**Отчет по проекту "Онлайн-кинотеатр"**

**Раздел 0**

**ФИО:** Куцыба Н.Д.

Гранин И.Д.

**Группа:** 6305-010302D  
**Проект:** Онлайн-кинотеатр  
**GitHub:** <https://github.com/Felisias/online_cinema>

**1. Архитектура проекта**

Проект представляет собой онлайн-кинотеатр, состоящий из:

* С помощью Django будет разработан пользовательский интерфейс и серверная часть приложения

**Взаимодействие компонентов**

1. Пользователь отправляет запрос с фронтенда к API (например, на просмотр списка фильмов).
2. Django обрабатывает запрос, взаимодействует с базой данных PostgreSQL и отправляет ответ в формате JSON.
3. Фронтенд получает ответ и отображает информацию пользователю.

**2. API (REST)**

**Основные эндпоинты**

**Пользователи (Users)**

* POST /api/users/register – регистрация нового пользователя.
* POST /api/users/login – авторизация.
* GET /api/users/me – информация о текущем пользователе.
* PUT /api/users/me – обновление данных пользователя.
* DELETE /api/users/me – удаление аккаунта.

**Фильмы (Movies)**

* GET /api/movies – список всех фильмов.
* GET /api/movies/{movieId} – информация о конкретном фильме.
* POST /api/movies – добавление нового фильма (для админов).
* PUT /api/movies/{movieId} – обновление информации о фильме (для админов).
* DELETE /api/movies/{movieId} – удаление фильма (для админов).

**Сериалы (Shows)**

* GET /api/shows – список всех сериалов.
* GET /api/shows/{showId} – информация о конкретном сериале.
* POST /api/shows – добавление нового сериала (для админов).
* PUT /api/shows/{showId} – обновление информации о сериале (для админов).
* DELETE /api/shows/{showId} – удаление сериала (для админов).

**Сезоны (Seasons)**

* GET /api/shows/{showId}/seasons – список сезонов для сериала.
* GET /api/shows/{showId}/seasons/{seasonId} – информация о конкретном сезоне.
* POST /api/shows/{showId}/seasons – добавление нового сезона (для админов).
* PUT /api/shows/{showId}/seasons/{seasonId} – обновление информации о сезоне (для админов).
* DELETE /api/shows/{showId}/seasons/{seasonId} – удаление сезона (для админов).

**Эпизоды (Episodes)**

* GET /api/shows/{showId}/episodes – список эпизодов для сериала.
* GET /api/shows/{showId}/episodes/{episodeId} – информация о конкретном эпизоде.
* POST /api/shows/{showId}/episodes – добавление нового эпизода (для админов).

**Жанры (Genres)**

* GET /api/genres/{genresId} – список жанров.
* POST /api/genres – добавление нового жанра (для админов).

**Поиск (Search)**

* GET /api/search?q=... – поиск фильмов/сериалов по названию, жанру, году.

**Отзывы/Рейтинги (Reviews/Ratings)**

* GET /api/movies/{movieId}/reviews – список отзывов для фильма.
* GET /api/shows/{showId}/reviews – список отзывов для сериала.
* POST /api/movies/{movieId}/reviews – добавление отзыва фильма.
* POST /api/shows/{showId}/reviews – добавление отзыва сериала.
* PUT /api/movies/{movieId}/reviews/{reviewId} – обновление отзыва фильма.
* PUT /api/shows/{showId}/reviews/{reviewId} – обновление отзыва сериала.
* DELETE /api/movies/{movieId}/reviews/{reviewId} – удаление отзыва фильма.
* DELETE /api/shows/{showId}/reviews/{reviewId} – удаление отзыва сериала.

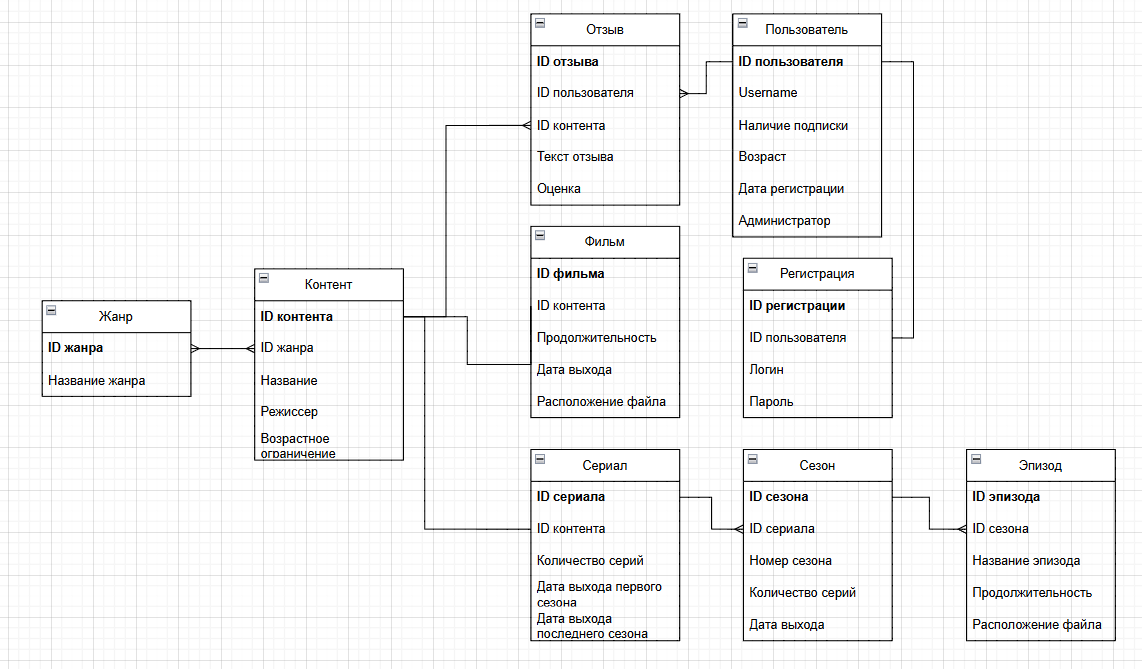
**Подписки и платежи (Subscriptions/Payments)**

* POST /api/users/me/subscriptions – покупка подписки.
* GET /api/users/me/subscriptions – информация о текущей подписке пользователя.

**3. Схема взаимодействия**

1. Пользователь регистрируется (POST /api/users/register) или входит в систему (POST /api/users/login).
2. Получает список фильмов (GET /api/movies).
3. Выбирает фильм, открывает его страницу (GET /api/movies/{movieId}).
4. Читает или оставляет отзыв (GET /api/movies/{movieId}/reviews, POST /api/movies/{movieId}/reviews).

**4. Схема базы данных**

****

База данных включает следующие сущности:

* Фильмы (ID, ID жанра, название, продолжительность, возрастное ограничение, дата выхода, режиссер, расположение файла)
* Сериалы (ID, ID жанра, название, количество серий, возрастное ограничение, дата выхода первого сезона, дата выхода последнего сезона, режиссер)
* Сезоны (ID, ID сериала, номер сезона, количество серий, дата выхода)
* Эпизоды (ID, ID сезона, название, продолжительность, расположение файла)
* Жанры (ID, название)
* Пользователи (ID, username, подписка, возраст, дата регистрации, статус администратора)
* Отзывы (ID, ID пользователя, ID фильма/сериала, текст, оценка)
* Регистрация (ID пользователя, логин, пароль)

**5. Стек технологий**

* **Фронтенд:** Django
* **Бэкенд:** Django
* **База данных:** PostgreSQL
* **Формат обмена данными:** JSON
* **Управление кодом:** Git, GitHub

**Дополнительно:** Docker и деплой пока не используются, но могут быть добавлены позже.

**Раздел 1**

**Отчет о выполнении лабораторной работы 1: Разработка базы данных**

**Цель работы**

Развернуть базу данных, разработать ORM-модели и реализовать скрипты для управления данными.

**Выполненные задачи**

**1. Настройка базы данных**

**Развертывание PostgreSQL в Docker**

Для работы с базой данных был использован PostgreSQL, развернутый в контейнере Docker. База данных была развернута с использованием следующей команды:

*docker run --name o\_cinema -e POSTGRES\_DB=cinema\_db -e POSTGRES\_USER=cinema\_user -e POSTGRES\_PASSWORD=cinema\_password -p 5432:5432 -d postgres:latest*

**Подключение к БД через DBeaver**

После развертывания базы данных было выполнено подключение к ней с использованием DBeaver. Параметры подключения:

* **Host:** localhost
* **Port:** 5432
* **Database:** cinema\_db
* **User:** cinema\_user
* **Password:** cinema\_password

**2. Разработка ORM-моделей**

**Определение сущностей и связей**

В рамках проекта были определены основные сущности:

* User (пользователь)
* Film (фильм)
* Review (отзыв)

Связи между сущностями:

* Один пользователь может оставить несколько отзывов (One-to-Many).
* Один фильм может иметь множество отзывов (One-to-Many).

**Реализация моделей с использованием Django ORM**

Для представления данных в базе была использована Django ORM. В файле models.py приложения cinema были определены модели с нужными полями и связями.

**Настройка миграций**

Были выполнены миграции для создания таблиц в базе данных:

*python manage.py makemigrations cinema*

*python manage.py migrate*

**3. Разработка модели пользователя**

**Определение структуры модели**

Была создана модель пользователя (User), включающая следующие поля:

* email (уникальный идентификатор);
* password (хешированный пароль);
* first\_name (имя);
* last\_name (фамилия);

**Настройка хеширования паролей**

Для безопасного хранения паролей было настроено их хеширование с использованием set\_password и проверки check\_password из Django.

**4. Функционал для работы с моделями**

**Написание скриптов для заполнения базы данных**

Был создан Python-скрипт для автоматического заполнения базы данных тестовыми данными. В скрипте использовалась библиотека Faker для генерации случайных пользователей и фильмов.

Пример команды для запуска скрипта:

python manage.py populate\_db

**Реализация API для работы с данными**

Были разработаны методы API для работы с моделями:

* **GET** /films/ — получить список фильмов;
* **POST** /films/ — добавить новый фильм;
* **GET** /films/<id>/ — получить информацию о конкретном фильме;
* **PUT/PATCH** /films/<id>/ — обновить информацию о фильме;
* **DELETE** /films/<id>/ — удалить фильм.

**Выводы**

В рамках лабораторной работы была успешно развернута база данных PostgreSQL в Docker, подключена через DBeaver, реализованы ORM-модели, выполнены миграции и настроены API-методы для работы с данными. Были написаны скрипты для автоматического заполнения базы данных и произведено тестирование API в Postman.

**Список команд, использованных в консоли**

python manage.py makemigrations cinema

python manage.py migrate

python manage.py showmigrations

python manage.py populate\_db

python manage.py runserver

python manage.py createsuperuser

**Раздел 2**

**Отчёт по лабораторной работе №2: Разработка API**

**Цель работы**

Разработать RESTful API для онлайн-кинотеатра с поддержкой базовых операций CRUD и возможностью работы с данными о фильмах, сериалах, сезонах, эпизодах, жанрах и отзывах пользователей.

**Задачи**

* Разработка CRUD-методов для всех моделей.
* Реализация маршрутов и контроллеров.
* Реализация сериализаторов для преобразования данных.
* Тестирование API с помощью Postman.
* Подготовка отчета и презентации.

**Структура API**

Используется Django + Django REST Framework. Все эндпоинты начинаются с /api/ и регистрируются через DefaultRouter.

**Основные маршруты:**

| **Метод** | **URL** | **Описание** |
| --- | --- | --- |
| GET | /api/users/ | Получить список пользователей |
| POST | /api/users/ | Создать пользователя |
| GET | /api/genres/ | Получить список жанров |
| POST | /api/genres/ | Добавить жанр |
| GET | /api/contents/ | Получить список контента |
| GET | /api/movies/ | Получить список фильмов |
| POST | /api/movies/ | Добавить фильм |
| GET | /api/series/ | Получить список сериалов |
| GET | /api/seasons/ | Получить список сезонов |
| GET | /api/episodes/ | Получить список эпизодов |
| GET | /api/reviews/ | Получить список отзывов |
| POST | /api/reviews/ | Добавить отзыв |
|  |  |  |

Аналогично PUT и DELETE.

Примеры запросов и ответов (Postman)

1. POST <http://127.0.0.1:8000/api/users/>

{

"email": "test@example.com",

"username": "testuser",

"password": "123456"

}

// в результате создаётся новый пользователь

1. GET <http://127.0.0.1:8000/api/movies/>

// получаем список фильмов

1. GET <http://127.0.0.1:8000/api/reviews/>

// получаем список отзывов

1. GET <http://127.0.0.1:8000/api/reviews/>1/

// получаем информацию об отзыве с ID 1

1. PUT <http://127.0.0.1:8000/api/movies/3/>

{

"id": 3,

"content": 5,

"duration": 170,

"release\_date": "2025-05-09",

"file\_path": "4.MP4"

}

// у фильма с ID 3 заменилось duration на 170

**Раздел 3**

**Лабораторная работа №3** (Авторизация пользователей с использованием JWT)

**1. Цель работы**

Целью лабораторной работы является реализация системы авторизации пользователей в веб-приложении онлайн-кинотеатра с использованием JWT-токенов и обеспечение безопасности REST API.

**2. Используемые технологии**

- Язык программирования: Python

- Фреймворк: Django + Django REST Framework

- JWT: Django Simple JWT

- База данных: PostgreSQL

- Тестирование API: Postman

- Docker

3. **Описание модели пользователя и аутентификации**

**Пользовательская модель:**

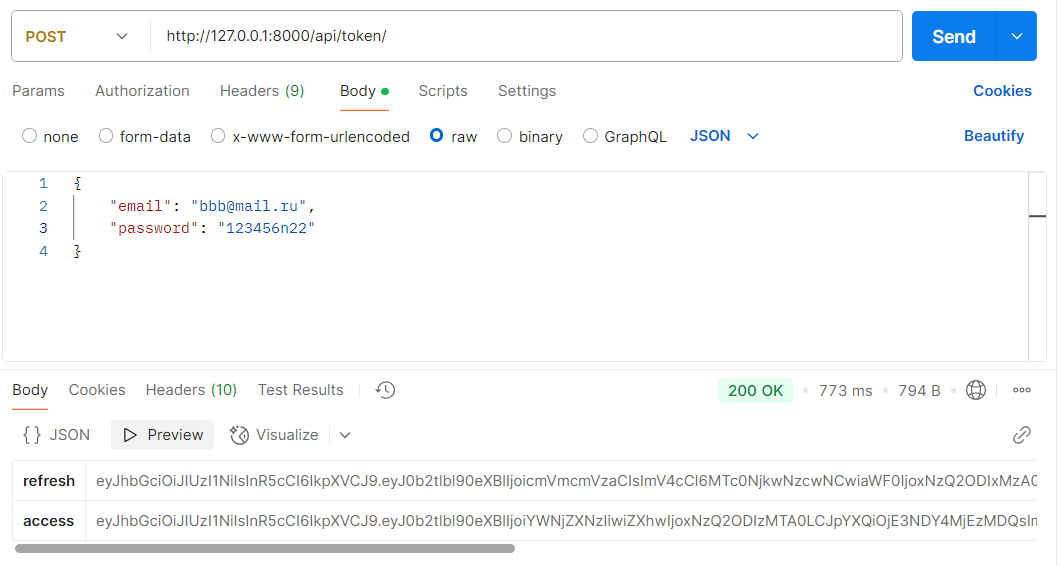
* id
* username
* email
* password (хэшируется)

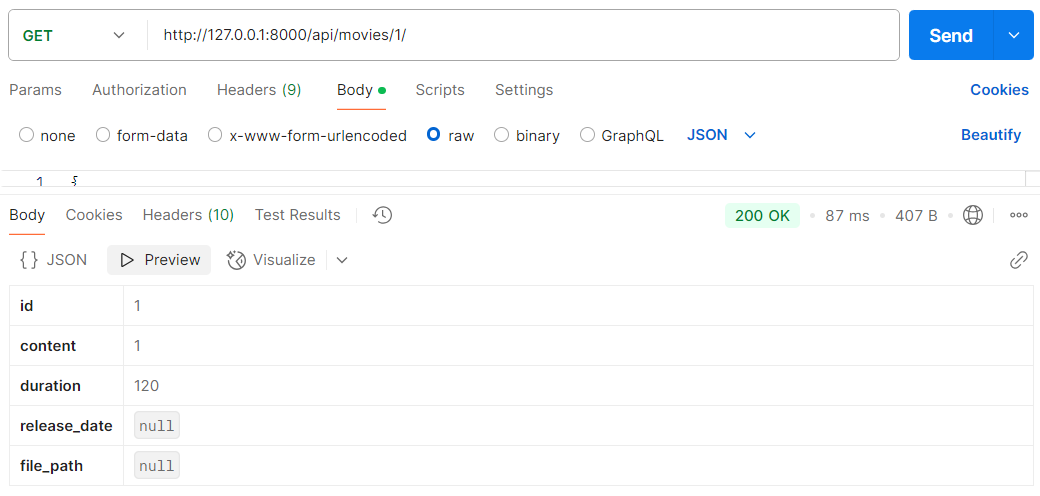
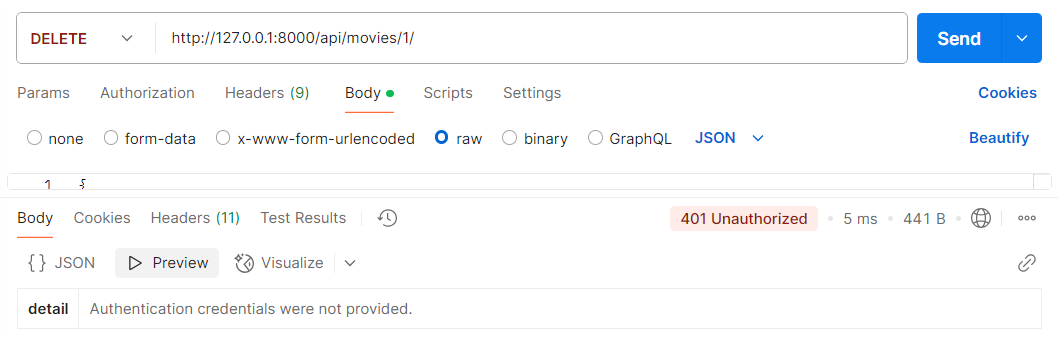


**JWT-аутентификация:**

Используется библиотека Django Simple JWT:

* /api/token/ — получение пары access/refresh токенов.
* /api/token/refresh/ — обновление access-токена.

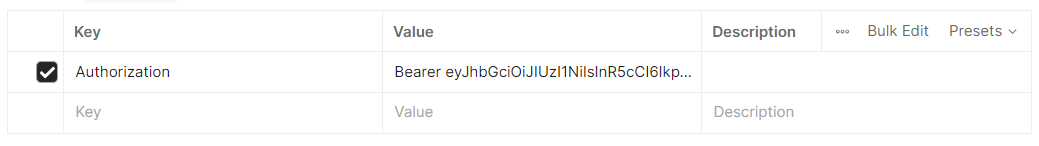
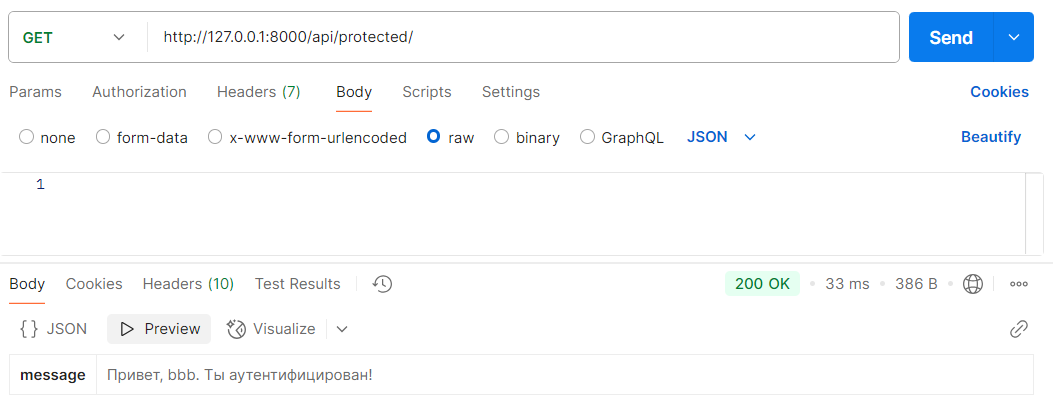


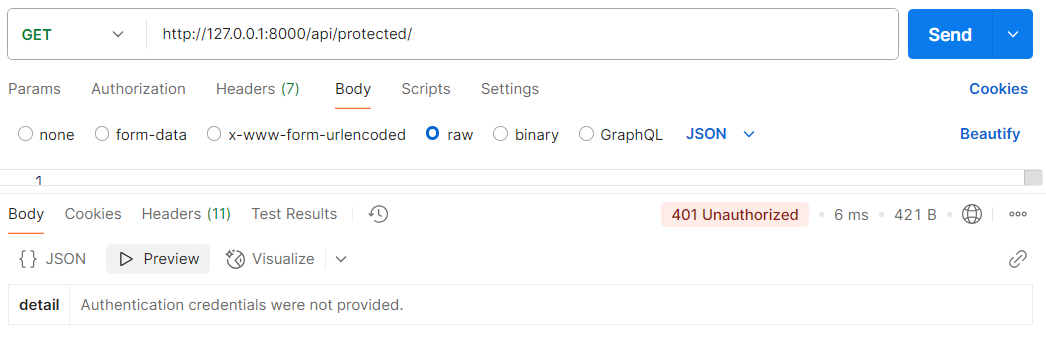
Пользователи без регистрации могут лишь просматривать контент, без возможности удалять, обновлять и создавать:

**Доступ к защищённому эндпоинту**

**URL:** GET /api/protected/  
**Заголовок:**

Authorization: Bearer <access\_token>



Если без токена:

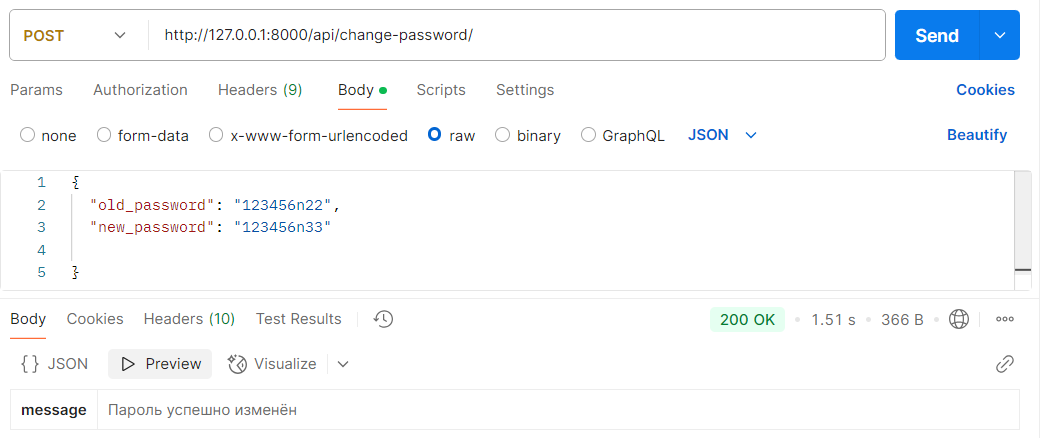
**Смена пароля**

**URL:** POST /api/change-password/  
**Запрос:**

{

"old\_password": "123456n22",

"new\_password": "123456n33"

}

**5. Выводы**

В ходе лабораторной работы реализована полноценная система аутентификации и авторизации пользователей с использованием JWT. Доступ к ресурсам API ограничен: для создания и изменения данных требуется аутентификация. Система обеспечивает регистрацию, вход, смену пароля и автоматическую привязку пользователей к создаваемым отзывам.

**Раздел 4**

**Лабораторная работа 4: Разработка пользовательского интерфейса и взаимодействие с API**

**Цель работы:**  
Создание клиентского веб-приложения с использованием Django, подключение его к серверу через REST API и реализация основных пользовательских сценариев.

**1. Описание структуры интерфейса**

Приложение реализует интерфейс онлайн-кинотеатра. Пользователь может:

* Зарегистрироваться и войти в аккаунт (страницы регистрации и авторизации).
* Просматривать список фильмов и сериалов.
* Переходить на отдельные страницы фильмов, сезонов, эпизодов.
* Оставлять отзывы к контенту (если авторизован).
* Видеть отзывы других пользователей.

**Основные страницы:**

* /register/ — страница регистрации.
* /login/ — страница входа.
* /films/ и /series/ — списки фильмов и сериалов.
* /films/<id>/ и /series/<id>/ — детальная страница фильма или сериала.
* /series/<id>/seasons/<id>/episodes/<id>/ — страница эпизода с отзывами.

Все страницы реализованы с использованием Django-шаблонов (Jinja2) и обрабатываются сервером.

**2. Код макетов (шаблонов)**

Примеры шаблонов:

**templates/login.html**

<form method="post" action="/login/">

{% csrf\_token %}

<input type="text" name="username" placeholder="Username">

<input type="password" name="password" placeholder="Password">

<button type="submit">Login</button>

</form>

**templates/register.html**

<form method="post" action="/register/">

{% csrf\_token %}

<input type="text" name="username" placeholder="Username">

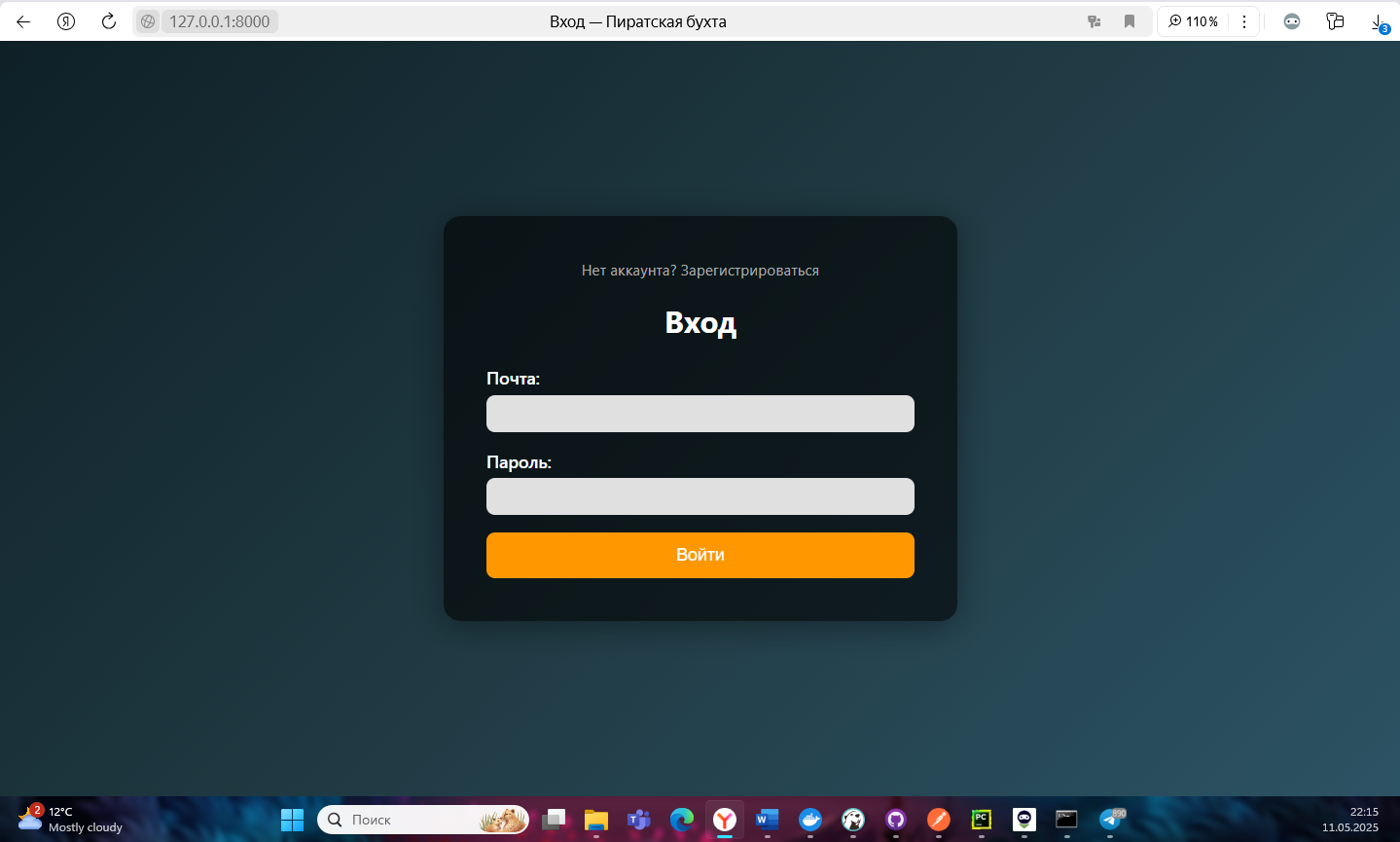
<input type="password" name="password" placeholder="Password">

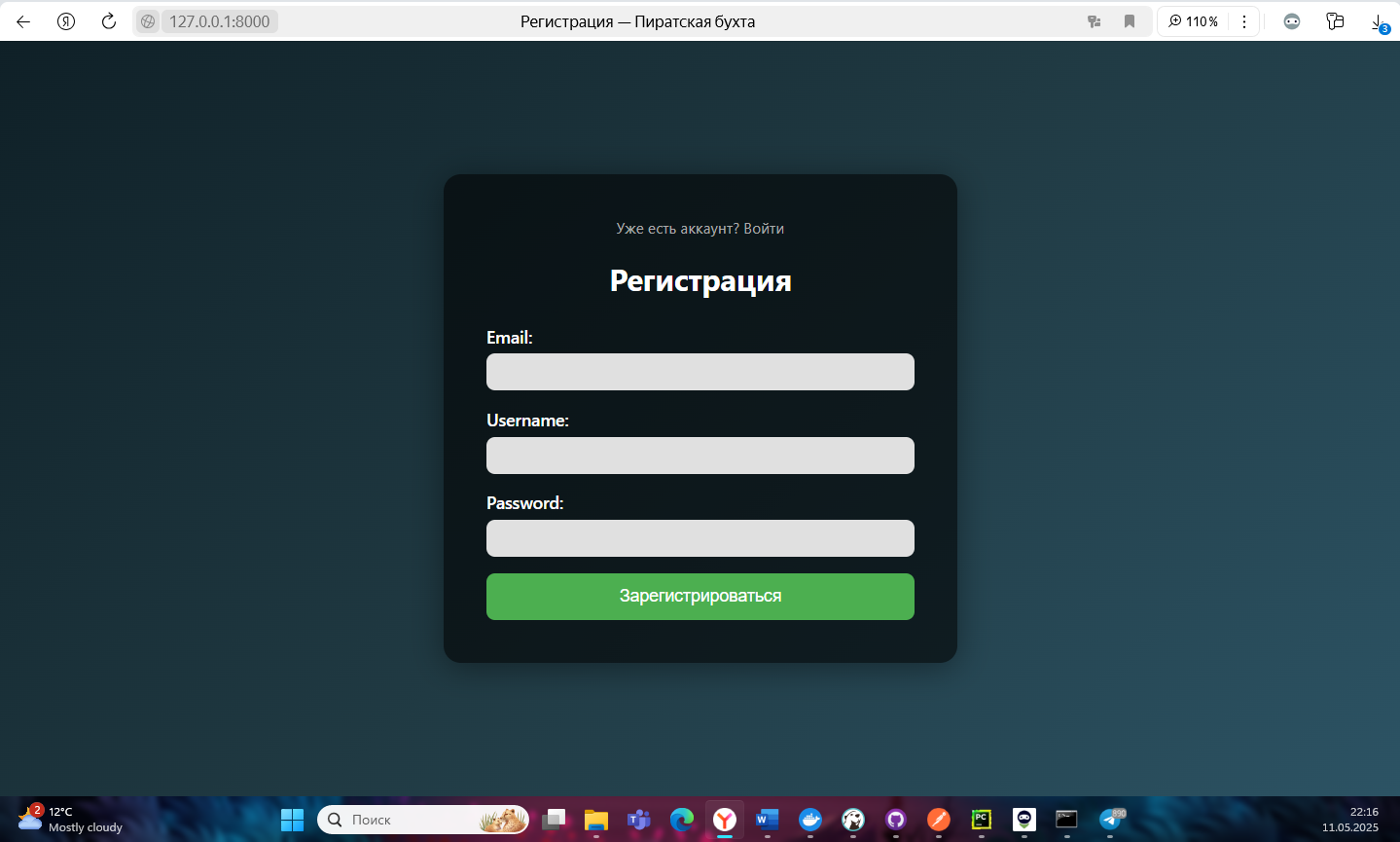
<input type="email" name="email" placeholder="Email">

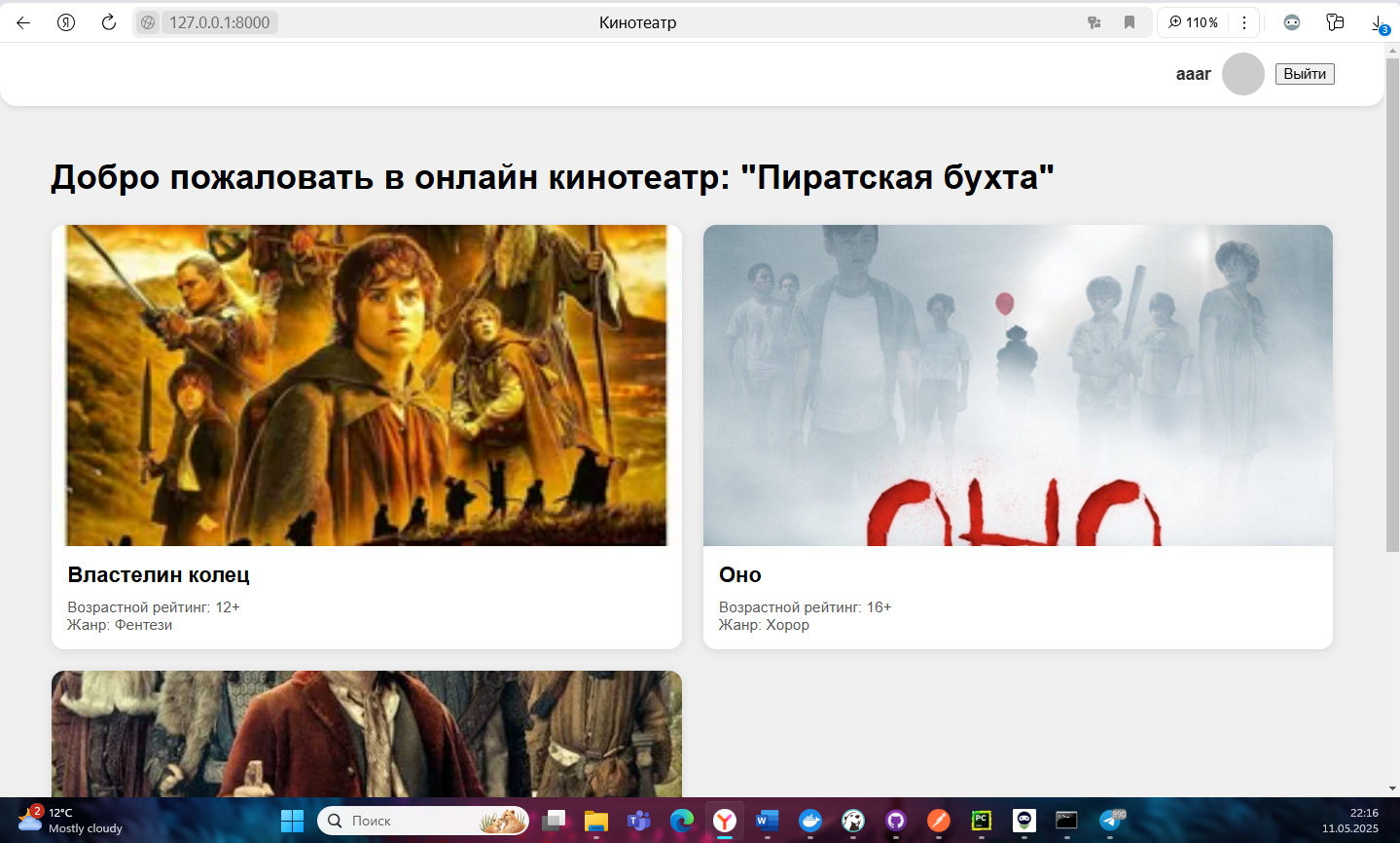
<button type="submit">Register</button>

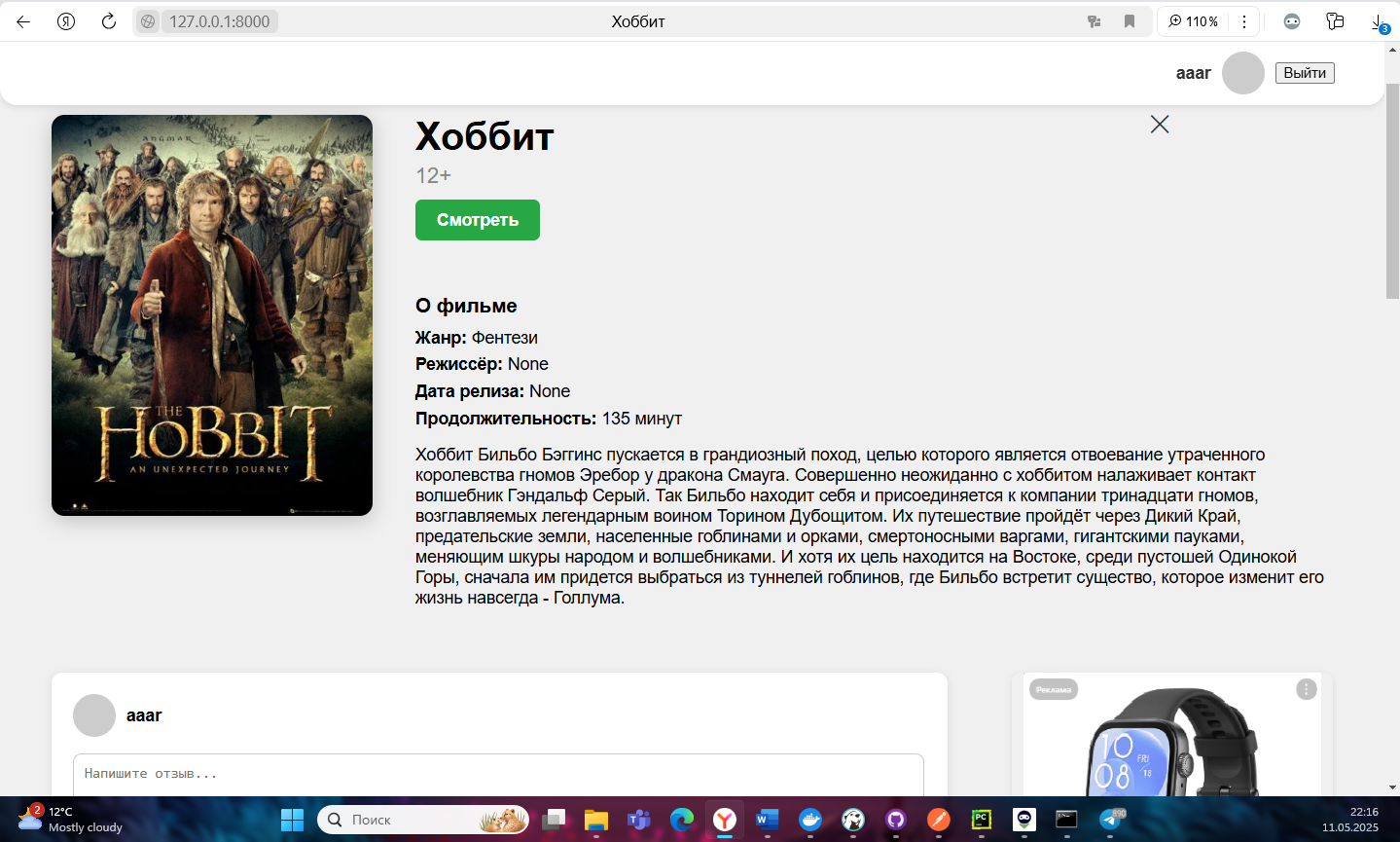
</form>

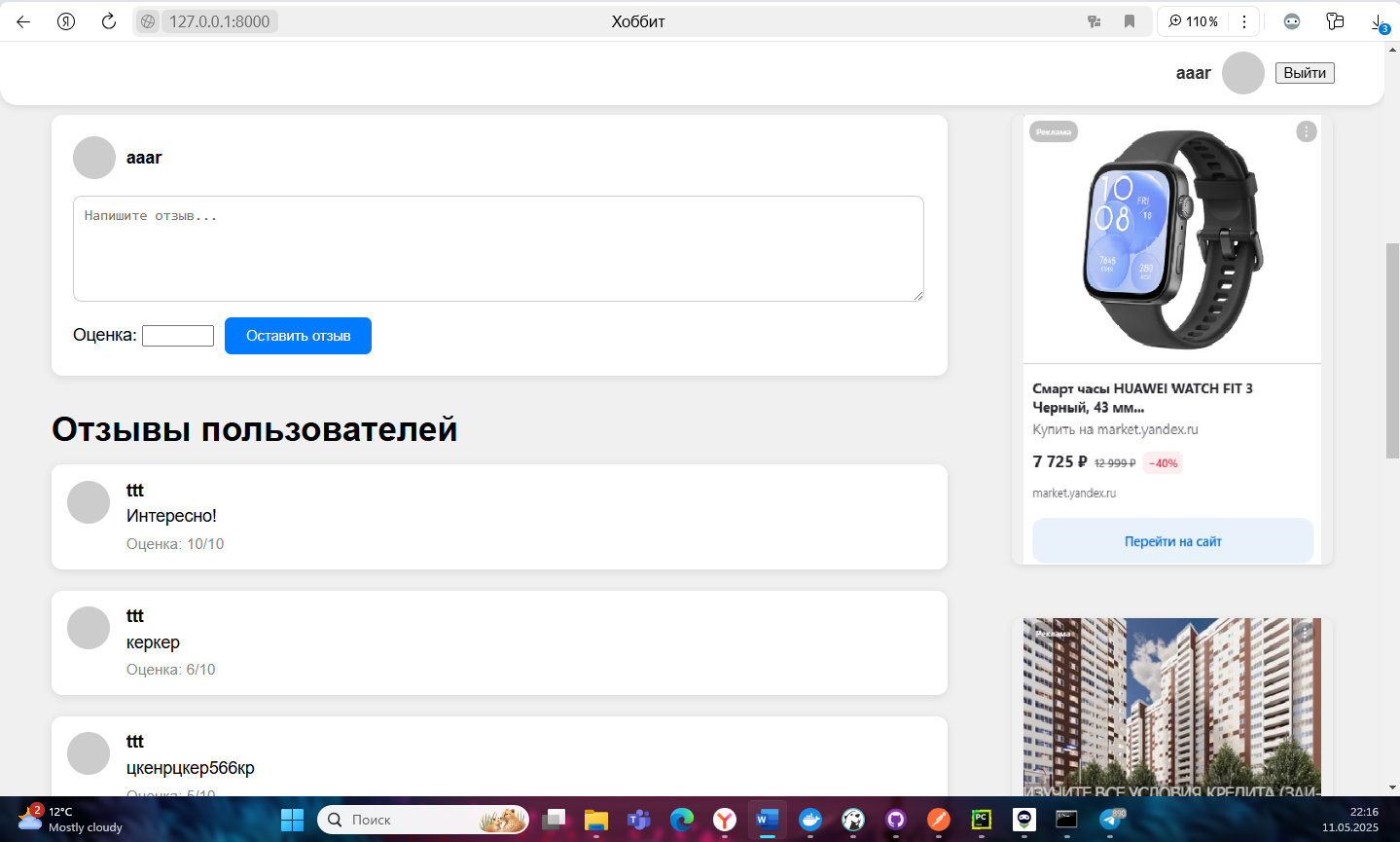
**3. Скриншоты страниц**











**4. Обработка данных и логика API**

**Регистрация и вход**

* При регистрации/авторизации данные отправляются через POST-запрос на сервер.
* В ответ сервер выдает JWT-токен.
* В views.py проверяется токен при каждом запросе, и пользователь определяется по нему.

**Пример кода обработки входа:**

def login\_view(request):

if request.method == 'POST':

username = request.POST['username']

password = request.POST['password']

response = requests.post(API\_URL + '/token/', data={'username': username, 'password': password})

if response.status\_code == 200:

token = response.json().get('access')

resp = redirect('/')

resp.set\_cookie('jwt\_token', token)

return resp

return render(request, 'login.html')

**Получение данных:**

* Данные фильмов, сериалов, отзывов получаются через requests.get(...) с заголовком Authorization: Bearer <token>.
* Пример:

response = requests.get(API\_URL + f'/films/{film\_id}/', headers={'Authorization': f'Bearer {token}'})

film = response.json()

return render(request, 'film\_detail.html', {'film': film})

**6. Вывод**

Все основные задачи лабораторной работы были выполнены:

* Созданы шаблоны на Django (Jinja2) для всех нужных страниц.
* Реализована регистрация и авторизация пользователей с использованием JWT.
* Данные загружаются через REST API и отображаются в интерфейсе.
* Реализовано отображение отзывов и проверка авторизации.
* Интерфейс протестирован: данные отображаются корректно, взаимодействие с сервером работает.

**Раздел 5**

**Отчёт по лабораторной работе №5** (Финализация приложения и упаковка в Docker)

**Цель работы**  
Завершить разработку приложения, связать клиентскую и серверную части, создать и запустить Docker-контейнеры для базы данных, backend и frontend, убедиться в их взаимодействии.

**Выполненные задачи**

1. Связаны фронтенд и бэкенд.
2. Устранены ошибки при передаче данных между компонентами.
3. Создан Docker-контейнер для PostgreSQL и сконфигурировано подключение.
4. Создан Dockerfile для backend-сервиса.
5. Создан docker-compose.yml для запуска всех компонентов.
6. Проведено тестирование контейнеров.
7. Подготовлена документация.

**Архитектура приложения**

Общая структура взаимодействия компонентов:

[Пользователь]  
↓  
[Frontend (Jinja2 + Django views)]  
↓  
[Django Backend (REST API)]  
↓  
[PostgreSQL DB (в Docker)]

Описание компонентов:

* **Frontend:** шаблоны Django на Jinja2. Формируют пользовательский интерфейс и передают данные на backend.
* **Backend:** Django REST Framework. Отвечает за обработку API-запросов, авторизацию, сериализацию и бизнес-логику.
* **База данных:** PostgreSQL в контейнере Docker.
* **Docker:** используется для развёртывания backend и базы данных.

**Связывание фронтенда и бэкенда**

* Все шаблоны (например, home.html, content\_detail.html) получают данные от API через requests.
* Авторизация пользователя реализована на основе JWT: токены передаются в HTTP-заголовках.
* Фронтенд-формы регистрации, входа и создания отзывов отправляют данные на API.

Пример запроса из views:

requests.get('http://127.0.0.1:8000/api/contents/<id>/', headers=headers)

**Docker: конфигурация базы данных**

Пример Dockerfile для PostgreSQL (используется официальный образ):

FROM postgres:latest

ENV POSTGRES\_DB=cinema\_db

ENV POSTGRES\_USER=cinema\_user

ENV POSTGRES\_PASSWORD=cinema\_password

EXPOSE 5432

Основной docker-compose.yml:

version: '3.8'

services:

db:

container\_name: o\_cinema

image: postgres:latest

restart: always

environment:

POSTGRES\_DB: cinema\_db

POSTGRES\_USER: cinema\_user

POSTGRES\_PASSWORD: cinema\_password

ports:

- "5438:5432"

volumes:

- cinema\_volume:/var/lib/postgresql/data

web:

build:

context: ../backend\_base/myproject

container\_name: cinema\_web

command: python manage.py runserver 0.0.0.0:8000

volumes:

- ../backend\_base/myproject:/app

ports:

- "8000:8000"

depends\_on:

- db

volumes:

cinema\_volume:

driver: local

driver\_opts:

type: 'none'

o: 'bind'

device: 'D:/GitHub/online\_cinema/database'

Содержимое .dockerignore:

venv/

\_\_pycache\_\_/

\*.pyc

\*.pyo

\*.pyd

db.sqlite3

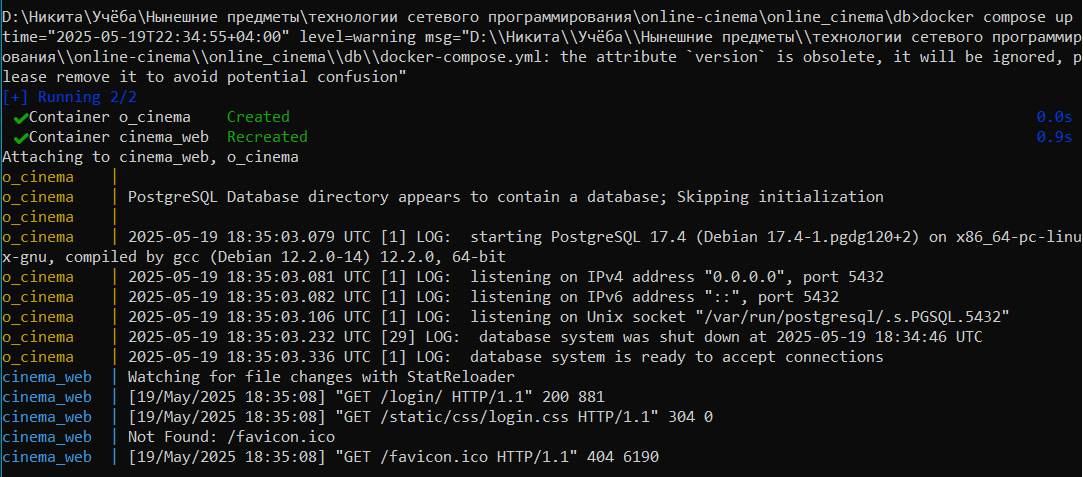
\*.log

**Тестирование работы контейнеров**

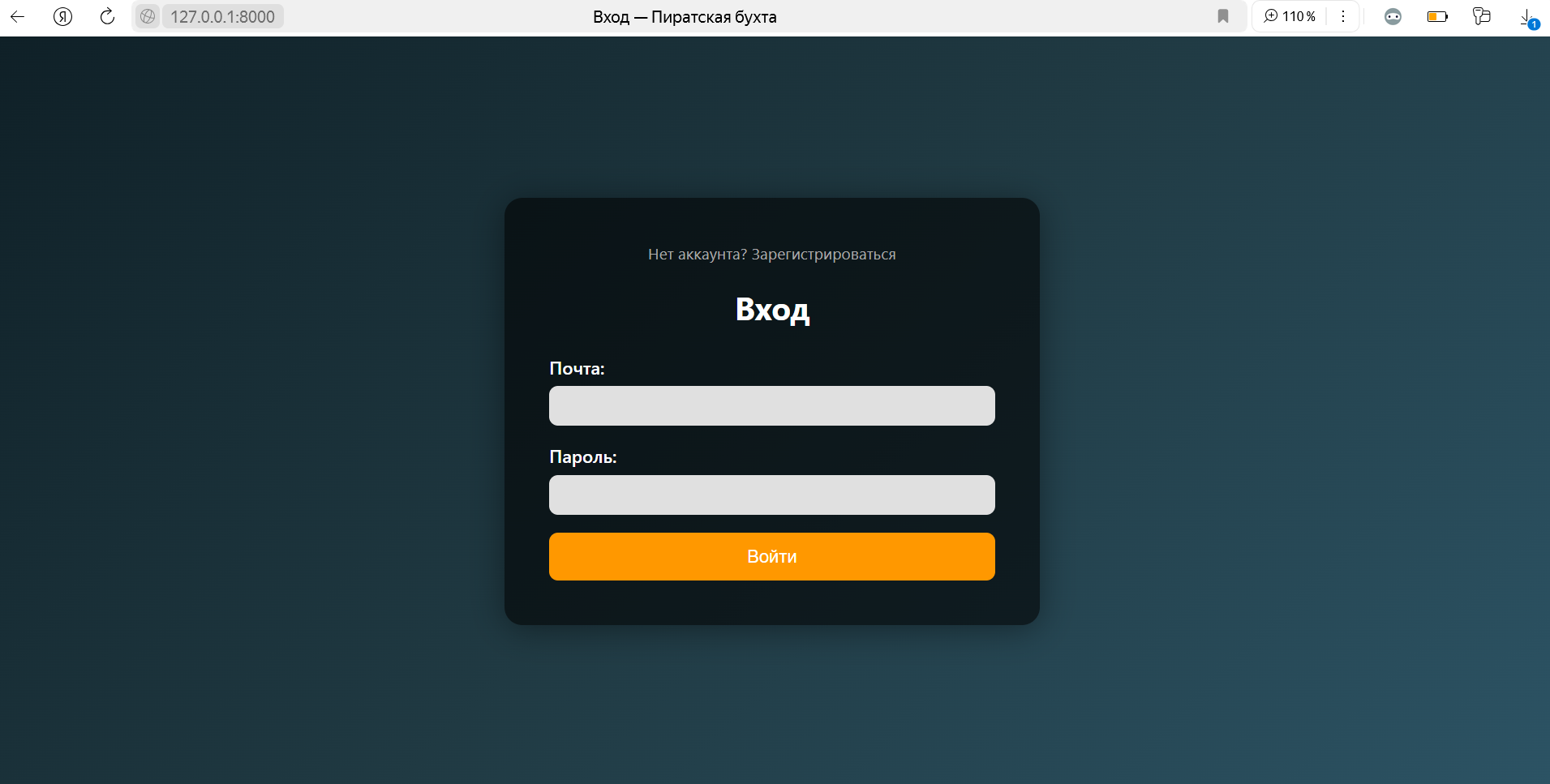
Контейнеры запускаются командой:

docker-compose up

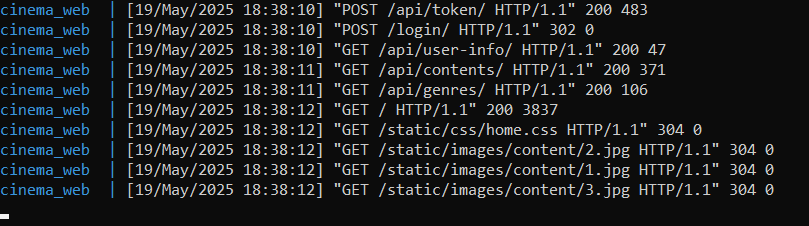
Скриншот с выполненной командой docker ps, показывающий запущенные контейнеры o\_cinema и cinema\_web.



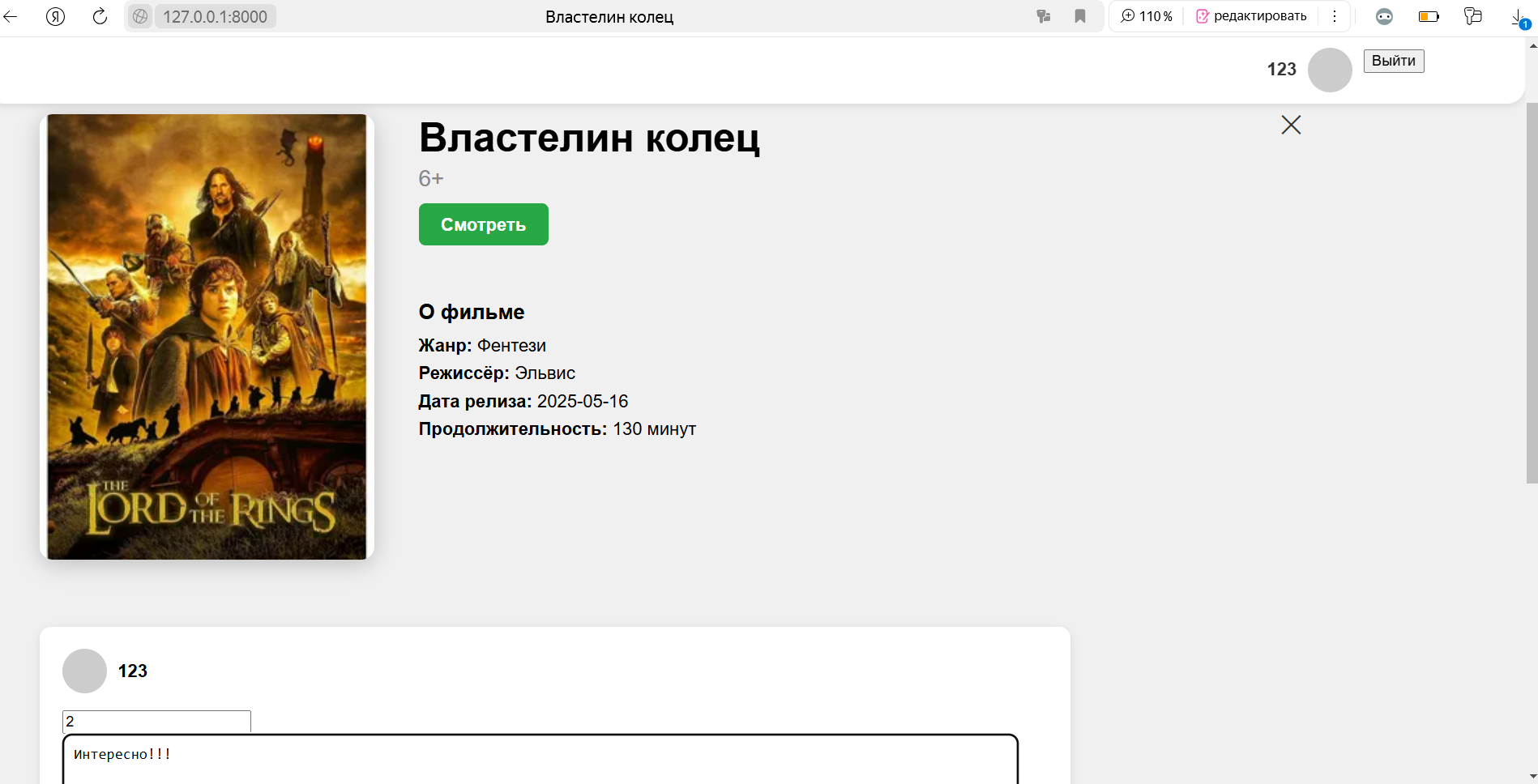
Скриншот открытой главной страницы фронтенда в браузере (по адресу http://localhost:8000/login/).



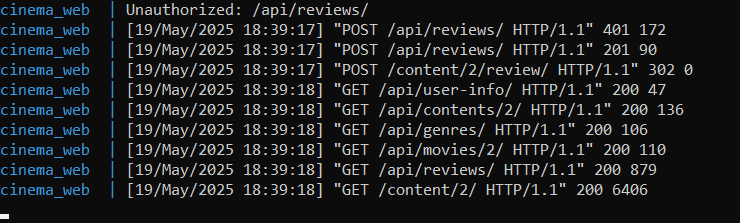
Скриншот ответа API на GET-запрос



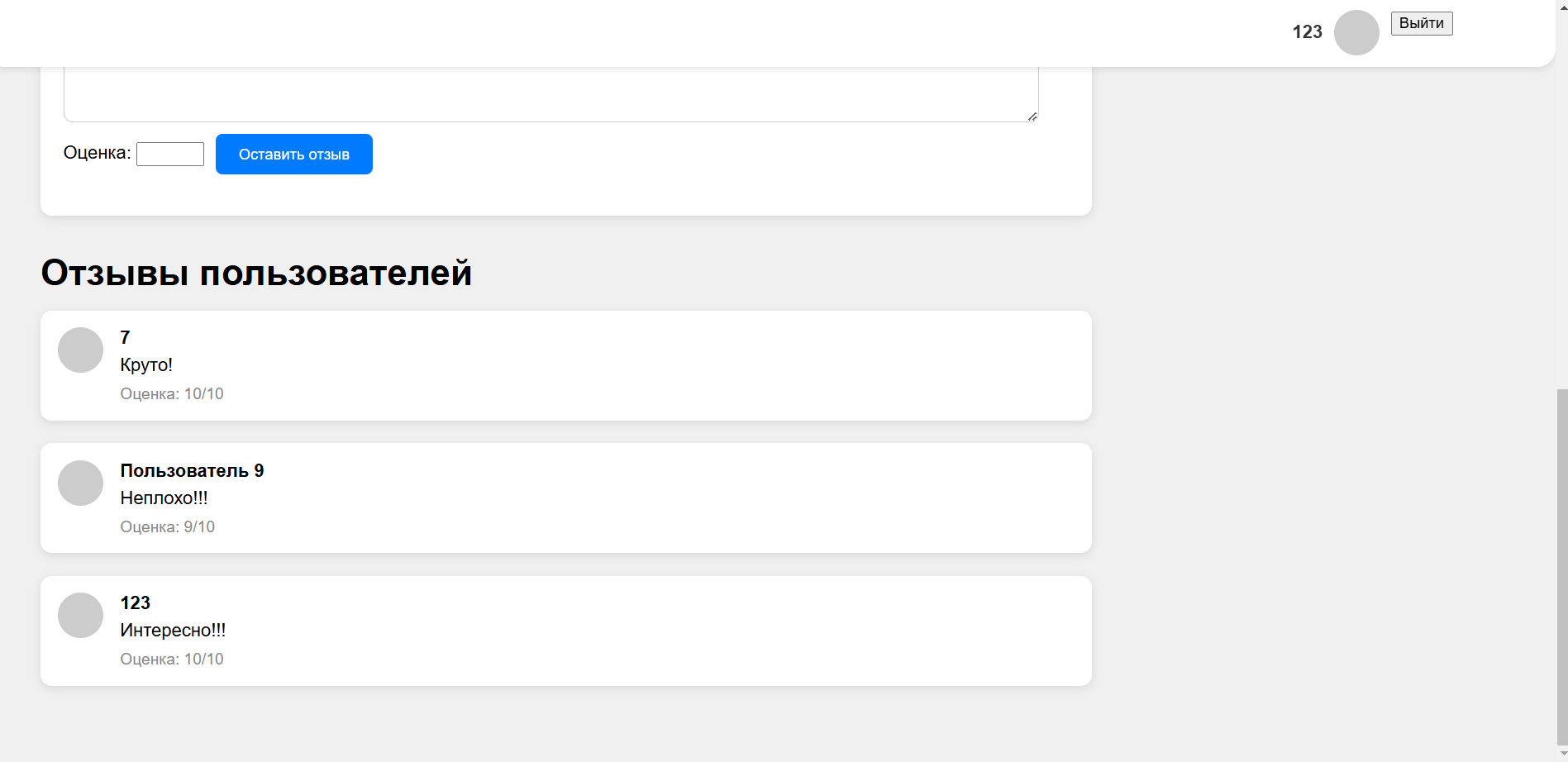
Скриншот работы с отзывами (добавление, просмотр).



Скриншот работы с отзывами (добавление, просмотр).



Скриншот работы с отзывами (добавление, просмотр).



**Описание API**

| **Метод** | **URL** | **Описание** | **Авторизация** |
| --- | --- | --- | --- |
| GET | /api/contents/ | Получение списка контента | Нет |
| GET | /api/contents/{id}/ | Получение конкретного контента | Нет |
| POST | /api/register/ | Регистрация пользователя | Нет |
| POST | /api/token/ | Получение JWT-токена | Нет |
| POST | /api/token/refresh/ | Обновление JWT-токена | Нет |
| POST | /api/reviews/ | Создание отзыва | Да |
| GET | /api/genres/ | Получение жанров | Нет |
| GET | /api/user-info/ | Получение информации о пользователе | Да |

Структура проекта

online\_cinema/

├── db/

│ ├── Dockerfile.txt

│ └── docker-compose.yml

├── backend\_base/

│ └── myproject/

│ ├── cinema/

│ │ ├── views\_api.py

│ │ ├── views\_frontend.py

│ │ ├── models.py

│ │ ├── serializers.py

│ │ └── templates/

│ ├── settings.py

│ ├── urls.py

│ └── manage.py

├── .dockerignore

├── requirements.txt

└── venv/

**Вывод**

В рамках лабораторной работы было:

* Завершено создание функционального приложения;
* Реализовано API и веб-интерфейс;
* Настроены контейнеры Docker для базы данных и backend-сервиса;
* Проверена связность всех компонентов;

Все цели и задачи лабораторной работы выполнены в полном объёме.