**Сервис для бесплатного просмотра фильмов с социальными функциями и пользовательскими рейтингами**

# Состав группы и репозиторий

* Хисматулин Вадим Русланович
* Маслов Денис Кириллович
* Семенов Александр Борисович

Ссылка на репозиторий: <https://github.com/r4mdevelopment/Watch>

# Технологический стек

* **Бэкенд (Серверная часть):** Python (FastAPI)
* **База данных:** PostgreSQL
* **Фронтенд:** HTML + CSS + JavaScript
* **ORM:** SQLAlchemy
* **Видеоплеер:** Kodik Player
* **Аутентификация:** JWT (JSON Web Tokens)
* **Контейнеризация:** Docker + Docker Compose
* **Внешние зависимости (API):**
  + The Movie Database (TMDB) - получение метаданных о фильмах
  + Kodik API - стриминг видеофайлов

# Проблема

Все популярные иностранные сервисы стриминга фильмов ушли из России, а за "Кинопоиск" и подобное приходится платить, но даже так не весь каталог фильмов там есть. Нужно создать сервис, в котором можно будет смотреть все фильмы БЕСПЛАТНО, а также ставить им оценки, делиться этими оценками с друзьями, создавать списки фильмов (например: свой топ-10). на сайте будет личный кабинет, который можно будет кастомизировать(менять фоны и подобное), создавать обсуждения фильмов/трейлеров/тизеров. Для авторов - делать анонсы своих фильмов или объявлять сбор средств.

# Выработка требований

Если рассматривать как в формате "как <роль>, я хочу <возможность>, чтобы <цель/выгода>":

**1. Просмотр контента:**

* **Как** анонимный пользователь, **я хочу** видеть на главной странице подборки фильмов (популярные, новинки, рекомендуемые), **чтобы** быстро найти что-то интересное для просмотра.
* **Как** пользователь, **я хочу** иметь возможность искать фильмы по названию, жанру и году выпуска, **чтобы** быстро найти конкретный фильм или открыть для себя новое кино.
* **Как** пользователь, **я хочу** нажимать на кнопку "Смотреть" и чтобы фильм воспроизводился в качественном встроенном плеере, **чтобы** мне не приходилось переходить на другие сайты.

**2. Управление аккаунтом и социализация:**

* **Как** новый пользователь, **я хочу** зарегистрироваться по email и паролю или через социальные сети, **чтобы** получить доступ ко всему функционалу сайта.
* **Как** пользователь, **я хочу** ставить фильмам оценку (от 1 до 10) и писать рецензии, **чтобы** делиться своим мнением и помочь другим пользователям выбрать фильм.
* **Как** пользователь, **я хочу** добавлять фильмы в списки "Просмотрено", "Буду смотреть" и создавать собственные списки (например, "Лучшие комедии 90-х"), **чтобы** систематизировать свою коллекцию и делиться ей с друзьями.
* **Как** пользователь, **я хочу** видеть профили других пользователей, их списки и рейтинги, **чтобы** находить единомышленников и новые фильмы через их вкусы.

**3. Кастомизация и дополнительные функции:**

* **Как** пользователь, **я хочу** менять аватарку и фон своего профиля, **чтобы** персонализировать свою страницу.
* **Как** пользователь, **я хочу** видеть раздел с обсуждениями (форум) к каждому фильму, **чтобы** читать мнения других и участвовать в дискуссиях после просмотра.

Если рассматривать пользовательские сценарии:

*Пользовательский сценарий 1:*

Я фанат фильмов MARVEL, который хочет рассказывать о своем опыте просмотра новинок и ставить им оценки. Я бы хотел иметь сайт, на котором соберу своих единомышленников для того, чтобы говорить об отсылках, деталях и забавных моментах в фильмах/сериалах. Круто было бы красиво оформить свой профиль, чтобы привлечь больше людей на мою страницу.

*Пользовательский сценарий 2:*

Я студент киновуза, который хочет снять свой дипломный проект, но никак не может найти команду и деньги на это. Хотелось бы иметь сайт, на котором я могу организовать сбор средств, а также поискать людей в команду, которых заинтересовал мой проект.

# Разработка архитектуры и детальное проектирование

1. **Характер нагрузки на сервис**

Ожидаемое количество пользователей: 10,000 уникальных пользователей в сутки

Пиковая нагрузка: до 2,000 одновременных подключений

Срок хранения данных: 7 лет

1. **Соотношение R/W нагрузки**

Чтение (Read): 85% - поиск фильмов, просмотр деталей, профилей, списков, рецензий

Запись (Write): 15% - регистрация, оценки, рецензии, создание списков

1. **Объемы трафика**

Средняя нагрузка: 10–50 запросов/сек

Пиковая нагрузка: до 200 – 500 запросов/сек (вечера, премьеры)

Трафик: 100 – 500 ГБ/мес

Устройства: 70% телефоны, 30% компьютеры

1. **Объемы дисковой системы**

База данных: 20–100 ГБ (фильмы, пользователи, отзывы, списки)

Файлы (аватары, обложки): 1–5 ГБ

При росте ×10: до 1 ТБ (включая резерв)

**C4 модель состоит из 4 уровней абстракции, где каждый следующий уровень детализирует предыдущий.**

**Диаграмма уровня 1: Контекст системы (System Context Diagram)**

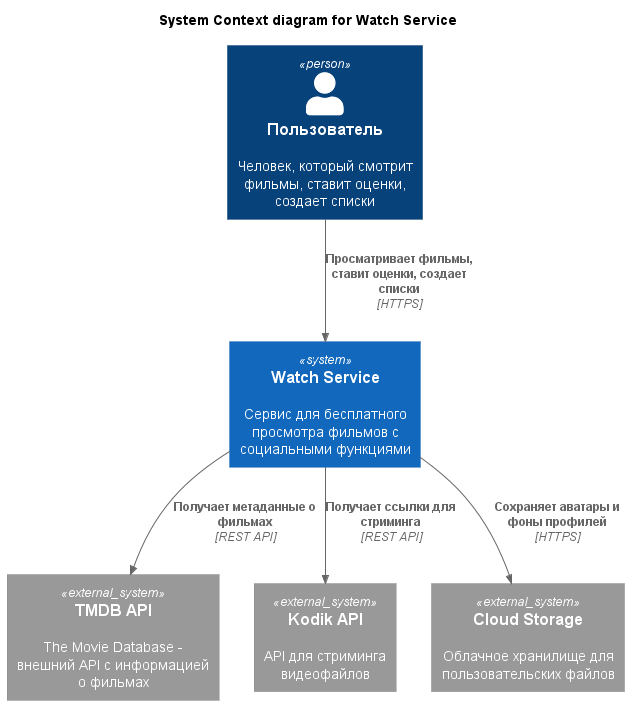
****

Диаграмма показывает систему в целом и ее взаимодействие с внешними системами и пользователями.

Основные элементы:

* Пользователь - человек, который использует сервис для просмотра фильмов, выставления оценок и создания списков
* Watch Service - основная система, предоставляющая функционал стриминга фильмов
* TMDB API - внешний сервис, предоставляющий метаданные о фильмах (постеры, описания, актерский состав)
* Kodik API - внешний сервис для стриминга видеофайлов
* Cloud Storage - облачное хранилище для пользовательских файлов

Взаимодействия:

* Пользователь взаимодействует с Watch Service через HTTPS протокол
* Watch Service получает информацию о фильмах от TMDB API через REST API
* Watch Service получает видео для воспроизведения от Kodik API через REST API
* Watch Service сохраняет аватары и фоны профилей в Cloud Storage через HTTPS

**Диаграмма уровня 2: Контейнеры (Containers)**

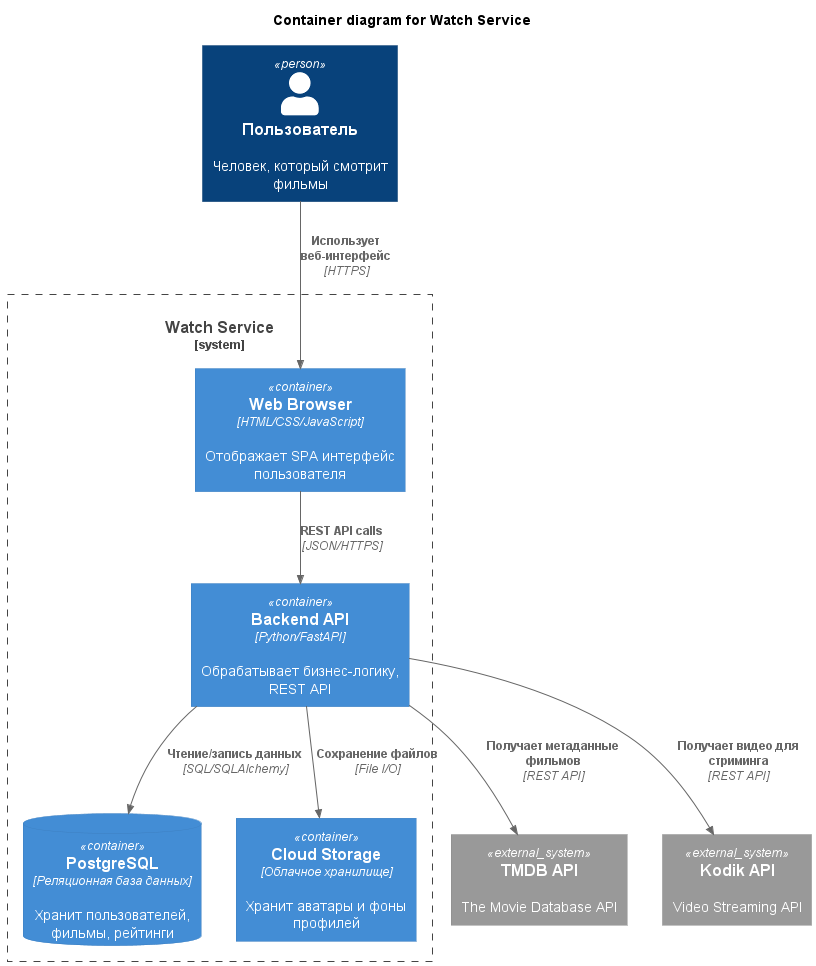


Диаграмма показывает высокоуровневую архитектуру системы и технические выборы.

Основные контейнеры:

* Web Browser - клиентское приложение на HTML/CSS/JavaScript, отображающее SPA интерфейс
* Backend API - серверное приложение на Python/FastAPI, обрабатывающее бизнес-логику
* PostgreSQL - реляционная база данных для хранения структурированных данных
* File Storage - локальное хранилище файлов для аватаров и фонов профилей

Взаимодействия между контейнерами:

* Браузер общается с бэкендом через REST API (JSON/HTTPS)
* Бэкенд взаимодействует с базой данных через SQLAlchemy
* Бэкенд сохраняет файлы в локальное хранилище
* Бэкенд получает данные от внешних API (TMDB, Kodik)

**Диаграмма уровня 3: Компоненты Backend API (Components)**

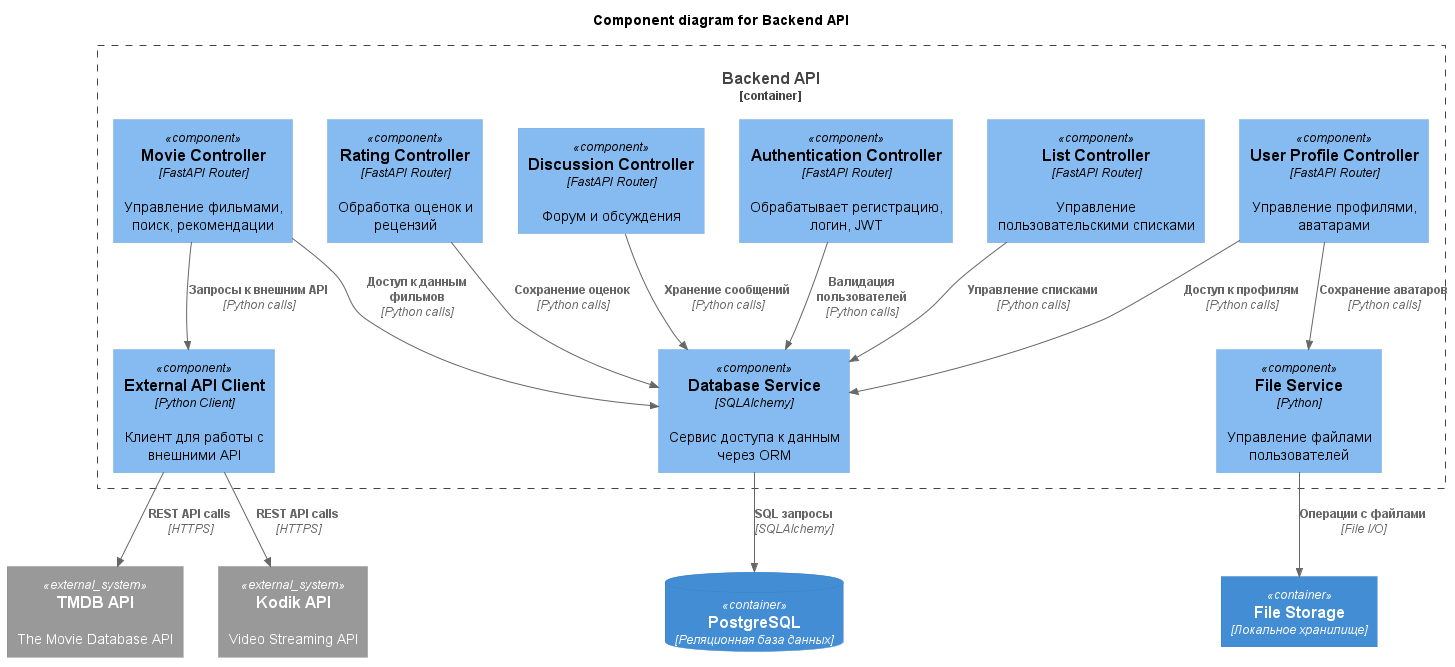
****

Диаграмма показывает внутреннюю структуру бэкенд приложения.

Контроллеры (обработчики запросов):

* Authentication Controller - управление аутентификацией (регистрация, вход, JWT)
* Movie Controller - работа с фильмами (поиск, рекомендации, информация)
* User Profile Controller - управление профилями пользователей
* Rating Controller - обработка оценок и рецензий
* List Controller - управление пользовательскими списками фильмов
* Discussion Controller - форум и обсуждения

Сервисные компоненты:

* External API Client - клиент для работы с внешними API (TMDB, Kodik)
* Database Service - сервис доступа к данным через SQLAlchemy
* File Service - управление файлами пользователей

Взаимодействия:

* Все контроллеры используют Database Service для доступа к данным
* Movie Controller использует External API Client для получения данных из TMDB
* Profile Controller использует File Service для работы с аватарами
* Database Service взаимодействует с PostgreSQL через SQLAlchemy
* File Service работает с локальным файловым хранилищем

**Диаграмма уровня 4: Код компонента аутентификации (Code)**

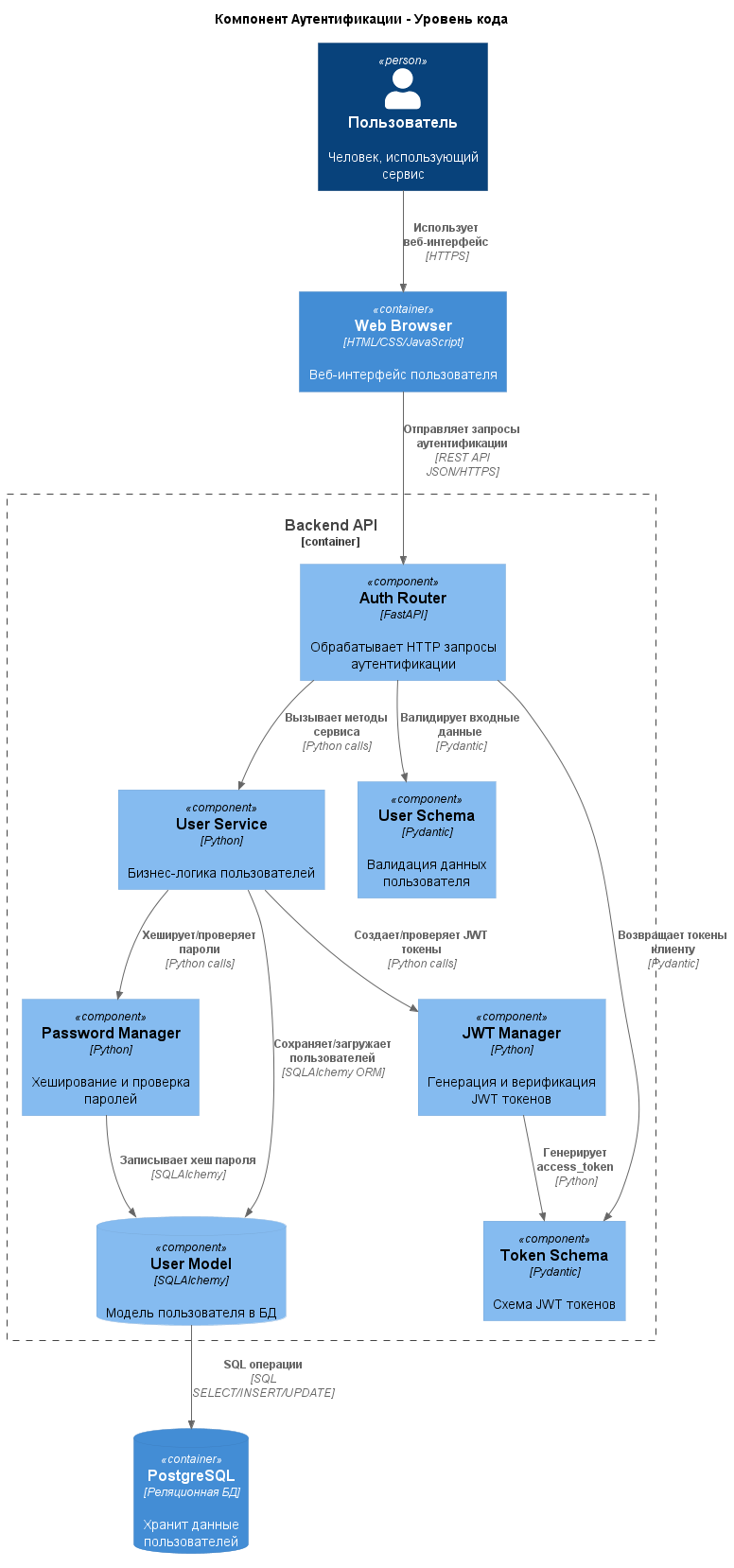


Диаграмма показывает детальную реализацию ключевого компонента системы.

Слои приложения:

* Presentation Layer: Auth Router (FastAPI маршруты)
* Business Layer: User Service (бизнес-логика)
* Security Layer: JWT Manager, Password Manager
* Data Layer: User Model, схемы валидации

Процесс аутентификации:

* Пользователь отправляет запрос через браузер
* Auth Router валидирует данные через Pydantic схемы
* User Service обрабатывает бизнес-логику
* Password Manager хеширует/проверяет пароли
* JWT Manager генерирует/верифицирует токены
* User Model сохраняет/загружает данные из PostgreSQL

Поток данных:

HTTP запросы → Валидация → Бизнес-логика → Безопасность → База данных

1. **Контракты API**

Аутентификация:

POST /api/auth/register

Body: {email: string, password: string, username: string}

Response: {user\_id: int, token: string, username: string}

Timeout: 500ms

POST /api/auth/login

Body: {email: string, password: string}

Response: {user\_id: int, token: string, username: string}

Timeout: 500ms

GET /api/users/{user\_id}

Headers: {Authorization: Bearer {token}}

Response: {user: {id, username, avatar\_url, created\_at}}

Timeout: 200ms

Фильмы:

GET /api/movies/popular?page=1

Response: {movies: [], total\_pages: int}

Timeout: 1s

GET /api/movies/{movie\_id}

Response: {movie: {id, title, description, rating, genres, release\_date}}

Timeout: 500ms

GET /api/movies/{movie\_id}/stream

Response: {stream\_url: string}

Timeout: 2s (внешний API)

GET /api/movies/search?q=avatar&genre=action&year=2020

Response: {movies: [], total\_results: int}

Timeout: 1s

Социальные функции:

POST /api/movies/{movie\_id}/rate

Body: {rating: int (1-10)}

Headers: {Authorization: Bearer {token}}

Response: {success: boolean, average\_rating: float}

Timeout: 500ms

POST /api/movies/{movie\_id}/review

Body: {content: string, rating: int}

Headers: {Authorization: Bearer {token}}

Response: {review\_id: int, created\_at: string}

Timeout: 500ms

GET /api/movies/{movie\_id}/reviews?page=1

Response: {reviews: [], total\_pages: int}

Timeout: 500ms

POST /api/lists

Body: {name: string, description: string, is\_public: boolean}

Headers: {Authorization: Bearer {token}}

Response: {list\_id: int, created\_at: string}

Timeout: 500ms

1. **Нефункциональные требования**

**Производительность**

* Стандартные запросы API: ≤ 500 мс
* Поиск и подборки: ≤ 1 сек
* Внешние API (TMDB, Kodik): ≤ 2 сек (с кэшированием)
* Время загрузки страницы: ≤ 2 сек
* Начало воспроизведения видео: ≤ 3 сек

**Надежность**

* Доступность: ≥ 99% uptime
* Время восстановления: <30 минут
* Резервное копирование: ежедневно

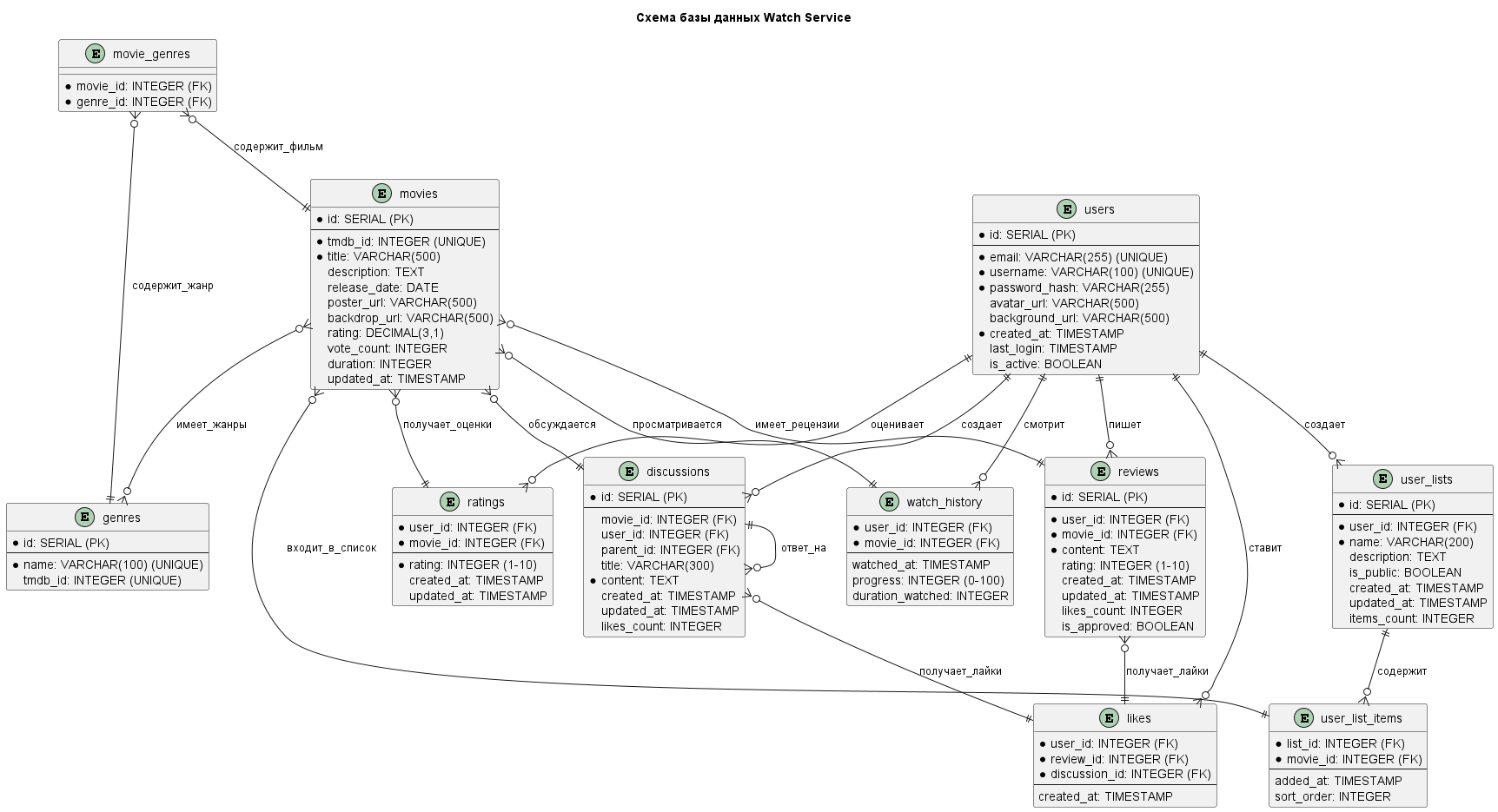
**Масштабируемость**

* Пользователи: до 10,000 активных пользователей
* Контейнеризация: Docker поддержка
* Rate limiting: 100 запросов/мин на пользователя

**Безопасность**

* HTTPS: обязательное шифрование
* JWT tokens: срок жизни 24 часа
* Валидация: Pydantic схемы
* SQL инъекция защита: SQLAlchemy

1. **Схема базы данных**

****

**Код:**

CREATE TABLE users (

id SERIAL PRIMARY KEY,

email VARCHAR(255) UNIQUE NOT NULL,

username VARCHAR(100) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

avatar\_url VARCHAR(500),

background\_url VARCHAR(500),

created\_at TIMESTAMP DEFAULT NOW(),

last\_login TIMESTAMP,

is\_active BOOLEAN DEFAULT TRUE

);

CREATE TABLE movies (

id SERIAL PRIMARY KEY,

tmdb\_id INTEGER UNIQUE NOT NULL,

title VARCHAR(500) NOT NULL,

description TEXT,

release\_date DATE,

poster\_url VARCHAR(500),

backdrop\_url VARCHAR(500),

rating DECIMAL(3,1),

vote\_count INTEGER DEFAULT 0,

duration INTEGER, -- в минутах

updated\_at TIMESTAMP DEFAULT NOW()

);

CREATE TABLE genres (

id SERIAL PRIMARY KEY,

name VARCHAR(100) UNIQUE NOT NULL,

tmdb\_id INTEGER UNIQUE

);

CREATE TABLE movie\_genres (

movie\_id INTEGER REFERENCES movies(id) ON DELETE CASCADE,

genre\_id INTEGER REFERENCES genres(id) ON DELETE CASCADE,

PRIMARY KEY (movie\_id, genre\_id)

);

CREATE TABLE ratings (

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

movie\_id INTEGER REFERENCES movies(id) ON DELETE CASCADE,

rating INTEGER CHECK (rating >= 1 AND rating <= 10),

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW(),

PRIMARY KEY (user\_id, movie\_id)

);

CREATE TABLE reviews (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

movie\_id INTEGER REFERENCES movies(id) ON DELETE CASCADE,

content TEXT NOT NULL,

rating INTEGER CHECK (rating >= 1 AND rating <= 10),

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW(),

likes\_count INTEGER DEFAULT 0,

is\_approved BOOLEAN DEFAULT TRUE

);

CREATE TABLE watch\_history (

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

movie\_id INTEGER REFERENCES movies(id) ON DELETE CASCADE,

watched\_at TIMESTAMP DEFAULT NOW(),

progress INTEGER DEFAULT 0 CHECK (progress >= 0 AND progress <= 100),

duration\_watched INTEGER DEFAULT 0, -- в секундах

PRIMARY KEY (user\_id, movie\_id)

);

CREATE TABLE user\_lists (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

name VARCHAR(200) NOT NULL,

description TEXT,

is\_public BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW(),

items\_count INTEGER DEFAULT 0

);

CREATE TABLE user\_list\_items (

list\_id INTEGER REFERENCES user\_lists(id) ON DELETE CASCADE,

movie\_id INTEGER REFERENCES movies(id) ON DELETE CASCADE,

added\_at TIMESTAMP DEFAULT NOW(),

sort\_order INTEGER DEFAULT 0,

PRIMARY KEY (list\_id, movie\_id)

);

CREATE TABLE discussions (

id SERIAL PRIMARY KEY,

movie\_id INTEGER REFERENCES movies(id) ON DELETE CASCADE,

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

parent\_id INTEGER REFERENCES discussions(id) ON DELETE CASCADE,

title VARCHAR(300),

content TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW(),

likes\_count INTEGER DEFAULT 0

);

CREATE TABLE likes (

user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,

review\_id INTEGER REFERENCES reviews(id) ON DELETE CASCADE,

discussion\_id INTEGER REFERENCES discussions(id) ON DELETE CASCADE,

created\_at TIMESTAMP DEFAULT NOW(),

PRIMARY KEY (user\_id, review\_id, discussion\_id),

CHECK (

(review\_id IS NOT NULL AND discussion\_id IS NULL) OR

(review\_id IS NULL AND discussion\_id IS NOT NULL)

)

);

Для обеспечения производительности используются индексы на часто запрашиваемых полях:

- Поиск пользователей (email)

- Поиск фильмов (название)

- Частые связи (user\_id, movie\_id)

1. **Схема масштабирования при росте нагрузки в 10 раз**

**Текущая → 100,000 пользователей/день**

