| **О Т Ч Е Т П О**  **З А Д А Н И Ю №2  на тему «Разработка программного продукта Movie-Picker»** | | |
| --- | --- | --- |
| по дисциплине «Технологии разработки качественного программного обеспечения» | | |
| Выполнили студенты гр. 5130904/20101 |  | Коба А. Ю.  Вдовина С.А. Месропян А.В.  Халиуллин А.Л. Фомин Н.А. |
|  | | |
| Руководитель |  | А.П. Маслаков  «1» октября 2025 г. |

## 

**Оглавление**

[Задание 2. High-Level Design (HLD) 3](#_Toc210301909)

[1. Макет дизайна интерфейса 3](#_Toc210301910)

[Модуль авторизации/аутентификации 3](#_Toc210301911)

[Модуль работы с фильмами 5](#_Toc210301912)

[Модуль списка “Хочу посмотреть” 7](#_Toc210301913)

[Модуль списка пользовательских рейтингов 7](#_Toc210301914)

[Модуль профиля пользователя 8](#_Toc210301915)

[2. Общая архитектура приложения 9](#_Toc210301916)

[Архитектура приложения 9](#_Toc210301917)

[Контекстная диаграмма (C4 - уровень 1) 10](#_Toc210301918)

[Контейнерная диаграмма (C4 - уровень 2) 11](#_Toc210301919)

[3. Стек технологий 12](#_Toc210301920)

[4. Схема базы данных 12](#_Toc210301921)

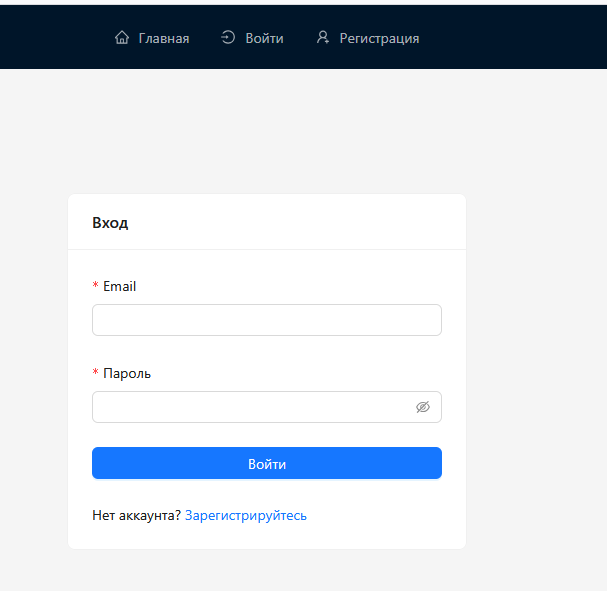
[5. Схема контрактов API 14](#_Toc210301922)

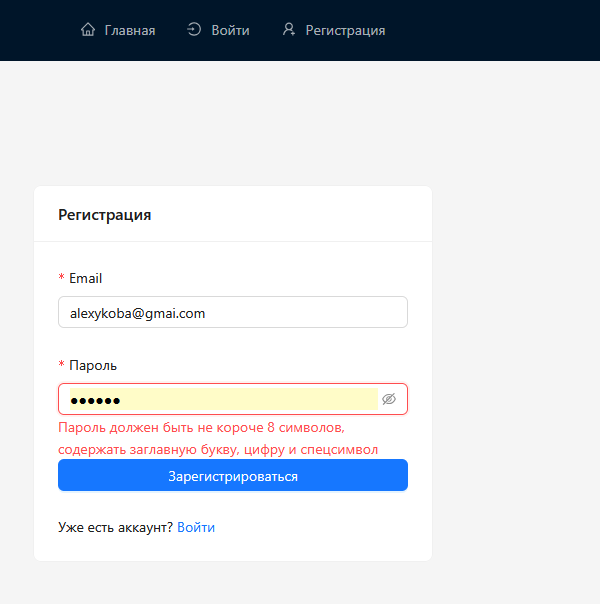
[6. Схема моделей данных 22](#_Toc210301923)

# Задание 2. High-Level Design (HLD)

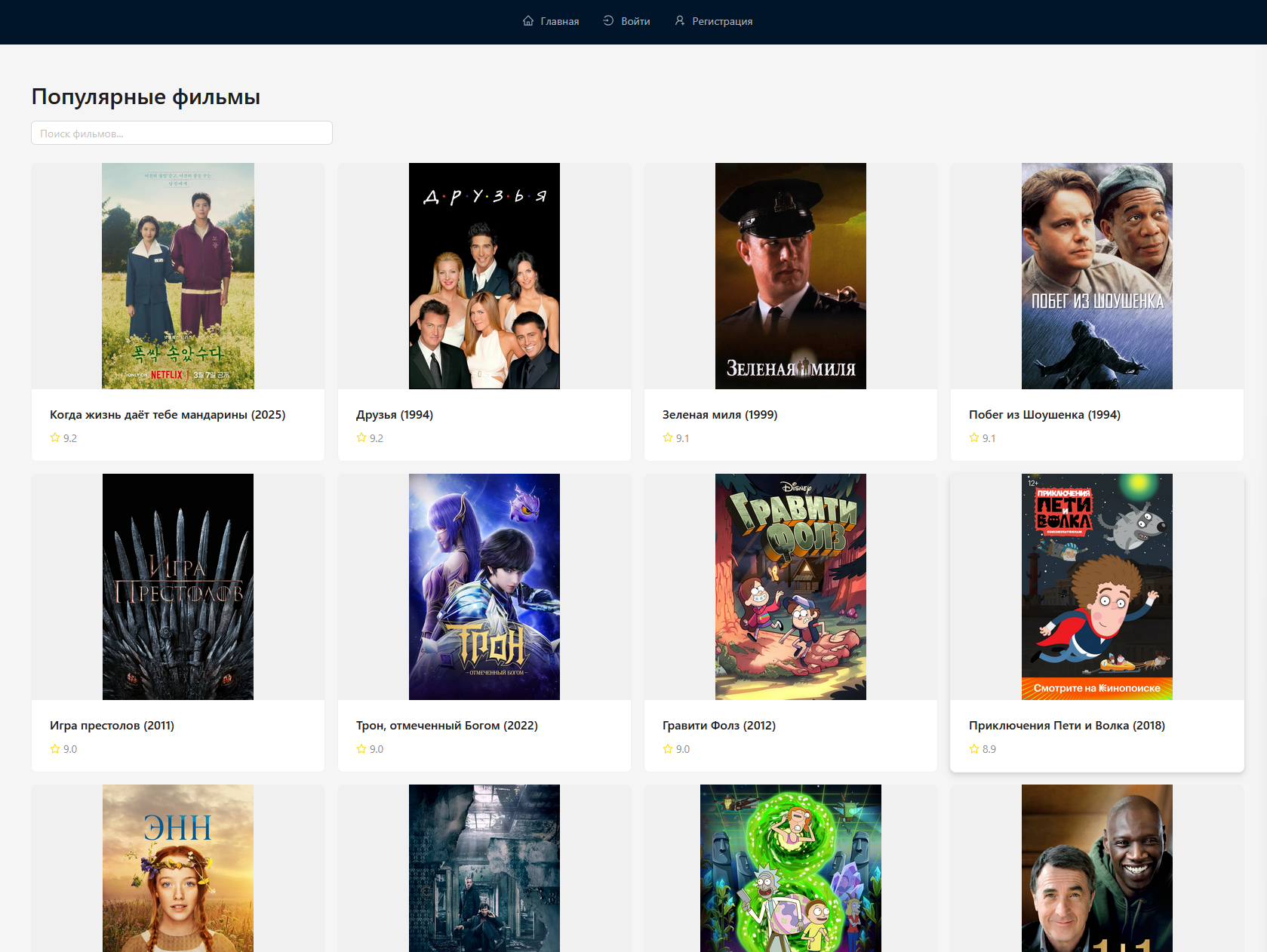
## Макет дизайна интерфейса

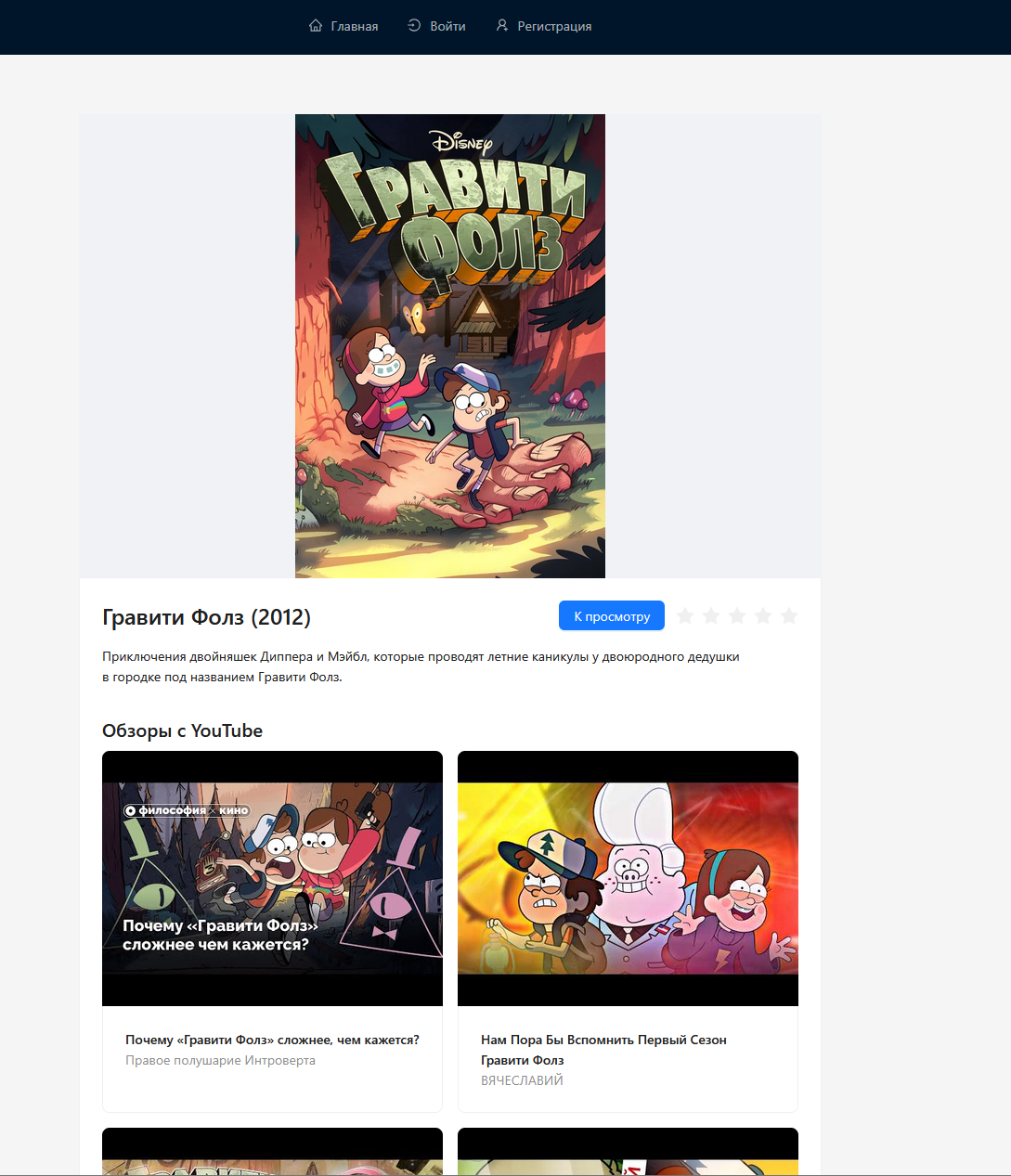
### Модуль авторизации/аутентификации



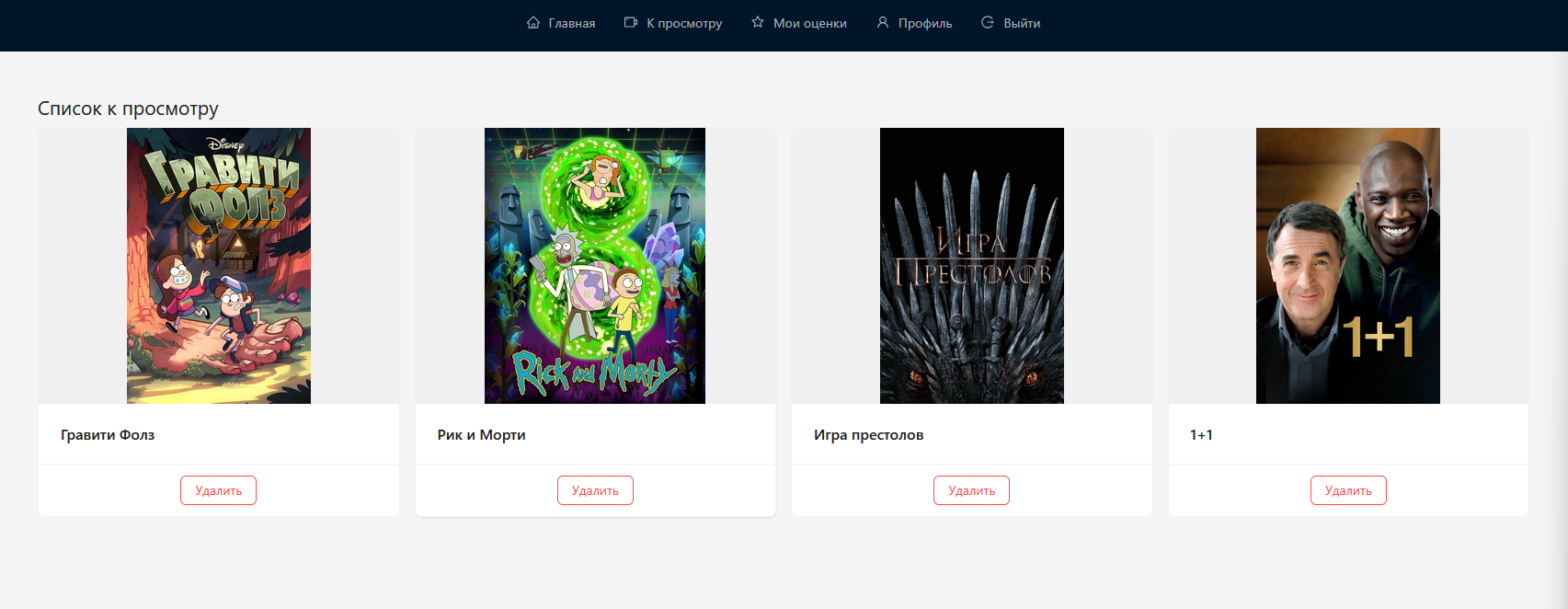


### Модуль работы с фильмами

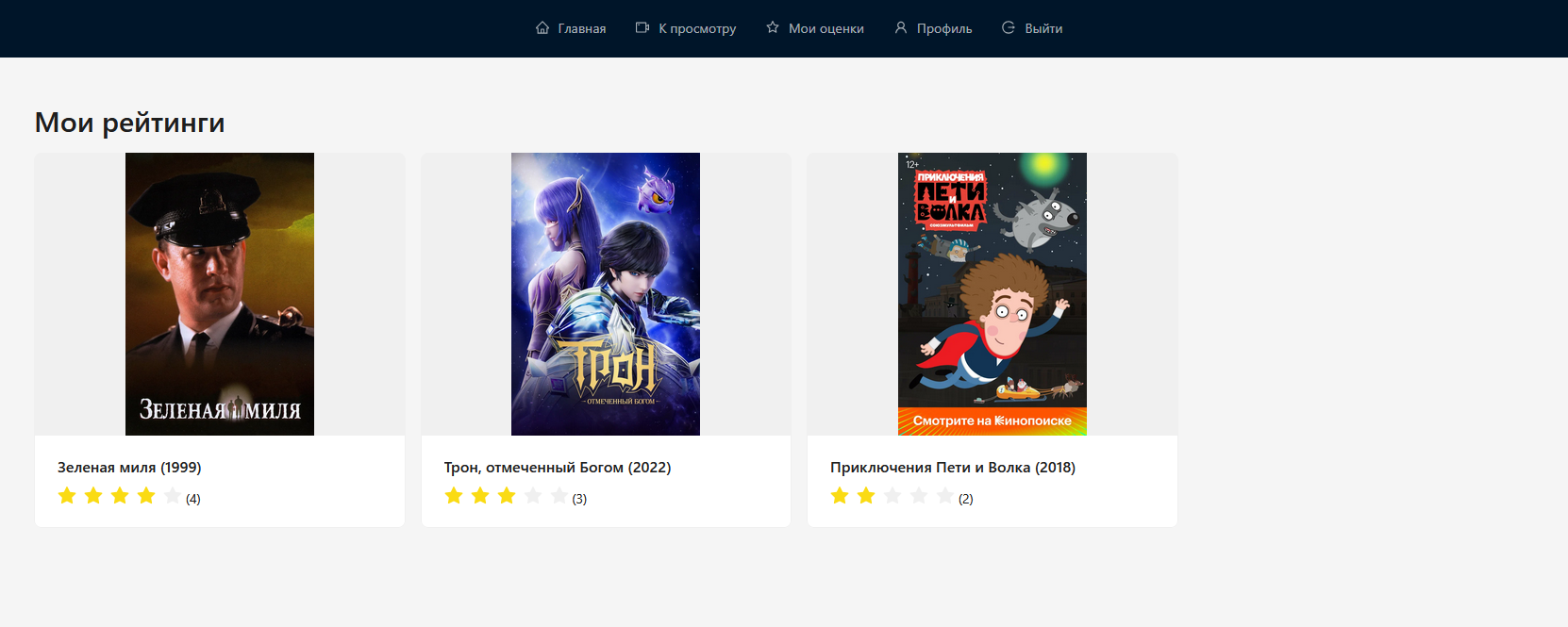




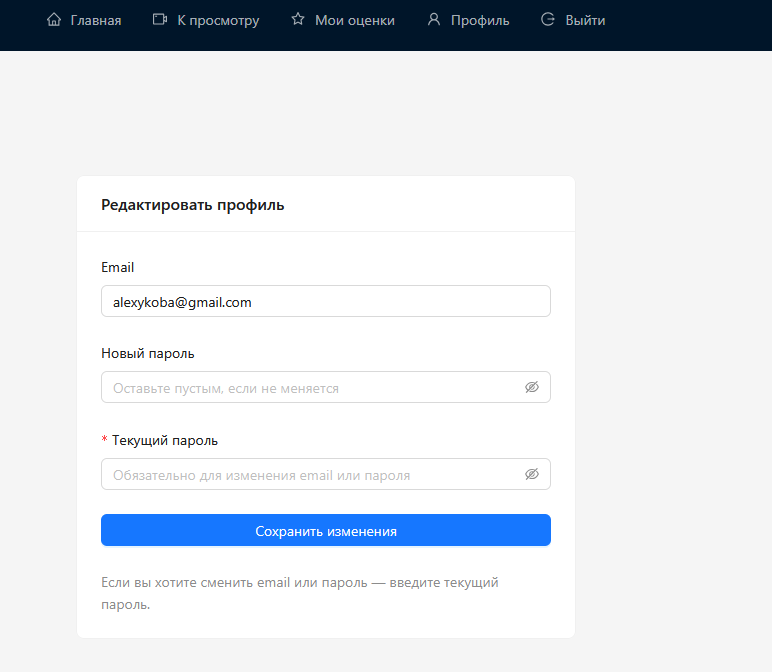
### Модуль списка “Хочу посмотреть”



### Модуль списка пользовательских рейтингов



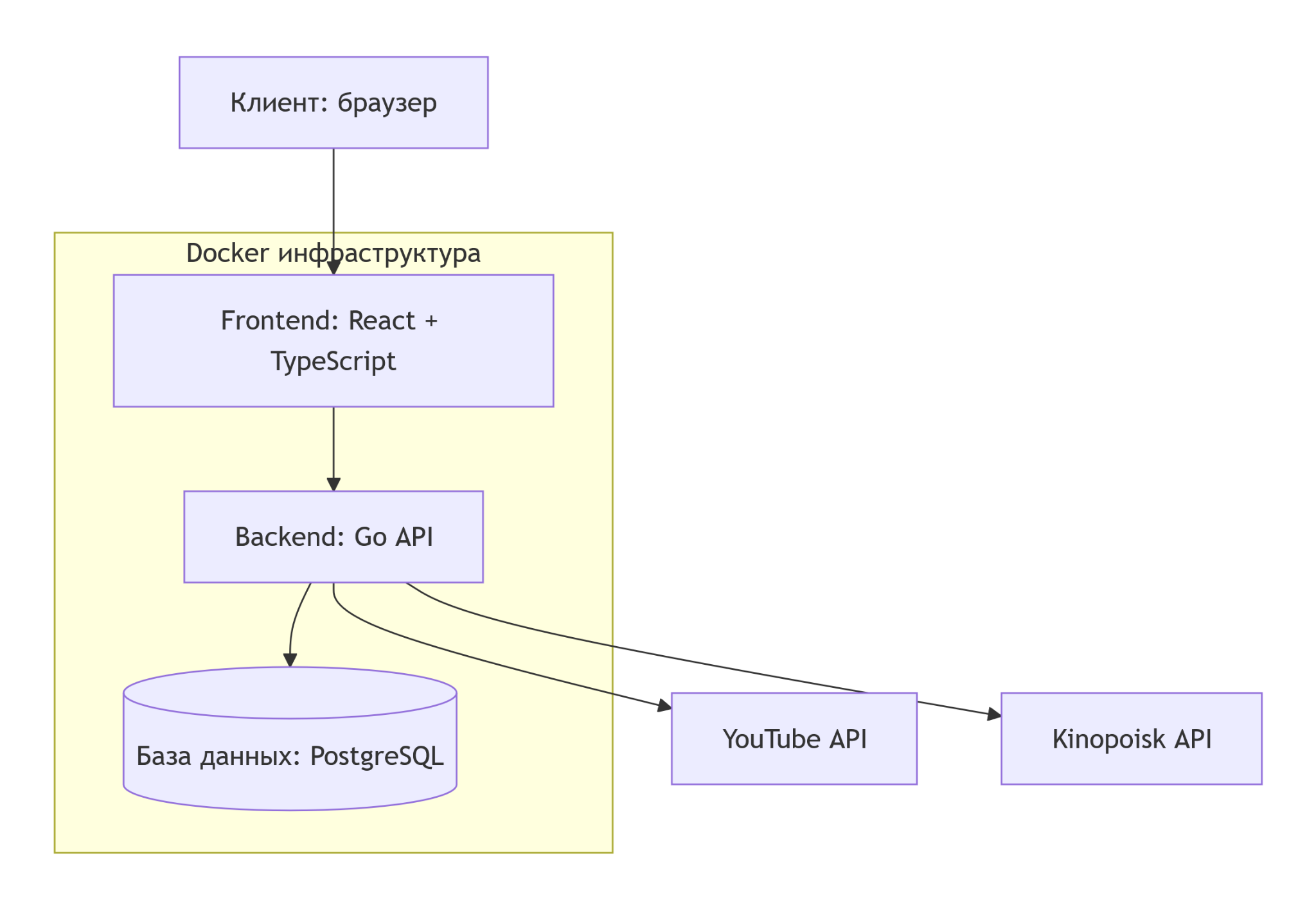
### Модуль профиля пользователя



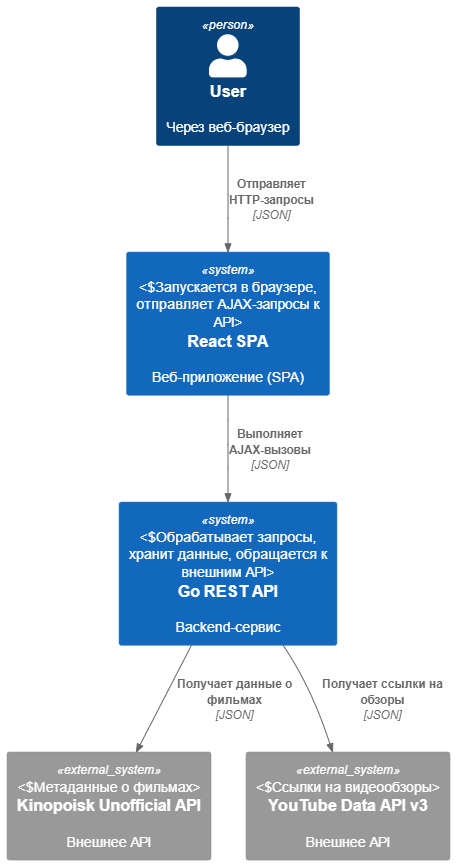
## 

## Общая архитектура приложения

### Архитектура приложения

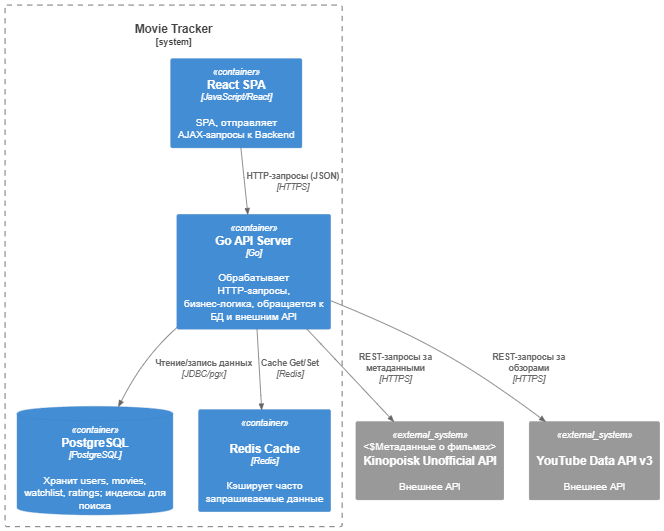
Приложение **Movie Picker** должно бытьпостроено по модели клиент–сервер и включает в себя два основных компонента: Frontend (клиентская часть) и Backend (серверная часть). Архитектура организована с учётом масштабируемости, безопасности и удобства разработки. Все компоненты разворачиваются в изолированных Docker-контейнерах.

### Контекстная диаграмма (C4 - уровень 1)



* **Компоненты:**
  1. **Пользователь (User)** – через веб‑браузер.
  2. **Веб‑приложение (React SPA)** – запускается в браузере, обращается к Backend.
  3. **Backend (Go REST API)** – обрабатывает запросы от Frontend, запрашивает Kinopoisk API и YouTube API.
  4. **Kinopoisk Unofficial API** – внешний сервис для получения метаданных о фильмах.
  5. **YouTube Data API v3** – внешний сервис для поиска видеообзоров.

### Контейнерная диаграмма (C4 - уровень 2)

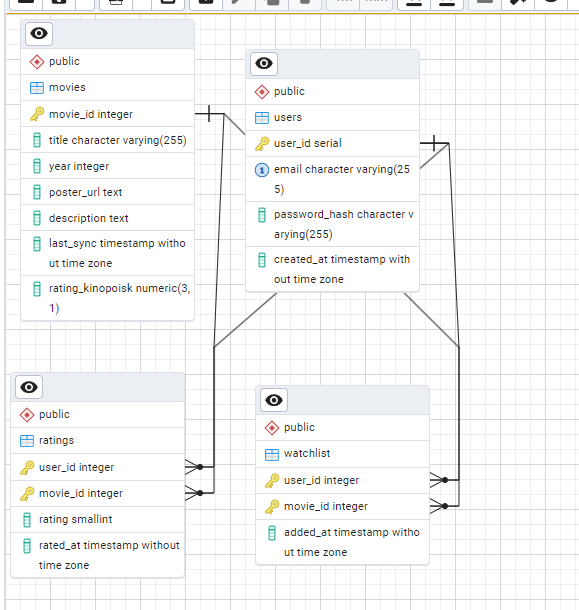


* **Контейнеры:**
  1. **React SPA (Frontend Container)**
     + Развёртывается как статический набор HTML/CSS/JS (в Docker Nginx или любой статический сервер).
     + Отправляет AJAX-запросы к API Server.
  2. **Go API Server (Backend Container)**
     + Обрабатывает HTTP-запросы, реализует бизнес‑логику.
     + Взаимодействует с PostgreSQL.
     + Вызывает внешний Kinopoisk API и YouTube API.
     + Выполняет аутентификацию/авторизацию (JWT).
  3. **PostgreSQL (Database Container)**
     + Хранит таблицы: users, movies, watchlist, ratings.
     + Индексы для ускорения поиска (по movie\_id, user\_id, title).
  4. **(Опционально) Redis/Cache (Cache Container)**
     + Если понадобится уменьшить задержки при частых запросах к одним и тем же фильмам, можно интегрировать Redis как кеш.

## 3. Стек технологий

* **Фронтенд** реализовывется на **React** с использованием **TypeScript**, библиотеки **Ant Design** и клиентского роутинга (**React Router**).
* **Бэкенд** реализовывается на **Go**, с использованием **Chi** в качестве маршрутизатора, **PostgreSQL** как СУБД, и **bcrypt** для безопасного хранения паролей.
* Реализована авторизация по **JWT**, взаимодействие с API, работа с YouTube-обзорами и обработка ошибок.

## 4. Схема базы данных

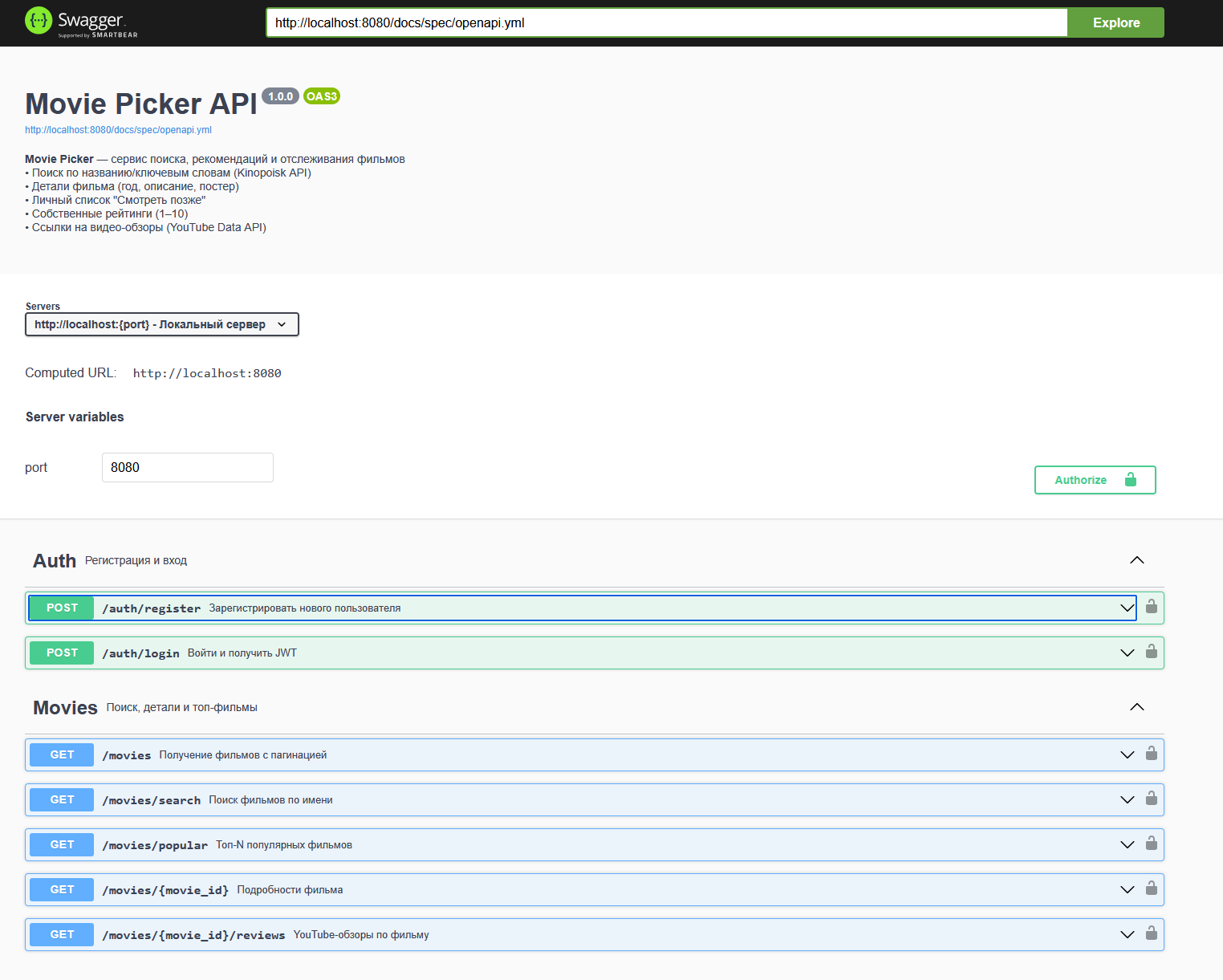


#### 

**Описание таблиц**

1. **users**
   * user\_id (SERIAL PK) — уникальный идентификатор пользователя.
   * email (VARCHAR, UNIQUE, NOT NULL) — адрес электронной почты.
   * password\_hash (VARCHAR, NOT NULL) — bcrypt-хэш пароля.
   * created\_at (TIMESTAMP, DEFAULT NOW()) — дата регистрации.
2. **movies**
   * movie\_id (INTEGER PK) — идентификатор фильма из Kinopoisk.
   * title (VARCHAR, NOT NULL) — название.
   * year (INTEGER) — год выпуска.
   * poster\_url (VARCHAR) — ссылка на постер.
   * description (TEXT) — краткое описание сюжета.
   * last\_sync (TIMESTAMP) — дата последнего обновления из Kinopoisk.
3. **watchlist**
   * watchlist\_id (SERIAL PK) *или* составной ключ (user\_id, movie\_id) (см. коммент ниже).
   * user\_id (INTEGER, FK → users.user\_id).
   * movie\_id (INTEGER, FK → movies.movie\_id).
   * added\_at (TIMESTAMP, DEFAULT NOW()).
4. **ratings**
   * rating\_id (SERIAL PK) *или* составной ключ (user\_id, movie\_id).
   * user\_id (INTEGER, FK → users.user\_id).
   * movie\_id (INTEGER, FK → movies.movie\_id).
   * rating (SMALLINT, NOT NULL, CHECK (rating BETWEEN 1 AND 10)).
   * rated\_at (TIMESTAMP, DEFAULT NOW()).

## 5. Схема контрактов API





**Модуль аутентификации (Auth)**

**Регистрация нового пользователя**

**Метод**: POST  
**Путь**: /auth/register  
**Описание**: Создание новой учетной записи пользователя  
**Тело запроса**:

* email (string, обязательный) - email пользователя
* password (string, обязательный) - пароль

**Ответ**: 201 Created  
Возвращает объект пользователя с полями:

* user\_id (integer) - идентификатор пользователя
* email (string) - email
* created\_at (string) - дата создания в формате datetime

**Вход в систему**

**Метод**: POST  
**Путь**: /auth/login  
**Описание**: Аутентификация пользователя и получение JWT токена  
**Тело запроса**:

* email (string, обязательный) - email пользователя
* password (string, обязательный) - пароль

**Ответ**: 200 OK  
Возвращает объект с токеном:

* access\_token (string) - JWT токен для доступа к API
* expires\_in (integer) - срок действия токена в секундах (3600)
* token\_type (string) - тип токена ("Bearer")

**Модуль фильмов (Movies)**

**Получение списка фильмов**

**Метод**: GET  
**Путь**: /movies  
**Описание**: Получение списка фильмов с поддержкой пагинации  
**Параметры запроса**:

* page (integer, опциональный) - номер страницы (по умолчанию 1)
* size (integer, опциональный) - количество элементов на странице (1-100, по умолчанию 10)

**Ответ**: 200 OK  
Возвращает массив объектов Movie

**Поиск фильмов**

**Метод**: GET  
**Путь**: /movies/search  
**Описание**: Поиск фильмов по названию  
**Параметры запроса**:

* q (string, обязательный) - поисковый запрос

**Ответ**: 200 OK  
Возвращает отфильтрованный массив объектов Movie

**Популярные фильмы**

**Метод**: GET  
**Путь**: /movies/popular  
**Описание**: Получение списка популярных фильмов  
**Параметры запроса**:

* limit (integer, опциональный) - ограничение количества (по умолчанию 10)

**Ответ**: 200 OK  
Возвращает массив популярных фильмов (объекты Movie)

**Детальная информация о фильме**

**Метод**: GET  
**Путь**: /movies/{movie\_id}  
**Описание**: Получение подробной информации о конкретном фильме  
**Параметры пути**:

* movie\_id (integer, обязательный) - идентификатор фильма

**Ответ**: 200 OK - возвращает объект Movie  
**Ответ**: 404 Not Found - если фильм не найден

**YouTube обзоры фильма**

**Метод**: GET  
**Путь**: /movies/{movie\_id}/reviews  
**Описание**: Получение видео-обзоров с YouTube для указанного фильма  
**Параметры пути**:

* movie\_id (integer, обязательный) - идентификатор фильма

**Ответ**: 200 OK  
Возвращает массив объектов YouTubeReviewItem

**Модуль пользователя (User)**

**Получение профиля**

**Метод**: GET  
**Путь**: /users/me  
**Описание**: Получение информации о текущем авторизованном пользователе  
**Требует аутентификации**: Да (JWT токен)

**Ответ**: 200 OK  
Возвращает объект User

**Обновление профиля**

**Метод**: PATCH  
**Путь**: /users/me  
**Описание**: Обновление email и пароля пользователя  
**Требует аутентификации**: Да (JWT токен)  
**Тело запроса**:

* email (string, обязательный) - новый email
* password (string, обязательный) - новый пароль

**Ответ**: 200 OK  
Возвращает обновленный объект User

**Модуль списка просмотра (Watchlist)**

**Получение списка**

**Метод**: GET  
**Путь**: /users/{user\_id}/watchlist  
**Описание**: Получение списка фильмов "Смотреть позже" для указанного пользователя  
**Требует аутентификации**: Да (JWT токен)  
**Параметры пути**:

* user\_id (integer, обязательный) - идентификатор пользователя

**Ответ**: 200 OK  
Возвращает массив объектов WatchlistItem

**Добавление в список**

**Метод**: POST  
**Путь**: /users/{user\_id}/watchlist  
**Описание**: Добавление фильма в список "Смотреть позже"  
**Требует аутентификации**: Да (JWT токен)  
**Параметры пути**:

* user\_id (integer, обязательный) - идентификатор пользователя

**Тело запроса**:

* movie\_id (integer, обязательный) - идентификатор фильма

**Ответ**: 201 Created  
Возвращает объект с подтверждением добавления

**Удаление из списка**

**Метод**: DELETE  
**Путь**: /users/{user\_id}/watchlist/{movie\_id}  
**Описание**: Удаление фильма из списка "Смотреть позже"  
**Требует аутентификации**: Да (JWT токен)  
**Параметры пути**:

* user\_id (integer, обязательный) - идентификатор пользователя
* movie\_id (integer, обязательный) - идентификатор фильма

**Ответ**: 204 No Content

**Модуль рейтингов (Ratings)**

**Получение рейтингов**

**Метод**: GET  
**Путь**: /users/{user\_id}/ratings  
**Описание**: Получение всех оценок, выставленных пользователем  
**Требует аутентификации**: Да (JWT токен)  
**Параметры пути**:

* user\_id (integer, обязательный) - идентификатор пользователя

**Ответ**: 200 OK  
Возвращает массив объектов Rating

**Добавление/обновление рейтинга**

**Метод**: POST  
**Путь**: /users/{user\_id}/ratings  
**Описание**: Добавление или обновление оценки фильма  
**Требует аутентификации**: Да (JWT токен)  
**Параметры пути**:

* user\_id (integer, обязательный) - идентификатор пользователя

**Тело запроса**:

* movie\_id (integer, обязательный) - идентификатор фильма
* rating (integer, обязательный) - оценка от 1 до 10

**Ответ**: 201 Created  
Возвращает объект с подтверждением сохранения рейтинга

**Удаление рейтинга**

**Метод**: DELETE  
**Путь**: /users/{user\_id}/ratings/{movie\_id}  
**Описание**: Удаление оценки фильма  
**Требует аутентификации**: Да (JWT токен)  
**Параметры пути**:

* user\_id (integer, обязательный) - идентификатор пользователя
* movie\_id (integer, обязательный) - идентификатор фильма

**Ответ**: 204 No Content

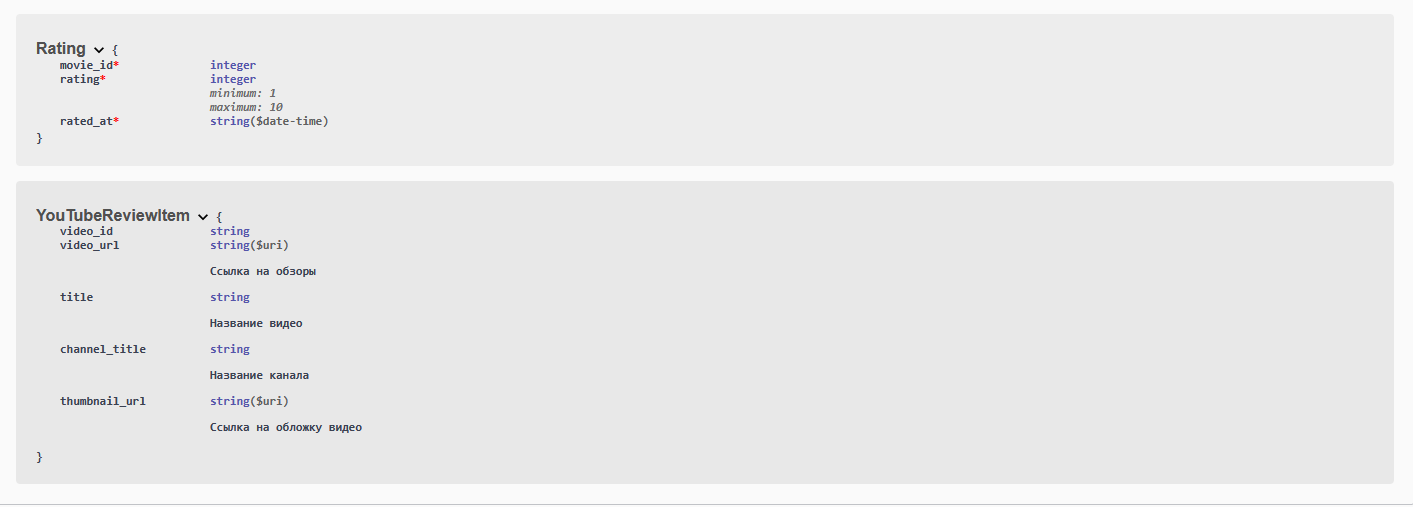
**Безопасность**

**Схема аутентификации**: JWT Bearer Token  
**Формат токена**: В заголовке Authorization: Bearer {token}  
**Срок действия**: 3600 секунд (1 час)

**Защищенные эндпоинты**: Все эндпоинты, кроме следующих, требуют JWT токен:

* POST /auth/register
* POST /auth/login
* GET /movies
* GET /movies/search
* GET /movies/popular
* GET /movies/{movie\_id}
* GET /movies/{movie\_id}/reviews

## 6. Схема моделей данных

**Модель User**

|  |
| --- |
| {  user\_id: integer;  email: string; // формат email  created\_at: string; // формат date-time } |

**Модель Movie**

|  |
| --- |
| {  movie\_id: integer; // внутренний идентификатор фильма  title: string; // название фильма  year: integer; // год выхода  poster\_url: string; // ссылка на постер (URI)  description: string; // краткое описание сюжета  ratingKinopoisk: number; // рейтинг на Кинопоиске (float) } |

**Модель WatchlistItem**

|  |
| --- |
| {  movie\_id: integer;  added\_at: string; // дата добавления (date-time)  title: string; // название фильма  poster\_url: string; // ссылка на постер (URI) } |

**Модель Rating**

|  |
| --- |
| {  movie\_id: integer;  rating: integer; // оценка от 1 до 10  rated\_at: string; // дата оценки (date-time) } |

**Модель YouTubeReviewItem**

|  |
| --- |
| {  video\_id: string; // идентификатор видео на YouTube  video\_url: string; // ссылка на видео (URI)  title: string; // название видео  channel\_title: string; // название канала  thumbnail\_url: string; // ссылка на миниатюру (URI) } |