Email = "test@example.com",

HashedPassword = "hashed",

Role = "user"

};

context.Users.Add(user);

// Создаем фильм

var movie = new Movie

{

MovieId = 1,

Title = "Test Movie",

Description = "Test Description",

ReleaseYear = 2020

};

context.Movies.Add(movie);

// Создаем статусы

var watchedStatus = new Status { StatusId = 1, Name = "Просмотрено" };

var ratedStatus = new Status { StatusId = 2, Name = "Оценен" };

context.Statuses.AddRange(watchedStatus, ratedStatus);

await context.SaveChangesAsync();

var controller = CreateController(context, 1);

var request = new UserMovieRateRequestDto

{

MovieId = 1,

Score = 8,

Comment = "Great movie"

};

// Act - выполнение действия

var result = await controller.RateMovie(request);

// Assert - проверка результата

var badRequestResult = Assert.IsType<BadRequestObjectResult>(result.Result);

var response = Assert.IsType<ApiResponse<UserMovieResponseDto>>(badRequestResult.Value);

Assert.Equal("Ошибка", response.Status);

Assert.Contains("Просмотрено", response.Message);

}

// Тест: Проверка существования пользователя при работе с токеном.

// Проверяет, что система корректно обрабатывает случай, когда пользователь

// был удален администратором, но его токен еще валиден.

[Fact]

public async Task GetUserMovies\_ShouldReject\_WhenUserDeleted()

{

// Arrange - подготовка данных

using var context = CreateDbContext();

// Пользователь НЕ создается в базе данных (симулируем удаление)

// Но токен все еще содержит его ID

var controller = CreateController(context, 1); // userId = 1, но пользователя нет в БД

// Act - выполнение действия

var result = await controller.GetUserMovies();

// Assert - проверка результата

var unauthorizedResult = Assert.IsType<UnauthorizedObjectResult>(result.Result);

var response = Assert.IsType<ApiResponse<IEnumerable<UserMovieResponseDto>>>(unauthorizedResult.Value);

Assert.Equal("Ошибка", response.Status);

Assert.Contains("не найден", response.Message);

}

// Тест: Можно изменить оценку фильма, если он уже в статусе "Оценен".

// Проверяет, что система разрешает изменение оценки для уже оцененных фильмов.

[Fact]

public async Task RateMovie\_ShouldSucceed\_WhenChangingRating()

{

// Arrange - подготовка данных

using var context = CreateDbContext();

// Создаем пользователя

var user = new User

{

UserId = 1,

Username = "testuser",

Email = "test@example.com",

HashedPassword = "hashed",

Role = "user"

};

context.Users.Add(user);

// Создаем фильм

var movie = new Movie

{

MovieId = 1,

Title = "Test Movie",

Description = "Test Description",

ReleaseYear = 2020

};

context.Movies.Add(movie);

// Создаем статусы

var watchedStatus = new Status { StatusId = 1, Name = "Просмотрено" };

var ratedStatus = new Status { StatusId = 2, Name = "Оценен" };

context.Statuses.AddRange(watchedStatus, ratedStatus);

// Создаем запись о фильме со статусом "Оценен" и оценкой 7

var userMovie = new UserMovie

{

UserId = 1,

MovieId = 1,

StatusId = 2, // "Оценен"

Score = 7,

Comment = "Старый комментарий",

AddedAt = DateTime.UtcNow

};

context.UserMovies.Add(userMovie);

await context.SaveChangesAsync();

var controller = CreateController(context, 1);

var request = new UserMovieRateRequestDto

{

MovieId = 1,

Score = 9, // Новая оценка

Comment = "Новый комментарий"

};

// Act - выполнение действия

var result = await controller.RateMovie(request);

// Assert - проверка результата

var okResult = Assert.IsType<OkObjectResult>(result.Result);

var response = Assert.IsType<ApiResponse<UserMovieResponseDto>>(okResult.Value);

Assert.Equal("Успешно", response.Status);

Assert.NotNull(response.Data);

Assert.Equal(9, response.Data.Score); // Новая оценка

Assert.Equal("Новый комментарий", response.Data.Comment);

Assert.Equal("Оценен", response.Data.StatusName);

// Проверяем, что оценка и комментарий обновились в базе данных

var updatedUserMovie = await context.UserMovies

.Include(um => um.Status)

.FirstOrDefaultAsync(um => um.UserId == 1 && um.MovieId == 1);

Assert.NotNull(updatedUserMovie);

Assert.Equal(2, updatedUserMovie.StatusId); // "Оценен"

Assert.Equal(9, updatedUserMovie.Score); // Новая оценка

Assert.Equal("Новый комментарий", updatedUserMovie.Comment);

}

// Тест: Проверка существования пользователя при оценке фильма.

// Проверяет, что система корректно обрабатывает случай удаленного пользователя

// при попытке оценить фильм.

[Fact]

public async Task RateMovie\_ShouldReject\_WhenUserDeleted()

{

// Arrange - подготовка данных

using var context = CreateDbContext();

// Пользователь НЕ создается в базе данных (симулируем удаление)

// Но токен все еще содержит его ID

// Создаем фильм и статусы для полноты теста

var movie = new Movie

{

MovieId = 1,

Title = "Test Movie",

Description = "Test Description",

ReleaseYear = 2020

};

context.Movies.Add(movie);

var watchedStatus = new Status { StatusId = 1, Name = "Просмотрено" };

var ratedStatus = new Status { StatusId = 2, Name = "Оценен" };

context.Statuses.AddRange(watchedStatus, ratedStatus);

await context.SaveChangesAsync();

var controller = CreateController(context, 1); // userId = 1, но пользователя нет в БД

var request = new UserMovieRateRequestDto

{

MovieId = 1,

Score = 8,

Comment = "Great movie"

};

// Act - выполнение действия

var result = await controller.RateMovie(request);

// Assert - проверка результата

var unauthorizedResult = Assert.IsType<UnauthorizedObjectResult>(result.Result);

var response = Assert.IsType<ApiResponse<UserMovieResponseDto>>(unauthorizedResult.Value);

Assert.Equal("Ошибка", response.Status);

Assert.Contains("не найден", response.Message);

}

}

**Функциональное тестирование с использованием Postman**

Postman используется в качестве основного инструмента функционального тестирования REST API приложения. Этот инструмент позволяет проводить комплексное тестирование API, имитируя реальные сценарии использования системы. Внутри него я протестировал все возможные сценарии использования приложения. В приложении Е будет оставлена ссылка на Postman коллекцию.

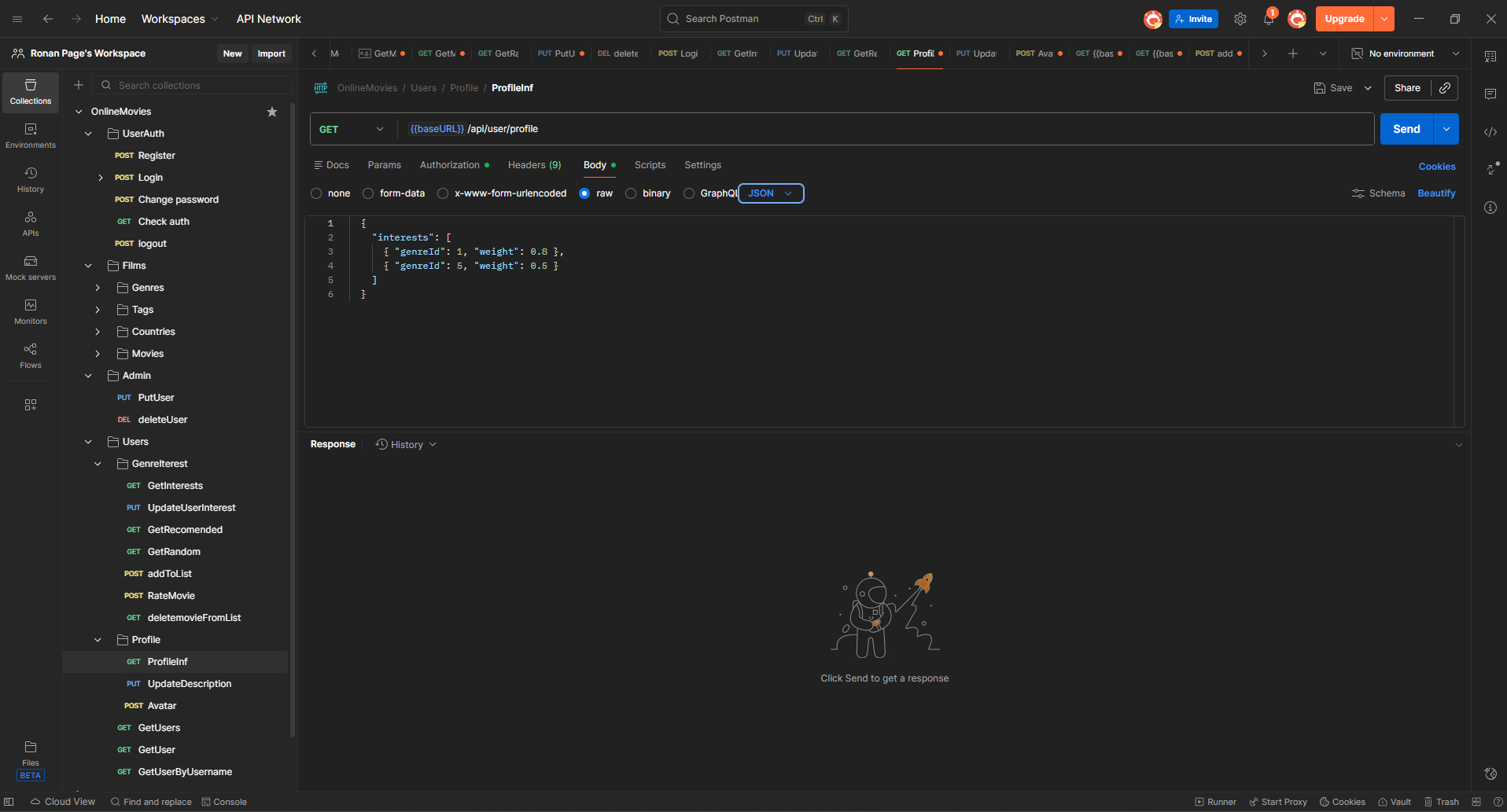


Рисунок 12. Функциональное тестирование через постман.

## Заключение

В ходе выполнения курсового проекта было проведено исследование предметной области «Онлайн-фильмотека» и разработано приложение для данной информационной системы, которое позволило объединить в одном месте обширный каталог фильмов с инструментами для его просмотра и поиска, функционал для ведения персональных списков просмотра, а также систему оценок и персональных рекомендаций. В процессе разработки были соблюдены все функциональные требования системы, а по итогам выполнения были достигнуты все поставленные цели.

## Список использованных источников

1. Microsoft. ASP.NET Core документация [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/aspnet/core/ (дата обращения: 09.12.2025).

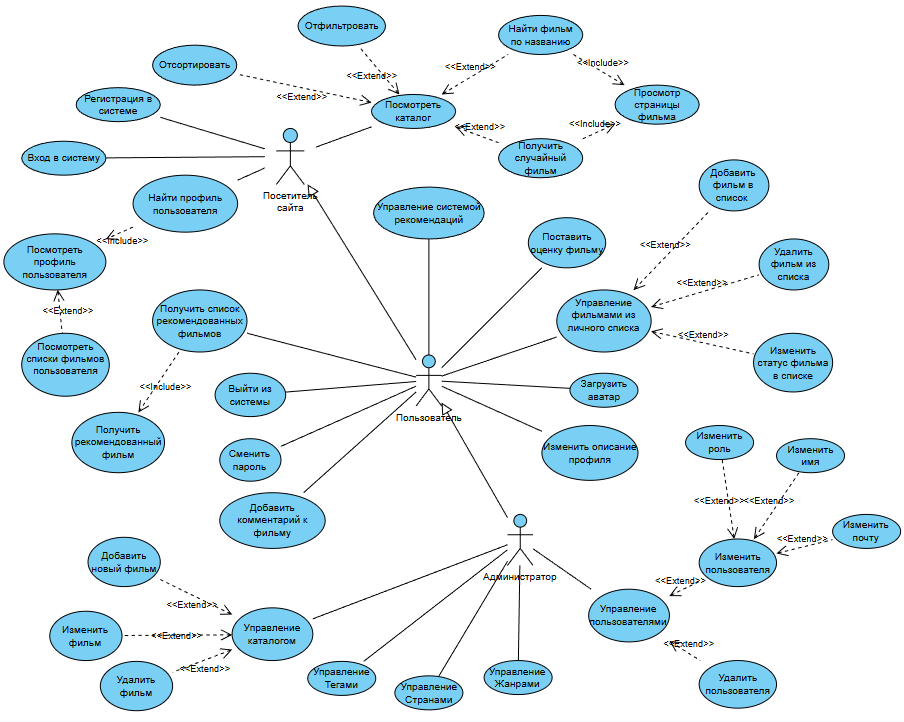
2. Microsoft. Entity Framework Core документация [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/ef/core/ (дата обращения: 08.12.2025).

3. xUnit.net. Документация по xUnit [Электронный ресурс]. – Режим доступа: https://xunit.net/ (дата обращения: 15.12.2025).

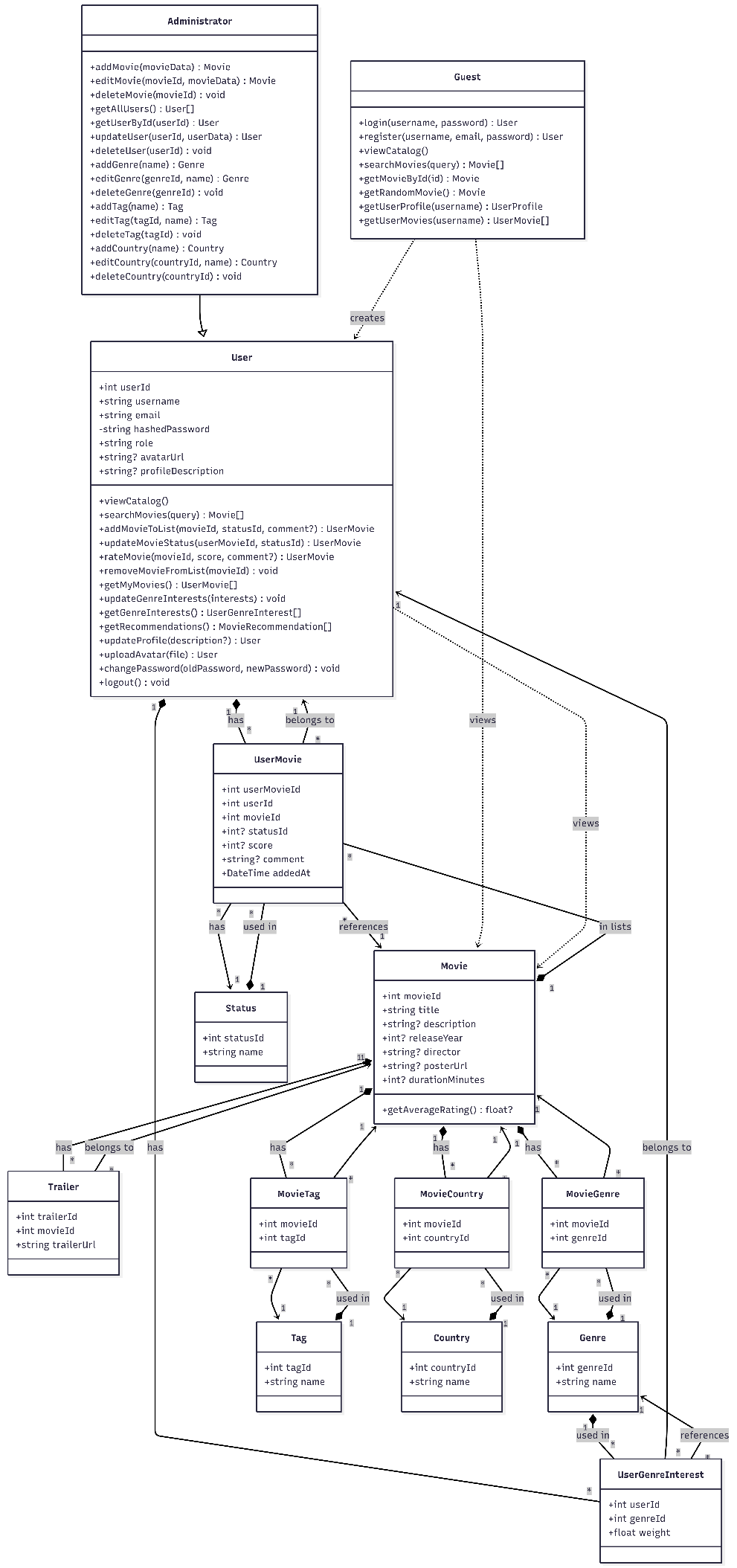
4. Postman Learning Center. API тестирование с Postman [Электронный ресурс]. – Режим доступа: https://learning.postman.com/docs/ (дата обращения: 08.12.2025).

5. JWT.io. Введение в JSON Web Tokens [Электронный ресурс]. – Режим доступа: https://jwt.io/introduction (дата обращения: 09.12.2025).

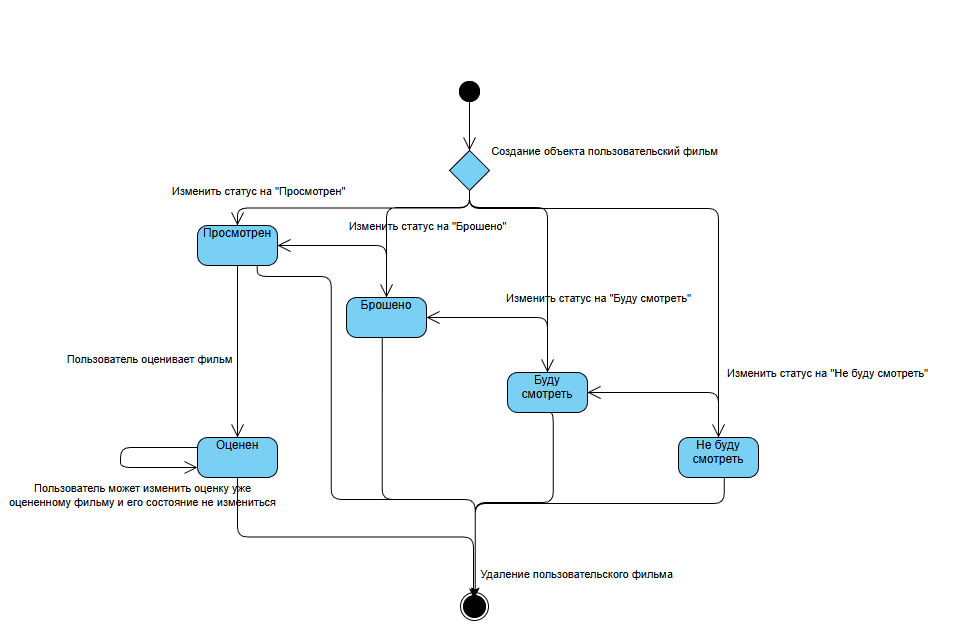
## Приложение А



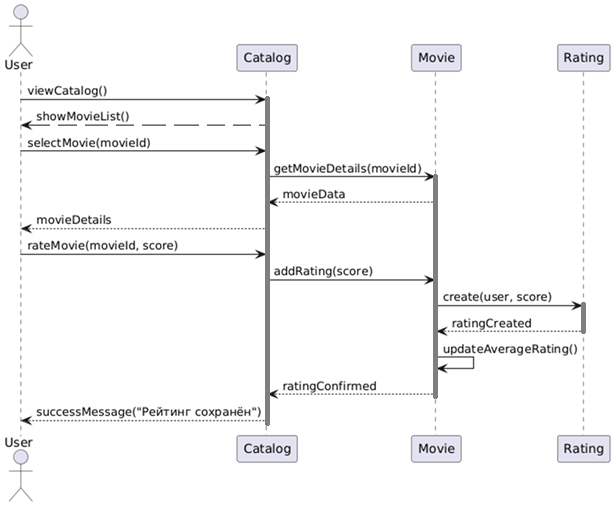
## Приложение Б



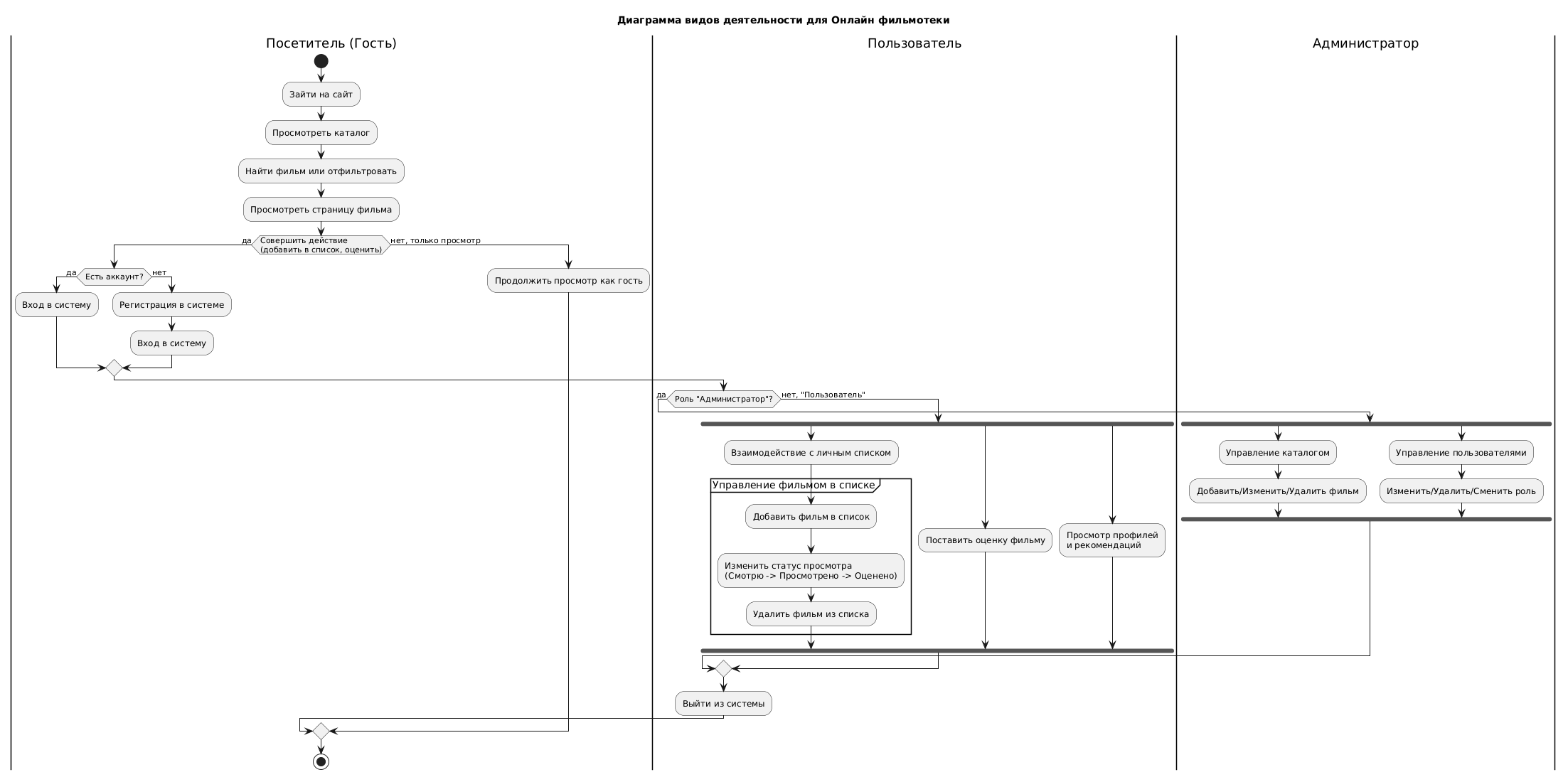
## Приложение В



## Приложение Г



## Приложение Д



## Приложение Е

<https://paromanprog-4560742.postman.co/workspace/Ronan-Page's-Workspace~b781b3f9-50fc-4d4f-8c6a-efe730bce75d/collection/48608033-9b31b84c-16d9-4120-85de-6422e5d7bd89?action=share&creator=48608033>