

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и администрирования
информационных систем

**Разработка серверной части
трекера фильмов Кинобаза
на языке Java**

Курсовая работа

02.03.03 – Математическое обеспечение и администрирование
информационных систем

Зав. кафедрой _____

д. ф.-м. н., проф. Артемов М. А.

Обучающийся _____

3 курс, 9 группа Загоровский А. В.

Руководитель _____

д. ф.-м. н., проф. Артемов М. А.

Воронеж 2023

Аннотация

Данная курсовая работа посвящена созданию онлайн-платформы, предназначенной для отслеживания просмотренных фильмов и сериалов. Также платформа предоставляет возможность комментирования, выставления оценок и обмена мнениями между пользователями. Данный ресурс также позволяет пользователю находить интересующие его фильмы по заданным критериям. В последствии поиск будет расширен системой умных рекомендаций.

Содержание

Содержание	3
Введение	4
1. Постановка задачи	5
2. Анализ задачи	6
2.1. Определение основных задач разрабатываемой онлайн-платформы ...	6
2.2. Сравнение с Кинопоиском	7
2.3. Предполагаемая архитектура онлайн-платформы	8
3. Средства реализации	9
4. Требования к аппаратному и программному обеспечению	11
5. Интерфейс пользователя	12
6. Реализация	18
6.1. Описания блоков архитектуры онлайн-платформы	18
6.2. Структура базы данных	18
6.3. Протокол обмена данными между клиентами и сервером	20
6.4. Структура серверной части	21
6.4.1. Схема работы серверного приложения	21
6.4.2. Описание основных классов, процедур и функций	22
7. План тестирования	24
Заключение	29
Список литературы	30
Приложение 1. Описание таблиц базы данных	31
Приложение 2. Sql-скрипт создания базы данных	34
Приложение 3. Листинг серверного приложения	35

Введение

Кинобаза – это веб-приложение, которое предназначено для отслеживания просмотренных фильмов и сериалов.

Существует много приложений, которые имеют схожий функционал. Однако, многие из них имеют ограниченные возможности, неудобные интерфейсы или неэффективные системы рекомендаций. Это затрудняет пользователей в поиске новых фильмов и сериалов.

Цель работы: спроектировать и разработать эффективную серверную часть трекера фильмов.

1. Постановка задачи

Разработать веб-приложение, предназначенное для отслеживания просмотров фильмов и сериалов, оценки контента, написания комментариев и поиска новых фильмов.

Для этого необходимо выполнить следующие задачи:

1. Разработать пользовательский интерфейс, который будет обеспечивать удобное отслеживание просмотренных фильмов и сериалов, а также возможность оценки и комментирования.
2. Спроектировать базу данных.
3. Реализовать взаимодействие серверной части с базой данных.
4. Разработать сервисы для управления:
 - 4.1. Регистрацией и аутентификацией.
 - 4.2. Комментариями.
 - 4.3. Оценками.
 - 4.4. Просмотрами.
 - 4.5. Профилем пользователя.
5. Обеспечить возможность поиска фильмов по названию, жанрам, актерам или режиссёрам.

2. Анализ задачи

2.1. Определение основных задач разрабатываемой онлайн-платформы

Основными задачами при разработке будут являться:

1. Управление пользовательскими функциями:
 - создание системы регистрации и аутентификации;
 - разработка функционала для создания и редактирования личных профилей пользователей;
 - обеспечение возможности обновления такой информации о пользователе, как имя, фотография профиля, пароль и почта.
2. Отслеживание просмотра фильмов:
 - реализация функциональности для отметки просмотренных фильмов и сериалов;
 - создание списков просмотренного, запланированного и просматриваемого пользователем.
3. Оценка и комментирование контента:
 - разработка сервиса для оценки фильмов и сериалов по 10 бальной системе;
 - добавление возможности написания комментариев.
4. Разработка функционала поиска фильмов по:
 - названию;
 - жанрам;
 - актёрам;
 - режиссёрам.

2.2. Сравнение с Кинопоиском

Сравнение можно провести по следующим пунктам:

1. Контент и база данных.

Так как «Кинобаза» была разработана недавно в отличие от «Кинопоиска», то имеет маленькую базу пользователей. В следствие чего система рекомендаций будет работать первое время не очень эффективно. Оба сервиса предлагают хорошую базу данных для хранения информации о фильмах.

2. Функциональность и возможности.

Обе платформы позволяют пользователям отслеживать фильмы, оценивать контент, писать комментарии и осуществлять поиск фильмов.

3. Коммуникация.

«Кинопоиск» имеет возможность добавлять пользователей в друзья и видеть последние просмотренные фильмы и оценки друзей. «Кинобаза» имеет только профиль, который можно просматривать любому человеку.

Онлайн-платформа «Кинобаза» имеет сходство с существующим ресурсом «Кинопоиск» в том, что оба предоставляют информацию о фильмах и сериалах. Однако, «Кинобаза» будет иметь преимущество в возможности добавления фильмов и сериалов самостоятельно, а также взаимодействие между пользователями в сообществах.

2.3. Предполагаемая архитектура онлайн-платформы

В соответствии с поставленными задачами структура «Кинобазы» может быть представлена в виде схемы, изображённой на рис. 2.1.

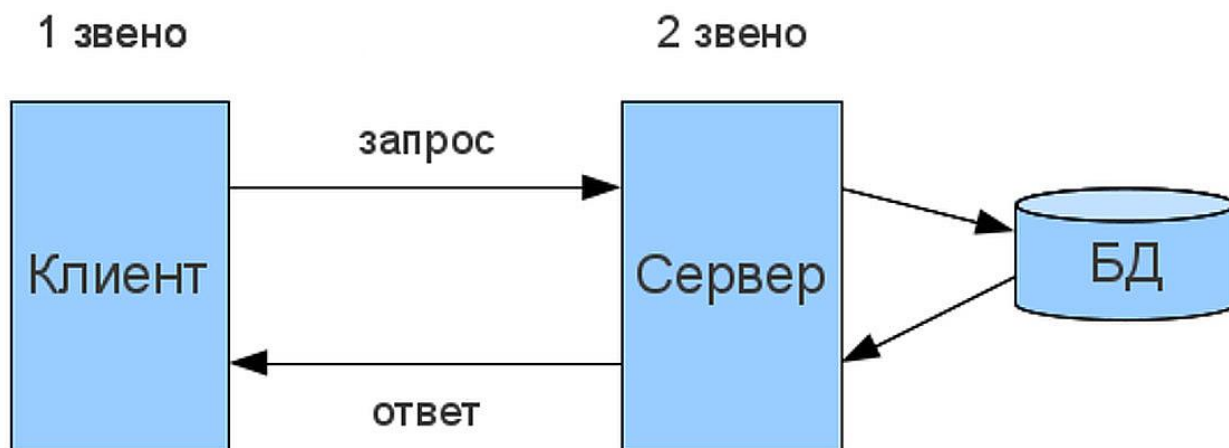


Рис. 2.1. Клиент-серверная архитектура

Данная архитектура имеет ряд преимуществ и соответствует требованиям и целям разработки веб-приложения. Вот некоторые из причин, по которым она была выбрана:

1. **Разделение ответственностей:** Клиентская часть занимается отображением данных и взаимодействием с пользователем, в то время как серверная часть отвечает за обработку запросов, бизнес-логику и доступ к базе данных.
2. **Масштабируемость:** разделение на клиентскую и серверную части позволяет гибко масштабировать систему. Можно масштабировать серверную часть, чтобы обрабатывать больше запросов и увеличить производительность, сохраняя клиентскую часть неизменной.
3. **Удобство разработки и поддержки:** разделение упрощает процесс разработки, тестирования и поддержки платформы. Каждая часть может быть разработана, оптимизирована и протестирована независимо друг от друга, что упрощает добавление новых функций.

В целом, выбор клиент-серверной архитектуры для разработки веб-приложения обусловлен её способностью обеспечить платформу гибкой масштабируемостью и удобством разработки.

3. Средства реализации

При реализации проекта использовались следующие средства:

1. Язык разработки – Java.
2. Среда разработки – IntelliJ IDEA Ultimate это интегрированная среда разработки, предназначенная для разработки Java-приложений.
3. Система контроля версий – GitHub это система контроля версий, используемая для хранения и управления исходным кодом приложения.
4. База данных – PostgreSQL это реляционная база данных, которая используется для хранения и организации данных приложения.
5. Сборщик проектов – Maven это сборщик проектов, используемый для автоматизации процесса сборки, тестирования и развертывания приложения.
6. Фреймворк – Spring Boot используется для упрощения создания и настройки приложений, предоставляет инструменты для управления зависимостями, конфигурации и разработки RESTful API.
7. Фреймворк – Lombok используется для автоматической генерации геттеров, сеттеров, конструкторов и других методов.
8. Библиотека – JSON Web Token используется для аутентификации и авторизации веб-приложения, обмена информацией между клиентом и сервером.
9. Библиотека – Hibernate Validator используется для валидации пользовательского ввода, обеспечивая целостность и корректность данных.
10. Библиотека – Flyway Core используется для автоматической миграции структуры базы данных при развертывании приложения, обеспечивая согласованность и легкость обновления.

11. Фреймворк – MapStruct используется для генерации автоматического кода преобразования объектов.
12. Фреймворк – Spring Security используется для обеспечения безопасности веб-приложений. Он предоставляет механизмы аутентификации и авторизации.
13. Фреймворк – Mockito используется для создания Mock-объектов и тестирования Java-приложений.
14. Swagger – это инструмент для создания, документирования и использования интерфейсов API.
15. Библиотека – Caffeine Cache используется для реализации кэширования данных в Java-приложениях.

Данные средства реализации позволили разработать функциональную и максимально эффективную онлайн платформу

4. Требования к аппаратному и программному обеспечению

Онлайн-платформа предназначена для использования в веб-браузере, например, Google Chrome, Mozilla Firefox, Safari и другие.

Минимальные требования приложения:

- процессор с тактовой частотой 1,8 ГГц или выше;
- 2 Гб оперативной памяти;
- 100 Мб дискового пространства для установки приложения.

Требования к аппаратному обеспечению сервера:

- многоядерный процессор с тактовой частотой не менее 2 ГГц;
- 4 Гб оперативной памяти;
- 9 Мб дискового пространства для базы данных.

Требования к программному обеспечению сервера:

- web-сервер Apache Tomcat;
- Java Runtime Environment (JRE) версии 17 или выше;
- PostgreSQL версии 15.2 или выше.

5. Интерфейс пользователя

В онлайн-платформе «Кинобаза» была разработана только серверная часть. Для взаимодействия с сервером, используется интерфейс Swagger API, представленный на рис. 5.1.

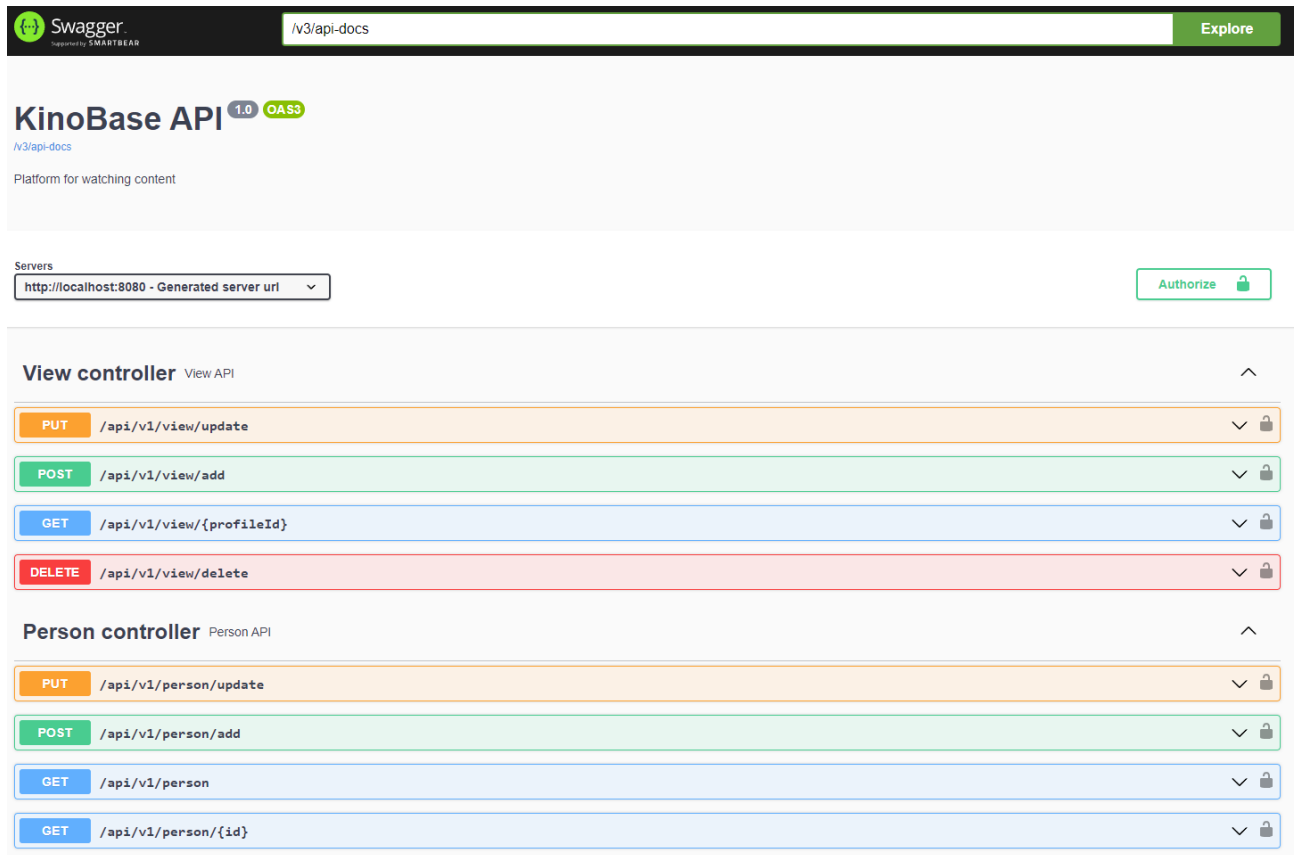


Рис 5.1. Интерфейс взаимодействия с сервером

Вместе с серверной частью был разработан дизайн клиентской части, который представляет визуальное оформление пользовательского интерфейса. Цель дизайна состоит в лучшем понимании функциональности, тем самым добавлением на серверную часть всех необходимых API методов для будущего взаимодействия с клиентской частью.

Главная страница сайта (рис. 5.2), которая является точкой входа для пользователя. В верхнем правом углу находится кнопка регистрации и иконка входа. Слева форма для поиска контента, а также кнопка меню (рис. 5.3). Ниже идёт панорама с самым просматриваемым фильмом за месяц. Далее идут фильмы-новинки и популярные за сегодня. В конце расположены топы по жанрам.

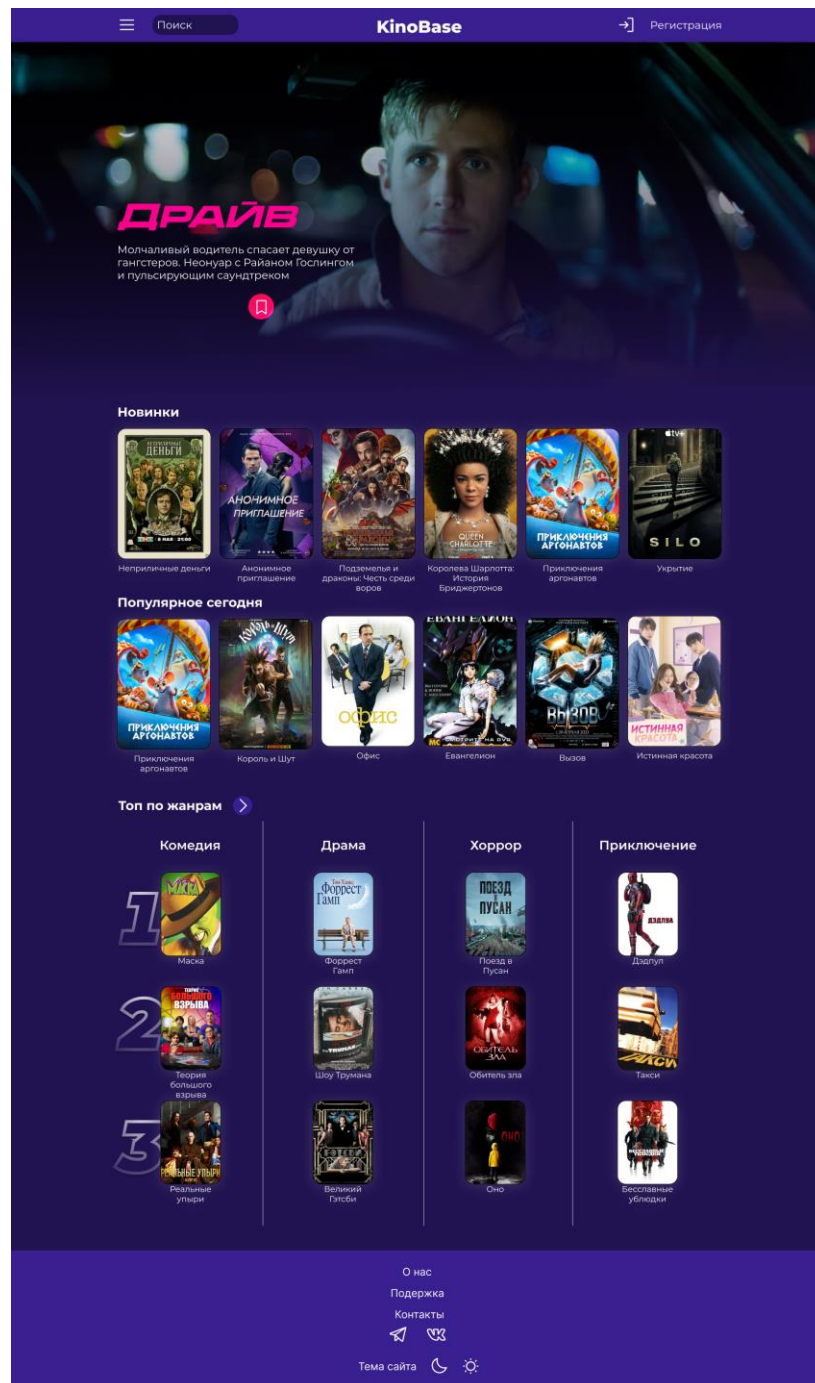


Рис. 5.2. Главная страница онлайн-платформы.

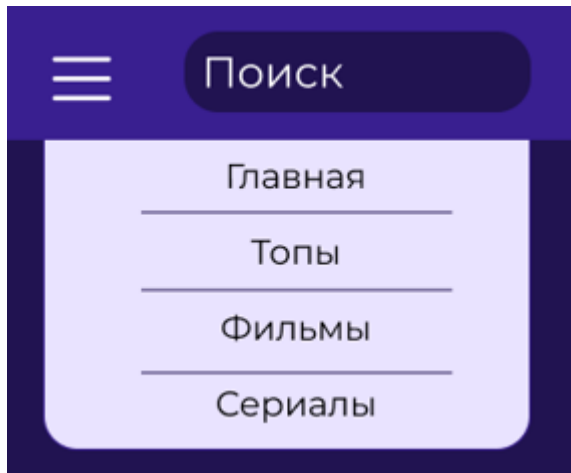


Рис. 5.3. Раскрытая форма гамбургера.

При переходе на страницу топов (рис. 5.4) есть 2 фильтра на выбор:

1. Выбор контента: фильмы или сериалы;
2. Выбор жанра, по которому будет отображаться топ.

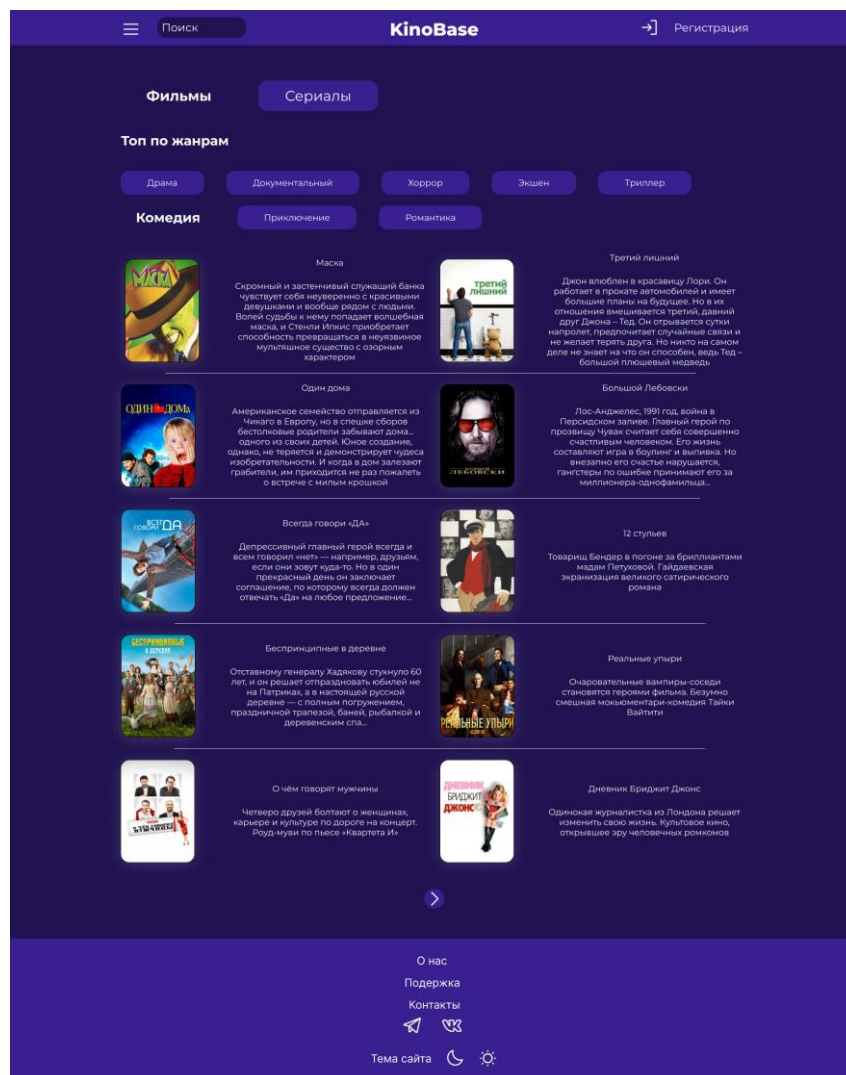


Рис. 5.4. Страница топов

При нажатии на кнопку регистрация перед пользователем появляется форма (рис. 5.5) с полями логин, пароль, никнейм, и почта. Ниже кнопки «Вперёд!» находится гиперссылка «Уже зарегистрирован», чтобы перейти на форму входа.

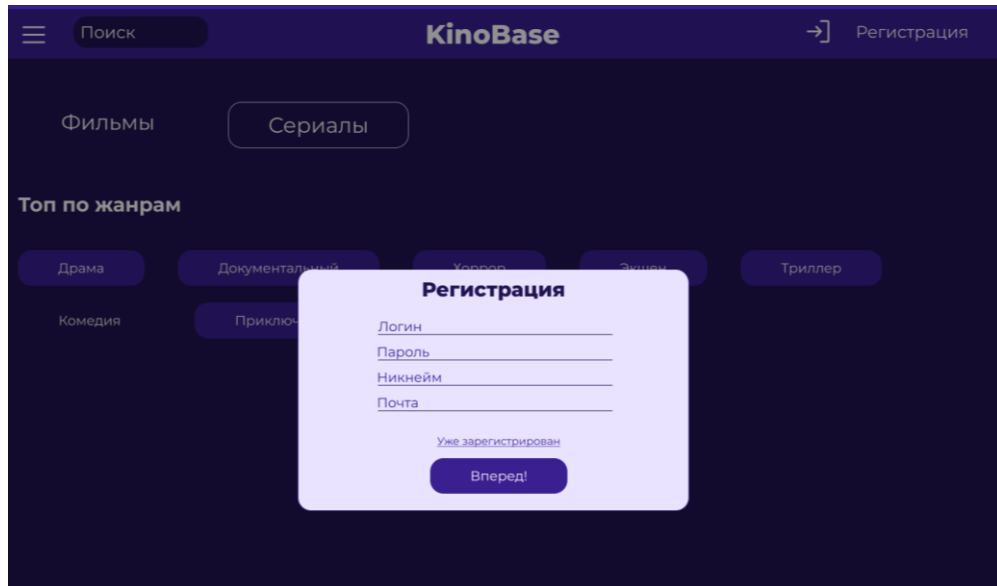
The image shows a web application interface for KinoBase. At the top, there is a dark blue header with a menu icon, a search bar labeled "Поиск", the site name "KinoBase", and a "Регистрация" link. Below the header, there are buttons for "Фильмы" and "Сериалы". A section titled "Топ по жанрам" displays several genre buttons: "Драма", "Документальный", "Ужасы", "Экшен", "Триллер", "Комедия", and "Приключения". Overlaid on this is a white registration form titled "Регистрация". The form contains four input fields: "Логин", "Пароль", "Никнейм", and "Почта". Below these fields is a link "Уже зарегистрирован" and a blue button labeled "Вперед!".

Рис. 5.5. Форма регистрации пользователя

При нажатии на иконку входа перед пользователем появляется форма (рис. 5.6) ввода логина и пароля. После нажатия на кнопку «Вперёд!», в случае успешной аутентификации, пользователь попадет в личный кабинет (рис. 5.7)

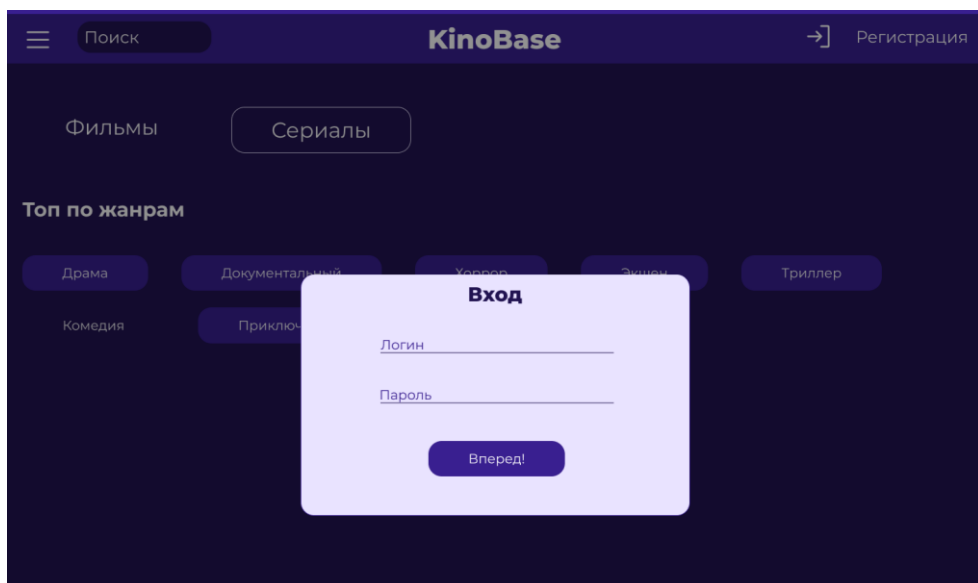
The image shows the same KinoBase web application interface as in the previous screenshot. Overlaid on the page is a white login form titled "Вход". It contains two input fields: "Логин" and "Пароль". Below these fields is a blue button labeled "Вперед!". The background elements, including the header, navigation buttons, and genre section, are identical to the registration form screenshot.

Рис. 5.6. Форма входа пользователя

В личном кабинете отображается никнейм пользователя, его логин, почта и возможность скрывать её от других пользователей, дата регистрации и количество просмотренных фильмов. Ниже идёт три пункта: «Смотрю», «Запланировано» и «Просмотрено». При нажатии на каждый будет отображаться контент в соответствующей категории. После идут последние отзывы пользователя, при нажатии на которые можно перейти на страницу фильма (рис. 5.8) к которому был оставлен отзыв.

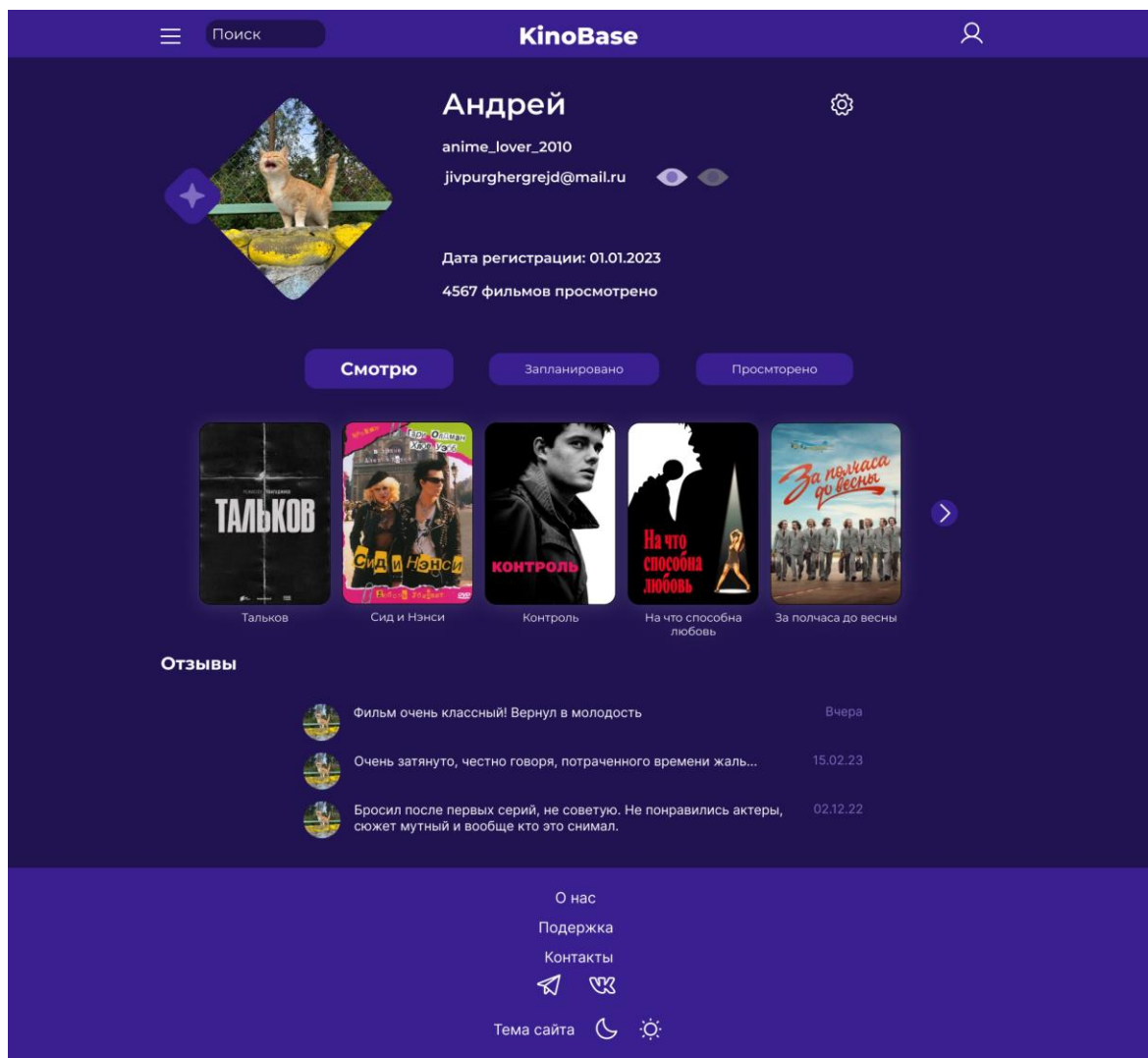


Рис. 5.7. Личный кабинет авторизованного пользователя

Каждый раз при нажатии на постер фильма пользователь переходит на страницу фильма (рис. 5.8), на которой содержится постер фильма, его название, средняя оценка, год выхода, жанры, количество серий (если это

сериал), страна, описание и оценка пользователя. Ниже идут отзывы о фильме, а также похожие фильмы.

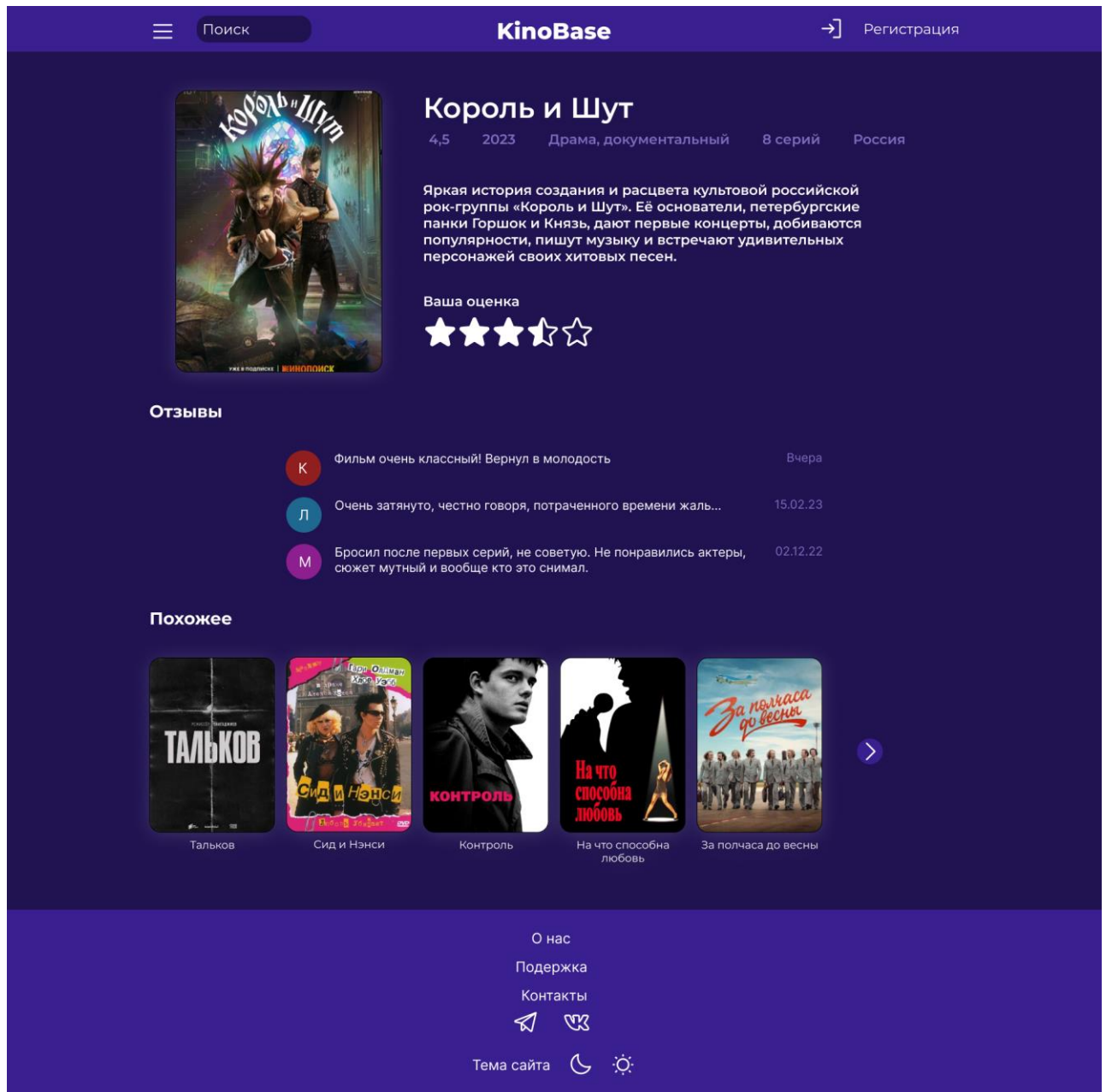


Рис. 5.8. Страница фильма

6. Реализация

6.1. Описания блоков архитектуры онлайн-платформы

Архитектура клиент-сервер представляет собой распределенную модель, в которой клиент и сервер взаимодействуют друг с другом для выполнения различных задач.

Клиентская часть отвечает за сбор пользовательских действий, в результате которых клиент формирует HTTP запросы на сервер. Например, при регистрации пользователя клиент отправит POST запрос с данными о пользователе. Клиент также обрабатывает полученные с сервера данные и отображает результаты пользователю в определённом формате, удобном для восприятия человеком.

Серверная часть отвечает за обработку HTTP запросов от клиентской части и выполнение соответствующих операций. Она содержит:

- бизнес-логику приложения, которая отвечает за обработку запросов, таких как проверку аутентификации, добавление фильмов, оценок и написание комментариев;
- взаимодействие с базой данных для чтения, обновления, удаления и добавления данных;
- функцию формирования ответов в формате JSON.

Серверная часть реализована на Java с использованием фреймворка Spring Boot.

База данных выполняет роль хранилища данных. Она предназначена для сохранения, организации и управления данными, которые будут использованы серверной частью.

6.2. Структура базы данных

Модель данных, описывающая структуру базы данных, разработана с помощью DataGrip v2023.1.1 и представлена на рис. 6.2.

База данных состоит из следующих таблиц.

Таблица `profile` содержит информацию о пользователе, а именно логин, хешированный пароль, его почту, никнейм и роль (администратор или пользователь)

Таблица `content` содержит информацию о контенте, который имеет атрибут `type`, по которому определяется фильм это или сериал. Также в таблице хранится информация про жанры, страны, название, описание, дата выхода, и постер.

Таблицы `view`, `rating` и `comment` содержат информацию про просмотры, оценки и комментарии пользователя к контенту. В таблице `view` отмечается просмотренный, запланированный и текущий просмотр контента и дата добавления.

Таблица `person` содержит информацию о актерах или режиссерах контента.

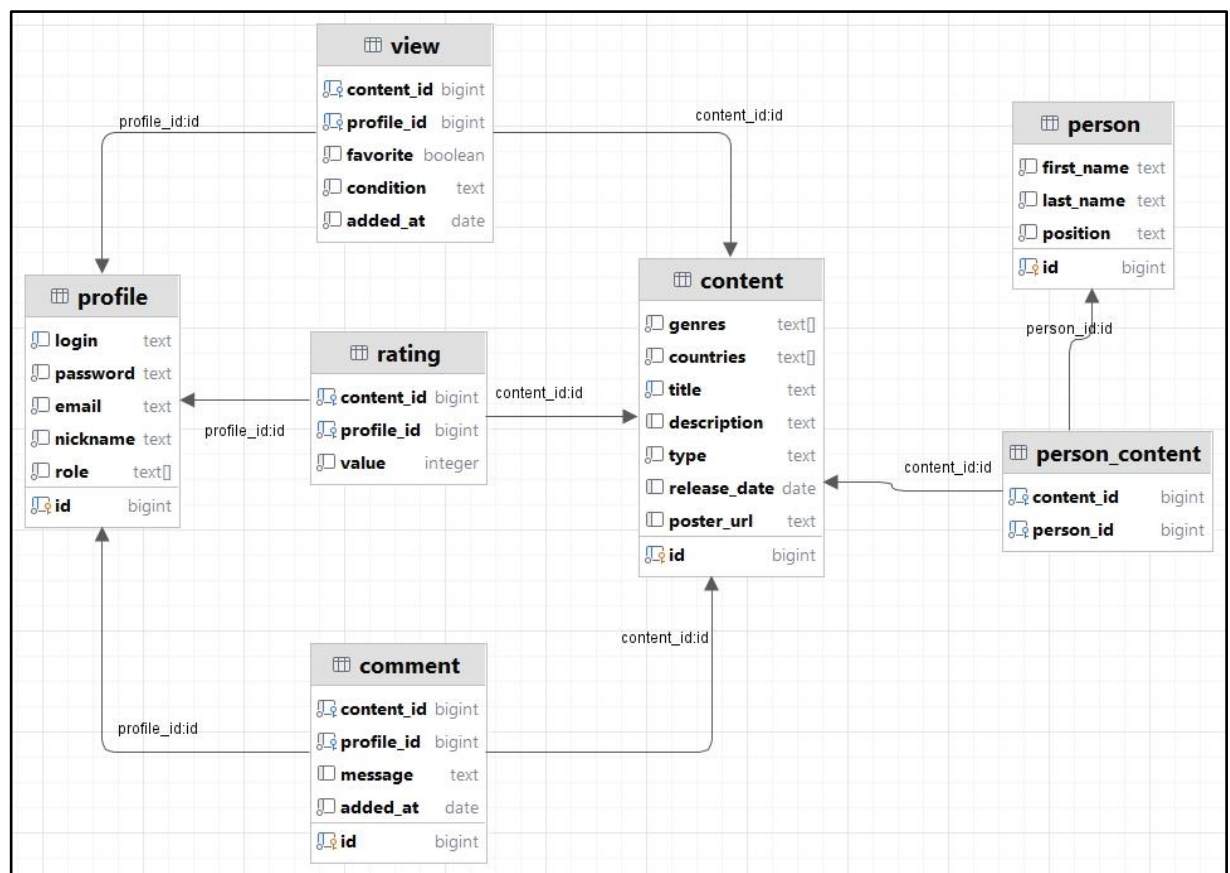


Рис 6.2. UML-диаграмма базы данных

В приложении 1 представлено более подробное описание каждой из таблиц.

В приложении 2 представлен sql-скрипт, предназначенный для создания описанной базы данных.

6.3. Протокол обмена данными между клиентами и сервером

Протокол обмена данными между клиентами и сервером определяет правила и формат передачи информации между клиентской и серверной частями приложения. Он обеспечивает стандартизацию коммуникации и позволяет разным компонентам взаимодействовать и обмениваться данными.

Для обмена данными был использован протокол HTTP (Hypertext Transfer Protocol), так как веб-приложение взаимодействует с сервером через HTTP-запросы и HTTP-ответы. Клиентская часть может отправлять запросы на сервер для выполнения операций, таких как получение списка фильмов, отправка оценки фильма или добавление комментария. Серверная часть обрабатывает эти запросы, взаимодействует с базой данных и формирует HTTP-ответы, содержащие необходимую информацию или результат выполненных операций.

Пример контроллера, обрабатывающего запросы для профиля

```
@RestController
@RequestMapping("/api/v1/profile")
@RequiredArgsConstructor
@Validated
@CacheConfig(cacheNames = {"profile", "view"})
@Tag(name = "Profile controller", description = "Profile API")
public class ProfileController {
    private final ProfileService profileService;
    private final ProfileMapper profileMapper;

    private final ViewService viewService;
    private final ViewMapper viewMapper;

    @GetMapping("/{id}")
    @Cacheable(cacheNames = {"profile"}, key = "#id")
    public ProfileDto getById(@PathVariable Long id) {
        Profile profile = profileService.getById(id);
```

```

        return profileMapper.toDto(profile);
    }

    @PutMapping("/update")
    @CachePut(cacheNames = {"profile"}, key = "#dto.id")
    public ProfileDto updateProfile(@Validated(OnUpdate.class)
    @RequestBody ProfileDto dto) {
        Profile profile = profileMapper.toEntity(dto);
        Profile updatedView = profileService.update(profile);
        return profileMapper.toDto(updatedView);
    }

    @GetMapping("/{id}/views")
    @Cacheable(cacheNames = {"view"}, key = "#id")
    public List<ViewDto> getAllByProfileId(@PathVariable Long
id) {
        List<View> views = viewService.getAllByProfileId(id);
        return viewMapper.toDto(views);
    }
    @DeleteMapping("/{id}")
    @CacheEvict(key = "#id")
    public void deleteById(@PathVariable Long id) {
        profileService.delete(id);
    }
}

```

6.4. Структура серверной части

6.4.1. Схема работы серверного приложения

Структура серверной части включает три уровня: контроллер, сервис и репозиторий:

1. Контроллер представляет собой компоненты, отвечающие за обработку запросов от клиентской части. Они принимают HTTP-запросы от клиента, извлекают необходимые параметры или данные из запроса и выполняют необходимые операции.
2. Сервис слой содержит бизнес-логику и выполняет операции над данными. Этот уровень служит связывающим звеном между контроллерами и репозиториями. Например, такие функции как поиск фильма, создание профиля пользователя или написания комментариев.

3. Репозиторий служит для взаимодействия с базой данных. Он содержит методы для выполнения операций чтения, записи, обновления и удаления данных из базы данных. Репозиторий обеспечивает абстракцию базы данных и скрывает детали взаимодействия с ней от остальных компонентов приложения.

Схема взаимодействия компонентов приведена на рис. 6.3.

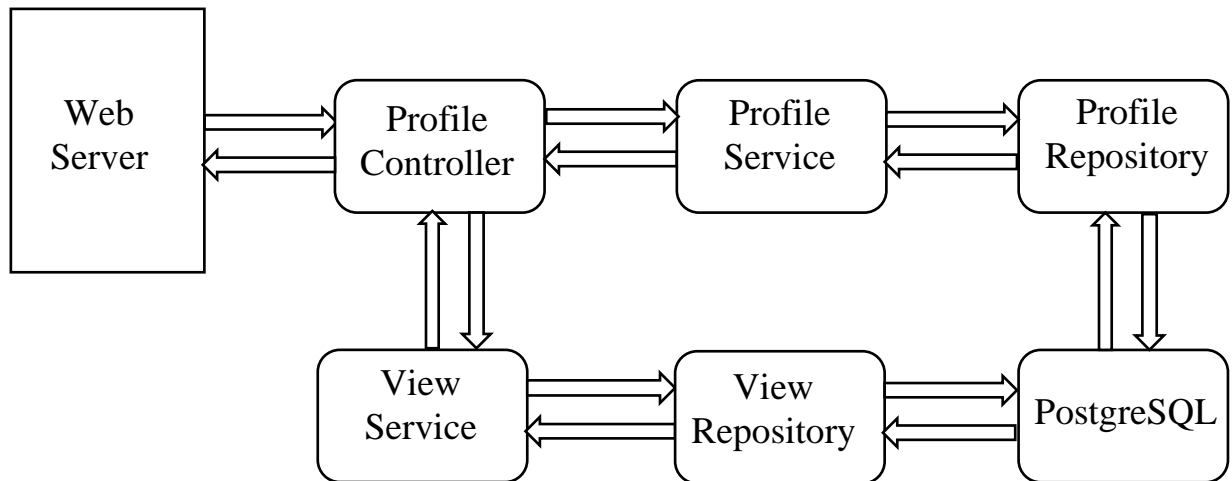


Рис. 6.3. Схема взаимодействия компонентов серверного приложения

6.4.2. Описание основных классов, процедур и функций

AuthController, отвечает за обработку HTTP запросов, связанных с аутентификацией и авторизацией пользователей. Он использует:

- AuthService – сервис для аутентификации пользователей и генерации JWT-токнов;
- ProfileService – сервис для работы с профилями пользователей, сохранение, изменение, удаление;
- ProfileMapper – класс, отвечающий за преобразование объектами типа Profile и ProfileDto.

Методы контроллера:

- login – обрабатывает POST-запрос на аутентификацию пользователя. Принимает объект JwtRequest, содержащий данные

для аутентификации. Возвращает объект JwtResponse с JWT-токеном;

- register – обрабатывает POST-запрос на регистрацию нового пользователя. Принимает объект ProfileDto, содержащий данные пользователя;
- refresh – обрабатывает POST-запрос на обновление JWT-токена.

ProfileService выполняет бизнес-логику связанную с профилем пользователей. Он использует:

- ProfileRepository – репозиторий, предоставляющий методы для работы с данными профилей пользователей в базе данных;
- PasswordEncoder – интерфейс, предоставляющий методы для шифрования паролей;
- ViewService, CommentService, RatingService – сервисы, отвечающие за операции над просмотрами, комментариями и рейтингами в системе.

Методы сервиса ProfileService:

- getById – получает профиль пользователя по его идентификатору. Если профиль не найден, выбрасывается исключение ResourceNotFoundException;
- getByLogin – получает профиль по его логину. При отсутствии профиля в базе данных выбрасывает аналогичную прошлую методу ошибку;
- update – обновляет профиль пользователя;
- create – создает новый профиль пользователя. Проверяет, что профиля с таким логином не существует. Шифрует пароль с помощью PasswordEncoder, сохраняет новый профиль с помощью репозитория в базе данных;
- delete – Удаляет профиль пользователя и связанные с ним просмотры, комментарии и оценки.

7. План тестирования

Тестирование разработанной онлайн-платформы «Кинобаза» состоит из следующих этапов:

1. Проверка регистрации нового пользователя и аутентификации.

Проверяется регистрация нового пользователя (рис. 7.1), добавление его в базу данных и возможность аутентификации (рис. 7.2) нового пользователя.

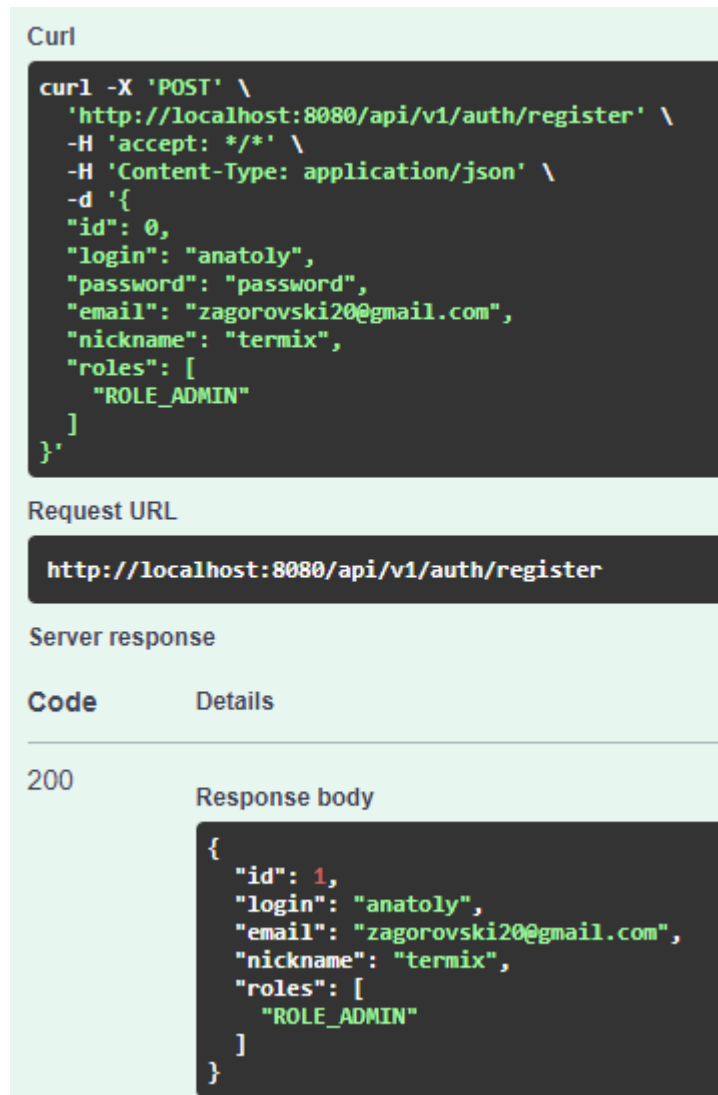


Рис. 7.1. Регистрация пользователя



Рис. 7.2. Аутентификация

2. Добавление фильма в список запланированных пользователя.

Проверяется корректность добавления фильма (рис. 7.3).

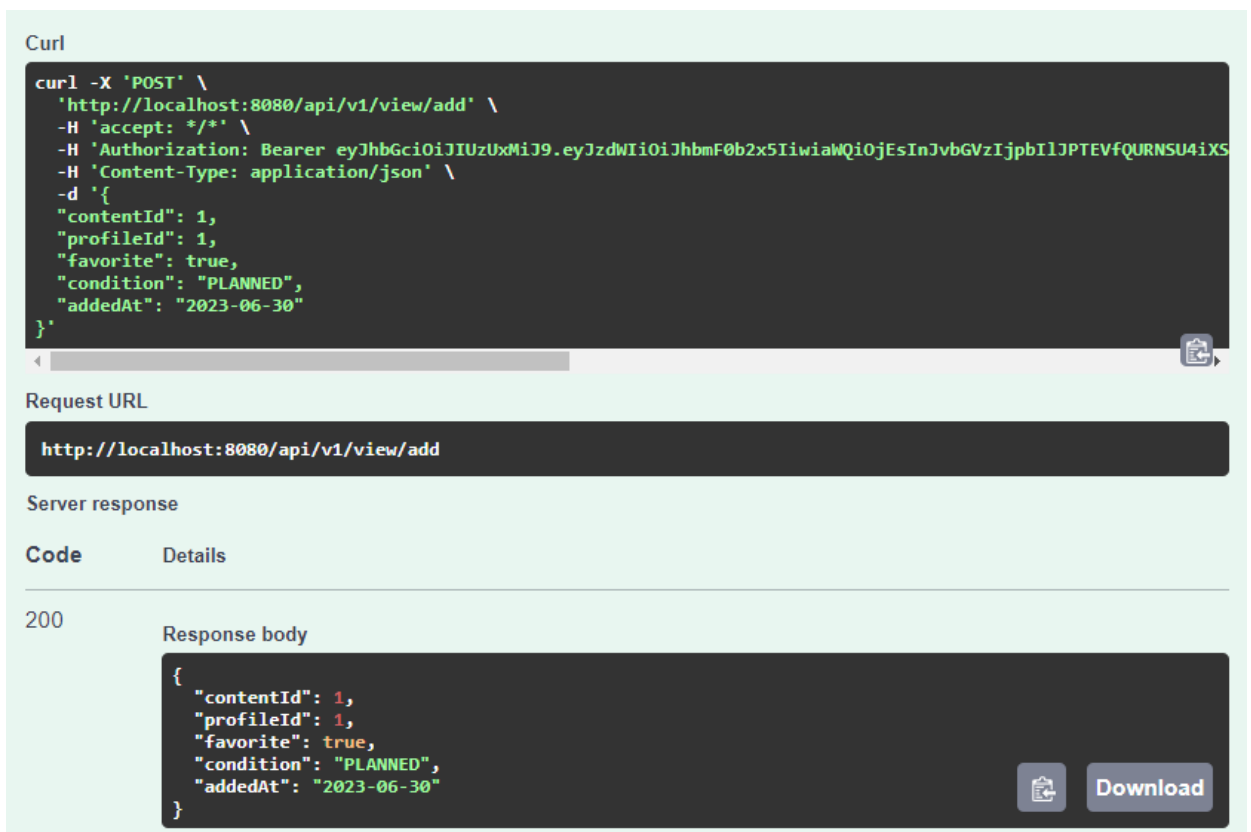


Рис 7.3. Добавление запланированного фильма

3. Проверка получения списка фильмов для пользователя.

Проверяется то, что контент правильно загружается для заданного пользователя с идентификатором 1 (рис. 7.4) из базы данных.

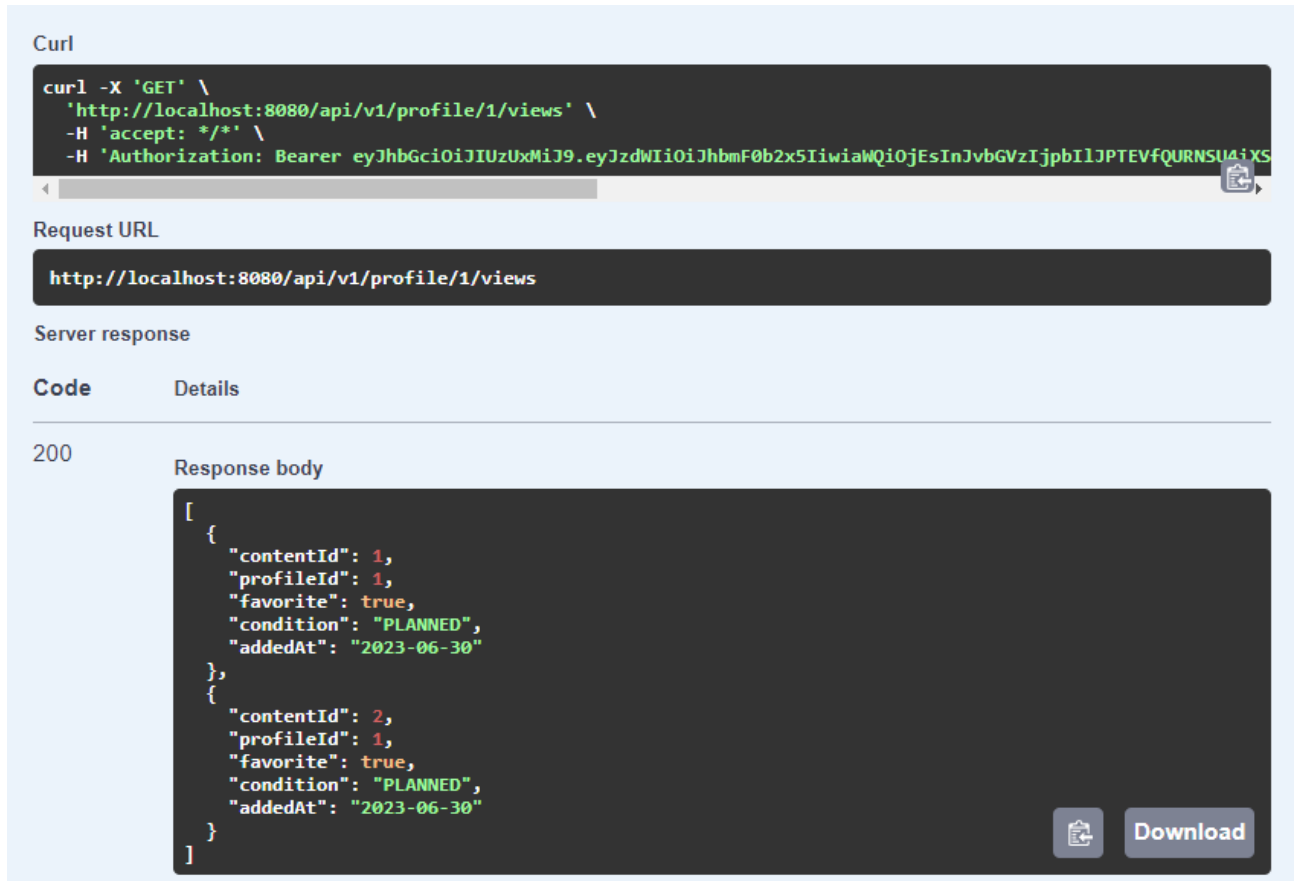


Рис. 7.4. Получение списка фильмов

4. Проверка поиска фильма по названию.

Проверяется правильная работа поиска по заданному названию фильма. При нахождении фильма (рис 7.5) в базе данных будет получена информацию о нём, если фильм не был найден, то вернётся 404 ошибка (рис 7.6).

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/v1/content/find/interstellar' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhbmF0b2x5IiwiaWQiOiJEsInJvbGVzIjpbIlJPTeVfQURNSU4iXS'
```

Request URL

`http://localhost:8080/api/v1/content/find/interstellar`

Server response

Code Details

200

Response body

```
{
  "id": 3,
  "genreList": [
    "ACTION",
    "DRAMA",
    "ADVENTURE"
  ],
  "countryList": [
    "UNITED_KINGDOM",
    "USA",
    "CANADA"
  ],
  "title": "Interstellar",
  "description": "Когда засуха, пыльные бури и вымирание растений приводят человечество к продовольственному кризису, коллектив исследователей и учёных отправляется сквозь червоточину (которая предположительно соединяет области пространства-времени через большое расстояние) в путешествие, чтобы превзойти прежние ограничения для космических путешествий человека и найти планету с подходящими для человечества условиями.",
  "type": "MOVIE",
  "releaseDate": "2014-01-01",
  "posterUrl": "https://i.pinimg.com/736x/92/27/e8/9227e8fab39e7d58df3278aa7244ee45--interstellar-drawing-art.jpg"
}
```

Download

Рис. 7.5. Результат поиска по названию

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/v1/content/find/The%20Shawshank%20Redemption' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhbmF0b2x5IiwiaWQiOiJEsInJvbGVzIjpbIlJPTeVfQURNSU4iXS'
```

Request URL

`http://localhost:8080/api/v1/content/find/The%20Shawshank%20Redemption`

Server response

Code Details

404

Undocumented Error: response status is 404

Response body

```
{
  "message": "Content not found",
  "errors": null
}
```

Download

Рис 7.6. Результат поиска по названию

Также важным аспектом тестирования является наличие юнит-тестов, которые позволяют проверить отдельные компоненты и функции

программного обеспечения на правильность и соответствие ожидаемым результатам.

Пример юнит-тестов класса `CommentServiceImplTest`:

- `testGetAllByContentId_Success` – проверяет успешное получение списка комментариев по идентификатору фильма или сериала;
- `testGetAllByContentId_NotFound` – проверяет случай, когда список комментариев не найден и ожидается выброс исключения;
- `testGetByMessage_Success` – проверяет случай, когда поиск комментария, в котором содержится заданное сообщение успешен;
- `testGetByMessage_NotFound` – проверяет случай, когда поиск комментария, в котором содержится заданное сообщение ничего не нашёл;
- `testUpdate` – проверяет успешность обновления комментария;
- `testCreate` – проверяет успешность создания комментария;
- `testDelete` – проверяет успешность удаления комментария.

Заключение

В результате выполнения поставленных задач была разработана онлайн-платформа «Кинобаза», предоставляющая пользователям удобный интерфейс для отслеживания просмотренных фильмов, оценки контента, написания комментариев и поиска новых фильмов. Были достигнуты следующие результаты:

1. Спроектирован пользовательский интерфейс, который обеспечивает удобное использование платформы.
2. Спроектирована база данных, которая хранит все необходимые данные о фильмах и сериалах, пользователях, комментариях и просмотрах.
3. Реализовано взаимодействие серверной части с базой данных, обеспечивающее сохранение, обновление, удаление и получение данных о контенте, пользователях, комментариях и просмотрах.
4. Разработаны сервисы для регистрации и аутентификации пользователей, а также для управления созданием, удалением и редактированием комментариев, отслеживания просмотров и оценивания контента.
5. Обеспечена возможность поиска фильмов по различным параметрам.

Онлайн-платформа успешно реализует поставленные задачи и предоставляет пользователям все необходимые функции для удовлетворения их потребностей в области кинопросмотра.

В планах разработать умную систему рекомендаций при поиске фильмов и сериалов, которая будет основана на предпочтениях и оценках пользователей с применением машинного обучения.

Список литературы

1. Уоллс К. Spring в действии – 6-е изд. / К. Уоллс ; пер. с англ. А. Н. Киселева – М. : ДМК Пресс, 2022. – 544 с.
2. Официальная документация фреймворка // Spring Boot Reference Documentation – URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle> (дата обращения: 05.04.2023).
3. Хорстманн К. Java. Библиотека профессионала, том 2. – 11-е изд. / К. Хорстманн ; пер. с англ. – СПб. : ООО «Диалектика», 2020. – 864 с.
4. Эккель Б. Философия Java. – 4-е полное изд. / Б. Эккель ; пер. с англ. ООО Издательство «Питер» СПб.: Питер, 2015 – 1168 с.
5. Блох Д. Java: эффективное программирование, 3-е / Д. Блох ; пер. с англ. – СПб : ООО «Диалектика», 2019. – 464 с.
6. Онлайн-платформа для просмотра фильмов Кинопоиск. – URL: <https://www.kinopoisk.ru/> (дата обращения: 25.03.2023).
7. Козмина. Ю. Spring 5 для профессионалов / Ю. Козмина, Р. Харроп, К. Шефер, К. Хо ; пер. с англ. – СПб. : ООО «Диалектика», 2019. – 1120 с.
8. Куроуз Д. Компьютерные сети. – 6-е изд / Д. Куроуз, К. Росс ; пер. с англ – Райтман М. – Москва : Издательство «Э», 2016. – 912 с.
9. Официальная документация PostgreSQL. – URL: <https://www.postgresql.org/docs/15/index.html> (дата обращения: 29.03.2023).
10. Мартин Р. Чистая архитектура / Р. Мартин ; пер. с англ – СПб. : ООО «Питер», 2018 – 352 с.

Приложение 1. Описание таблиц базы данных

Profile

Название	Домен	Обязатель- ный	PK	FK	Уникаль- ный	Описание
id	bigint	+	+	–	+	
login	text	+	–	–	+	Логин
password	text	+	–	–	–	Хешированный пароль
email	text	+	–	–	+	Почта
nickname	text	+	–	–	–	Псевдоним
role	text[]	+	–	–	–	Роль может быть «ROLE_USER» или «ROLE_ADMIN»

Альтернативные ключи: login, email.

Content

Название	Домен	Обязатель- ный	PK	FK	Уникаль- ный	Описание
id	bigint	+	+	–	+	
genres	text[]	+	–	–	–	Жанры
countries	text[]	+	–	–	–	Страны в которых снимался контент
title	text	+	–	–	+	Название контента
description	text	–	–	–	–	Описание
type	text	+	–	–	–	Проверка на то, что тип является «MOVIE» или «TV_SHOW»
release_date	date	–	–	–	–	Дата выпуска
poster_url	text	–	–	–	–	Ссылка на картинку

Альтернативные ключи: title.

Person

Название	Домен	Обязатель- ный	PK	FK	Уникаль- ный	Описание
id	bigint	+	+	–	+	
first_name	text	+	–	–	–	Имя
last_name	text	+	–	–	–	Фамилия

position	text	+	–	–	–	Должность человека «ACTOR» или «DIRECTOR»
----------	------	---	---	---	---	---

Альтернативные ключи: нет.

Person_content

Название	Домен	Обязатель- ный	PK	FK	Уникаль- ный	Описание
content_id	bigint	+	–	+	+	Идентификатор контента
person_id	bigint	+	–	+	+	Идентификатор человека

Альтернативные ключи: нет.

Rating

Название	Домен	Обязатель- ный	PK	FK	Уникаль- ный	Описание
content_id	bigint	+	–	+	+	Идентификатор контента
profile_id	bigint	+	–	+	+	Идентификатор пользователя
value	integer	+	–	–	–	Оценка от 1 до 10

Альтернативные ключи: нет.

Comment

Название	Домен	Обязатель- ный	PK	FK	Уникаль- ный	Описание
id	bigint	+	+	–	+	
content_id	bigint	+	–	+	+	Идентификатор контента
profile_id	bigint	+	–	+	+	Идентификатор пользователя
message	text	+	–	–	–	Сообщение комментария
added_at	date	+	–	–	–	Дата написания комментария

Альтернативные ключи: нет.

View

Название	Домен	Обязатель- ный	PK	FK	Уникаль- ный	Описание
content_id	bigint	+	—	+	+	Идентификатор контента
profile_id	bigint	+	—	+	+	Идентификатор пользователя
favorite	boolean	+	—	—	—	Флаг показывающий является ли контент любимым
condition	text	+	—	—	—	Состояние контента, по умолчанию «WATCHED». Может быть «WATCHED», «WATCHING» или «PLANNED »
added_at	date	+	—	—	—	Дата добавления в список

Альтернативные ключи: нет.

Приложение 2. Sql-скрипт создания базы данных

```

CREATE TABLE profile
(
    id          SERIAL8 PRIMARY KEY,
    login       TEXT      NOT NULL UNIQUE,
    password    TEXT      NOT NULL,
    email       TEXT      NOT NULL,
    nickname    TEXT      NOT NULL,
    role        TEXT[]    NOT NULL
);
CREATE TABLE person
(
    id          SERIAL8 PRIMARY KEY,
    first_name  TEXT      NOT NULL,
    last_name   TEXT      NOT NULL,
    position    TEXT      NOT NULL
);
CREATE TABLE content
(
    id          SERIAL8 PRIMARY KEY,
    genres      TEXT[]    NOT NULL,
    countries   TEXT[]    NOT NULL,
    title       TEXT      UNIQUE NOT NULL,
    description  TEXT      NULL,
    type        TEXT      NOT NULL CHECK ( type IN ('MOVIE', 'TV_SHOW')),
    release_date DATE      NULL,
    poster_url  TEXT      NULL
);
CREATE TABLE person_content
(
    content_id BIGINT REFERENCES content (id) NOT NULL,
    person_id  BIGINT REFERENCES person (id) NOT NULL,
    UNIQUE (person_id, content_id)
);
CREATE TABLE rating
(
    content_id BIGINT REFERENCES content (id) NOT NULL,
    profile_id BIGINT REFERENCES profile (id) NOT NULL,
    value       INTEGER CHECK ( value >= 1 AND value <= 10) NOT NULL,
    UNIQUE (content_id, profile_id)
);
CREATE TABLE comment
(
    id          SERIAL8 PRIMARY KEY,
    content_id  BIGINT REFERENCES content (id) NOT NULL,
    profile_id  BIGINT REFERENCES profile (id) NOT NULL,
    message     TEXT      NOT NULL,
    added_at    DATE      DEFAULT NOW()      NOT NULL
);
CREATE TABLE view
(
    content_id  BIGINT REFERENCES content (id) NOT NULL,
    profile_id  BIGINT REFERENCES profile (id) NOT NULL,
    favorite    BOOLEAN DEFAULT false        NOT NULL,
    condition   TEXT      DEFAULT 'WATCHED'   NOT NULL,
    added_at    DATE      DEFAULT NOW()      NOT NULL,
    UNIQUE (content_id, profile_id)
);

```

Приложение 3. Листинг серверного приложения

```
// ApplicationConfig
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor(onConstructor = @__(@Lazy))
public class ApplicationConfig {
    private final ApplicationContext applicationContext;
    private final JwtTokenProvider jwtTokenProvider;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager
    authenticationManager(AuthenticationConfiguration configuration) throws
    Exception {
        return configuration.getAuthenticationManager();
    }

    @Bean
    public OpenAPI openAPI() {
        return new OpenAPI()
            .addSecurityItem(new
    SecurityRequirement().addList("bearerAuth"))
            .components(
                new Components()
                    .addSecuritySchemes("bearerAuth",
                        new SecurityScheme()

    .type(SecurityScheme.Type.HTTP)

                                .scheme("bearer")
                                .bearerFormat("JWT")

                            )
                )
            .info(new Info()
                .title("KinoBase API")
                .description("Platform for watching content")
                .version("1.0")
            );
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws
    Exception {
        httpSecurity
            .csrf().disable()
            .cors()
            .and()
            .httpBasic().disable()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .exceptionHandling()
            .authenticationEntryPoint((request, response, authException)
-> {
                response.setStatus(HttpStatus.UNAUTHORIZED.value());
                response.getWriter().write("unauthorized");
            })
            .accessDeniedHandler((request, response,
```

```

accessDeniedException) -> {
    response.setStatus(HttpStatus.FORBIDDEN.value());
    response.getWriter().write("forbidden");
})
.and()
.authorizeHttpRequests()
.requestMatchers("/api/v1/auth/**").permitAll()
.requestMatchers("/swagger-ui/**").permitAll()
.requestMatchers("/v3/api-docs/**").permitAll()
.anyRequest().authenticated()
.and()
.anonymous().disable()
.addFilterBefore(new JwtTokenFilter(jwtTokenProvider),
UsernamePasswordAuthenticationFilter.class);

    return httpSecurity.build();
}
}

// DataSourceConfig
@Configuration
public class DataSourceConfig {
    @Bean
    @Primary
    @ConfigurationProperties("app.datasource.main")
    public HikariDataSource hikariDataSource() {
        return DataSourceBuilder
            .create()
            .type(HikariDataSource.class)
            .build();
    }

    @Bean
    public JdbcTemplate jdbcTemplate(HikariDataSource hikariDataSource) {
        return new JdbcTemplate(hikariDataSource);
    }
}

// entities
@Data
public class Comment {
    private Long id;
    private Long contentId;
    private Long profileId;
    private String comment;
    private LocalDate addedAt;
}

@Data
public class Content {
    private Long id;
    private Set<Genre> genreList;
    private Set<Country> countryList;
    private String title;
    private String description;
    private TypeOfContent type;
    private LocalDate releaseDate;
    private String posterUrl;
}

@Data
public class Person {
    private Long id;

```

```

        private String firstName;
        private String lastName;
        private Position position;
    }

    @Data
    public class PersonContent {
        private Long personId;
        private Long contentId;
    }

    @Data
    public class Profile {
        private Long id;
        private String login;
        private String password;
        private String email;
        private String nickname;
        private Set<Role> roles;
    }

    @Data
    public class Rating {
        private Long contentId;
        private Long profileId;
        private Integer value;
    }

    @Data
    public class View {
        private Long contentId;
        private Long profileId;
        private boolean favorite;
        private Condition condition;
        private LocalDate addedAt;
    }

    // ProfileRepositoryImpl
    @Repository
    @RequiredArgsConstructor
    @Slf4j
    public class ProfileRepositoryImpl implements ProfileRepository {
        private final JdbcTemplate jdbcTemplate;

        @Override
        public Optional<Profile> findById(Long id) {
            log.info(getClass().getName() + " findById пошёл в бд" );
            String query = "SELECT id, login, password, email, nickname, role
FROM profile WHERE id = ?";
            try {
                return Optional.ofNullable(jdbcTemplate.queryForObject(query, new
ProfileRowMapper(), id));
            } catch (EmptyResultDataAccessException e) {
                return Optional.empty();
            }
        }

        @Override
        public Optional<Profile> findByLogin(String login) {
            log.info(getClass().getName() + " findByLogin пошёл в бд" );
            String query = "SELECT id, login, password, email, nickname, role
FROM profile WHERE login = ?";
            try {
                return Optional.ofNullable(jdbcTemplate.queryForObject(query, new

```

```

ProfileRowMapper(), login));
    } catch (EmptyResultDataAccessException e) {
        return Optional.empty();
    }
}

@Override
public void update(Profile profile) {
    log.info(getClass().getName() + " update пошёл в бд" );
    String query = "UPDATE profile SET password = ?, nickname = ?, email
= ? WHERE id = ?";
    jdbcTemplate.update(query, profile.getPassword(),
profile.getNickname(), profile.getEmail(), profile.getId());
}

@Override
public void create(Profile profile) {
    log.info(getClass().getName() + " create пошёл в бд" );
    String query = "INSERT INTO profile(login, password, email, nickname,
role) VALUES (?, ?, ?, ?, ?)";
    jdbcTemplate.update(query,
        profile.getLogin(),
        profile.getPassword(),
        profile.getEmail(),
        profile.getNickname(),
        profile.getRoles().stream()
            .map(Role::name)
            .toArray(String[]::new)
    );
}

@Override
public void delete(Long id) {
    log.info(getClass().getName() + " delete пошёл в бд" );
    String query = "DELETE FROM profile WHERE id = ?";
    jdbcTemplate.update(query, id);
}
}

// ProfileRowMapper
public class ProfileRowMapper implements RowMapper<Profile> {

    @Override
    public Profile mapRow(ResultSet rs, int rowNum) throws SQLException {
        Profile profile = new Profile();

        profile.setId(rs.getLong("id"));
        profile.setLogin(rs.getString("login"));
        profile.setPassword(rs.getString("password"));
        profile.setEmail(rs.getString("email"));
        profile.setNickname(rs.getString("nickname"));
        profile.setRoles(getRoles(rs.getStringArray("role")));

        return profile;
    }

    private Set<Role> getRoles(Array array) throws SQLException {
        String[] roles = (String[]) array.toArray();
        return Arrays.stream(roles)
            .map(Role::valueOf)
            .collect(Collectors.toSet());
    }
}

```

```

    }
}

// ProfileServiceImpl

@Service
@RequiredArgsConstructor
@Slf4j
public class ProfileServiceImpl implements ProfileService {
    private final ProfileRepository profileRepository;
    private final PasswordEncoder passwordEncoder;

    private final ViewService viewService;
    private final CommentService commentService;
    private final RatingService ratingService;

    @Override
    @Transactional(readOnly = true)
    public Profile getById(Long id) {
        return profileRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Profile not
found"));
    }

    @Override
    @Transactional(readOnly = true)
    public Profile getByLogin(String login) {
        return profileRepository.findByLogin(login)
            .orElseThrow(() -> new ResourceNotFoundException("Profile not
found"));
    }

    @Override
    @Transactional
    public Profile update(Profile profile) {
        profile.setPassword(passwordEncoder.encode(profile.getPassword()));
        profileRepository.update(profile);
        return profile;
    }

    @Override
    @Transactional
    public Profile create(Profile profile) {
        if (profileRepository.findByLogin(profile.getLogin()).isPresent()) {
            throw new IllegalStateException("Profile already exists.");
        }
        profile.setPassword(passwordEncoder.encode(profile.getPassword()));
        profileRepository.create(profile);

        profile.setId(profileRepository.findByLogin(profile.getLogin()).get().getId());
    };

    return profile;
}

    @Override
    @Transactional
    public void delete(Long id) {
        viewService.deleteAllByProfileId(id);
        commentService.deleteAllByProfileId(id);
        ratingService.deleteAllByProfileId(id);
        profileRepository.delete(id);
    }
}

```

```

// ProfileController

@RestController
@RequestMapping("/api/v1/profile")
@RequiredArgsConstructor
@Validated
@CacheConfig(cacheNames = {"profile", "view"})
@Tag(name = "Profile controller", description = "Profile API")
public class ProfileController {
    private final ProfileService profileService;
    private final ProfileMapper profileMapper;

    private final ViewService viewService;
    private final ViewMapper viewMapper;

    @GetMapping("/{id}")
    @Cacheable(cacheNames = {"profile"}, key = "#id")
    public ProfileDto getById(@PathVariable Long id) {
        Profile profile = profileService.getById(id);
        return profileMapper.toDto(profile);
    }

    @PutMapping("/update")
    @CachePut(cacheNames = {"profile"}, key = "#dto.id")
    public ProfileDto updateProfile(@Validated(OnUpdate.class) @RequestBody
ProfileDto dto) {
        Profile profile = profileMapper.toEntity(dto);
        Profile updatedView = profileService.update(profile);
        return profileMapper.toDto(updatedView);
    }

    @GetMapping("/{id}/views")
    @Cacheable(cacheNames = {"view"}, key = "#id")
    public List<ViewDto> getAllByProfileId(@PathVariable Long id) {
        List<View> views = viewService.getAllByProfileId(id);
        return viewMapper.toDto(views);
    }

    @DeleteMapping("/{id}")
    @CacheEvict(key = "#id")
    public void deleteById(@PathVariable Long id) {
        profileService.delete(id);
    }
}

//ProfileDto
@Data
@Schema(description = "Profile DTO")
public class ProfileDto {

    private Long id;

    @Schema(example = "anatoly")
    @NotNull(message = "login cannot be empty", groups = {OnCreate.class,
OnUpdate.class})
    private String login;

    @Schema(example = "password")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    @NotNull(message = "password cannot be empty", groups = {OnCreate.class,
OnUpdate.class})

```



```

        private String password;

        @Schema(example = "zagorovski20@gmail.com")
        @Email
        @NotNull(message = "email cannot be empty", groups = {OnCreate.class,
OnUpdate.class})
        private String email;

        @Schema(example = "termix")
        private String nickname;

        @Schema(example = "[\"ROLE_ADMIN\"]")
        private Set<Role> roles;
    }

    // JwtRequest
    @Data
    @Schema(description = "Request for login")
    public class JwtRequest {

        @Schema(example = "anatoly")
        @NotNull(message = "login cannot be empty")
        private String login;

        @Schema(example = "password")
        @NotNull(message = "password cannot be empty")
        private String password;
    }

    // JwtResponse
    @Data
    public class JwtResponse {
        private Long id;
        private String login;
        private String accessToken;
        private String refreshToken;
    }

    // JwtEntity
    @Data
    @AllArgsConstructor
    public class JwtEntity implements UserDetails {
        private Long id;
        private final String login;
        private final String password;
        private final String email;
        private final String nickname;
        private final Collection<? extends GrantedAuthority> authorities;

        @Override
        public Collection<? extends GrantedAuthority> getAuthorities() {
            return authorities;
        }

        @Override
        public String getPassword() {
            return password;
        }

        @Override
        public String getUsername() {
            return login;
        }
    }

```

```

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}
}

//JwtEntityFactory
public class JwtEntityFactory {
    public static JwtEntity create(Profile profile) {
        return new JwtEntity(
            profile.getId(),
            profile.getLogin(),
            profile.getPassword(),
            profile.getEmail(),
            profile.getNickname(),
            mapToGrantedAuthorities(new ArrayList<>(profile.getRoles()))
        );
    }

    private static List<GrantedAuthority> mapToGrantedAuthorities(List<Role>
roles) {
        return roles.stream()
            .map(Enum::name)
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());
    }
}

// JwtTokenFilter
@AllArgsConstructor
public class JwtTokenFilter extends GenericFilterBean {
    private final JwtTokenProvider jwtTokenProvider;

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        String bearerToken = ((HttpServletRequest)
servletRequest).getHeader("Authorization");
        if (bearerToken != null &&
            bearerToken.startsWith("Bearer ")) {
            bearerToken = bearerToken.substring(7);
            if (jwtTokenProvider.validateToken(bearerToken)) {
                try {
                    Authentication authentication =
jwtTokenProvider.getAuthentication(bearerToken);
                    if (authentication != null) {
                        SecurityContextHolder.getContext().setAuthentication(authentication);

```

```

        }
        } catch (ResourceNotFoundException ignored) {
        }
    }
}
filterChain.doFilter(servletRequest, servletResponse);
}
}
// JwtTokenProvider

@Service
@RequiredArgsConstructor
public class JwtTokenProvider {
    private final JwtProperties jwtProperties;
    private final UserDetailsService userDetailsService;
    private final ProfileService profileService;
    private Key key;

    @PostConstruct
    public void init() {
        this.key = Keys.hmacShaKeyFor(jwtProperties.getSecret().getBytes());
    }

    public String createAccessToken(Long profileId, String login, Set<Role>
roles) {
        Claims claims = Jwts.claims().setSubject(login);
        claims.put("id", profileId);
        claims.put("roles", resolveRoles(roles));
        Date now = new Date();
        Date validity = new Date(now.getTime() + jwtProperties.getAccess());
        return Jwts.builder()
            .setClaims(claims)
            .setIssuedAt(now)
            .setExpiration(validity)
            .signWith(key)
            .compact();
    }

    private List<String> resolveRoles(Set<Role> roles) {
        return roles.stream()
            .map(Enum::name)
            .collect(Collectors.toList());
    }

    public String createRefreshToken(Long profileId, String login) {
        Claims claims = Jwts.claims().setSubject(login);
        claims.put("id", profileId);
        Date now = new Date();
        Date validity = new Date(now.getTime() + jwtProperties.getRefresh());
        return Jwts.builder()
            .setClaims(claims)
            .signWith(key)
            .compact();
    }

    public JwtResponse refreshUserTokens(String refreshToken) {
        JwtResponse jwtResponse = new JwtResponse();
        if (!validateToken(refreshToken)) {
            throw new AccessDeniedException();
        }
        Long profileId = Long.valueOf(getId(refreshToken));
        Profile profile = profileService.getById(profileId);
        jwtResponse.setId(profileId);
    }
}

```

```

        jwtResponse.setLogin(profile.getLogin());
        jwtResponse.setAccessToken(createAccessToken(profileId,
profile.getLogin(), profile.getRoles()));
        return jwtResponse;
    }

    public boolean validateToken(String token) {
        Jws<Claims> claims = Jwts
            .parserBuilder()
            .setSigningKey(key)
            .build()
            .parseClaimsJws(token);
        return !claims.getBody().getExpiration().before(new Date());
    }

    private String getId(String token) {
        return Jwts
            .parserBuilder()
            .setSigningKey(key)
            .build()
            .parseClaimsJws(token)
            .getBody()
            .get("id")
            .toString();
    }

    private String getLogin(String token) {
        return Jwts
            .parserBuilder()
            .setSigningKey(key)
            .build()
            .parseClaimsJws(token)
            .getBody()
            .getSubject();
    }

    public Authentication getAuthentication(String token) {
        String login = getLogin(token);
        UserDetails userDetails =
userDetailsService.loadUserByUsername(login);
        return new UsernamePasswordAuthenticationToken(userDetails, "",
userDetails.getAuthorities());
    }
}
// JwtUserDetailsService

@Service
@RequiredArgsConstructor
@CacheConfig(cacheNames = {"profile"})
public class JwtUserDetailsService implements UserDetailsService {
    private final ProfileService profileService;

    @Override
    @Cacheable(key = "#username")
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Profile profile = profileService.getByLogin(username);
        return JwtEntityFactory.create(profile);
    }
}

// application.yml

```

```

spring:
  cache:
    cache-names: content, comment, person, personContent, profile, rating,
view
    caffeine:
      spec: maximumSize=100, expireAfterAccess=30m
app:
  datasource:
    main:
      driver-class-name: org.postgresql.Driver
      jdbc-url: jdbc:postgresql://localhost:5432/KinoBase
      password: '1234'
      username: postgres
      maximum-pool-size: 15
server:
  error:
    include-binding-errors: always
    include-message: always
security:
  jwt:
    secret:
aXJnZXVqaGFpdWhyZ2lodWpyZ2FlcmdpaGRqb2lqa2Fsc2RyZ2lraGFzaW91a2pyaGdjb2k7a3Nnc
mg=
    access: 3600000
    refresh: 2592000000
springdoc:
  override-with-generic-response: false

```