

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное учреждение высшего  
образования «Самарский национальный исследовательский университет имени  
академика С.П. Королева»  
(Самарский университет)

Естественнонаучный институт  
Механико-математический факультет

Кафедра информатики и вычислительной  
математики Направление подготовки  
02.03.03 Математическое обеспечение и  
администрирование информационных  
систем  
Направленность (профиль) «Разработка и  
администрирование информационных  
систем»

Курсовая работа  
по дисциплине «Базы данных и СУБД»

**Разработка информационной системы «Онлайн-Кинотеатр»**

Выполнил студент курса 3 группы  
4345-020303D Ильметов Семён  
Сергеевич

---

Научный руководитель к. ф.-м. н.,  
Семенова И.В.

---

Работа защищена  
«\_\_\_» \_\_\_\_\_ 2024 г.  
Оценка \_\_\_\_\_  
зав. кафедрой ИиВМ  
д. ф.-м. н., профессор  
Степанов А.Н.

---

Самара 2024

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	4
1 Описание и анализ предметной области.....	6
1.1 Анализ предметной области «Онлайн-кинотеатр».....	6
1.2 Обзор систем-аналогов.....	7
1.2.1 Информационная система «Кинопоиск».....	7
1.2.2 Информационная система «Okko».....	8
1.4 Функциональные и нефункциональные требования к информационной системе.....	10
2 Проектирование и реализация базы данных «Онлайн-кинотеатр» .....	13
2.1 Логическая модель базы данных .....	13
2.2 Нормализация реляционных отношений.....	14
2.3 Обоснование выбора СУБД .....	15
2.4 Физическая модель базы данных.....	16
3 Проектирование и реализация информационной системы «Онлайн-кинотеатр» .....	22
3.1 Клиент-серверная архитектура информационной системы.....	22
3.2 Диаграмма вариантов использования .....	23
3.3 Диаграмма последовательности .....	25
3.4 Выбор средств реализации.....	26
3.5 Реализация серверной части .....	27
3.6 Реализация клиентской части .....	31
3.7 Тестирование информационной системы.....	36
3.7.1 Модульное тестирование.....	36
3.7.2 Нагрузочное тестирование .....	37
3.7.3 Тестирование кроссбраузерности .....	38
ЗАКЛЮЧЕНИЕ.....	41
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	42
ПРИЛОЖЕНИЕ А.....	44

ПРИЛОЖЕНИЕ Б .....	53
ПРИЛОЖЕНИЕ В .....	64
ПРИЛОЖЕНИЕ Г .....	67

## ВВЕДЕНИЕ

В настоящее время информационные технологии внедряются во все большее число отраслей человеческой деятельности. Они необходимы и полезны не только в производственных, технических, экономических сферах, но и для поиска информации о том, как и где провести свой досуг.

Человек сталкивается с ситуациями, когда ему необходимо среди большого числа фильмов и сериалов, доступных в интернете, выбрать то, что является подходящим по требуемым критериям или настроению, будь то комедия для веселого вечера или драма для глубоких размышлений. Учитывая степень развития информационных технологий, человек может производить поиск информации о фильмах и сериалах на различных сайтах и платформах. Однако гораздо удобнее иметь единую систему, содержащую информацию обо всех доступных фильмах и сериалах в одном месте - онлайн-кинотеатр.

Таким образом, актуальной является разработка информационной системы «Онлайн-кинотеатр», которая позволит пользователю удобно и быстро выполнить необходимый поиск, учитывая все его запросы, а также будет содержать всю актуальную и подробную информацию о каждом фильме.

Целью курсовой работы в рамках освоения компетенций ОПК-3, ОПК-4 и ОПК-5 является разработка информационной системы «Онлайн-кинотеатр», которая позволит пользователю осуществлять поиск заведения, удовлетворяющего заданным критериям.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) Провести анализ предметной области.
- 2) Построить информационную модель данных (ОПК-3, ОПК-4).
- 3) Изучить существующие системы-аналоги.
- 4) Сформулировать функциональные и нефункциональные требования в системе (ОПК-3).
- 5) Построить логическую модель данных, провести ее нормализацию (ОПК-3, ОПК-4).
- 6) Выбрать СУБД (ОПК-5).
- 7) Построить физическую модель данных (ОПК-3, ОПК-4).
- 8) Реализовать базу данных в выбранной СУБД (ОПК-3, ОПК-5).
- 9) Спроектировать приложение для работы с созданной базой данных (ОПК-3, ОПК-4).
- 10) Выбрать средства реализации приложения для работы с созданной базой данных (ОПК-5).
- 11) Реализовать приложение для работы с созданной базой данных (ОПК-3).

- 12) Провести тестирование реализованной системы.
- 13) Провести проверку работоспособности реализованной системы, оценить полученные результаты. (ОПК-3, ОПК-5).
- 14) Разработать инструкцию пользователя для созданной системы (ОПК-4).

## 1 Описание и анализ предметной области

### 1.1 Анализ предметной области «Онлайн-кинотеатр»

Современное информационное общество стремительно развивается, при этом удовлетворение потребностей пользователей в разнообразных развлекательных контенттах становится все более актуальным. В данном контексте, создание инновационных платформ для онлайн-кинопросмотра приобретает особую значимость. Информационная система онлайн-кинотеатр представляет собой ответ на растущий запрос к удобному, доступному и персонализированному потреблению киноконтента.

Определим схему взаимодействия пользователя при походе в обычный кинотеатр. Любой нормальный пользователь для начала должен перейти на сайт кинотеатра и найти подходящий ему фильм. Благо сайты кинотеатров выглядят более-менее современно, а иначе пришлось бы ещё разбираться как с ним взаимодействовать. После выбора фильма из маленького списка возможных, пользователь должен выбрать время, место, ряд и учесть ещё пару десятков факторов, таких как: “Как добраться до кинотеатра?”, “Брать с собой еду или нет?” и др. В конце концов может оказаться, что пользователь не успевает на сеанс фильма, а ещё хуже если он оплатил билеты. Так же может оказаться что пользователю не понравится фильм, и он потратит зря своё время на выбор фильма, поездку до кинотеатра и т.д.

Исходя из описания работы оффлайн кинотеатра можно выделить следующие проблемы:

- Цены на билеты могут иметь высокую стоимость, завися от фильма, мест, типа сеанса и т.п
- Шум соседей по залу может сильно отвлекать и мешать наслаждаться просмотром.
- Нахождение в кинотеатре предполагает нахождение в комнате с звукоизоляцией. и в случае чрезвычайных ситуаций, таких как пожар, теракт, зрители могут оказаться в ловушке.
- Просмотр ограничивается одним разом. Для просмотра этого же фильма необходимо заново оплатить билет.

Онлайн-кинотеатр устраняет данные недостатки оффлайн-кинотеатров и предлагает множество технологичных решений, которые облегчат пользователю просмотр фильмов. Основной проблемой оффлайн-кинотеатра и достоинством онлайн-кинотеатра является то, что он работает постоянно, и пользователь может посмотреть фильм в любое удобное для него время. Пользователю предлагается для просмотра любой фильм, хранящийся в

системе. Для просмотра пользователю необходимо авторизоваться. Для этого пользователь должен ввести логин в качестве почты или телефона и пароль. В случае, если пользователь попадает на сайт впервые, система предлагает ему зарегистрироваться. Все данные пользователей, в целях безопасности, согласно Федеральному закон "О персональных данных" от 27.07.2006 N 152-ФЗ хранятся в зашифрованном виде. После авторизации в системе перед пользователем предстаёт выбор. На главном окне представлены рекомендованные фильмы, сериалы и другие категории. Пользователь может начать смотреть их или перейти на отдельную страницу с интересующей категорией. Основным фактором показа фильма в онлайн кинотеатре является лицензия, описанная в статье 5.1. Прокатное удостоверение на фильм в федеральном законе от 22.08.1996 N 126-ФЗ (ред. от 19.10.2023) "О государственной поддержке кинематографии Российской Федерации". Также пользователь может найти нужный ему фильм для просмотра, благодаря поисковой панели. Для этого пользователь в графе поиска выбирает необходимые для поиска параметры, такие как: название, жанр, дата выхода, страна, год и т.д. Далее система предложит пользователю варианты, подходящие по критерию, для выбора. Так же система предлагает оценивать и оставлять отзывы фильмы, сериалы и т.д. Для этого пользователь должен перейти на страницу с просматриваемым контентом и оставить оценку или написать отзыв к фильму. Данные отзывы могут видеть все пользователи системы

Таким же образом, на основе анализа работы онлайн-кинотеатра, можно выделить следующие недостатки:

1. Отсутствие или наличие временной лицензии на прокат некоторых фильмов.
2. Для просмотра интересующих фильмов, как правило, требуется подписка.
3. Из-за проблем с интернетом зритель может не полноценно насладиться просмотром.

## 1.2 Обзор систем-аналогов

На сегодняшний день существует немало информационных систем, позволяющих человеку осуществлять поиск заведения по различным критериям, а также в выбранном заведении бронировать столик. Среди них можно выделить: «Кинопоиск», «Окко».

### 1.2.1 Информационная система «Кинопоиск»

Кинопоиск предоставляет информацию о кинофильмах, телесериалах, в том числе кадры, трейлеры, постеры, обои, а также данные о персонах, связанных с кино и

телепроизводством: актёрах, режиссёрах, продюсерах, сценаристах, операторах, композиторах, художниках и монтажёрах. Посетители могут ставить оценки фильмам и сериалам, добавлять их в ожидаемые, писать рецензии, покупать билеты в кинотеатры на сайте с компьютера или мобильных устройств. Важной особенностью, которую стоит подчеркнуть является функция скачивания фильма для просмотра в оффлайн-режиме, что решает третью проблему, но только на мобильных устройствах. Имеется онлайн-кинотеатр с фильмами и сериалами по подписке «Яндекс. Плюс» или за отдельную плату. «Кинопоиск» входит в топ-25 самых посещаемых площадок рунета, месячная аудитория сервиса составляет более 20 миллионов человек.

Не смотря на популярность данного сервиса, Кинопоиск имеет ряд недостатков:

- Дорогостоящая подписка не даёт полноценно насладиться просмотром фильма.
- Система рекомендаций нацелена на продвижение отечественных проектов, а не на то, что может понравиться пользователю.
- В связи с отзывом лицензий на фильмы в виду со случившимися обстоятельствами в стране, у пользователя пропадает возможность для просмотра мировых бестселлеров.

### 1.2.2 Информационная система «Okko»

Okko предоставляет такие же возможности для просмотра, что и Кинопоиск, однако имеет некоторые отличия. Okko представляет старые и новые фильмы и сериалы Киностудии Крупного плана, Мосфильм, Союзмультфильм, Киностудия Горького и ВГТРК. Имеет договор с американской группой ViacomCBS на покупку 3 тысяч наименований, включая авторские права по дистрибуции брендов Nickelodeon, Nick Jr., Comedy Central, Channel 5 и MTV. В рамках договора сервис получил эксклюзивные права на цифровые премьеры ViacomCBS, при этом часть фильмотеки доступна пользователям сервиса до показа в телевизионном эфире. Также имеет эксклюзивные права на показ кинопремии «Оскар и музыкальной премии «Грэмми». Тем не менее данный сервис тоже имеет ряд недостатков:

Просмотр фильмов без подписки не предоставляется пользователю

Сервис путём зрительного обмана при оформлении подписки вводит пользователей в заблуждение.

### 1.3 Информационная модель «Онлайн-кинотеатр»



На основании проведенного анализа предметной области была построена информационная модель в нотации Питера Чена. Такая нотация применяется для графической интерпретации предметной области в терминах сущностей и связей, иллюстрирующих ее абстрактное представление на логическом и концептуальном уровнях. Предложенная Питером Ченом графическая нотация ERD предполагает свой набор правил для изображения различных элементов [4]. Независимая сущность отображается на диаграммах прямоугольником с одинарной рамкой, зависимая – с двойной. Атрибуты отображаются за пределами графического обозначения сущности в виде эллипсов, связанных одинарной линией с ним. Атрибуты, входящие в первичный ключ сущности, выделяются подчеркиванием имени. Связь между сущностями показывается в виде ромба с указанием имени связи внутри него. При этом, если ромб имеет двойную рамку, то связь – идентифицирующая, одинарную – неидентифицирующая. Мощность (кардинальность) связи обозначается числами или буквами.

Для предметной области «Онлайн-кинотеатр» были выделены следующие основные сущности:

1) Фильм – сущность, определяющая информацию о конкретном фильме. Сущность должна иметь такие атрибуты, как название, жанр, год, дата выхода, отзывы, категория, бюджет, сборы, рейтинг, постер, фотографии, длительность и страны;

2) Пользователь – сущность, определяющая информацию о человеке, желающем посмотреть фильм. Атрибутами данной сущности являются ФИО, пол, адрес электронной почты и фото;

Далее были выявлены следующие связи сущностей.

– сущности «Пользователи» и «Фильм» связаны между собой связью «многие ко многим»: один пользователь может выбрать несколько фильмов, один фильм может быть выбрано несколькими пользователями;

– сущность «Фильм» связана с атрибутом отзывы связью «один ко многим»: один фильм может содержать несколько отзывов;

– сущность «Фильм» связана с атрибутом страна связью «многие ко многим»: один фильм может быть создан разными странами, и каждая страна может участвовать в создании фильмов;

– сущность «Фильм» связана с атрибутом съёмочная группа связью «один ко многим»: один фильм может содержать несколько участников съёмочной группы;

На рисунке 2.1 показана созданная информационная модель предметной области «Онлайн-кинотеатр».

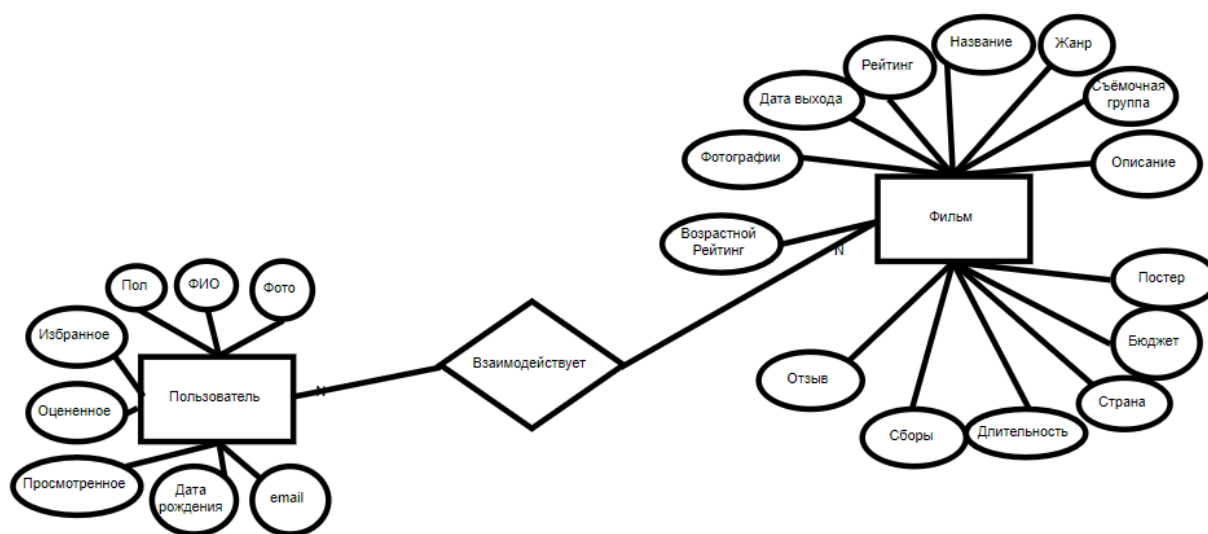


Рисунок 2.1 – Информационная модель предметной области

#### 1.4 Функциональные и нефункциональные требования к информационной системе

В результате анализа предметной области были сформулированы функциональные и нефункциональные требования, которым должна удовлетворять разрабатываемая информационная система «Онлайн-кинотеатр». У информационной системы должно быть 4 уровня доступа: незарегистрированный пользователь, зарегистрированный пользователь, , модератор.

Функциональные требования:

1. Возможность регистрации и аутентификации. Пользователь имеет возможность регистрации учётной записи в ИС. Для регистрации пользователю необходимо ввести логин, ФИО, адрес почты и указать пароль. Регистрация возможна только в случае, если на данный адрес или логин не была произведена регистрация. В противном случае система выдаёт ошибку. При аутентификации пользователь указывает адрес почты или номер телефона и пароль. Система проверяет соответствие хеша вводимого пароля с хешом пароля, записанным в системе. Аутентификация возможна только на созданную учётную запись. В случае несоответствия система выдаёт ошибку.
2. Возможность хранения, добавления, изменения основной информации о фильмах, а также о людях, участвующих в данных фильмах.

3. Возможность хранения информации о пользователях. Все конфиденциальные данные о пользователях хранятся в зашифрованном виде.

4. Авторизация:

При входе система определяет возможности, доступные пользователю, в зависимости от выданной ему роли.

Виды ролей:

1. Гость (незарегистрированный пользователь). Роль гостя присваивается любому пользователю, который использует ИС. Имеет минимальные права, такие как:
  - Просмотр информации о фильмах, об отзывах пользователей, участников съёмочных групп
  - Возможность регистрации, вход в учётную запись
  - Поиск фильмов, участников съёмочных групп
2. Зарегистрированный пользователь — пользователь, успешно вошедший в систему. Имеет стандартный набор возможностей, необходимых для комфортного использования ИС, такие как:
  - Просмотр, добавление, изменение отзывов, оценок о фильмах.
  - Просмотр фильмов, участников съёмочных групп.
  - Добавление фильма в категорию избранное, просмотренное.
  - Поиск фильмов, участников съёмочных групп
  - Просмотр, редактирование, удаление персональных данных.
  - Возможность выхода из ИС.
3. Модератор. Модератор имеет повышенный набор возможностей, необходимых для администрирования ИС, такие как:
  - Все возможности зарегистрированного пользователя.
  - Просмотр, добавление, редактирование, удаление отзывов, о фильмах.
  - Просмотр, редактирование, удаление персональных данных пользователей.
  - Блокирование возможностей оставления отзывов/аккаунта любого пользователя, кроме модератора.
  - Удаление/изменение любого отзыва
  - Добавление/изменение информации о фильме.

Нефункциональные требования:

Требования к реализации:

1. С представляет собой клиент-серверное веб-приложение;
2. Для реализации ИС используется реляционная база данных;
3. В качестве СУБД используется PostgreSQL 16;

4. ИС должна поддерживать кроссплатформенной и кроссбраузерность;
5. В качестве среды разработки используется Visual Studio Code 1.86.1, фреймворк Django 5.0.3;

Требования к совместимости:

1. пользовательские интерфейсы должны корректно и одинаково отображаться в следующих браузерах: Google Chrome 96+, Yandex-браузер 22.1.0.2510+, Microsoft Edge 113.0.1774.50.

Требования к безопасности:

1. ИС должна предусматривать безопасное хранение конфиденциальной информации о пользователя.
2. ИС должна предусматривать идентификацию, аутентификацию и авторизацию.
3. ИС должна предусматривать разграничение прав для пользователя и модератора;

## 2 Проектирование и реализация базы данных «Онлайн-кинотеатр»

### 2.1 Логическая модель базы данных

Опираясь на ER-диаграмму и используя нотацию Мартина, была построена логическая модель базы данных информационной системы «Онлайн-кинотеатра». На рисунке 2.2 показана построенная логическая модель базы данных проектируемой системы. При проектировании были учтены особенности реляционной модели баз данных, каждой сущности было сопоставлено одноименное отношение, в котором определены потенциальные и внешние ключи. Таким образом, созданная логическая модель базы данных содержит следующие отношения:

- а) сущности «Пользователь» было сопоставлено отношение «Пользователь» с полями: номер пользователя, являющееся потенциальным ключом, имя, фамилия, отчество, логин, пароль, дата рождения, адрес электронной почты, фото, статус блокировки чата;
- б) сущности «Фильм» было сопоставлено отношение «Фильм» с полями: номер фильма, название, слоган, описание, постер, дата выхода в мире и в России, бюджет, сбор, длительность, возрастной рейтинг, рейтинг МРАА, ссылка на трейлер, фильм. Отношение имеет потенциальный ключ – номер фильма;

В связи с тем, что некоторые сущности, присутствующие в ER-диаграмме, имеют связь «многие-ко-многим», необходимо добавить дополнительные отношения в логическую модель базы данных для разбиения такого типа связи на две связи типа «один-ко-многим». Таким образом, логическая модель базы данных содержит следующие отношения: «Пользователь», «Фильм», «Участник съёмочной группы», «Жанр», «Страна», «Избранное», «Просмотренное», «Отзыв», «Оценка», «Звёзды», «Должность», «Тип кухни», «Статус». «Фильм-Жанр», «Фильм-Страна», «Фильм группа».

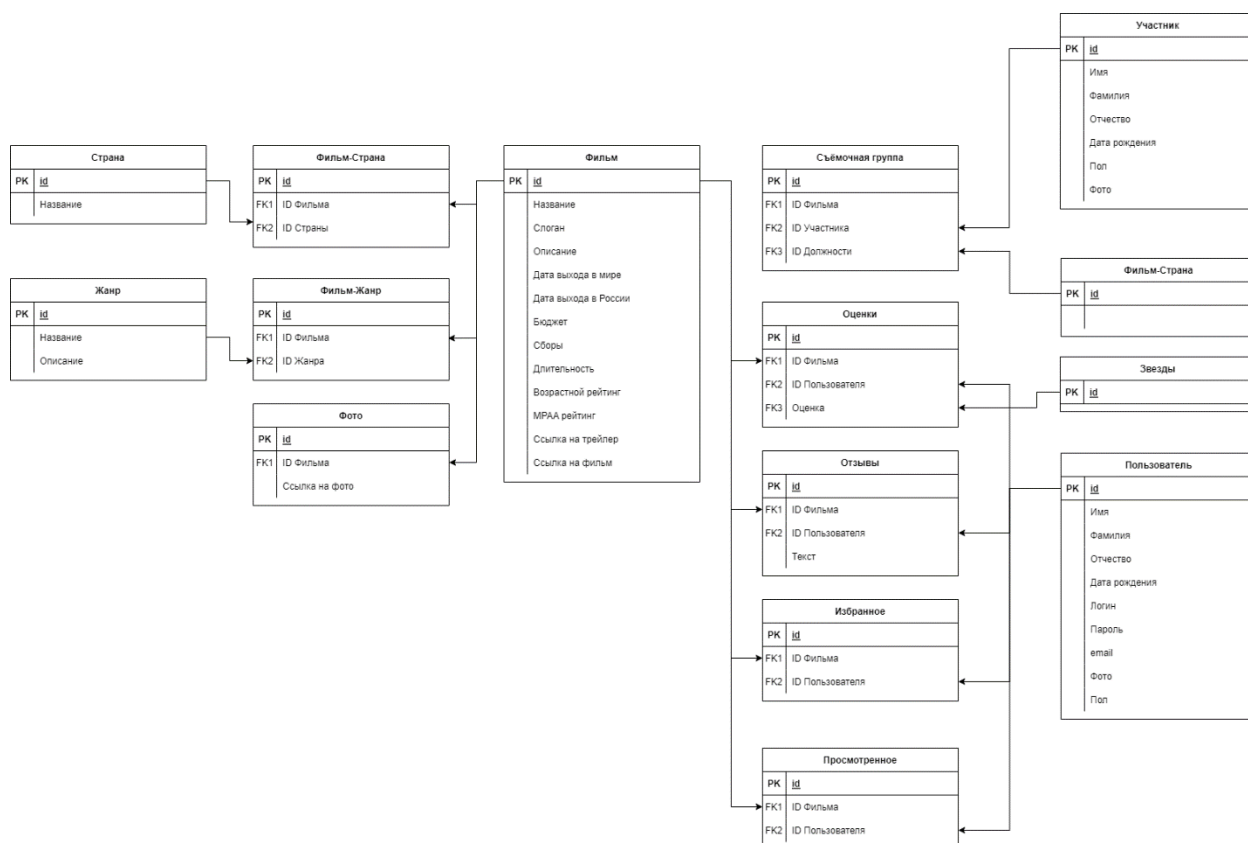


Рисунок 2.1 – Логическая модель базы данных

## 2.2 Нормализация реляционных отношений

Основной задачей, решаемой в процессе проектирования БД, является задача нормализации ее отношений. Процесс проектирования БД с использованием метода нормальных форм является итерационным и заключается в последовательном переводе отношений из первой нормальной формы в нормальные формы более высокого порядка по определенным правилам. Каждая следующая нормальная форма ограничивает определенный тип функциональных зависимостей, устраняет соответствующие аномалии при выполнении операций над отношениями БД и сохраняет свойства предшествующих нормальных форм. Если база данных находится как минимум в третьей нормальной форме, то она считается нормализованной. Отношение находится в первой нормальной форме тогда и только тогда, когда все его атрибуты являются простым (имеют единственное значение) [1].

Рассмотрим, как изначально представлялось отношение «Пользователь» и его поле «ФИО». Очевидно, что это поле состоит из трёх значений, а именно «Фамилия», «Имя» и «Отчество». Поэтому при приведении отношения к первой нормальной форме было реализовано разбиение поля «ФИО», результат представлен на рисунке 2.2.

Добавить фото

Рисунок 2.2 – Приведение отношения «Пользователь» к первой нормальной форме

После того, как все отношения приведены к первой нормальной форме, необходимо проверить условия второй нормальной формы и при необходимости привести к ней. Отношение находится во второй нормальной форме тогда и только тогда, когда оно находится в первой нормальной форме и каждый неключевой атрибут неприводимо зависит от потенциального ключа. При декомпозиции отношениям «Фильм», возникают связи «многие-ко-многим» с отношениями «Съёмочная группа», «Страна» и «Жанр», которые необходимо разбить на связи типа «один-ко-многим», добавив дополнительное отношение «Участник съёмочной группы», «Должность», «Фильм-Страна» и «Фильм-Жанр». Аналогичным образом в отношении «Пользователь» было выявлено 4 связи «многие-ко-многим» с отношением «Фильм», таким образом была произведена декомпозиция данного отношения и было добавлено 4 отношения, состоящее из 2 связей типа «один-ко-многим»: «Отзывы», «Просмотренное», «Оценки», «Избранное».

Добавить фото

Рисунок 2.3 – Приведение отношения «Фильм» ко второй нормальной форме

Добавить фото

Рисунок 2.4 – Приведение отношения «Пользователь» ко второй нормальной форме

Таким образом, условие второй нормальной формы для проектируемой базы данных выполняется. Отношение находится в третьей нормальной форме, когда оно находится во второй нормальной форме, и каждый неключевой атрибут нетранзитивно зависит от потенциального ключа [1]. Нетранзитивная зависимость – это зависимость, при которой неключевые атрибуты не зависят от значений других неключевых атрибутов. Поскольку для каждого отношения неключевые атрибуты не зависят друг от друга, но зависят от потенциального ключа, то все отношения проектируемой базы данных находятся в третьей нормальной форме..

### 2.3 Обоснование выбора СУБД

Выбор системы управления баз данных (СУБД) представляет собой сложную многопараметрическую задачу и является одним из важных этапов при разработке информационных систем на основе баз данных. В качестве СУБД для реализации проектируемой реляционной модели данных была выбрана PostgreSQL. Система управления базами данных PostgreSQL является свободной объектно-реляционной СУБД.

К преимуществам и функциональным возможностям выбранной СУБД относится:

- 1) надежность: надёжность СУБД PostgreSQL обеспечивается соответствием принципам ACID (атомарность, согласованность, изолированность, надежность), наличием Write Ahead Logging (WAL) — общепринятого механизма

протоколирования всех существующих транзакций. Сюда же стоит отнести и возможность восстановления базы данных Point in Time Recovery (PITR), репликацию, поддержку целостности данных на уровне схемы;

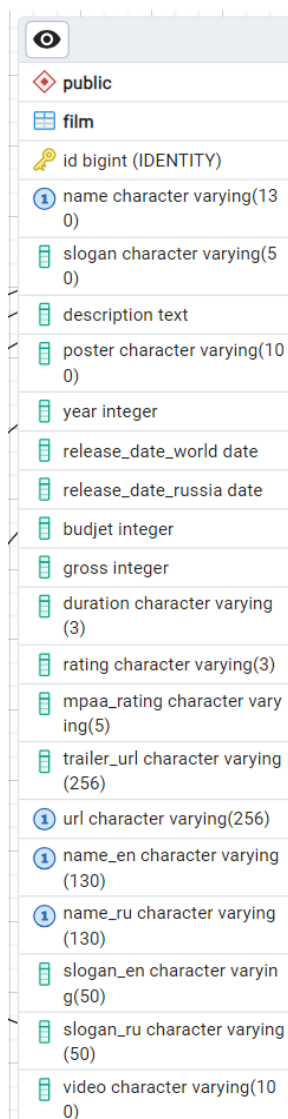
- 2) производительность: в СУБД PostgreSQL она основана на применении индексов, наличии гибкой системы блокировок и интеллектуального планировщика запросов, использовании системы управления буферами памяти и кэширования;
- 3) расширяемость;
- 4) поддержка SQL;
- 5) PostgreSQL поддерживает много разных типов и структур данных, в том числе сетевые адреса, данные в текстовом формате JSON и геометрические данные для координат геопозиций. Все эти форматы можно хранить и обрабатывать в СУБД.

Также стоит отметить, что данная СУБД не ограничивает максимальный размер баз данных, работает на большинстве распространенных операционных системах, таких как MacOS, Windows, BSD, Linux и поддерживает широкий набор языков программирования: Python, C/C++, Delphi, Erlang, Go, Java, JavaScript, Lisp, .Net, R, Tcl.

#### 2.4 Физическая модель базы данных

На основе логической модели данных и особенностей типов данных выбранной СУБД PostgreSQL была построена физическая модель базы данных. Отношения, определенные в логической модели, на этом этапе проектирования становятся таблицами. Рассмотрим отношение «Фильм» и соответствующую ему таблицу «Film», представленную на рисунке 2.4.





Attribute	Type	Constraints
id	bigint (IDENTITY)	Primary Key
name	character varying(130)	
slogan	character varying(50)	
description	text	
poster	character varying(100)	
year	integer	
release_date_world	date	
release_date_russia	date	
budget	integer	
gross	integer	
duration	character varying(3)	
rating	character varying(3)	
mpaa_rating	character varying(5)	
trailer_url	character varying(256)	
url	character varying(256)	
name_en	character varying(130)	
name_ru	character varying(130)	
slogan_en	character varying(50)	
slogan_ru	character varying(50)	
video	character varying(100)	

Рисунок 2.5 – Таблица «Film» в физической модели базы данных

Атрибут «id», являющийся потенциальным ключом, определен на домене «номер фильма», который описывается следующим образом:

- 1) Номер фильма;
- 2) Целочисленный тип;
- 3) BigInteger.

Атрибуты «name», «name\_ru», «name\_en», определен на домене «Наименование фильма», который описывается следующим образом:

- 1) Название фильма;
- 2) Текст переменной длины;
- 3) Varchar (130);
- 4) Атрибуты «name», «name\_ru» состоят из символов кириллицы, а атрибут «name\_en» состоит из символов латиницы.

Атрибут «budget» определен на домене «Бюджет», который описывается следующим образом:

- 1) Бюджет фильма;
- 2) Целочисленный тип;
- 3) Integer:
- 4) Записывается в долларах США.

Атрибут «gross» определен на домене «Сбор», который описывается следующим образом:

- 1) Сбор фильма;
- 2) Целочисленный тип;
- 3) Integer:
- 4) Записывается в долларах США.

Атрибут «url» определен на домене «Адрес фильма», который описывается следующим образом:

- 1) Адрес фильма;
- 2) Текст переменной длины;
- 3) Varchar(256):
- 4) Записывается латиницей

Атрибут «rating» определен на домене «Возрастной рейтинг фильма», который описывается следующим образом:

- 1) Рейтинг фильма;
- 2) Текст переменной длины;
- 3) Varchar(3):
- 4) Состоит из возраста и символа “+”

Атрибут «mpaa\_rating» определен на домене «МРАА рейтинг фильма», который описывается следующим образом:

- 1) МРАА рейтинг фильма;
- 2) Текст переменной длины;
- 3) Varchar(3):

Атрибуты «slogan», «slogan\_ru» и «slogan\_en» определены на домене «Слоган фильма», который описывается следующим образом:

- 1) Слоган фильма;
- 2) Текст переменной длины;
- 3) varchar (50);

- 4) Атрибуты «slogan», «slogan\_ru» состоят из символов кириллицы, а атрибут «slogan\_en» состоит из символов латиницы.

Атрибут «description», являющийся внешним ключом, определен на домене «Описание фильма» родительского отношения «Улица», который описывается следующим образом:

- 1) Описание фильма;
- 2) Текст переменной длины;
- 3) TEXT.

Атрибут «poster» определен на домене «постер фильма», который описывается следующим образом:

- 1) Постер фильма;
- 2) Текст переменной длины;
- 3) Varchar (10).
- 4) Представляет собой относительный путь к файлу в структуре проекта, начинается с media/films

Атрибут «year» определен на домене «Год выхода», который описывается следующим образом:

- 4) Год выхода;
- 5) Целочисленный тип;
- 6) Integer:
- 7) Год начинается может принимать значения начиная с 1980г по нынешний.

Атрибут «release\_date\_world» определен на домене «Дата выхода», который описывается следующим образом:

- 1) Дата выхода в мире;
- 2) Дата;
- 3) DATE.

Атрибут «release\_date\_russia» определен на домене «Дата выхода», который описывается следующим образом:

- 1) Дата выхода в России;
- 2) Дата;
- 3) DATE.

Атрибут «trailer\_url» определен на домене «Трейлер фильма», который описывается следующим образом:

- 1) Трейлер фильма;
- 2) Текст переменной длины;



```

budget integer,
gross integer,
duration character varying(3) COLLATE pg_catalog."default",
rating character varying(3) COLLATE pg_catalog."default",
mpaa_rating character varying(5) COLLATE pg_catalog."default",
trailer_url character varying(256) COLLATE pg_catalog."default",
url character varying(256) COLLATE pg_catalog."default",
name_en character varying(130) COLLATE pg_catalog."default",
name_ru character varying(130) COLLATE pg_catalog."default",
slogan_en character varying(50) COLLATE pg_catalog."default",
slogan_ru character varying(50) COLLATE pg_catalog."default",
video character varying(100) COLLATE pg_catalog."default",
CONSTRAINT film_pkey PRIMARY KEY (id),
CONSTRAINT film_name_en_key UNIQUE (name_en),
CONSTRAINT film_name_key UNIQUE (name),
CONSTRAINT film_name_ru_key UNIQUE (name_ru),
CONSTRAINT film_url_e02a1a2e_uniq UNIQUE (url)
);

```

### 3 Проектирование и реализация информационной системы «Онлайн-кинотеатр»

#### 3.1 Клиент-серверная архитектура информационной системы

Архитектура информационной системы - концепция, определяющая модель, структуру, выполняемые функции и взаимосвязь компонентов информационной системы.

На основе предъявляемых функциональных требований проектируемая информационная система представлена как веб-приложение. Веб-приложение – это клиент–серверное приложение, в котором клиент (веб-интерфейс) запускается в веб-браузере, а серверная часть запускается на веб-сервере [6]. Для информационной системы «Онлайн-кинотеатр» была выбрана трехуровневая архитектура, представленная на рисунке 3.1.

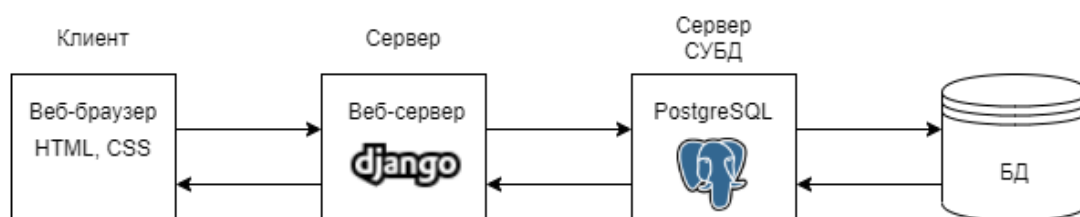


Рисунок 3.1 – Трехуровневая архитектура клиент-сервер

Трехуровневая архитектура клиент-сервер — это модель, в которой функции приложения распределены между тремя различными слоями, каждый из которых выполняет свои конкретные функции.

- 1) Клиентский уровень. Этот уровень представляет собой пользовательский интерфейс, который позволяет взаимодействовать с приложением. В проектируемой информационной системе клиентом является веб-браузер.
- 2) Серверный уровень. Этот уровень представляет собой службы и приложения, которые обрабатывают запросы от клиентского уровня и взаимодействуют с уровнем хранения данных. На серверном уровне происходит обработка бизнес-логики, а также доступ к данным и их обработка.
- 3) Уровень хранения данных. Данный уровень является нижним уровнем в архитектуре трехуровневой клиент-серверной модели. Данный уровень включает в себя хранение, доступ и обработку данных. На этом уровне данные хранятся в базе данных, и работа с данными осуществляется с помощью языков запросов, таких как SQL. Этот уровень обеспечивает постоянный доступ к данным, не зависимо от работоспособности уровней приложения и представления.

Таким образом, трехуровневая архитектура клиент-сервер позволяет разделять функционирование приложения на три уровня, что делает приложение более модульным, более эффективным и более легким в сопровождении и обновлении.

Преимущества трехуровневой архитектуры клиент-сервер:

- 1) Масштабируемость. Трехуровневая архитектура позволяет распределять нагрузку на различные уровни, что обеспечивает более эффективное использование ресурсов и масштабируемость приложения;
- 2) Безопасность. Разделение приложения на три уровня позволяет обеспечить более высокий уровень безопасности, так как каждый уровень может быть защищен от несанкционированного доступа;
- 3) Гибкость. Трехуровневая архитектура позволяет легко изменять и модифицировать отдельные уровни приложения без влияния на другие уровни;
- 4) Расширяемость. Трехуровневая архитектура позволяет легко добавлять новые функции и возможности в приложение, не затрагивая другие уровни;
- 5) Надежность. Разделение приложения на три уровня позволяет обеспечить более высокую надежность и устойчивость к сбоям, так как каждый уровень может быть отказоустойчивым и иметь механизмы восстановления после сбоев.

### 3.2 Диаграмма вариантов использования

Процесс проектирования информационной системы включает в себя создание UML-диаграмм. Диаграмма вариантов использования отражает функциональные возможности и требования проектируемой системы с использованием действующих лиц и вариантов использования.

Проектируемую систему «Онлайн-кинотеатр» могут использовать 4 актёра: «Незарегистрированный пользователь», «Зарегистрированный пользователь», «Модератор». Актёры «Зарегистрированный пользователь» и являются наследниками актёра «Неавторизованный пользователь», а актёр «Модератор» является наследником «Зарегистрированный пользователь».

Актеру ««Незарегистрированный пользователь» доступны следующие варианты использования:

- 1) «Регистрация»;
- 2) «Просмотр информации о фильмах»;
- 3) «Поиск фильма»;
- 4) «Поиск фильма по названию»;
- 5) «Поиск фильма по году выхода»;
- 6) «Поиск фильма по жанрам»;

- 7) «Поиск фильма по возрастному рейтингу»;
- 8) «Поиск фильма по рейтингу фильма».

Актеру «Зарегистрированный пользователь» доступны варианты использования:

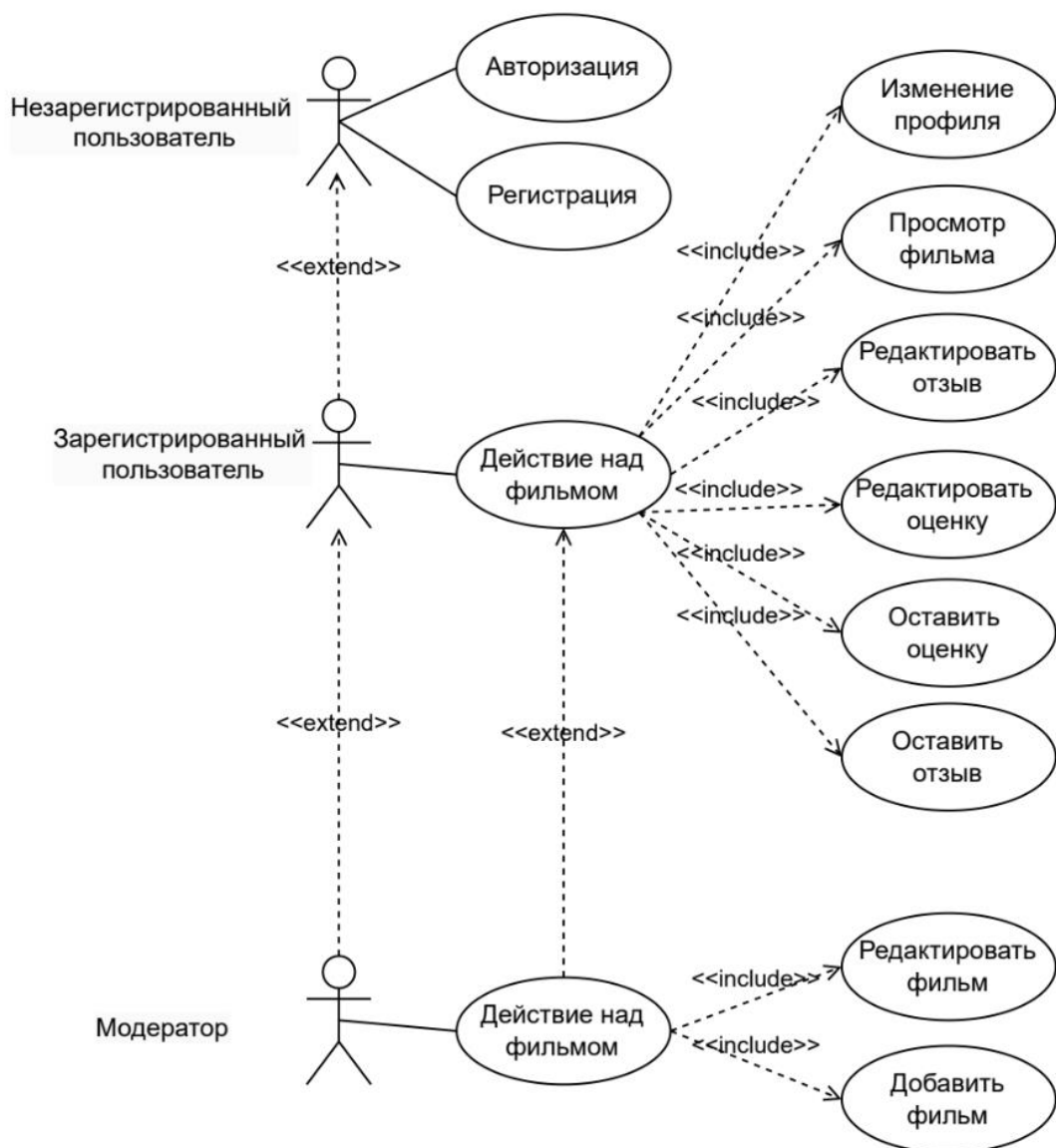
- 1) Наследует все варианты использования актёра «Незарегистрированный пользователь»;
- 2) «Добавление фильма в избранное»;
- 3) «Удаление фильма из избранного»;
- 4) «Добавление фильма в просмотренное»;
- 5) «Удаление фильма из просмотренное»;
- 6) «Оставить оценку фильму»;
- 7) «Оставить отзыв о фильме».
- 8) «Удалить отзыв о фильме».
- 9) «Просмотр профиля».
- 10) «Редактирование профиля».

Актеру «Модератор» доступны варианты использования:

- 1) Наследует все варианты использования актёра «Зарегистрированный пользователь»;
- 2) «Просмотр, добавление, удаление и редактирование информации о фильмах»;
- 3) «Просмотр, добавление, удаление и редактирование информации об актёрах»;
- 4) «Просмотр, добавление, удаление и редактирование информации о пользователях».
- 5) «Удаление отзыва пользователя»;

На рисунке 3.2 представлена диаграмма вариантов использования для проектируемой системы.





### 3.2 – Диаграмма вариантов использования информационной системы «Онлайн-кинотеатр»

#### 3.3 Диаграмма последовательности

Диаграмма последовательности – это тип диаграммы, который позволяет описать взаимодействие между объектами в системе в виде последовательности сообщений, действий и операций, отображая порядок выполнения действий и обмена информацией между объектами во времени. Диаграмма последовательности наглядно отображает временной аспект взаимодействия. Она имеет два измерения. Одно измерение (слева-направо) указывает на порядок вовлечения экземпляров сущностей во взаимодействие. Крайним слева на диаграмме отображается экземпляр актера или объект, который является инициатором взаимодействия. Правее отображается другой экземпляр сущности, который

непосредственно взаимодействует с первым и т.д. Второе измерение (сверху-вниз) указывает на порядок обмена сообщениями. Начальному моменту времени соответствует самая верхняя часть диаграммы.

На рисунке 3.3 представлена диаграмма последовательности для варианта использования «Процесс поиска фильма пользователем».

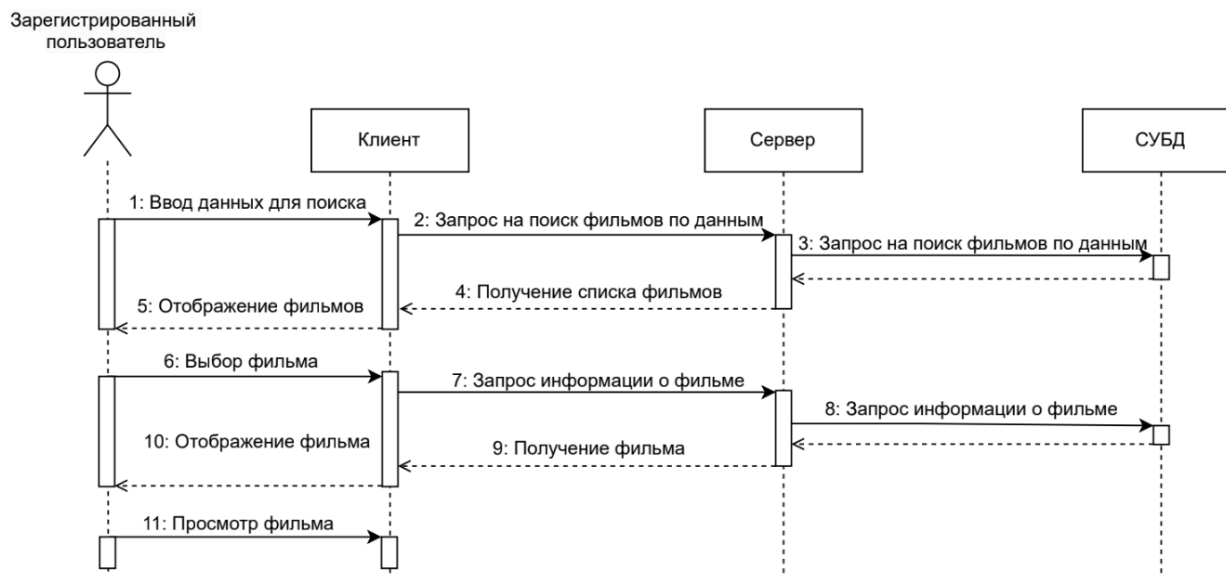


Рисунок 3.3 – Диаграмма последовательности для варианта использования «Процесс поиска фильма пользователем»

В спроектированной диаграмме последовательности пользователь заполняет критерии поиска. Клиентская часть информационной системы формирует запрос на поиск фильма к серверной части. Серверная часть в свою очередь проверяет данные и отправляет запрос на поиск к базе данных информационной системы. После завершения поиска серверная часть возвращает клиентской список найденных фильмов. Клиентская часть при получении информации о найденных заведений отображает ее пользователю. Далее пользователь совершает выбор нужного фильма, аналогичным образом происходит взаимодействие клиентской части, серверной части и базы данных. В результате, информация о выбранном фильме отображается пользователю.

### 3.4 Выбор средств реализации

Для реализации информационной системы был выбран язык программирования Python и фреймворк Django 5.0.3 Django («Джанго») — это свободный фреймворк для разработки веб-приложений и сайтов на языке Python. В качестве среды разработки для языка программирования Python была выбрана Visual Studio Code 1.89.1 К преимуществам Django 5.0.3 можно отнести:

- 1) Универсальность. Django подходит для разработки веб-сайтов и приложений любого типа. Он может работать с любыми типами файлов, различными базами данных;
- 2) Безопасность. Django включает механизмы предотвращения SQL-инъекций (XSS) и подделки межсайтовых запросов (CSRF);
- 3) Масштабируемость. Архитектура Django строится на независимости составляющих частей. Любой компонент можно заменить или модифицировать, не затрагивая другие;
- 4) Подробная документация и большое сообщество программистов;
- 5) Широкие возможности для работы с базами данных. Django поддерживает множество реляционных СУБД, таких как MySQL, PostgreSQL, и SQLite, а также NoSQL базы данных;
- 6) Поддержка ORM. Django имеет встроенный ORM (Object-Relational Mapping), который облегчает работу с базой данных и управление данными в приложении.

В целом, фреймворк Django является мощным инструментом для создания веб-приложений, который обладает широким спектром функций и возможностей.

База данных для проектируемой информационной системы реализована в СУБД PostgreSQL 16. Веб-интерфейс написан на языке гипертекстовой разметки HTML и языка описания внешнего вида CSS.

### 3.5 Реализация серверной части

Серверная часть получает запросы от клиента, обрабатывает их и полученный результат отправляет обратно клиенту. Для реализации серверной части используется СУБД PostgreSQL 16., язык программирования Python и веб-фреймворк Django 5.0.3

Фреймворк Django реализует архитектурный паттерн Model-View-Template или сокращенно MVT, который по смыслу является аналогом более распространенного паттерна MVC (Model-View-Controller). MVT представляет собой шаблон архитектуры программного обеспечения. Схематично архитектуру MVT в Django можно представить следующим образом:

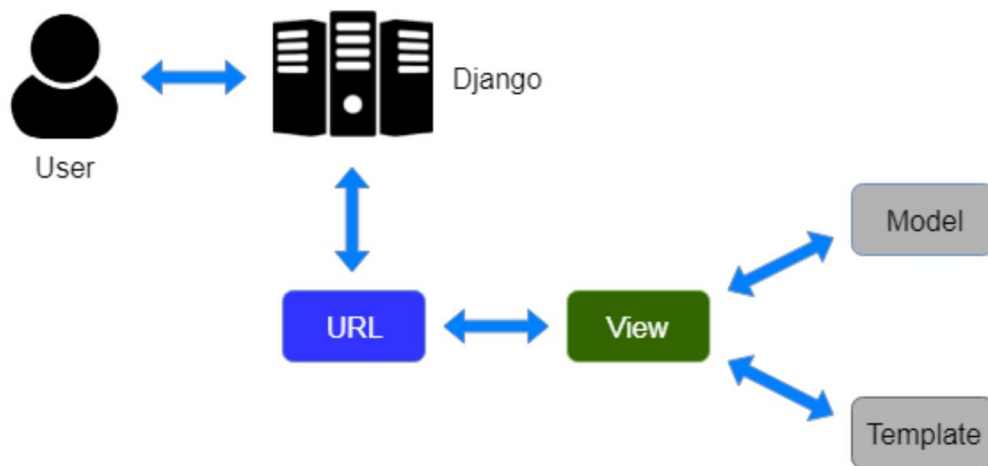


Рисунок 3.4 – Схема архитектуры MVT в Django

Основные элементы паттерна:

- 1) Model — описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных.
- 2) View – получает запрос, обрабатывает его и отправляет в ответ пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними. Для создания ответа может применять Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления). То есть представление получает данные от модели и предоставляет шаблонам (templates) доступ к этим данным или предварительно обрабатывает данные и затем предоставляет к ним доступ шаблонам;
- 3) Template – представляет логику представления в виде сгенерированной разметки html. В MVC этому компоненту соответствует View, то есть представления.

При каждом переходе пользователя по ссылке браузер отправляет HTTP запрос серверу, URL маршрутизатор определяет, с каким представлением (View) сопоставляется данный запрос и передает этот запрос выбранному представлению (View). Представление получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные, или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template).

Для реализации функциональной возможности пользователя просматривать список фильмов на просмотр были использованы модели Film, Genre, которые определяют таблицы «Фильм», «Жанр» соответственно, используется URL:

```
path("", views.MovieView.as_view(), name="list"),
```

Для просмотра списка фильмов использованы представления MovieView, FilterMovieView и Search. Первые два представления наследуются от класса FilmFilters, который служит в роли фильтра фильмов по разным критериям, таких как жанр, год, рейтинг и др.

```
class FilmFilters:

    def get_genres(self):
        return Genre.objects.all()

    def get_years(self):
        return Film.objects.all().values("year")

    def get_stars(self):
        return ScoreStar.objects.all().values("value")

    def get_rating(self):
        return Film.objects.all().values("rating")

    def get_mpaa_rating(self):
        return Film.objects.all().values("mpaa_rating")

class MovieView(FilmFilters, ListView):
    model = Film
    queryset = Film.objects.all().order_by('id')
    template_name = "films/film_list.html"
    paginate_by = 6

class FilterMovieView(FilmFilters, ListView):

    template_name = "films/film_list.html"
    paginate_by = 1

    def get_queryset(self):
        queryset = Film.objects.filter(
            Q(year__in=self.request.GET.getlist("year")) |
            Q(genres__in=self.request.GET.getlist("genre")) |
            Q(rating__in=self.request.GET.getlist("rating")) |
            Q(mpaa_rating__in=self.request.GET.getlist("mpaa_rating"))
        ).distinct()
        return queryset
```

```

def get_context_data(self, *args, **kwargs):
    context = super().get_context_data(*args, **kwargs)
    context["year"] = ''.join([f"year={x}&" for x in
self.request.GET.getlist("year")])
    context["genre"] = ''.join([f"genre={x}&" for x in
self.request.GET.getlist("genre")])
    context["rating"] = ''.join([f"rating={x}&" for x in
self.request.GET.getlist("rating")])
    context["mpaa_rating"] = ''.join([f"mpaa_rating={x}&" for x in
self.request.GET.getlist("mpaa_rating")])
    return context

class Search(ListView):

    """Поиск фильмов"""
    template_name = "films/film_list.html"

    def get_queryset(self):
        return
Film.objects.filter(name__icontains=self.request.GET.get("q"))

    def get_context_data(self, *args, **kwargs):
        context = super().get_context_data(*args, **kwargs)
        context["q"] = f'q={self.request.GET.get("q")}&'
        return context

```

Помимо этого, в разрабатываемой информационной системы были реализованы следующие основные функции на стороне сервера: поддержка двух языков: английский и русский, регистрация пользователя, аутентификация пользователей, возможность пользователю добавить фильм в список просмотренного и избранного, оценивать и оставлять отзывы к фильму, осуществлять поиск фильма по названию, фильтр фильма по критериям: «Жанр», «Год», «Рейтинг», «Возрастной рейтинг», редактирование пользователем личной информации. Листинг функций, реализующих функционал системы, представлен в приложении Б.

Для работы с базами данных в проекте Django в файле settings.py определен параметр DATABASES. Переменная DATABASES содержит набор конфигураций подключений к базам данных в виде словаря. Ключи в этом словаре - названия подключений. Конфигурация каждого подключения может состоять из ряда параметров. Параметр ENGINE указывает на используемый движок для доступа к БД. Подключение к базе данных выглядит следующим образом:

```

DATABASES = {
    'default': {

```

```

        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'cinema',
        'USER': 'postgres',
        'PASSWORD': 'adadod26',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

```

Также в разрабатываемой информационной системе поддерживается безопасность благодаря тому, что Django предоставляет гибкую систему хранения паролей и по умолчанию использует PBKDF2 [7]. PBKDF2 является стандартом для производства ключей на основе паролей и может использоваться для защиты паролей различных приложениях. Важным свойством PBKDF2 является возможность увеличить время вычисления производства ключей путем увеличения количества итераций, что делает атаку на пароли методом перебора затратной в вычислительном отношении.

### 3.6 Реализация клиентской части

Клиентская часть реализует пользовательский интерфейс веб-приложения и загружается в виде веб-страниц браузера. Интерфейс клиентского приложения реализован с помощью:

HTML – это стандартный язык разметки гипертекста, который используется для описания структуры содержимого веб-страниц, а именно, расположения элементов друг относительно друга, их внешнего представления, их функций.

CSS — это язык таблиц стилей, который позволяет создать оформление страницы. Он имеет множество возможностей для настройки дизайна страницы, таких как задание цветов, шрифтов, размеров, расположения элементов и т. Д

Как было сказано в параграфе 3.5, шаблон (Template) — это основной инструмент для создания пользовательского интерфейса в Django. Шаблоны (файлы с расширением .html) хранятся в папке «templates» и используются для представления данных приложения. В папке «static» расположены файлы с расширением «.css», в которых определяется внешний вид страниц и их элементов.

Незарегистрированный пользователь попадает на главную страницу, на которой он может просматривать информацию о различных фильмах, актёрах, осуществлять поиск по наименованию фильма, фильтровать заведения по различным критериям.

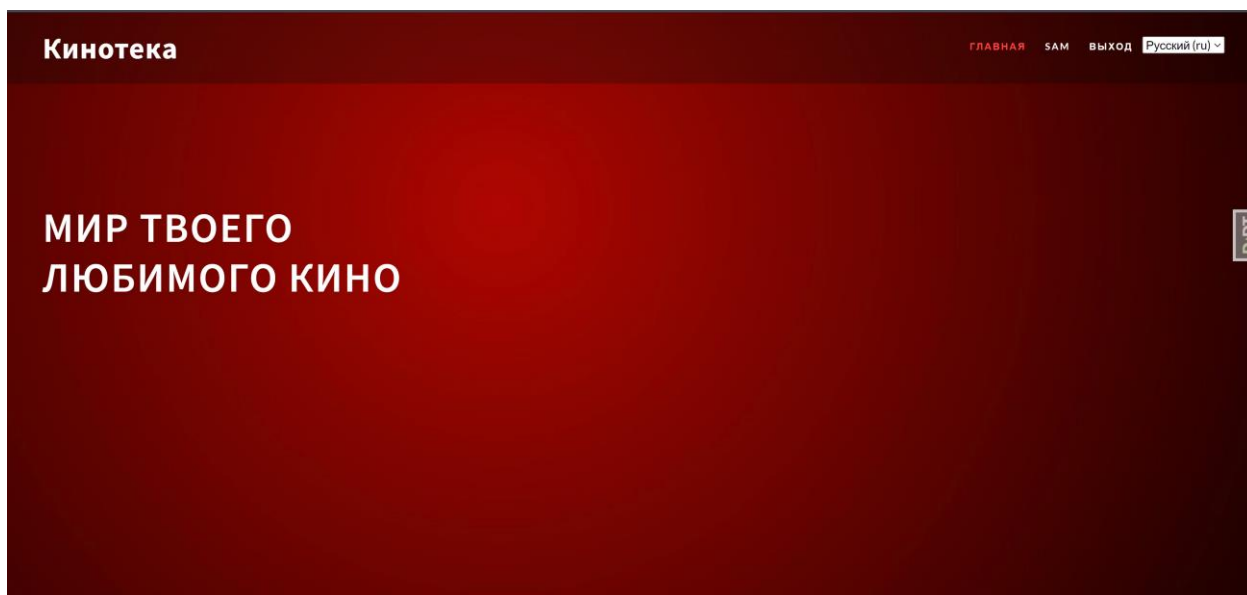


Рисунок 3.5 – Главная страница информационной системы «Онлайн-кинотеатр»

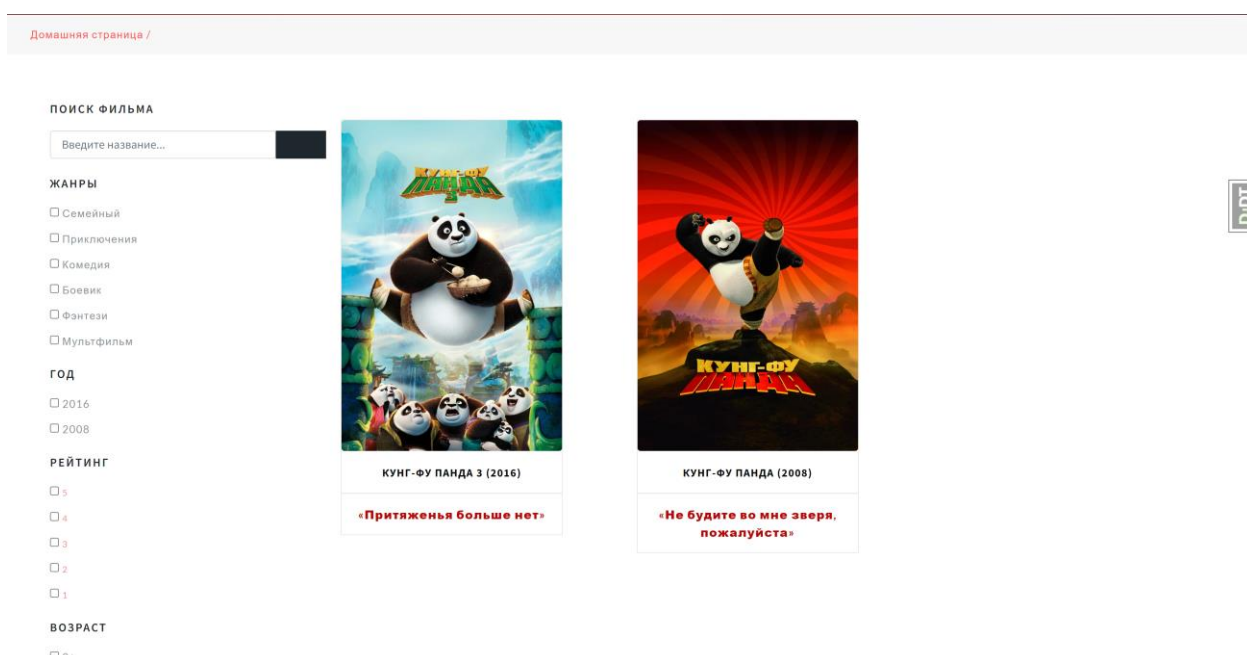


Рисунок 3.6 – Главная страница информационной системы «Онлайн-кинотеатр»

На рисунке 3.11 представлена страница с информацией о конкретном фильме, на которой размещены необходимые данные, также на этой странице есть кнопка «Добавить в избранное», «Добавить в просмотренное», «Просмотреть фильм», «Просмотреть трейлер», есть так же возможность оставить оценку фильму, рейтинг которого подсчитывается как среднее значение среди всех оценок, оставить и удалить собственный отзыв, в заключении есть возможность перейти на страницу с конкретным актёром, чтобы посмотреть информацию о нём.



**ПОИСК ФИЛЬМА**
Кунг-фу Панда (2008)

---

**ЖАНРЫ**

- ☒ Семейный
- ☒ Приключения
- ☒ Комедия
- ☒ Боевик
- ☒ Фэнтези
- ☒ Мультфильм

**ГОД**

- ☒ 2016
- ☒ 2008

**РЕЙТИНГ**

- ☒ 5
- ☒ 4
- ☒ 3
- ☒ 2
- ☒ 1

**ВОЗРАСТ**


- ☒ 0+
- ☒ 6+


**МРАА**

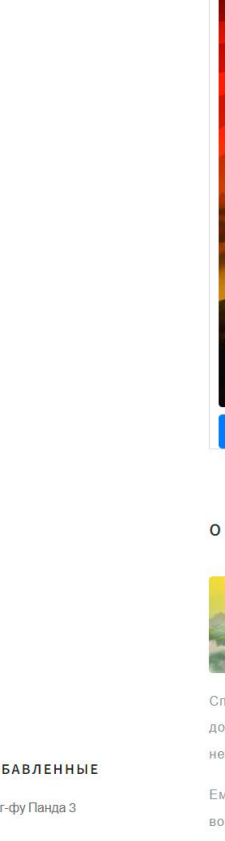
- ☒ PG

Найти

**ПОСЛЕДНИЕ ДОБАВЛЕННЫЕ**



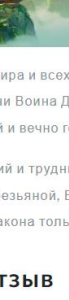

Кунг-фу Панда 3


Кунг-фу Панда



[Посмотреть фильм](#)
[Посмотреть трейлер](#)

### О ФИЛЬМЕ КУНГ-ФУ ПАНДА


Спасение Долины Мира и всех ее обитателей от непобедимого и безжалостного мастера Тай Лунга должно лечь на плечи Воина Дракона, Избранного среди лучших из лучших, коим становится... неуклюжий, ленивый и вечно голодный панда По.

Ему предстоит долгий и трудный путь к вершинам мастерства кунг-фу бок о бок с легендарными воинами: Тигрой, Обезьяной, Богомопом, Гадюкой и Журавлем. По постигнет тайну древнего Свитка и станет Воином Дракона только в том случае, если сможет поверить в себя...

### Оставить отзыв

Отправить

### Отзывы



**Семён Ильметов (nem)**  
 Фильм детства, обожаю

[Удалить](#)

Кунг-фу Панда (2008)

## Kung Fu Panda

Год 2008

Страна: США Великобритания

**Слоган:** «Не будьте во мне зверя, пожалуйста»

**Режиссёр:** Марк Осборн

Оператор:

Сценарий: Джонатан Айбел

Композитор: Ханс Циммер

Актёр: [Джек Блэк](#)

**Жанр:** Мультфильм Фэнтези Боевик Комедия

## Приключения Семейный

Премьера в мире: 15 мая 2008 г.

**Премьера в России:** 5 июня 2008 г.

Бюджет: 130000000 \$

Сборы: 631744560 \$

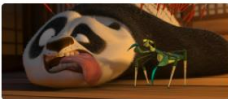
Рейтинг: 5,0 Ваша оценка:



- Добавить в избранное

Deutsche Endpunkte

## О ФИЛЬМЕ КУНГ-ФУ ПАНДА



Спасение Долины Мира и всех ее обитателей от непобедимого и безжалостного мастера Тай Лунга должно лечь на плечи Воина Дракона, Избранного среди лучших из лучших, коим становится... неуклюжий, ленивый и вечно голодный панда По.

Ему предстоит долгий и трудный путь к вершинам мастерства кунг-фу бокс с легендарными воинами: Тигрой, Обезьяной, Богомолком, Гадюкой и Журавлем. По постигнет тайну древнего Свитка и станет Воином Дракона только в том случае, если сможет поверить в себя...

[Оставить отзыв](#)

Отправить

## Отзывы



Семён Ильметов (nem)

Фильм детства, обожаю

Удалить

Рисунок 3.7 – Страница с информацией о фильме

Страница регистрации имеет вид, представленный на рисунке 3.12.

Домашняя страница /

**Регистрация**

Имя пользователя:

Пароль:

Подтверждение пароля:

Почта:

Имя:

Фамилия:

Отчество (Опционально):

**Зарегистрироваться**

**Библиотека**

D/DT

Рисунок 3.8 – Страница регистрации

Страница для авторизации пользователя представлена на рисунке 3.13

Домашняя страница /

**Вход**

Имя пользователя:

Пароль:

**Войти**

**Библиотека**

Описание

РАЗДЕЛЫ

СТРАНИЦЫ

КОНТАКТНАЯ ИНФОРМАЦИЯ

D/DT

Рисунок 3.9 – Страница авторизации

Авторизованный пользователь имеет все возможности перечисленные выше и доступ ко всем страницам, приведенным на рисунках выше, а также ему предоставляется возможность добавлять и удалять выбранные фильмы в «Избранное» и «Просмотренное» (Рисунок 3.14), страница «Профиль» (Рисунок 3.17), где расположена информация о пользователе и списке его оцененных, просмотренных и избранных фильмов, а также страница для редактирования информации о пользователе (Рисунок 3.18).

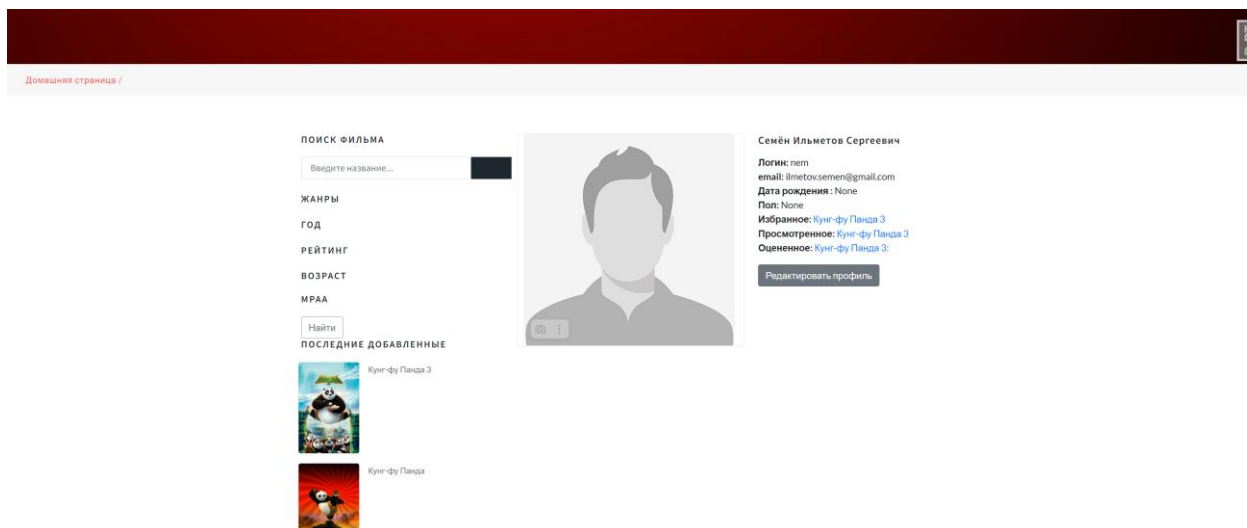


Рисунок 3.10 – Страница «Профиль» с информацией о пользователе

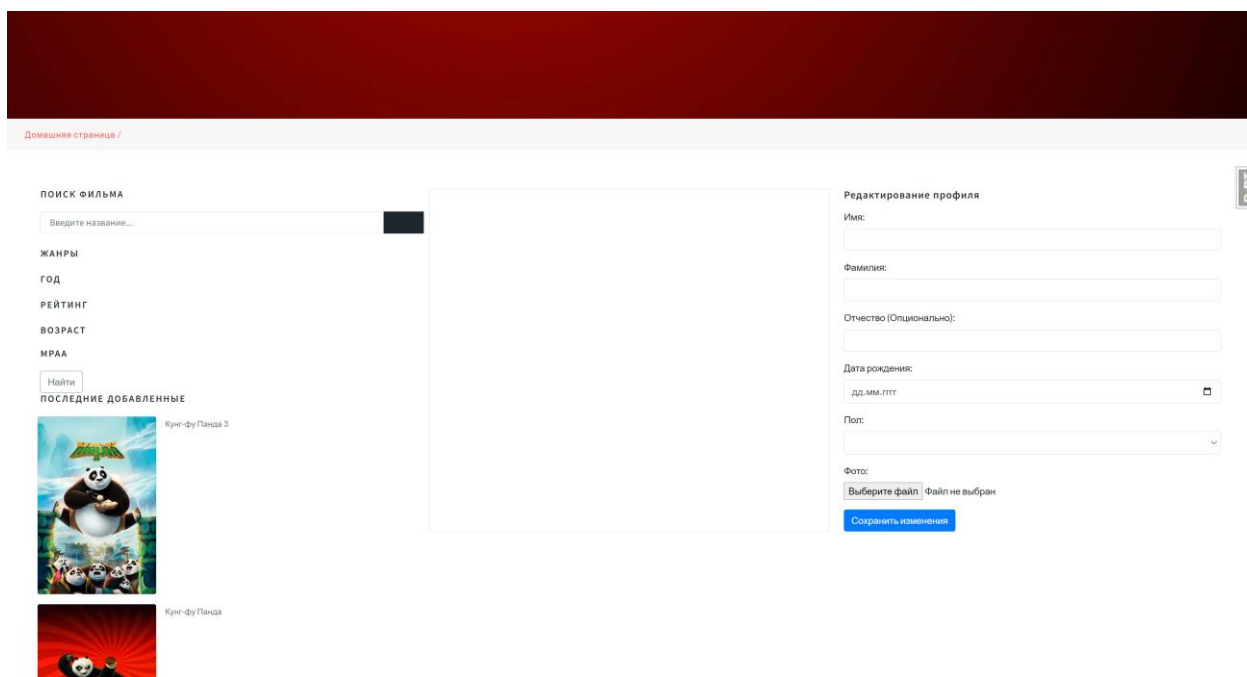


Рисунок 3.11 – Страница «Редактирование профиля» с возможностью редактировать информацию о пользователе

Для понимания работы клиентской части веб–приложения рассмотрим некоторые функции. В html шаблоне регистрации пользователем используются следующие методы, представленные в приложении В:

<form method="post" >— сообщает серверу о методе запроса post. Метод post посылает на сервер данные в запросе браузера;

{% csrf\_token %} — это токен, который используется в Django для защиты от атак межсайтовой подделки запросов (CSRF). CSRF-атака возникает, когда злоумышленник отправляет запрос от имени авторизованного пользователя без его согласия. Этот тег генерирует уникальный токен для каждого запроса и проверяет его при получении запроса на сервере. Если токен не совпадает, то запрос отклоняется как подозрительный;

В html шаблоне с заявками на бронирование в конкретном заведении для менеджера используются следующие методы, представленные в приложении В:

### 3.7 Тестирование информационной системы

Тестирование – это процесс, который используется для получения актуальной информации о системе. Для информационной системы «Онлайн-кинотеатр» было проведено функциональное тестирование на модульном и системном уровнях, а также нефункциональное тестирование: нагрузочное и кроссбраузерности.

#### 3.7.1 Модульное тестирование

Модульное (компонентное) тестирование направлено на проверку отдельных небольших частей приложения, которые можно исследовать изолированно от других подобных частей. В Django автоматизация модульного тестирования будет выполняться с использованием встроенного в фреймворк инструмента для тестирования - Django Test Framework. Для написания модульных тестов необходимо создать файл с тестами, который будет находиться в директории приложения. В этом файле можно определить класс для тестирования, который будет наследоваться от `django.test.TestCase`.

Для примера рассмотрим реализацию модульных тестов для форм (Form). Тест для проверки регистрации описан в Приложении В.

Всего было написано и запущено 19 модульных тестов (рисунок 3.28)

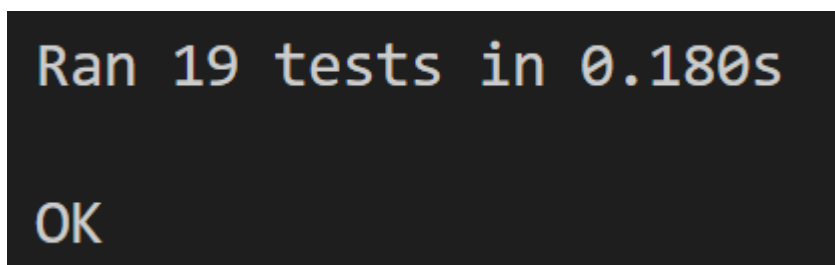


Рисунок 3.12 – Протокол модульного тестирования для информационной системы «Онлайн-кинотеатр»

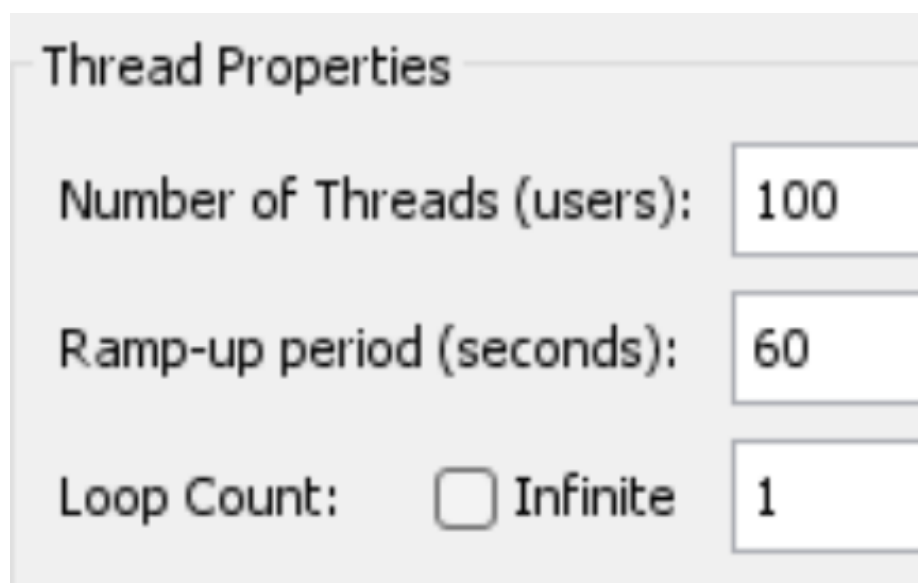
### 3.7.2 Нагрузочное тестирование

Далее был выполнен переход из проверки функциональных требований системы к проверке нефункциональных требований. Одним из видов нефункционального тестирования является тестирование производительности. Для тестирования веб-приложения «Онлайн-кинотеатр» было выбрано нагрузочное тестирование, основная цель которого - проверка способности системы обрабатывать растущие объёмы ожидаемой реалистичной нагрузки. Нагрузочное тестирование проводилось в операционной системе Windows 10 с помощью инструмента Apache JMeter.

Для тестирования были выбраны следующие сценарии:

- 1) реалистичные условия: 1 виртуальный пользователей в течение 60 секунды делал 1 запрос к главной странице сайта;
- 2) реалистичные условия: 100 виртуальных пользователей в течение 60 секунд делали 1 запрос к главной странице сайта;
- 3) реалистичные условия: 200 виртуальных пользователей в течение 60 секунд делали 1 запрос к главной странице сайта;
- 4) пиковые условия: 300 виртуальных пользователей в течение 60 секунд делали 1 запрос к главной странице сайта;
- 5) пиковые условия: 400 виртуальных пользователей в течение 60 секунд делали 1 запрос к главной странице сайта.
- 6) пиковые условия: 450 виртуальных пользователей в течение 60 секунд делали 1 запрос к главной странице сайта.

Пример заполнения параметров для тестирования представлен на рисунке 3.25



Thread Properties

Number of Threads (users): 100

Ramp-up period (seconds): 60

Loop Count: ☐ Infinite 1

Рисунок 3.13 – Параметры нагрузочного тестирования

Результаты нагрузочного тестирования приведены в таблице 1.

Таблица 1 – Результаты нагрузочного тестирования

Сценарий тестирования	Sample	AVG, ms	Min, ms	Max, ms	Error, %	Throughput
1 пользователь	1	190	190	190	0.00%	5.3/sec
100 пользователей	100	162	112	376	0.00%	1,7/sec
200 пользователей	200	207	140	463	0.00%	3.3/sec
300 пользователей	300	234	147	506	0.00%	5/sec
400 пользователей	400	254	143	441	0.00%	6.7/sec
450 пользователей	463	19125	11	77551	23.11%	3.9/sec

Параметры, участвующие в тестировании:

- 1) sample – количество отправленных запросов;
- 2) average– среднее арифметическое для всех ответов, миллисекунды;
- 3) min – минимальное время отклика, миллисекунды;
- 4) max – максимальное время отклика, миллисекунды;
- 5) errorrate – процент неудачных тестов;
- 6) throughput – сколько запросов в секунду обрабатывает сервер.

На основании проведенного нагрузочного тестирования главной страницы веб-приложения в Jmeter было установлено, что при параметрах 450 пользователей за 60 секунду процент ошибок составил 23%, что свидетельствует о том, что сервер не может обработать все запрашиваемые запросы при такой нагрузке. При меньшем количестве пользователей (менее 450) процент ошибок составил 0.

### 3.7.3 Тестирование кроссбраузерности

Также информационная система «Онлайн-кинотеатр» была проверена на кроссбраузерность. Кроссбраузерность — это способность веб-приложения корректно работать на различных браузерах и платформах. Тестирование проводилось в следующих браузерах: Google Chrome 113.0.5672.127, Яндекс браузер 23.3.4.605, Microsoft Edge 113.0.1774.50.

Для перечисленных браузеров тестирование состоит в том, чтобы открыть веб-приложение в выбранном браузере, ожидаемым результатом, соответственно, является корректное отображение страниц. Полученные результаты, представленные на рисунках ниже, подтверждают реализацию требования к кроссбраузерности.

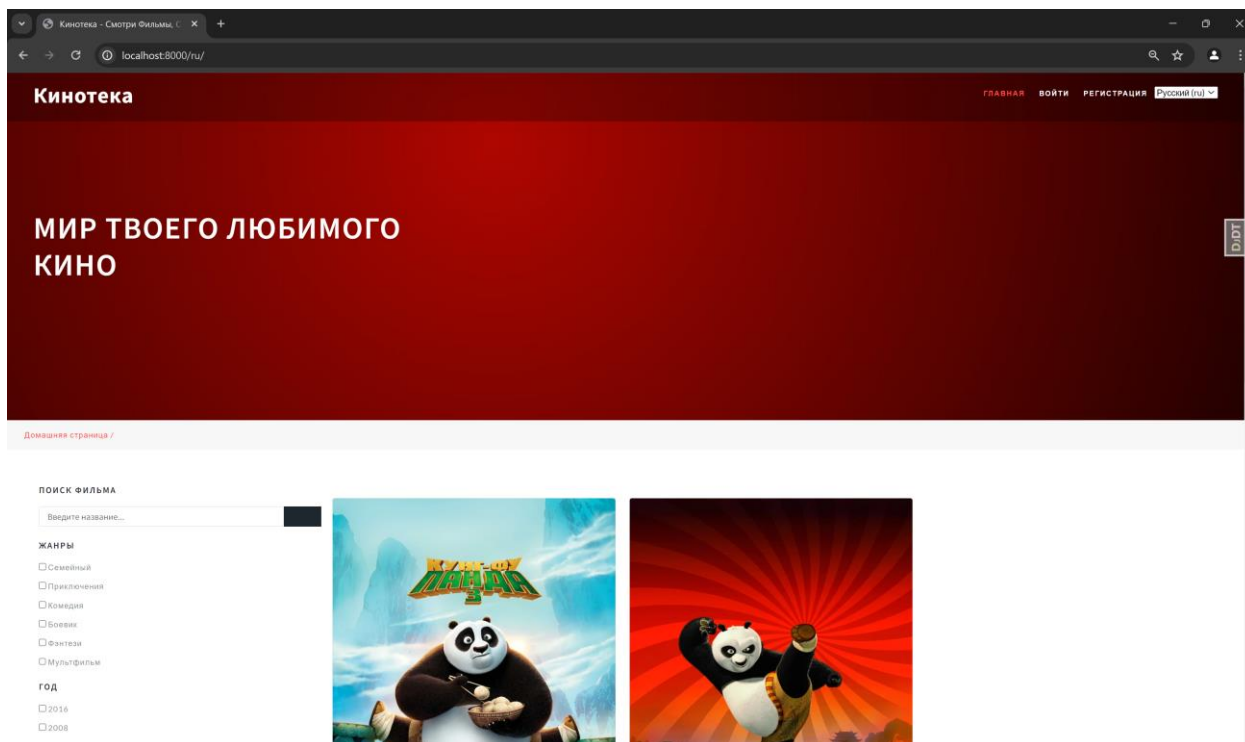


Рисунок 3.14 – Информационная система «Онлайн-кинотеатр» в браузере Google Chrome

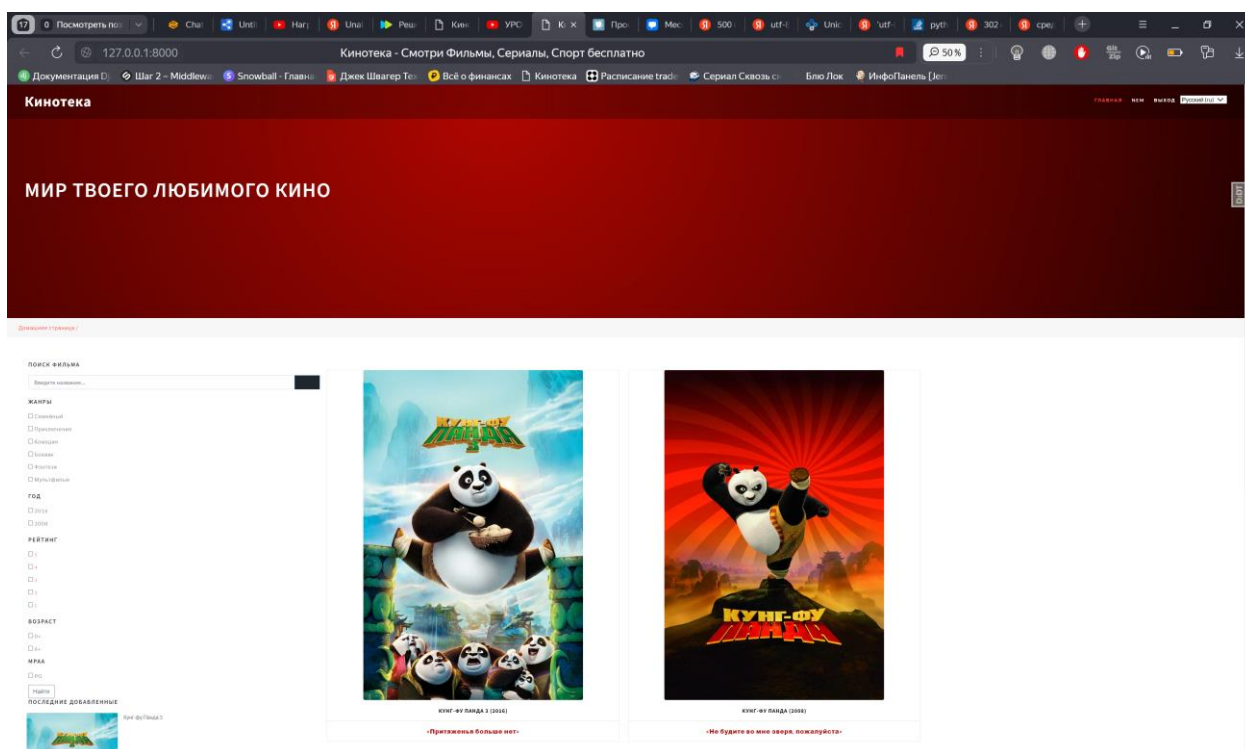


Рисунок 3.15 – Информационная система «Онлайн-кинотеатр» в Яндекс браузере

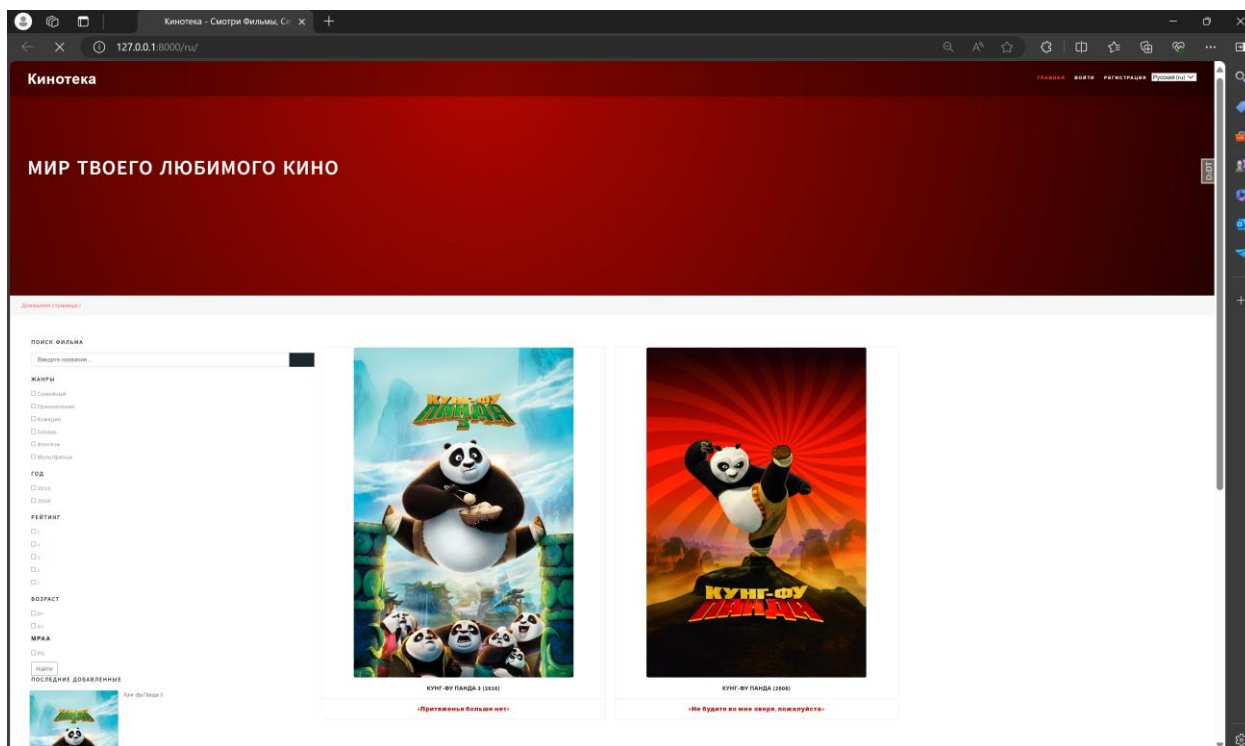


Рисунок 3.16 – Информационная система «Онлайн-кинотеатр» в браузере Microsoft Edge

### 3.7.4 Интеграционное тестирование

В рамках интеграционного тестирования проверяется взаимодействие следующих компонентов приложения:

#### 1) Представления (View)

Для проверки используются методы HTTP клиента, а именно:

- 1) GET – для проверки получения данных;
- 2) POST – для проверки создания и обновления данных;
- 3) DELETE – для проверки удаления данных.

Для интеграционного тестирования использован Postman — это мощный и интуитивно понятный инструмент, предназначенный специально для тестирования и разработки API. Подготовлено окружение для хранения значений, используемых во время тестирования, например, адреса обращения к серверу и токенов пользователей. Создана коллекция, объединяющая все связанные запросы.

Протестированы GET запросы для получения информации о фильмах, конкретном фильме, профиле, странице регистрации и авторизации.



GET ⌵ http://127.0.0.1:8000/ru/filter/?genre=6&year=2008&rating=6%2B Send ⌵

Params ● Authorization Headers (7) Body Scripts ● Settings Cookies

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	genre	6			
<input checked="" type="checkbox"/>	year	2008			
<input checked="" type="checkbox"/>	rating	6%2B			

body Cookies (1) Headers (13) Test Results (1/1) 200 OK 202 ms 26.11 KB Save as example ...

Рисунок 3.17 – Проверка GET запроса на вывод фильмов по фильтру

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы спроектирована, реализована и протестирована информационная система «Онлайн-кинотеатр», позволяющая пользователю просматривать информацию о фильмах, актёрах, оставлять отзывы, оценки, добавлять их в «Избранное», «Просмотренное», осуществлять поиск фильмов по названию, фильтровать фильмы по критериям: «Жанр», «Год», «Рейтинг», «Возрастной рейтинг», редактирование личной информации. Для менеджера система предоставляет следующие возможности: редактировать информацию о фильме, обрабатывать отзывы к фильмам. Для информационной системы было разработано руководство пользователя, представленное в приложении Г.

В процессе выполнения работы было сделано следующее:

- 1) проведён анализ предметной области;
- 2) определены функциональные и нефункциональные требования к системе;
- 3) на основе анализа предметной области была построена информационная модель «Онлайн-кинотеатр» и соответствующая ей логическая модель базы данных;
- 4) в СУБД PostgreSQL реализована физическая модель базы данных;
- 5) с использованием языка гипертекстовой разметки HTML, языка таблиц стилей CSS и фреймворка Django реализовано клиентское приложение;
- 6) с использованием языка программирования Python, веб-фреймворка Django реализовано серверное приложение;
- 7) реализована страница поиска заведений по различным критериям;
- 8) реализована страница с заполнением и отправкой заявки на бронирование пользователем;
- 9) реализована страница с заявками на бронирования, принадлежащих пользователю;
- 10) реализована страница «Профиль» с данными пользователя;
- 11) реализовано формирование системой отчета с информацией о пользователе в виде документа с расширением pdf, который может быть сохранен и просмотрен модератором;
- 12) реализовано формирование системой отчета с информацией о фильмах, добавленных пользователем в «Избранное», «Просмотренное» в виде документа с расширением pdf, который может быть сохранен и просмотрен пользователем;
- 13) проведено модульное тестирование, нагрузочное тестирование и тестирование кроссбраузерности.

14) разработана инструкция пользователя для созданной системы.

Таким образом, в ходе выполнения курсовой работы были сформированы систематические знания основных положений и концепций современных информационных технологий для проектирования и реализации информационных систем на основе баз данных, основных стандартов, норм и правил разработки технической документации для информационных систем на основе баз данных, методики установки СУБД и администрирования информационных систем и баз данных, в том числе отечественного производства; проблем и тенденций развития рынка СУБД, были сформированы умения использовать основные положения и концепции современных информационных технологий для проектирования и реализации информационных систем на основе баз данных в профессиональной деятельности, выбирать необходимые стандарты, нормы и правила для подготовки конкретной технической документации, использовать основные стандарты, нормы и правила при подготовке технической документации для информационных систем на основе баз данных, использовать методику установки СУБД, реализовывать техническое сопровождение СУБД, информационных систем и баз данных, в том числе отечественного производства; применять знания проблем и тенденций развития рынка программного обеспечения при выборе СУБД для конкретной информационной системы, а также приобретены навыки разработки информационных систем на основе баз данных, подготовки технической документации для информационных систем на основе баз данных, установки СУБД, проектирования, реализации и администрирования информационных систем на основе баз данных, что свидетельствует о том, что компетенции ОПК–3, ОПК–4 и ОПК–5 освоены.

#### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Хоменко, А. Д. Базы данных. Учебник для ВУЗов. –М: Корона Принт, 2004–736 с.

## ПРИЛОЖЕНИЕ А

### Листинг функций серверной части

```
class GenreYear:

    def get_genres(self):
        return Genre.objects.all()

    def get_years(self):
        return Film.objects.all().values("year")

    def get_stars(self):
        return ScoreStar.objects.all().values("value")

    def get_rating(self):
        return Film.objects.all().values("rating")

    def get_mpaa_rating(self):
        return Film.objects.all().values("mpaa_rating")
```

```
class MovieView(GenreYear, ListView):
    model = Film
    queryset = Film.objects.all().order_by('id')
    template_name = "films/film_list.html"
    paginate_by = 6
```

Листинг А.1 – Представление вывода списка фильмов на главной странице

```
class MovieDetailView(GenreYear, DetailView):

    model = Film
    template_name = "films/film_detail.html"
    slug_field = "url"
    queryset = Film.objects.all().select_related()

    def get_context_data(self, **kwargs: Any):
        context = super().get_context_data(**kwargs)
        context['star_form'] = RatingForm()

        if self.request.user.is_authenticated:

            user = User.objects.get(user=self.request.user)
            context['user'] = user

        return context
```

Листинг А.2 – Представление вывода информации о фильме

```
class GenreYear:
```

```

def get_genres(self):
    return Genre.objects.all()

def get_years(self):
    return Film.objects.all().values("year")

def get_stars(self):
    return ScoreStar.objects.all().values("value")

def get_rating(self):
    return Film.objects.all().values("rating")

def get_mpaa_rating(self):
    return Film.objects.all().values("mpaa_rating")

class MovieDetailView(GenreYear, DetailView):

    model = Film
    template_name = "films/film_detail.html"
    slug_field = "url"
    queryset = Film.objects.all().select_related()

    def get_context_data(self, **kwargs: Any):
        context = super().get_context_data(**kwargs)
        context['star_form'] = RatingForm()

        if self.request.user.is_authenticated:

            user = User.objects.get(user=self.request.user)
            context['user'] = user

        return context

```

Листинг А.3 – Представление вывода информации об актёре

```

class FilterMovieView(GenreYear, ListView):

    template_name = "films/film_list.html"
    paginate_by = 1

    def get_queryset(self):
        queryset = Film.objects.filter(
            Q(year__in=self.request.GET.getlist("year")) |
            Q(genres__in=self.request.GET.getlist("genre")) |
            Q(rating__in=self.request.GET.getlist("rating")) |
            Q(mpaa_rating__in=self.request.GET.getlist("mpaa_rating"))
        ).distinct()
        return queryset

```

```

def get_context_data(self, *args, **kwargs):
    context = super().get_context_data(*args, **kwargs)
    context["year"] = ''.join([f"year={x}&" for x in
self.request.GET.getlist("year")])
    context["genre"] = ''.join([f"genre={x}&" for x in
self.request.GET.getlist("genre")])
    context["rating"] = ''.join([f"rating={x}&" for x in
self.request.GET.getlist("rating")])
    context["mpaa_rating"] = ''.join([f"mpaa_rating={x}&" for x in
self.request.GET.getlist("mpaa_rating")])
    return context

```

Листинг А.4 – Представление фильтра фильмов ко критериям

```

class Search(ListView):
    """Поиск фильмов"""
    template_name = "films/film_list.html"

    def get_queryset(self):
        return Film.objects.filter(name__icontains=self.request.GET.get("q"))

    def get_context_data(self, *args, **kwargs):
        context = super().get_context_data(*args, **kwargs)
        context["q"] = f'q={self.request.GET.get("q")}&'
        return context

```

Листинг А.5– Представление поиск фильмов

```

class AddReviewView(View):
    template_name = "films/film_detail.html"

    def post(self, request, slug):
        film = get_object_or_404(Film, id=slug)
        if request.user.is_authenticated:
            if 'submit_review' in request.POST:
                form = ReviewForm(request.POST)
                if form.is_valid():
                    review = form.save(commit=False)
                    review.film = film
                    review.user = request.user.user
                    review.save()
            elif 'delete_review' in request.POST:
                review_id = request.POST.get('delete_review')
                review = get_object_or_404(Review, review_id=review_id)
                review.delete()

        return redirect(film.get_absolute_url())

    def get(self, request, slug):
        film = get_object_or_404(Film, id=slug)
        reviews = film.review_set.all()
        form = ReviewForm()
        context = {'film': film, 'reviews': reviews, 'form': form}

```

```
return render(request, self.template_name, context)
```

Листинг Б.5 – Представление добавление отзыва

```
class AddStarRating(View):
    """Добавление рейтинга фильму"""

    template_name = "films/film_detail.html"

    def post(self, request, slug):
        form = RatingForm(request.POST)
        film = get_object_or_404(Film, url=slug)

        if form.is_valid():
            if request.user.is_authenticated:
                user = User.objects.get(user=request.user)
                Score.objects.update_or_create(
                    user=user,
                    film=film,
                    defaults={'star': form.cleaned_data['star']}
                )
            return redirect(film.get_absolute_url())
```

Листинг А.6 – Представление добавление рейтинга фильму

```
class MarkAsViewedView(View):
    template_name = "films/film_detail.html"

    def post(self, request, film_id):
        film = get_object_or_404(Film, id=film_id)
        form = ViewedForm(request.POST or None)
        user = User.objects.get(user=request.user)
        if form.is_valid():
            if request.user.is_authenticated:
                if film in user.views.all():
                    user.views.remove(film)
                else:
                    user.views.add(film)
            return redirect(film.get_absolute_url())
```

Листинг А.7 – Представление добавление фильма в «Просмотренное»

```
class AddToFavoriteView(View):
    template_name = "films/film_detail.html"

    def post(self, request, film_id):
        film = get_object_or_404(Film, id=film_id)
        form = FavoriteForm(request.POST or None)
        user = User.objects.get(user=request.user)
        if form.is_valid():
            if request.user.is_authenticated:
                if film in user.favorite.all():
                    user.favorite.remove(film)
                else:
```



```

        user.favorite.add(film)
    return redirect(film.get_absolute_url())

```

Листинг Б.7 – Представление добавление фильма в «Избранное»

```

def sign_in(request):
    if request.method == 'GET':
        if request.user.is_authenticated:
            return redirect('/')

    form = LoginForm()
    return render(request, 'registration/login.html', {'form': form})

    elif request.method == 'POST':
        form = LoginForm(request.POST)

        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']
            user = authenticate(request, username=username, password=password)
            if user:
                login(request, user)
                messages.success(request, f'Здравствуйте {username.title()},
добро пожаловать!')
                return redirect('/')

            messages.error(request, f'Неверное имя пользователя или пароль')
            return render(request, 'registration/login.html', {'form': form})

```

Листинг А.8 – Представление входа в аккаунт

```

def sign_out(request):
    logout(request)
    messages.success(request, f'Вы вышли из системы')
    return redirect('/')

```

Листинг А.9 – Представление выхода из аккаунта

```

def sign_up(request):
    if request.method == 'GET':
        form = RegisterForm()
        return render(request, 'registration/register.html', {'form': form})

    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            messages.success(request, 'Вы успешно зарегистрировались')
            return redirect('/')
        else:
            return render(request, 'registration/register.html', {'form': form})

```

Листинг А.10 – Представление регистрации

```

@login_required
def user_profile(request):
    user_profile = get_object_or_404(User, user=request.user)

```

```
return render(request, 'profile/profile.html', {'profile': user_profile})
```

Листинг А.11 – Представление вывода страницы пользователя

```
@login_required
def profile_edit(request):
    user_instance = request.user
    if request.method == 'POST':
        form = UserProfileForm(request.POST, request.FILES,
instance=user_instance)
        if form.is_valid():
            form.save()
            return render(request, 'profile/profile_edit.html')
    else:
        form = UserProfileForm(instance=user_instance)
    return render(request, 'profile/profile_edit.html', {'form': form})
```

Листинг Б.12 – Представление редактирование профиля пользователя

```
def generate_pdf_report(request, profile_id):

    pdfmetrics.registerFont(TTFont('MyFont', 'static/font.ttf'))

    try:
        profile = User.objects.get(id=profile_id)
    except User.DoesNotExist:
        return HttpResponse("User does not exist.")

    buffer = BytesIO()
    pdf = canvas.Canvas(buffer, pagesize=A4)
    pdf.setFont('MyFont', 12)
    pdf.drawString(1*inch, 10*inch, "Отчёт о пользователе:
{}".format(profile.surname + " " + profile.name + " " + profile.lastname))
    pdf.drawString(1*inch, 9*inch, "Логин: {}".format(profile.login))
    pdf.drawString(1*inch, 8*inch, "Email: {}".format(profile.email))
    pdf.drawString(1*inch, 7*inch, "Дата рождения:
{}".format(profile.date_of_birth))
    pdf.drawString(1*inch, 6*inch, "Пол: {}".format(profile.gender))

    favorite_films = profile.get_favorite()
    viewed_films = profile.get_views()
    scores = profile.get_scores()

    data = [["Избранное:"]]
    data.extend([[f"Фильм: {film.name}"] for film in favorite_films])
    table = Table(data, colWidths=[6*inch])
    table.setStyle([
        ('FONTNAME', (0,0), (-1,-1), 'MyFont'),
        ('FONTSIZE', (0,0), (-1,-1), 12),
        ('TEXTCOLOR', (0,0), (-1,-1), colors.black),
        ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
        ('ALIGN', (0,0), (-1,-1), 'LEFT'),
    ])
    ))
```

```

table.wrapOn(pdf, 6*inch, 4*inch)
table.drawOn(pdf, 1*inch, 5*inch)

data = [["Просмотренное:"]]
data.extend([[f"Фильм: {film.name}"] for film in viewed_films])
table = Table(data, colWidths=[6*inch])
table.setStyle([
    ('FONTNAME', (0,0), (-1,-1), 'MyFont'),
    ('FONTSIZE', (0,0), (-1,-1), 12),
    ('TEXTCOLOR', (0,0), (-1,-1), colors.black),
    ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
    ('ALIGN', (0,0), (-1,-1), 'LEFT'),
])
table.wrapOn(pdf, 6*inch, 4*inch)
table.drawOn(pdf, 1*inch, 3*inch)

data = [["Оцененное:"]]
data.extend([[f"Фильм: {score.film.name}", f"Оценка: {score.star.value}"] for
score in scores])
table = Table(data, colWidths=[3*inch, 3*inch])
table.setStyle([
    ('FONTNAME', (0,0), (-1,-1), 'MyFont'),
    ('FONTSIZE', (0,0), (-1,-1), 12),
    ('TEXTCOLOR', (0,0), (-1,-1), colors.black),
    ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
    ('ALIGN', (0,0), (-1,-1), 'LEFT'),
])
table.wrapOn(pdf, 6*inch, 4*inch)
table.drawOn(pdf, 1*inch, 1*inch)

pdf.save()
buffer.seek(0)
return HttpResponse(buffer.getvalue(), content_type='application/pdf')

```

Листинг А.12 – Представление составление отчёта «Сведение о фильме»

```

@login_required
def generate_film_pdf_report(request, film_id):

    pdfmetrics.registerFont(TTFont('MyFont', 'static/font.ttf'))

    try:
        film = Film.objects.get(id=film_id)
    except Film.DoesNotExist:
        return HttpResponse("Film does not exist.")

    buffer = BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=A4)
    elements = []

```

```

# Set up styles
styles = getSampleStyleSheet()

# Define custom styles with the custom font
normal_style = ParagraphStyle(
    'Normal',
    parent=styles['Normal'],
    fontName='MyFont',
    fontSize=12
)

heading_style = ParagraphStyle(
    'Heading2',
    parent=styles['Heading2'],
    fontName='MyFont',
    fontSize=18
)

# Title
title = Paragraph(f"{film.name} ({film.year})", heading_style)
elements.append(title)

# Film details
details = [
    f"Год: {film.year}",
    f"Страна: {'', '.join(country.name for country in film.countries.all())}",
    f"Слоган: {film.slogan}",
    f"Жанры: {'', '.join(genre.name for genre in film.genres.all())}",
    f"Дата выхода (Мир): {film.release_date_world}",
    f"Дата выхода (Россия): {film.release_date_russia}",
    f"Бюджет: {film.budget} $",
    f"Кассовый сбор: {film.gross} $",
    f"Рейтинг: {film.get_average_rating()}",
    f"Количество отзывов: {film.get_review_count()}",
    f"Количество оценок: {film.get_score_count()}",
    f"Добавлено в избранное: {film.get_favorites_count()}",
    f"Просмотрено: {film.get_watched_count()}",
]

for detail in details:
    elements.append(Paragraph(detail, normal_style))

elements.append(Paragraph(f"О фильме {film.name}", heading_style))
elements.append(Paragraph(film.description, normal_style))

doc.build(elements)
buffer.seek(0)
return HttpResponse(buffer.getvalue(), content_type='application/pdf')

```

Листинг А.13 – Представление составление отчёта «Сведения об пользователе»

## ПРИЛОЖЕНИЕ Б

### Листинг html шаблонов для клиентской части

```
{% extends 'films/base.html' %}
{% load i18n %}
{% block title %} {% get_current_language as LANGUAGE_CODE %} {% if LANGUAGE_CODE
== 'ru' %} Кинотека - Смотри Фильмы бесплатно {% else %} Kinoteka - Watch Movies
for free {% endif %}{% endblock title %}
{% block header %} {% endblock header %}
{% block film %}
    <div class="left-ads-display col-lg-9">
        <div class="row">
            {% for film in film_list %}
                <div class="col-md-4 product-men">
                    <div class="product-shoe-info editContent text-center mt-lg-
4">
                        <div class="men-thumb-item">
                            
                        </div>
                        <div class="item-info-product">
                            <h4 class="">
                                <a href="{ { film.get_absolute_url }}"
class="editContent">
                                    { { film.name } } { {'(' } { { film.year } } { {')' } }
                                </a>
                            </h4>
                            <div class="product_price">
                                <div class="grid-price">
                                    <span class="money editContent">{ {
film.slogan } }</span>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            {% endfor %}
        </div>
        <div class="grid-img-right mt-4 text-right bg bg1" >
            {% include 'include/pagination.html' %}
        </div>
    </div>
{% endblock film %}
```

### Листинг Б.1 – html шаблон страницы фильмов

```
{% extends 'films/base.html' %}
{% load i18n film_tag %}
{% block title %} { { film.name } } ( { { film.year } } ) {% endblock title %}
{% block header %} {% endblock header %}
{% block container %}
    <div class="container py-md-3">
```

```

{% endblock container %}

{% block sidebar %}
    <div class="side-bar col-lg-4">
{% endblock sidebar %}

{% block film %}
    <div class="left-ads-display col-lg-8">
        <div class="row">
            <div class="desc1-left col-md-6">
                
                <a href="{{ film.video_url }}" target="_blank" class="btn btn-primary mt-2">
                    {% if LANGUAGE_CODE == 'ru' %} Посмотреть фильм {% else %}
Watch Movie {% endif %}
                </a>
                <a href="{{ film.trailer_url }}" target="_blank" class="btn btn-secondary mt-2">
                    {% if LANGUAGE_CODE == 'ru' %} Посмотреть трейлер {% else %}
Watch Trailer {% endif %}
                </a>
                <a href="{% url 'generate_film_pdf_report' film.id %}" class="btn btn-secondary mt-2">Сгенерировать отчёт в PDF</a>
            </div>
            <div class="desc1-right col-md-6 pl-lg-4">
                <h3 class="editContent" style="outline: none; cursor: inherit;">
                    {{ film.name }} ({{ film.year }})
                </h3>
                <h5 class="editContent">{{ film.name_en }}</h5>
                <ul>
                    <li style="list-style: none">
                        {% get_current_language as LANGUAGE_CODE %}
                        <span><b>{% if LANGUAGE_CODE == 'ru' %} Год {% else %}
Year {% endif %}</b>{{ film.year }}</span>
                    </li>
                    <li style="list-style: none">
                        <span><b>{% if LANGUAGE_CODE == 'ru' %} Страна: {% else %}
%} Country: {% endif %}</b>
                            {% for country in film.countries.all %}
                                {{ country.name }}{% if not forloop.last %} , {%
else %}. {% endif %}
                            {% endfor %}
                        </span>
                    </li>
                    <li style="list-style: none">
                        <span><b>{% if LANGUAGE_CODE == 'ru' %} Слоган: {% else %}
%} Tagline: {% endif %}</b> {{ film.slogan }}</span>
                    </li>
                </ul>
            </div>
        </div>
    </div>

```

```

        <li style="list-style: none">
            <span><b>{% if LANGUAGE_CODE == 'ru' %} Режиссёр: {% else
}% Director: {% endif %}</b>
                {% for film_member_post in
film.filmmemberpost_set.all %}
                    {% if film_member_post.post.name == 'Режиссёр' or
film_member_post.post.name == 'Director' %}
                        <a href="{%
film_member_post.member.get_absolute_url %}">
                            {{ film_member_post.member.name }} {{
film_member_post.member.surname }} {{ film_member_post.member.last_name }}{% if
not forloop.last %} , {% else %}. {% endif %}
                        </a>
                    {% endif %}
                {% endfor %}
            </span>
        </li>
        <li style="list-style: none">
            <span><b>{% if LANGUAGE_CODE == 'ru' %} Оператор: {% else
}% Operator: {% endif %}</b>
                {% for film_member_post in
film.filmmemberpost_set.all %}
                    {% if film_member_post.post.name == 'Оператор' or
film_member_post.post.name == 'Operator' %}
                        <a href="{%
film_member_post.member.get_absolute_url %}">
                            {{ film_member_post.member.name }} {{
film_member_post.member.surname }} {{ film_member_post.member.last_name }}{% if
not forloop.last %} , {% else %}. {% endif %}
                        </a>
                    {% endif %}
                {% endfor %}
            </span>
        </li>
        <li style="list-style: none">
            <span><b>{% if LANGUAGE_CODE == 'ru' %} Сценарий: {% else
}% Scenario: {% endif %}</b>
                {% for film_member_post in
film.filmmemberpost_set.all %}
                    {% if film_member_post.post.name == 'Сценарист'
or film_member_post.post.name == 'Scenarist' %}
                        <a href="{%
film_member_post.member.get_absolute_url %}">
                            {{ film_member_post.member.name }} {{
film_member_post.member.surname }} {{ film_member_post.member.last_name }}{% if
not forloop.last %} , {% else %}. {% endif %}
                        </a>
                    {% endif %}
                {% endfor %}
            </span>
        </li>
    
```

```

        </span>
    </li>
    <li style="list-style: none">
        <span><b>{% if LANGUAGE_CODE == 'ru' %} Композитор: {%
else %} Compositor: {% endif %}</b>
            {% for film_member_post in
film.filmmemberpost_set.all %}
                {% if film_member_post.post.name == 'Композитор'
or film_member_post.post.name == 'Compositor' %}
                    <a href="{%
film_member_post.member.get_absolute_url %}">
                        {{ film_member_post.member.name }} {{
film_member_post.member.surname }} {{ film_member_post.member.last_name }}{% if
not forloop.last %} , {% else %}. {% endif %}
                    </a>
                {% endif %}
            {% endfor %}
        </span>
    </li>
    <li style="list-style: none">
        <span><b>{% if LANGUAGE_CODE == 'ru' %} Актёр: {% else %}
Actor: {% endif %}</b>
            {% for film_member_post in
film.filmmemberpost_set.all %}
                {% if film_member_post.post.name == 'Актёр' or
film_member_post.post.name == 'Actor'%}
                    <a href="{%
film_member_post.member.get_absolute_url %}">
                        {{ film_member_post.member.name }} {{
film_member_post.member.surname }} {{ film_member_post.member.last_name }}{% if
not forloop.last %} , {% else %}. {% endif %}
                    </a>
                {% endif %}
            {% endfor %}
        </span>
    </li>
    <li style="list-style: none">
        <span><b>{% if LANGUAGE_CODE == 'ru' %} Жанр: {% else %}
Genres: {% endif %}</b>
            {% for genre in film.genres.all %}
                {{ genre.name }}{% if not forloop.last %} , {% else
%}. {% endif %}
            {% endfor %}
        </span>
    </li>
    <li style="list-style: none">
        <span><b>{% if LANGUAGE_CODE == 'ru' %} Премьера в мире:
{% else %} Release date (World) {% endif %}</b> {{ film.release_date_world
}}</span>
    </li>

```



```

        <li style="list-style: none">
            <span><b>{% if LANGUAGE_CODE == 'ru' %} Премьера в
России: {% else %} Release date (Russia) {% endif %} </b> {{
film.release_date_russia }}</span>
        </li>
        <li style="list-style: none">
            <span><b>{% if LANGUAGE_CODE == 'ru' %} Бюджет: {% else
%} Budget {% endif %} </b> {{ film.budget }} $</span>
        </li>
        <li style="list-style: none">
            <span><b>{% if LANGUAGE_CODE == 'ru' %} Сборы: {% else %}
Gross worldwide: {% endif %} </b> {{ film.gross }} $</span>
        </li>
        <li style="list-style: none">
            <span><b>{% if LANGUAGE_CODE == 'ru' %} Рейтинг: {% else
%} Rating: {% endif %} </b>{{film.get_average_rating}}</span>
        </li>
        <li>
            <form action="{% url 'add_rating' film.url %}"
method="post" name="rating">
                {% csrf_token %}
                <span class="rating">
                    {% for k, v in star_form.fields.star.choices %}
                        <input id="rating{{ v }}" type="radio"
name="star" value="{{ k }}">
                            <label for="rating{{ v }}">{{ k }}</label>
                    {% endfor %}
                </span>
            </form>
            <script>
                const ratingInputs =
document.querySelectorAll('input[name="star"]');
                const ratingForm =
document.querySelector('form[name="rating"]');
                ratingInputs.forEach(input => {
                    input.addEventListener('click', () => {
                        ratingForm.submit();
                    });
                });
            </script>
        </li>
        <li>
            <form action="{% url 'add_to_favorite' film.id %}"
method="post">
                {% csrf_token %}
                <button type="submit" class="btn btn-outline-danger">
                    {% if film in user.get_favorite %}
                        {% if LANGUAGE_CODE == 'ru' %} Удалить из
избранного {% else %} Delete Favorite {% endif %}
                    {% else %}

```

```

                                {% if LANGUAGE_CODE == 'ru' %} Добавить в
избранное {% else %} Add Favorite {% endif %}
                                {% endif %}
                                </button>
                                </form>
                                </li>
                                <li>
                                    <form action="{% url 'mark_as_viewed' film.id %}"
method="post">
                                        {% csrf_token %}
                                        <button type="submit" class="btn btn-outline-
primary">
                                            {% if film in user.get_views %}
                                                {% if LANGUAGE_CODE == 'ru' %} Удалить из
просмотренного {% else %} Delete Viewed {% endif %}
                                                {% else %}
                                                    {% if LANGUAGE_CODE == 'ru' %} Добавить в
просмотренное {% else %} Add Viewed {% endif %}
                                                    {% endif %}
                                                </button>
                                            </form>
                                        </li>
                                </div>
                                </div>
                                <div class="row sub-para-w3layouts mt-5">
                                    <h3 class="shop-sing editContent" style="outline: none; cursor:
inherit;">
                                        {% if LANGUAGE_CODE == 'ru' %} О фильме {% else %} About Movie {%
endif %} {{ film.name }}
                                    </h3>
                                    <p>
                                        {% for image in film.photo_set.all %}
                                            
                                        {% endfor %}
                                    </p>
                                    <p class="editContent" style="outline: none; cursor: inherit;">
                                        {{ film.description|safe }}
                                    </p>
                                </div>
                                <div class="row">
                                    <div class="single-form-left">
                                        <div class="contact-single">
                                            <h3 class="editContent" style="outline: none; cursor:
inherit;">
                                                <span class="sub-tittle editContent"
style="outline: none; cursor: inherit;">
                                                    </span>{% if LANGUAGE_CODE == 'ru' %} Оставить отзыв {%
else %} Leave Review {% endif %}
                                            </h3>

```

```

        <form action="{% url 'add_review' film.id %}" method="post"
class="mt-4">
            {% csrf_token %}
            <div class="form-group editContent"
                <label for="contactcomment" class="editContent"
                    {% if LANGUAGE_CODE == 'ru' %} Ваш комментарий *
{% else %} Your Comment {% endif %}
                </label>
                <textarea class="form-control-review border" rows="5"
name="text"
                    id="contactcomment" required=""></textarea>
            </div>
            <button type="submit"
                name="submit_review"
                class="mt-3 btn btn-success btn-block py-3"
                style="outline: none; cursor: inherit;">
                {% if LANGUAGE_CODE == 'ru' %} Отправить {% else
%} Send {% endif %}
            </button>
        </form>
    </div>
</div>
</div>
{% for review in film.get_review %}
    <div class="media py-5">
        
        <div class="media-body mt-4">
            <h5 class="mt-0 editContent"
                style="outline: none; cursor: inherit;">
                {{ review.user.name }} {{ review.user.surname }}
            </h5>
            <p class="mt-2 editContent">
                {{ review.text }}
            </p>
            {% if request.user.is_authenticated and review.user ==
request.user.user or request.user.is_superuser %}
                <form action="{% url 'add_review' film.id %}"
method="post">
                    {% csrf_token %}
                    <!-- <button type="submit" name="edit_review"
value="{{ review.review_id }}">Редактировать</button> -->
                    <button class="btn btn-delete" type="submit"
name="delete_review" value="{{ review.review_id }}">Удалить</button>
                </form>
            {% endif %}
        </div>
    </div>

```

```

        </div>
    {% endfor %}
    <div class="grid-img-right mt-4 text-right bg bg1" >
        {% include 'include/pagination.html' %}
    </div>
</div>
</div>
{% endblock film %}

```

Листинг Б.2 – html шаблон страницы фильма

```

{% extends 'films/base.html' %}
{% block title %}{{ member.name}} {{ member.surname}} {{ member.last_name}} {%
endblock title %}
{% block container %}
    <div class="container py-md-3">
{% endblock container %}

{% block sidebar %}
    <div class="side-bar col-lg-4">
{% endblock sidebar %}

{% block film %}
    <div class="left-ads-display col-lg-8">
        <div class="row">
            <div class="desc1-left col-md-6">
                
            </div>
            <div class="desc1-right col-md-6 pl-lg-4">
                <h4 class="editContent" style="outline: none; cursor: inherit;">
                    {{ member.name}} {{ member.surname}} {{ member.last_name}}
                </h4>
                <h5 class="editContent">
                    {{ member.name_en}} {{ member.surname_en}} {{
member.last_name_en}}
                </h5>
                <ul>
                    <li style="list-style: none">
                        <span><b>Карьера:</b>
                            {% for post in member.members_posts.all.distinct %}
                                {{post.name}}
                            {% endfor %}
                        </span>
                    </li>
                    <li style="list-style: none">
                        <span><b>Дата рождения :</b>
                            {{ member.date_of_birth }}
                        </span>
                    </li>
                    <li style="list-style: none">
                        <span><b>Пол:</b> {{ member.gender }}</span>

```

```

        </li>
    </div>
</div>
<div class="row sub-para-w3layouts mt-5">
    <h3 class="shop-sing editContent" style="outline: none; cursor:
inherit;">
        06 Актёре:
    </h3>
    <p class="editContent" style="outline: none; cursor: inherit;">
        {{ member.description|safe }}
    </p>
</div>
</div>

```

{% endblock film %}

Листинг Б.3 – html шаблон страницы актёра

```

{% extends 'base.html' %}
{% load static %}
{% block title %}{{ profile.name }} {{ profile.surname }} {{ profile.lastname }} {%
endblock title %}

```

```

{% block content %}
<div class="container py-5">
    <div class="row">
        {% block sidebar %}
        <div class="col-md-4 mb-4 mb-md-0">
            <div class="card h-100">
                
                <div class="card-body d-flex flex-column">
                    <h5 class="card-title mb-3">{{ profile.name }} {{
profile.surname }} {{ profile.lastname }}</h5>
                    <p class="card-text mb-auto">
                        <b>Логин:</b> {{ profile.login }}<br>
                        <b>Email:</b> {{ profile.email }}<br>
                        <b>Дата рождения:</b> {{ profile.date_of_birth }}<br>
                        <b>Пол:</b> {{ profile.gender }}
                    </p>
                    <a href="{% url 'profile_edit' %}" class="btn btn-primary mt-
auto">Редактировать профиль</a>
                    <a href="{% url 'generate_pdf_report' profile.id %}"
class="btn btn-secondary mt-2">Сгенерировать отчёт в PDF</a>
                </div>
            </div>
        </div>
        {% block profile %}
        <div class="col-md-8">
            <div class="card">

```

```

        <div class="card-body">
            <div class="row row-cols-1 row-cols-md-3">
                <div class="col mb-4">
                    <h6 class="mb-3">Избранное:</h6>
                    <ul class="list-group list-group-flush">
                        {% for film in profile.get_favorite %}
                            <li class="list-group-item"><a href="{{
film.get_absolute_url }}">{{ film.name }}</a></li>
                        {% endfor %}
                    </ul>
                </div>
                <div class="col mb-4">
                    <h6 class="mb-3">Просмотренное:</h6>
                    <ul class="list-group list-group-flush">
                        {% for film in profile.get_views %}
                            <li class="list-group-item"><a href="{{
film.get_absolute_url }}">{{ film.name }}</a></li>
                        {% endfor %}
                    </ul>
                </div>
                <div class="col mb-4">
                    <h6 class="mb-3">Оцененное:</h6>
                    <ul class="list-group list-group-flush">
                        {% for score in profile.get_scores %}
                            <li class="list-group-item"><a href="{{
score.film.get_absolute_url }}">{{ score.film }} </a><a>- {{ score.star.value
}}</a></li>
                        {% endfor %}
                    </ul>
                </div>
            </div>
        </div>
    {% endblock profile %}
</div>
{% endblock content %}

```

Листинг Б.4 – html шаблон страницы профиля

```

{% extends 'registration/base.html' %}
{% load i18n film_tag %}
{% block title %}{% if LANGUAGE_CODE == 'ru' %} Вход {% else %} Login {% endif
%}{% endblock title %}
{% block header %}{% endblock header %}
{% block authorization %}
<div class="container d-flex justify-content-center align-items-center"
style="min-height: 10vh;">
    <div class="card p-4 shadow" style="max-width: 400px; width: 100%;">
        {% get_current_language as LANGUAGE_CODE %}

```

```

        <h2 class="text-center mb-4">{% if LANGUAGE_CODE == 'ru' %} Вход {% else
    %} Login {% endif %}</h2>
        <form method="post">
            {% csrf_token %}
            <div class="form-group mb-3">
                <label for="{{ form.username.id_for_label }}">{% if LANGUAGE_CODE
    == 'ru' %} Имя пользователя: {% else %} Username: {% endif %}</label>
                {{ form.username|add_class:"form-control mx-auto" }}
            </div>
            <div class="form-group mb-3">
                <label for="{{ form.password.id_for_label }}">{% if LANGUAGE_CODE
    == 'ru' %} Пароль {% else %} Password {% endif %}</label>
                {{ form.password|add_class:"form-control mx-auto" }}
            </div>
            <button type="submit" class="btn btn-danger btn-block">{% if
    LANGUAGE_CODE == 'ru' %} Войти {% else %} Login {% endif %}</button>
        </form>
    </div>
</div>
{% endblock %}

```

Листинг Б.5 – html шаблон страницы авторизации

```

{% extends 'registration/base.html' %}
{% load i18n film_tag %}
{% block title %}{% if LANGUAGE_CODE == 'ru' %} Регистрация {% else %} SingUp {%
    endif %}{% endblock title %}
{% block header %}{% endblock header %}
{% block authorization %}
<div class="container d-flex justify-content-center align-items-center"
    style="min-height: 10vh;">
    <div class="card p-4 shadow" style="max-width: 500px; width: 100%;">
        {% get_current_language as LANGUAGE_CODE %}
        <h2 class="text-center mb-4">{% if LANGUAGE_CODE == 'ru' %} Регистрация
    {% else %} SignUp {% endif %}</h2>
        <form method="post" class="text-center">
            {% csrf_token %}
            <div class="form-group mb-3">
                <label for="{{ form.username.id_for_label }}" class="text-left w-
    100">{% if LANGUAGE_CODE == 'ru' %} Имя пользователя: {% else %} login {% endif
    %}</label>
                {{ form.username|add_class:"form-control mx-auto" }}
            </div>
            <div class="form-group mb-3">
                <label for="{{ form.password1.id_for_label }}" class="text-left
    w-100">{% if LANGUAGE_CODE == 'ru' %} Пароль {% else %} Password {% endif
    %}</label>
                {{ form.password1|add_class:"form-control mx-auto" }}
            </div>
            <div class="form-group mb-3">

```

```

        <label for="{{ form.password2.id_for_label }}" class="text-left
w-100">{% if LANGUAGE_CODE == 'ru' %} Подтверждение пароля: {% else %} Confirm
Password {% endif %}</label>
        {{ form.password2|add_class:"form-control mx-auto" }}
    </div>
    <div class="form-group mb-3">
        <label for="{{ form.email.id_for_label }}" class="text-left w-
100">{% if LANGUAGE_CODE == 'ru' %} Почта: {% else %} Email {% endif %}</label>
        {{ form.email|add_class:"form-control mx-auto" }}
    </div>
    <div class="form-group mb-3">
        <label for="{{ form.name.id_for_label }}" class="text-left w-
100">{% if LANGUAGE_CODE == 'ru' %} Имя: {% else %} Name {% endif %}</label>
        {{ form.name|add_class:"form-control mx-auto" }}
    </div>
    <div class="form-group mb-3">
        <label for="{{ form.surname.id_for_label }}" class="text-left w-
100">{% if LANGUAGE_CODE == 'ru' %} Фамилия: {% else %} Surname: {% endif
%}</label>
        {{ form.surname|add_class:"form-control mx-auto" }}
    </div>
    <div class="form-group mb-3">
        <label for="{{ form.lastname.id_for_label }}" class="text-left w-
100">{% if LANGUAGE_CODE == 'ru' %} Отчество (Опционально): {% else %} Lastname
(Optional) {% endif %}</label>
        {{ form.lastname|add_class:"form-control mx-auto" }}
    </div>
    <button type="submit" class="btn btn-danger btn-block">{% if
LANGUAGE_CODE == 'ru' %} Зарегистрироваться {% else %} SingUp {% endif
%}</button>
    </form>
</div>
</div>
{% endblock %}

```

Листинг Б.6 – html шаблон страницы регистрации

## ПРИЛОЖЕНИЕ В

Листинг тестов для информационной системы

```

class TestForms(TestCase):

    def test_register_form_valid_data(self):
        form = RegisterForm(data={
            'username' : 'username',
            'password1' : 'password',
            'password2' : 'password',
            'email': 'email',
            'name': 'name',
            'surname': 'surname',

```



```

        'lastname': 'lastname'
    })

    self.assertTrue(form.is_valid())

def test_register_form_has_no_data(self):
    form = RegisterForm(data={})

    self.assertFalse(form.is_valid())
    self.assertEqual(len(form.errors), 7)

def test_login_form_valid_data(self):
    form = LoginForm(data={
        "username" : 'username',
        'password' : 'password',
    })

    self.assertTrue(form.is_valid())

def test_login_form_has_no_data(self):
    form = LoginForm(data={})

    self.assertFalse(form.is_valid())
    self.assertEqual(len(form.errors), 2)

```

Листинг В.1 – Тестирование форм

```

class TestViews(TestCase):
    def setUp(self):
        self.client = Client()
        self.list_url = reverse('list')
        self.detail_url = reverse('film_detail', args=['film_slug'])
        self.film = Film.objects.create(name='film', url = "film_slug")

    def test_film_list_get(self):
        response = self.client.get(self.list_url)

        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, "films/film_list.html")

    def test_film_detail_get(self):
        response = self.client.get(self.detail_url)

        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, "films/film_detail.html")

```

Листинг В.2 – Тестирование представлений

```

class TestUrls(SimpleTestCase):

```

```

def test_film_list(self):
    url = reverse('list')
    self.assertEqual(resolve(url).func.view_class, MovieView)

def test_filter(self):
    url = reverse('filter')
    self.assertEqual(resolve(url).func.view_class, FilterMovieView)

def test_search(self):
    url = reverse('search')
    self.assertEqual(resolve(url).func.view_class, Search)

def test_film_detail(self):
    url = reverse('film_detail', args=['some-slug'])
    self.assertEqual(resolve(url).func.view_class, MovieDetailView)

def test_add_rating(self):
    url = reverse('add-rating', args=['some-slug'])
    self.assertEqual(resolve(url).func.view_class, AddStarRating)

def test_member_detail(self):
    url = reverse('member_detail', args=['some-slug'])
    self.assertEqual(resolve(url).func.view_class, MemberView)

def test_add_review(self):
    url = reverse('add_review', args=[0])
    self.assertEqual(resolve(url).func.view_class, AddReviewView)

def test_add_favorite(self):
    url = reverse('add_to_favorite', args=[0])
    self.assertEqual(resolve(url).func.view_class, AddToFavoriteView)

def test_add_rating(self):
    url = reverse('mark_as_viewed', args=[0])
    self.assertEqual(resolve(url).func.view_class, MarkAsViewedView)

```

Листинг В.3 – Тестирование ссылок

```

class TestModels(TestCase):

    def setUp(self):
        self.film = Film.objects.create(name='film', url = "film_slug")
        self.user = User.objects.create(name='user', surname
='surname', login='login', password = 'password', email = 'email')
        self.member = Member.objects.create(name='user', surname='surname')

    def test_film_has_already_poster_on_default(self):
        self.assertEqual(self.film.poster, "films/default_film_image.jpg")

    def test_member_has_already_photo_on_default(self):

```

```

self.assertEqual(self.member.photo,"members/default_member_image.jpeg")

def test_user_has_already_photo_on_default(self):
    self.assertEqual(self.user.photo,"users/default_user_image.jpeg")

def test_film_average_rating(self):
    self.assertEqual(self.film.get_average_rating(),0)

def test_user_add_views(self):

    self.user.views.add(self.film)

    self.assertEqual(len(self.user.views.all()),1)

```

#### Листинг В.4 – Тестирование моделей

### ПРИЛОЖЕНИЕ Г

#### Руководство пользователя

##### Г.1 Назначение системы

Информационная система «Онлайн-кинотеатр» позволяет пользователю смотреть нужную информацию о фильмах, смотреть их и делиться впечатлениями. В системе реализовано 3 типа пользователя: незарегистрированный пользователь, зарегистрированный пользователь, модератор.

##### Г.2 Условия работы системы

Для корректной работы информационной системы необходимо наличие соответствующих программных и аппаратных средств. Для корректного функционирования клиентской части необходимо:

- 1) Требования к техническому обеспечению:
  - тип ЭВМ: x86-64 совместимый;
  - процессор с тактовой частотой 2,1 ГГц или выше;
  - клавиатура или иное устройство ввода;
  - мышь или другое манипулирующее ввода;
  - размер ОП – 8 Гб, тип DDR4, частота памяти 2400 МГц;
- 2) Требования к программному обеспечению:
  - ОС Windows 7,8,10,11, имеющая браузер;
  - Google Chrome 96+, Yandex-браузер 22.1.0.2510+, Microsoft Edge 113.0.1774.50.

Для корректного функционирования серверной части необходимо:

- 1) Требования к техническому обеспечению:
  - тип ЭВМ: x86-64 совместимый;

- процессор с тактовой частотой 2,1 ГГц или выше;
- клавиатура или иное устройство ввода;
- мышь или другое манипулирующее ввода;
- дисплей с разрешением не менее 1920x1080 пикселей;
- размер ОП – 8 Гб, тип DDR4, частота памяти 2400 МГц;

## 2) Требования к программному обеспечению:

- операционная система Windows 10,11;
- браузер Google Chrome 96+, Yandex-браузер 22.1.0.2510+, Microsoft Edge 113.0.1774.50;
- СУБД PostgreSQL 16 и выше;
- Python 3.12.2 и выше;
- PyCharm Community Edition 2021.3.2 и выше;

## Г.3 Установка системы

Система поставляется в виде git-репозиторий. Данный репозиторий можно скачать себе на машину, прописав команду `git clone https://github.com/Jinqu3/OnlineCinema.git -r requirements.txt`.

После, как только репозиторий клонировался на машину, открыть среду разработки Visual Studio Code и в файле `settings.py` необходимо настроить подключение с базой данных с помощью параметра `DATABASES`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'наименование базы данных',
        'USER': 'пользователь в СУБД',
        'PASSWORD': 'пароль',
        'HOST': '127.0.0.1',
        'PORT': '5432'
    }
}
```

После этого с помощью команды `python manage.py migrate` необходимо выполнить миграции. И создать пользователя с правами администратора с помощью команды `python manage.py createsuperuser`.

## Г.4 Работа с системой

### Г.4.1 Регистрация в системе

Для пользователя регистрация осуществляется по ссылке «register/» на странице регистрации (см. Рисунок Г.1). Здесь необходимо заполнить такие поля как логин, имя, фамилия, отчество(при наличии) пользователя, адрес электронной почты, пароль и подтверждение пароля.

**Регистрация**

Имя пользователя:

Пароль:

Подтверждение пароля:

Почта:

Имя:

Фамилия:

Отчество (Опционально):

**Зарегистрироваться**

Рисунок Г.1 – Страница регистрации пользователя

Вход в систему осуществляется по ссылке «/login», на странице необходимо ввести свой логин и пароль (см. Рисунок Г.2).

**Вход**

Имя пользователя:

Пароль:

**Войти**

Рисунок Г.2 – Страница регистрации входа

#### Г.4.2 Главная страница

На главной странице есть возможность просматривать информацию о различных фильмах, осуществлять поиск по названию фильма, фильтровать заведения по жанру, году и т.д (см. Рисунок Г.3).

Также при нажатии на кнопку «Забронировать» незарегистрированный пользователь получит сообщение с предупреждением. А зарегистрированный пользователь, после авторизации, имеет возможность забронировать столик. Аналогичным образом на странице с информацией о заведении при нажатии на кнопку «Добавить в избранное» незарегистрированный пользователь получит сообщение с предупреждением. А

зарегистрированный пользователь, после авторизации, имеет возможность добавить информацию о заведении в избранное (см. Рисунок Г.5).

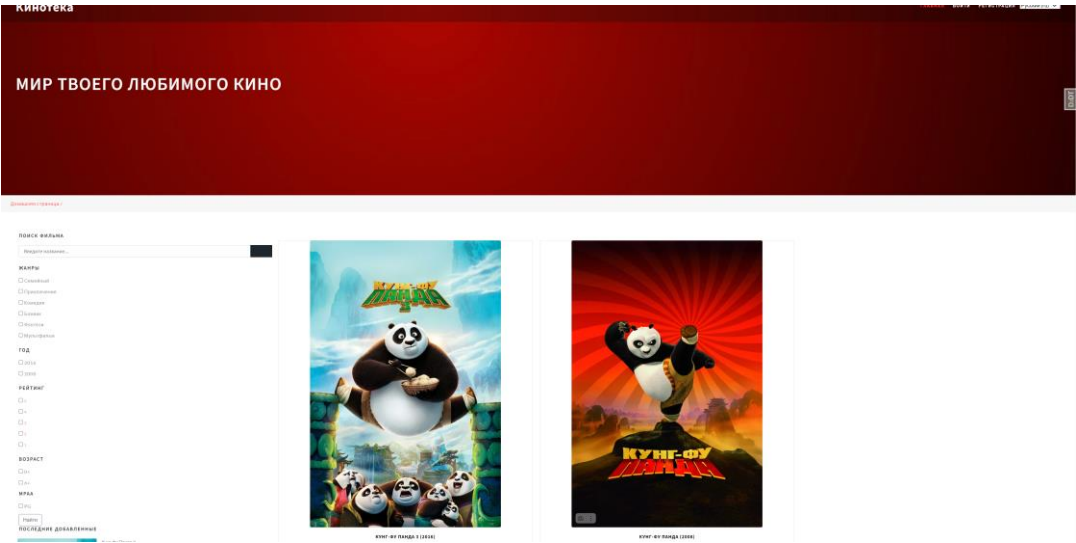


Рисунок Г.4 – Главная страница

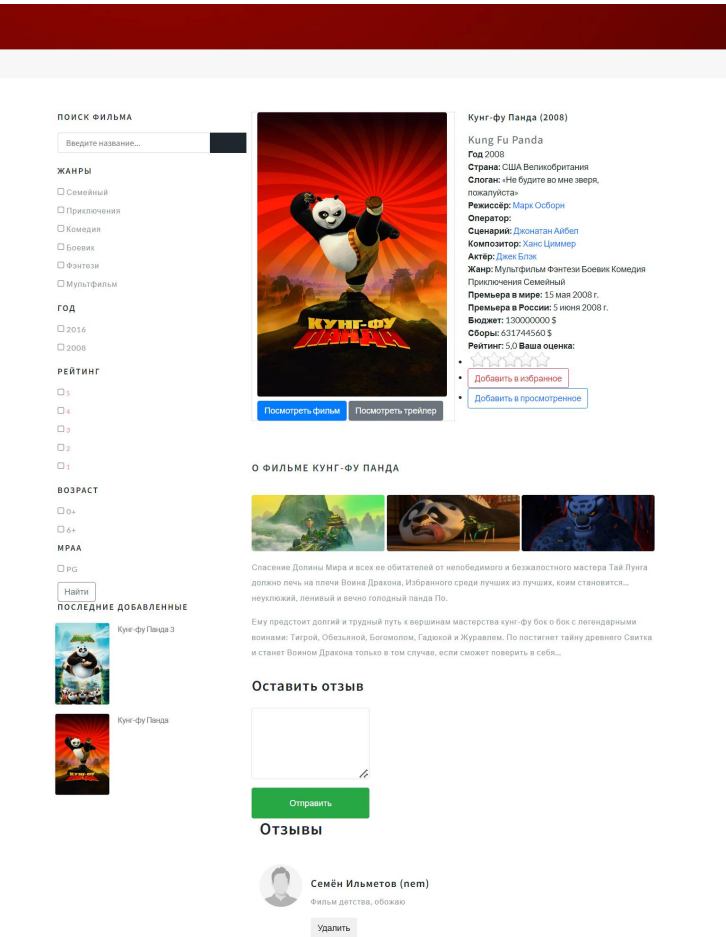


Рисунок Г.5 – Страница с информацией о конкретном фильме

### Г.4.3 Избранное

Авторизованный пользователь может добавлять и удалять фильм в «Избранное», «Просмотренное», ставить оценку и оставлять отзыв. Данная информация отображается в его профиле (см. Рисунок Г.6).

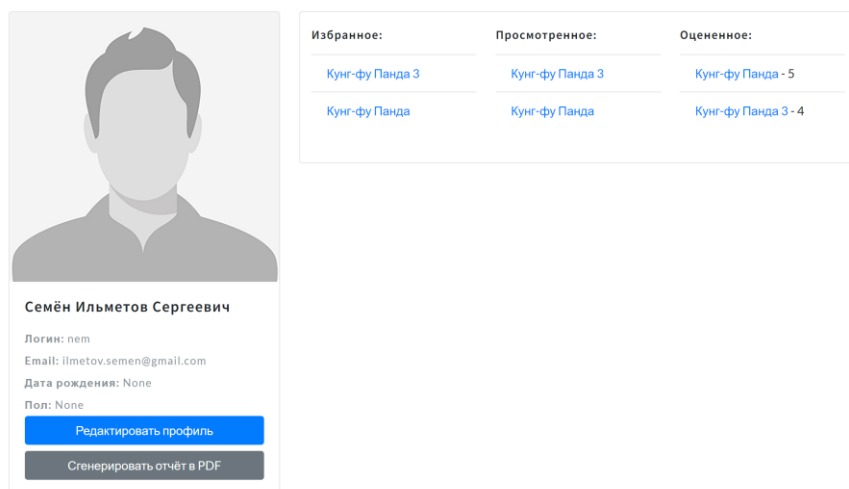


Рисунок Г.6– Страница «Избранное» для пользователя

На странице «Профиля» пользователь при нажатии на кнопку «Сгенерировать отчёт в pdf» получает файл с расширением «.pdf» с информацией обо всех фильмах, добавленных в «Избранное», «Просмотренное», а также обо всех оцененных фильмах (см. Рисунок Г.7)

Модератор на странице «Фильма» при нажатии на кнопку «Сгенерировать отчёт в pdf» получает файл с расширением «.pdf» с подробной информацией о фильме.

Отчёт о пользователе: Ильметов Семён Сергеевич

Логин: nem

Email: ilmetov.semen@gmail.com

Дата рождения: None

Пол: None

Избранное:  
Фильм: Кунг-фу Панда 3  
Фильм: Кунг-фу Панда

Просмотренное:  
Фильм: Кунг-фу Панда 3  
Фильм: Кунг-фу Панда

Оцененное:  
Фильм: Кунг-фу Панда  
Фильм: Кунг-фу Панда 3

Оценка: 5  
Оценка: 4

Рисунок Г.7– Отчёт для пользователя

#### Г.4.4 Добавление отзыва фильму

Авторизованному пользователю доступна возможность оставлять отзыв на фильмы (см. Рисунок Г.8), для этого необходимо ввести текст и нажать на кнопку «Отправить». Все отзывы пользователей добавляются на страницу фильма.(см. Рисунок Г.8).

### Оставить отзыв


Супееер

Отправить


Рисунок Г.7 – Создание отзыва на фильм

#### Оставить отзыв

Отправить




Семён Ильметов (sam)  
Фильм просто пушка!



Семён Ильметов (1234)  
Туфта

Удалить




Семён Ильметов (pet)  
Супееер

Рисунок Г.8 – Страница с отправленными заявками на бронирование для авторизованного пользователя

Для модератора при работе с отзывами добавляется возможность удалить отзыв любого пользователя (см. Рисунок Г.9).






**Семён Ильметов (sam)**  
Фильм просто пушка!

Удалить


---



**Семён Ильметов (1234)**  
Туфта

Удалить

---



**Семён Ильметов (nem)**  
Супееер

Удалить

---

Рисунок Г.9 – Просмотр отзывов к фильму от лица модератора