

**ZION – Movie Website**

**Plan**

## 1️⃣ Начальная настройка проекта

📌 **Что делаем?**

* Установка Django, DRF, PostgreSQL (если нужно).
* Настройка settings.py (CORS, базы данных, DRF).

📚 **Что изучаем?**

* Django: [Официальная документация](https://docs.djangoproject.com/en/5.0/intro/)
* Django Rest Framework (DRF): Официальный гайд
* PostgreSQL: Основы SQL на W3Schools
* CORS (разрешаем API для фронтенда): [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS)

### ****Шаг 1: Установка Django, DRF, PostgreSQL****

#### 1️⃣ Создание **виртуального окружения**

Перед установкой зависимостей рекомендуется создать **виртуальное окружение** для проекта:

python -m venv venv

Активируем его:

* **Windows:** venv\Scripts\activate
* **Linux/macOS:** source venv/bin/activate

#### 2️⃣ **Установка зависимостей**

pip install django djangorestframework psycopg2-binary django-cors-headers

**Что мы установили?**

* django – основной фреймворк.
* djangorestframework – API-фреймворк для Django.
* psycopg2-binary – драйвер PostgreSQL для Django.
* django-cors-headers – разрешает запросы к API с фронтенда.

#### 3️⃣ **Создание проекта и приложения**

django-admin startproject movie\_website .

cd movie\_website

django-admin startapp movies

## Шаг 2: Настройка *settings.py*

Открываем movie\_website/settings.py и вносим изменения.

#### 1️⃣ **Добавляем приложения**

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

# Наше приложение

'movies',

# Django Rest Framework

'rest\_framework',

# CORS для фронтенда

'corsheaders',

]

#### 2️⃣ **Настраиваем CORS (разрешаем fронтенду доступ к API)**

Добавляем в MIDDLEWARE:

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'corsheaders.middleware.CorsMiddleware', # Добавляем сюда

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.middleware.authentication.AuthenticationMiddleware',

'django.middleware.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

Добавляем настройку внизу settings.py:

CORS\_ALLOW\_ALL\_ORIGINS = True # Разрешаем запросы со всех доменов

(В будущем можно ограничить доступ только для конкретного фронтенда.)

## Шаг 3: Настройка базы данных (PostgreSQL)

#### 1️⃣ Создай файл **docker-compose.yml**

version: '3.1'

services:

db:

image: postgres

restart: always

ports:

- 5432:5432

volumes:

- postgres\_data:/var/lib/postgresql/data

environment:

POSTGRES\_PASSWORD: 4444

POSTGRES\_USER: root

POSTGRES\_DB: ZION

adminer:

image: adminer

restart: always

ports:

- 8080:8080

volumes:

postgres\_data:

#### **2️⃣ Запусти контейнеры**

В терминале (в той же папке, где docker-compose.yml) запусти:

docker-compose up -d

**Флаги:**

-d – запуск в фоне.

#### **3️⃣ Проверка работы**

После успешного запуска:

* **PostgreSQL** доступен на localhost:5432
* **Adminer** доступен в браузере на http://localhost:8080

**Вход в Adminer:**

* **System**: PostgreSQL
* **Server**: db (если не работает, попробуй localhost)
* **Username**: root
* **Password**: 4444
* **Database**: ZION

#### **Настроим Django для работы с контейнером**

Открой settings.py и в DATABASES пропиши:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'ZION',

'USER': 'root',

'PASSWORD': '4444',

'HOST': 'db’, # Используем имя контейнера

'PORT': '5432',

}

}

#### **Применяем миграции**

После изменения настроек запусти:

python manage.py migrate

Если все настроено правильно, миграции выполнятся успешно.

## Шаг 4: Применение миграций и запуск сервера

После настройки **применяем миграции**:

python manage.py migrate

Создаем **суперпользователя** (если нужен доступ в админку):

python manage.py createsuperuser

Запускаем сервер:

python manage.py runserver

Теперь API доступен на **http://127.0.0.1:8000/** 🚀

## 2️⃣ Разработка Backend (Django + DRF)

📌 **Что делаем?**

* Создаем модель Movie (ORM)
* Настраиваем сериализаторы (serializers.py)
* Создаем API:
  + /movies/ – список фильмов
  + /movies/{id}/ – детальная страница
  + /movies/search/?query=... – поиск фильмов

📚 **Что изучаем?**

* Django ORM (работа с моделями): [Django Docs](https://docs.djangoproject.com/en/5.0/topics/db/models/)
* DRF: работа с сериализаторами: DRF Serializers
* API-представления в DRF: Generic Views
* Тестирование API с Postman: Postman Docs

## **Шаг 1: Создаем модель** Movie

Открываем movies/models.py и добавляем:

python

CopyEdit

from django.db import models

class Movie(models.Model):

title = models.CharField(max\_length=255)

description = models.TextField()

release\_date = models.DateField()

poster\_url = models.URLField()

rating = models.FloatField(default=0.0)

def \_\_str\_\_(self):

return self.title

📌 **Объяснение полей:**

* title – название фильма.
* description – описание.
* release\_date – дата выхода.
* poster\_url – ссылка на постер.
* rating – рейтинг фильма.

### ****Применяем миграции****

bash

CopyEdit

python manage.py makemigrations movies

python manage.py migrate

## 🔹 **Шаг 2: Создаем сериализатор**

Открываем movies/serializers.py и добавляем:

python

CopyEdit

from rest\_framework import serializers

from .models import Movie

class MovieSerializer(serializers.ModelSerializer):

class Meta:

model = Movie

fields = '\_\_all\_\_'

📌 **Объяснение:**

* MovieSerializer преобразует объект Movie в JSON.

## 🔹 **Шаг 3: API Views**

Открываем movies/views.py и добавляем:

python

CopyEdit

from rest\_framework import generics

from rest\_framework.response import Response

from django.db.models import Q

from .models import Movie

from .serializers import MovieSerializer

class MovieListAPIView(generics.ListAPIView):

queryset = Movie.objects.all()

serializer\_class = MovieSerializer

class MovieDetailAPIView(generics.RetrieveAPIView):

queryset = Movie.objects.all()

serializer\_class = MovieSerializer

class MovieSearchAPIView(generics.ListAPIView):

serializer\_class = MovieSerializer

def get\_queryset(self):

query = self.request.query\_params.get('query', '')

return Movie.objects.filter(Q(title\_\_icontains=query) | Q(description\_\_icontains=query))

📌 **Что сделали?**

* MovieListAPIView – возвращает список фильмов (/movies/).
* MovieDetailAPIView – возвращает фильм по id (/movies/{id}/).
* MovieSearchAPIView – ищет фильмы по названию или описанию (/movies/search/?query=...).

## 🔹 **Шаг 4: Настроим маршруты**

Открываем movies/urls.py и добавляем:

python

CopyEdit

from django.urls import path

from .views import MovieListAPIView, MovieDetailAPIView, MovieSearchAPIView

urlpatterns = [

path('movies/', MovieListAPIView.as\_view(), name='movie-list'),

path('movies/<int:pk>/', MovieDetailAPIView.as\_view(), name='movie-detail'),

path('movies/search/', MovieSearchAPIView.as\_view(), name='movie-search'),

]

Добавляем в movie\_website/urls.py:

python

CopyEdit

from django.urls import path, include

urlpatterns = [

path('api/', include('movies.urls')),

]

## 🔹 **Шаг 5: Тестируем API**

Запускаем сервер:

bash

CopyEdit

python manage.py runserver

Открываем **Postman** или браузер и тестируем:

✅ **Получить список фильмов**:  
🔗 GET http://127.0.0.1:8000/api/movies/

✅ **Получить фильм по id**:  
🔗 GET http://127.0.0.1:8000/api/movies/1/

✅ **Поиск фильмов**:  
🔗 GET http://127.0.0.1:8000/api/movies/search/?query=Matrix

## 3️⃣ Интеграция с TMDB API

📌 **Что делаем?**

* Регистрируем API-ключ на [TMDB](https://www.themoviedb.org/)
* Создаем функцию для получения фильмов из TMDB
* Автоматически загружаем фильмы в базу

📚 **Что изучаем?**

* Работа с requests в Python: Real Python Guide
* API TMDB (как делать запросы): TMDB API Docs
* Хранение API-ключей в Django (переменные окружения): [Django-environ](https://django-environ.readthedocs.io/en/latest/)

## 4️⃣ Разработка Frontend (HTML, CSS, JavaScript)

📌 **Что делаем?**

* Разрабатываем базовый **layout** (шаблоны Django или чистый HTML)
* Верстаем страницы: Home, Movie List, Movie Details, Search
* Добавляем стили (Flexbox, Grid)

📚 **Что изучаем?**

* HTML + CSS: [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Learn/HTML)
* Flexbox & Grid (CSS): CSS Tricks – A Complete Guide
* Bootstrap (ускоряем верстку): Bootstrap Docs

### Detail

#### **1. Базовая структура проекта**

📌 **Цели**:

* Определить структуру проекта (файлы, папки).
* Настроить основные стили, базовый layout.
* Подготовить HTML-шаблоны.

📁 **Структура файлов**:

/zion-frontend

│── /assets

│ ├── /css

│ │ ├── styles.css

│ │ ├── animations.css

│ │ ├── theme.css

│ ├── /js

│ │ ├── script.js

│ │ ├── api.js

│ │ ├── effects.js

│ ├── /img

│ │ ├── arcade-screen.gif

│ │ ├── neon-bg.jpg

│ │ ├── glitch-logo.png

│── index.html

│── movies.html

│── details.html

│── search.html

#### **🔹 2. Главная страница (index.html)**

📌 **Цели**:

* Создать **заставку** с VHS-эффектами.
* Реализовать **глитч-анимацию** для заголовка.
* Добавить **кнопку "Press Start"** для перехода к списку фильмов.
* Боковые аркадные автоматы с VHS-экранами.
* Неоновая сетка на заднем фоне.

📌 **Технологии**:

* **HTML** – структура.
* **CSS (Flexbox, Grid, @keyframes)** – оформление.
* **JS (анимации, переходы, динамика)** – логика эффектов.

📌 **Элементы**: ✅ **Неоновая сетка** (движущиеся линии, перспективный эффект).  
✅ **Аркадные автоматы** (VHS-анимация).  
✅ **Глитч-заголовок** (эффект мигающего текста).  
✅ **Кнопка "Press Start"** (анимированный hover).

#### **🔹 3. Страница списка фильмов (movies.html)**

📌 **Цели**:

* Отобразить список фильмов **в виде VHS-карточек**.
* Подключить API для загрузки данных.
* Реализовать **глитч-наведение** на карточки.
* Добавить **фильтры и сортировку** по жанру/рейтингу.

📌 **Технологии**:

* **Fetch API** – получение данных.
* **CSS Grid** – расположение карточек.
* **JS (events, filter, sort)** – интерактивные элементы.

📌 **Элементы**:  
✅ **Карточки VHS** (тень, искажения, глитч).  
✅ **Жанры, рейтинг** (отображение в стиле 8-бит).  
✅ **Фильтры** (динамическое обновление списка).  
✅ **Ленивая загрузка** (загрузка новых фильмов при прокрутке).

#### **🔹 4. Детальная страница фильма (details.html)**

📌 **Цели**:

* Отобразить **подробную информацию** о фильме.
* Дать пользователю **возможность добавить в "Избранное"**.
* Сделать VHS-эффект для фона постера.
* Добавить **отображение рейтинга в 8-битном стиле**.

📌 **Технологии**:

* **Fetch API** – загрузка деталей фильма.
* **JS LocalStorage** – хранение избранного.
* **CSS Filters** – VHS-эффекты.

📌 **Элементы**:  
✅ **Постер фильма в VHS-рамке**.  
✅ **Название + жанры + рейтинг**.  
✅ **Описание с неоновым свечением**.  
✅ **Кнопка "Добавить в избранное"**.

#### **🔹 5. Страница поиска (search.html)**

📌 **Цели**:

* Реализовать **поиск фильмов** с **неоновой подсветкой ввода**.
* Подключить API /movies/search/?query=....
* Сделать **анимированную загрузку результатов**.

📌 **Технологии**:

* **Fetch API** – получение результатов.
* **CSS (animations, glow)** – стилизация.
* **JS (input event, debounce)** – обработка ввода.

📌 **Элементы**:  
✅ **Поле поиска с неоновым свечением**.  
✅ **Результаты в VHS-карточках**.  
✅ **Глитч-анимация при загрузке**.

#### **🔹 6. API-интеграция**

📌 **Цели**:

* Подключение к API /movies/, /movies/{id}/, /movies/search/.
* Реализация загрузки данных **с плавными эффектами**.

📌 **Технологии**:

* **Fetch API + async/await**.
* **JS Promises**.

📌 **Элементы**:  
✅ **Отображение данных из API**.  
✅ **Обновление списка фильмов без перезагрузки**.  
✅ **Хранение избранного в LocalStorage**.

#### **🔹 7. Анимации и эффекты**

📌 **Цели**:

* Добавить **глитч-анимации** (текст, кнопки).
* Реализовать **VHS-сканлайны**.
* **Неоновый hover-эффект** для кнопок.
* **Анимированные переходы между страницами**.

📌 **Технологии**:

* **CSS Keyframes**.
* **JS Animations**.

📌 **Элементы**:  
✅ **Глитч-заголовки** (смещение текста в RGB).  
✅ **Неоновая кнопка (hover + box-shadow)**.  
✅ **VHS-сканлайны (filter + opacity)**.

#### **🔹 8. Завершающий этап (Тестирование и Деплой)**

📌 **Цели**:

* Оптимизация стилей и JS-кода.
* Тестирование UI/UX.
* Деплой на **Netlify/GitHub Pages**.

📌 **Элементы**:  
✅ **Использование lazy-loading** для фильмов.  
✅ **Минификация CSS и JS**.  
✅ **Хостинг проекта в открытый доступ**.

#### **🚀 Итог: Поэтапный процесс разработки**

1️⃣ **Настройка проекта** – структура, базовые стили.  
2️⃣ **Главная страница** – неоновая сетка, аркады, кнопка "Press Start".  
3️⃣ **Список фильмов** – VHS-карточки, сортировка, API-запросы.  
4️⃣ **Детальная страница фильма** – постер, рейтинг, описание, избранное.  
5️⃣ **Поиск** – динамический ввод, подсветка, обновление списка.  
6️⃣ **API-интеграция** – получение данных, обновление интерфейса.  
7️⃣ **Анимации и эффекты** – глитч, VHS, неон.  
8️⃣ **Финал** – тестирование, оптимизация, деплой.

## 5️⃣ Подключение фронтенда к API (Fetch API)

📌 **Что делаем?**

* Используем Fetch API/AJAX для запросов к бэкенду
* Отображаем фильмы, полученные через API
* Реализуем поиск фильмов

📚 **Что изучаем?**

* Fetch API (основы работы с API на фронтенде): [MDN Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
* Работа с async/await в JavaScript: [MDN Async/Await](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Promises)
* Отображение данных на странице (JS DOM): [MDN Manipulating the DOM](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model)

## 6️⃣ Аутентификация пользователей (если требуется)

📌 **Что делаем?**

* Настраиваем Django Authentication
* Добавляем JWT (если API)
* Реализуем регистрацию/вход через Django

📚 **Что изучаем?**

* Django User Authentication: [Django Docs](https://docs.djangoproject.com/en/5.0/topics/auth/)
* JWT в Django: [DRF Simple JWT](https://django-rest-framework-simplejwt.readthedocs.io/en/latest/)
* OAuth2 (если требуется авторизация через Google/GitHub): [OAuth2 in Django](https://django-allauth.readthedocs.io/en/latest/)

## 7️⃣ CRUD-операции (если нужно)

📌 **Что делаем?**

* Добавляем возможность добавлять/удалять фильмы
* Реализуем избранное (добавление фильмов в "любимые")

📚 **Что изучаем?**

* Django Forms (создание форм): [Django Forms Docs](https://docs.djangoproject.com/en/5.0/topics/forms/)
* DRF API для CRUD-операций: DRF ViewSets
* JavaScript LocalStorage (если избранное без бэка): [MDN LocalStorage](https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage)

## 8️⃣ Деплой проекта

📌 **Что делаем?**

* Разворачиваем на **Railway/Vercel/Render** (бэкенд)
* Фронтенд на **Netlify/GitHub Pages**
* Оптимизируем статику

## 📚 Что изучаем?

* Деплой Django на Railway: Railway Docs
* Docker для Django (если нужно): Docker + Django Tutorial
* CI/CD (GitHub Actions): [GitHub Actions Docs](https://docs.github.com/en/actions)