

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет \_\_\_\_\_«Информатика и вычислительная техника» \_

(наименование факультета)

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

(наименование кафедры)

Зав. кафедрой «ПОВТиАС»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_В. В. Долгов

(подпись) (И.О.Ф)

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г.

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе по дисциплине (модулю)\_\_\_Объектно-ориентированное\_\_\_\_\_\_\_\_ программирование­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование учебной дисциплины (модуля))

на тему: Объектно-ориентированная реализация базы данных «Видеотеки»

Автор проекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.В.Голосуев

(подпись) (И.О.Ф.)

Направление/специальность, профиль/специализация:

09.03.04 «Программная инженерия»

код направления наименование направления (специальности)

\_ ПОВТиАС \_

наименование профиля (специализации)

Обозначение курсовой работы КР.290000.000 Группа ВПР33

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Б. В. Габрельян

подпись (И. О. Ф.)

Проект (работа) защищен(а) \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

дата оценка подпись

Ростов-на-Дону

2022



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

(наименование факультета)

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

(наименование кафедры)

Зав. кафедрой «ПОВТиАС»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Долгов

(подпись) (И.О.Ф)

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022г.

**ЗАДАНИЕ**

к курсовой работе по дисциплине (модулю) \_\_\_Объектно-ориентированное\_\_\_\_\_\_ программирование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование дисциплины (модуля))

Обучающийся Голосуев Данил Витальевич Группа  ВПР33

Тема Объектно-ориентированная реализация базы данных «Видеотеки»

Срок представления проекта (работы) к защите «\_\_» \_\_\_\_\_\_\_\_\_ 2022 г.   
Исходные данные для курсовой работы

1. Задание на выполнение курсовой работы

2. Head First. Паттерны проектирования. Обновленное юбилейное издание. — СПб.: Питер, 2018. — 656 с.

Содержание пояснительной записки

ВВЕДЕНИЕ:

В разделе «Введение» описание фильмов и сериалов, а также актуальность фильмов и сериалов в современном мире.

Наименование и содержание разделов:

1. Аналитический обзор. В данном разделе происходит описание предметной области, аргументация необходимости использования информационных систем в области поиска интересующих видео и сохранения собственной истории просмотра. Постановка основной задачи.

2. Алгоритмическое конструирование. В данном разделе описан основной алгоритм работы программного средства (ПС). Рассмотрен алгоритм, применяемый для фильтрации видео по входным параметрам, а также алгоритмы авторизации пользователя и обработки запросов.

3. Программное конструирование. Раздел описывает обоснование выбора языка программирования, среды разработки, фреймворка. Описываются основные классы, относящиеся к серверной части и к клиентской.

4. Тестирование работоспособности. В данном разделе продемонстрированы основные этапы тестирования программного средства, такие как авторизация, фильтрация видео, подробный вывод информации о видео и собственного списка пользователя.

ЗАКЛЮЧЕНИЕ:

В рамках данной курсовой работы была разработана информационная система видеотеки. Все поставленные задачи решены.

Перечень графического материала:

1. Рисунок 2.1 – Алгоритм обработки простого запроса

2. Рисунок 2.2 – Алгоритм аутентификации пользователя

3. Рисунок 2.3 – Алгоритм поиска видео по критериям

4. Рисунок 4.1 – Форма аутентификации

5. Рисунок 4.2 – Отображение всех видео

6. Рисунок 4.3 – Фильтрация видео

7. Рисунок 4.4 – Страница статусов

8. Рисунок 4.5 – Страница видео

Руководитель проекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_Б. В. Габрельян\_\_

подпись, дата И.О.Ф.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. В. Голосуев

подпись, дата И.О.Ф.

СОДЕРЖАНИЕ

Изм.

Лист

№ докум.

Подпись

Дата

Лист

4

КР.290000.000

Разраб.

Голосуев Д. В.

Провер.

Габрельян Б. В.

Объектно-ориентированная реализация базы данных «Видеотеки».

Лист

Листов

56

[Введение 6](#_Toc122371140)

[1 Аналитический обзор игры 7](#_Toc122371141)

[1.1 Понятие информационной системы и видеотеки 7](#_Toc122371142)

[1.2 Потребность в информационной системе 7](#_Toc122371143)

[1.3 Постановка задачи 8](#_Toc122371144)

[1.4 Вывод по главе 9](#_Toc122371145)

[2 Алгоритмы программного средства 11](#_Toc122371146)

[2.1 Общий алгоритм работы программного средства 11](#_Toc122371147)

[2.2 Алгоритм аутентификации пользователя 13](#_Toc122371148)

[2.3 Алгоритм поиска видео по критериям 14](#_Toc122371149)

[2.4 Выводы по главе 15](#_Toc122371150)

[3. Программное конструирование 16](#_Toc122371151)

[3.1 Обоснование выбора средств разработки 16](#_Toc122371152)

[3.3 Описание основных классов серверной части 17](#_Toc122371153)

[3.3.1 Класс «Video» 17](#_Toc122371154)

[3.3.2 Класс «VideoService» 18](#_Toc122371155)

[3.3.3 Класс «VideoController» 18](#_Toc122371156)

[3.3.4 Интерфейс «VideoRepository» 19](#_Toc122371157)

[3.3.5 Класс «Status» 20](#_Toc122371158)

[3.3.6 Класс «User» 20](#_Toc122371159)

[3.4 Описание основных классов клиентской части 21](#_Toc122371160)

[3.4.1 Класс «Structure» 21](#_Toc122371161)

[3.4.2 Класс «Pages» 22](#_Toc122371162)

[3.4.3 Класс «VideoService» 23](#_Toc122371163)

[3.4.4 Класс «UserService» 24](#_Toc122371164)

[3.5 Вывод по главе 24](#_Toc122371165)

[4. Тестирование приложения 25](#_Toc122371166)

[4.1 Описание процесса тестирования 25](#_Toc122371167)

[4.2 Вывод по главе 28](#_Toc122371168)

[Заключение 29](#_Toc122371169)

[Список использованных источников 30](#_Toc122371170)

[Приложение А 32](#_Toc122371171)

[UML-диаграмма классов 32](#_Toc122371172)

[Приложение Б 34](#_Toc122371173)

[Фрагменты кода, содержащие базовые алгоритмы 34](#_Toc122371174)

Введение

Фильмы и сериалы уже давно являются большой и растущей частью нашей культуры. Сложно найти человека, у которого не любимого фильма или сериала, продолжения которого он ждет с нетерпением, либо смотрит их днями напролет, или же сопровождает с ними свой обеденный перерыв.

Но что такое кино? Это своеобразный сюжет, который обладает смыслом, выражается он в самых различных образах. Получается, человек, который хочет получить новые ощущения и эмоции, при просмотре фильма одновременно получает и определённую линию сюжета и её отображения. Во время просмотра фильма не нужно напрягаться, включать своё воображение и фантазировать для сформирования зрительного образа в своей голове. Хорошо посмотреть в фильм в такие моменты, когда необходимо дать мозгу «отдохнуть» и телу расслабиться. Также стоит отметить, что люди частенько используют кино как своеобразную разрядку в своей жизни, при этом полностью забывая о том, что существуют книги.

Очень часто в нашей жизни бывает так, что истина и правда находятся рядом. Порой просто необходимо «пощекотать» свои нервы над ужасающим триллером, или поплакать над романтической историей любви, восхитится рыцарем и смелым героем, и всё это не будет лишним. С хороших героев можно брать пример. Фильм способен оказывать на человека и его эмоциональное состояние большое влияние, порой после просмотра какого-нибудь фильма у человека возникает желание изменить свою жизнь, и он начинает это осуществлять. Самое главное, чтобы эти перемены были к лучшему. [1]

1 Аналитический обзор игры

В данном разделе рассматриваются задачи компьютерной игры «Endless Runner», приводятся ее особенности, определяется цель работы и задачи.

* 1. Понятие информационной системы и видеотеки

Информационная система (ИС) — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию.

ИС предназначена для своевременного обеспечения надлежащих людей надлежащей информацией, то есть для удовлетворения конкретных информационных потребностей в рамках определённой предметной области, при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы, базы данных и информационные услуги. [2]

По толковому словарю Ожегова видеотека — это учреждение, собирающее и хранящее видеофильмы, а также само такое собрание. [3].

1.2 Потребность в информационной системе

В настоящее время выпускается огромное количество видеоматериалов, может быть сложно выбрать что-то, подходящее по жанру и сюжету. Возникает потребность в удобном поиске интересующих видеоматериалов.

Видеотека подобно библиотеке помогает определиться с выбором фильма или сериала. Поиск в информационной системе происходит быстрее, чем ручной выбор видео по жанрам и уровня рейтинга.

После выбора и просмотра видеоматериала может возникнуть желание сохранить информацию о просмотренном видео, это можно сделать разными способами:

* записать на бумаге;
* записать в заметки на электронном устройстве;
* запомнить и стараться не забыть.

В таком случае возникает потребность в эффективном и удобном способе хранения информации о просмотренных видео, а также о запланированных видео на будущий просмотр. Более того, если зритель прерывает свой просмотр, то он может не найти информации, на каком эпизоде сериала он остановился. Это можно решить вышеперечисленными способами, но самым эффективным является использование информационной системы.

1.3 Постановка задачи

В результате аналитического обзора была поставлена цель: разработать программное средство, реализующее информационную систему видеотеки.

Программное средство должно быть реализовано с использование клиент-серверной архитектуры. Сервер ­должен использовать фреймворк Spring [4] совместно с базой данных PostgreSQL [5]. Клиентская часть должна работать в большинстве современных браузеров, используя библиотеку React [6].

Для достижения поставленной цели необходимо решить следующие задачи:

* описать основной алгоритм работы программного средства (ПС);
* выбор языка программирования и среды;
* программная реализация;
* разработать модуль интуитивно-понятного интерфейса (ПС);
* провести тестирование ПС;
* написать пояснительную записку (ПЗ).

В программном средстве будут реализованы следующие функции:

* отображение всех видео из видеотеки;
* отображение всех видео, соответствующих критериям поиска;
* отображение страницы конкретного видео;
* добавление и обновление видео через форму;
* регистрация и аутентификация пользователей;
* возможность ведения каждым пользователем собственного списка видео со статусом и прогрессом просмотра.

1.4 Вывод по главе

В данной главе были описаны причины необходимости существования информационных систем для удобства просмотра видеоматериалов, были выявлены основные задачи и методы необходимые для реализации информационной системы видеотеки. Также на основе проведенного обзора была поставлена задача и описаны функции, которые будет выполнять разрабатываемое программное средство.

2 Алгоритмы программного средства

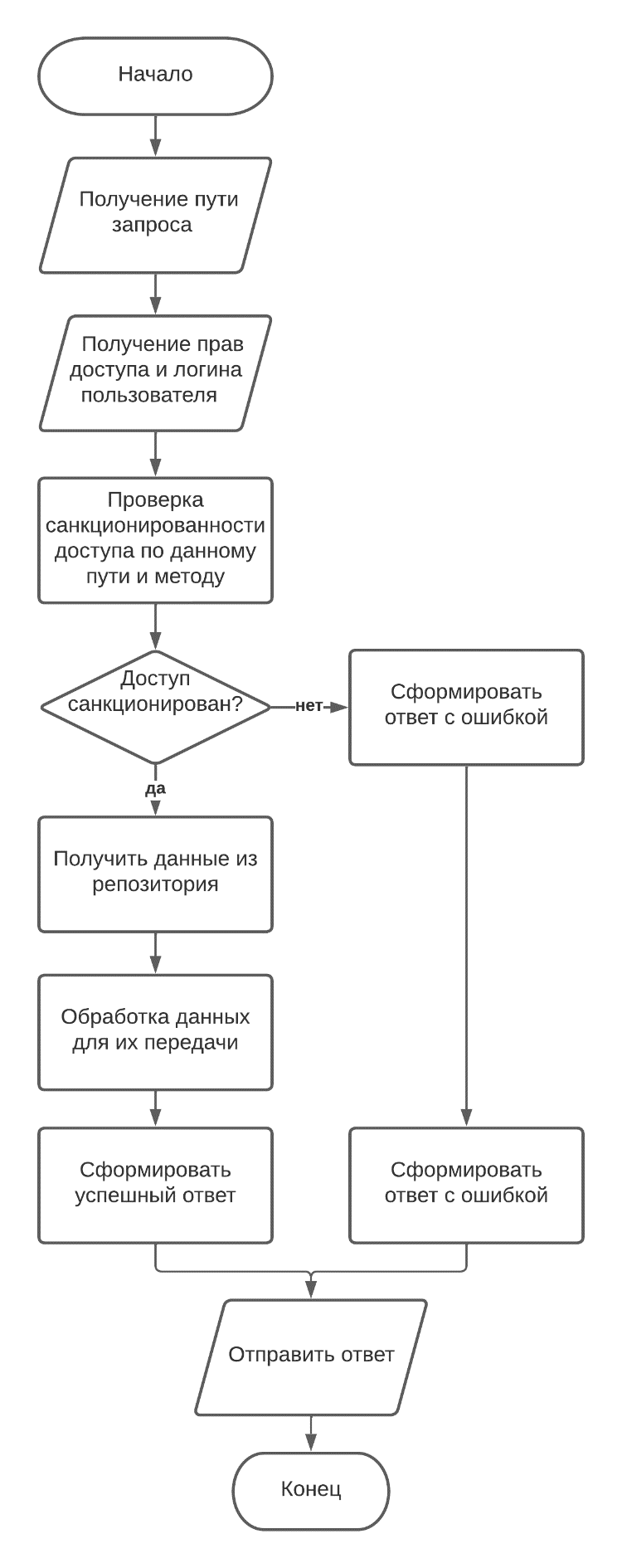
В данном разделе рассматриваются общий алгоритм работы программного средства (ПС).

2.1 Общий алгоритм работы программного средства

Программное средство предполагает использование клиент-серверной структуры передачи данных.

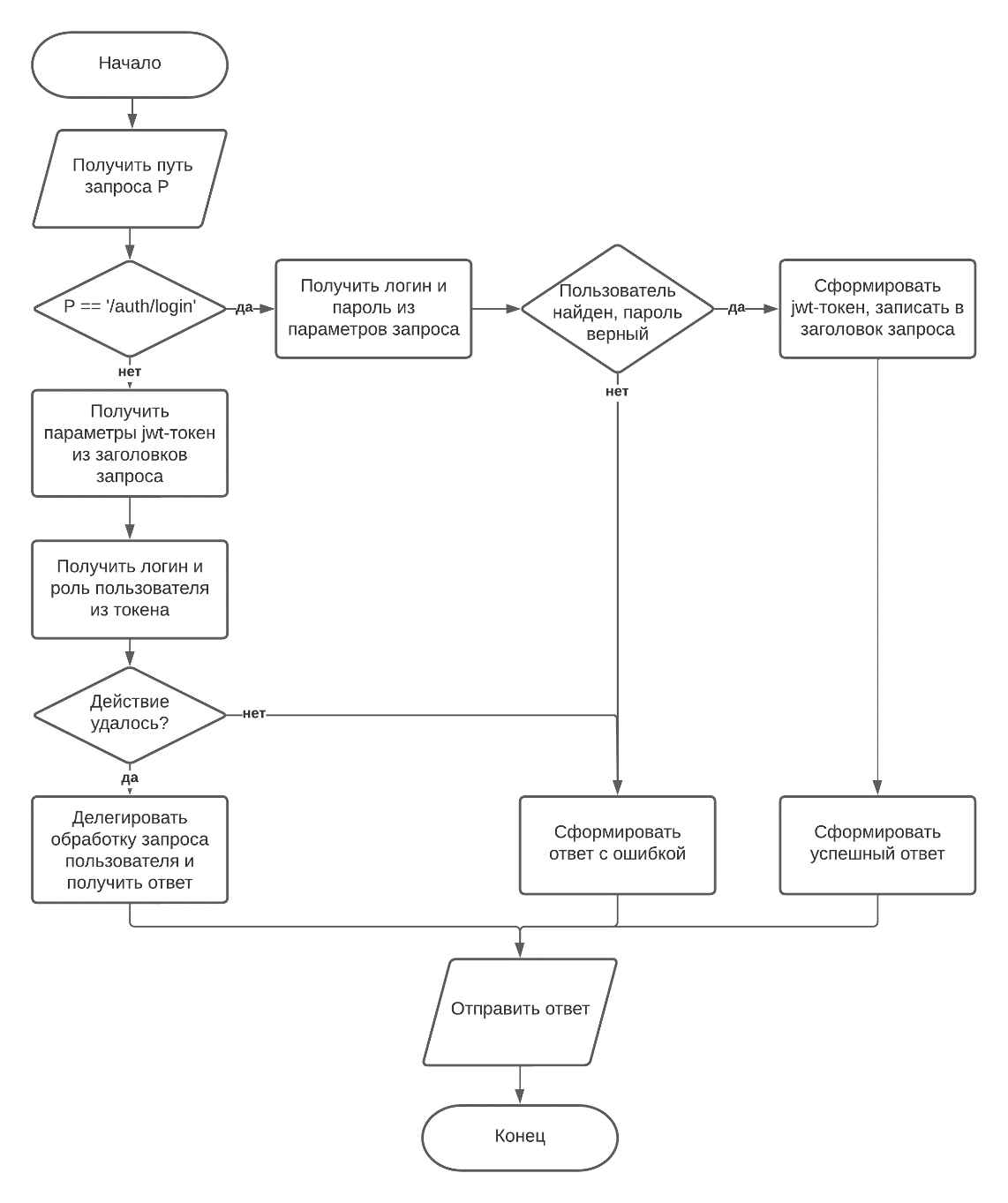
* Клиентское приложение формирует запрос и отправляет его на сервер
* Серверное приложение получает запрос
  + Обращается к базе данных
  + Формирует и возвращает ответ клиентскому приложению

Данный алгоритм ответа на запрос более подробно изображен на рисунке 2.1. На вход он получает путь запроса и параметры аутентификации, как результат возвращает данные из базы данных или генерирует ответ с ошибкой.

Рисунок 2.1 – Алгоритм обработки простого запроса

2.2 Алгоритм аутентификации пользователя

Для проверки санкционированности доступа к ресурсу, необходимо иметь возможность идентифицировать пользователя и определить его права доступа к информационной системе, которые определяются его ролью. Алгоритм аутентификации пользователя представлен на рисунке 2.2, на вход он получает путь запроса, а как результат делегирует обработку запроса или возвращает ответ с ошибкой.

Рисунок 2.2 – Алгоритм аутентификации пользователя

2.3 Алгоритм поиска видео по критериям

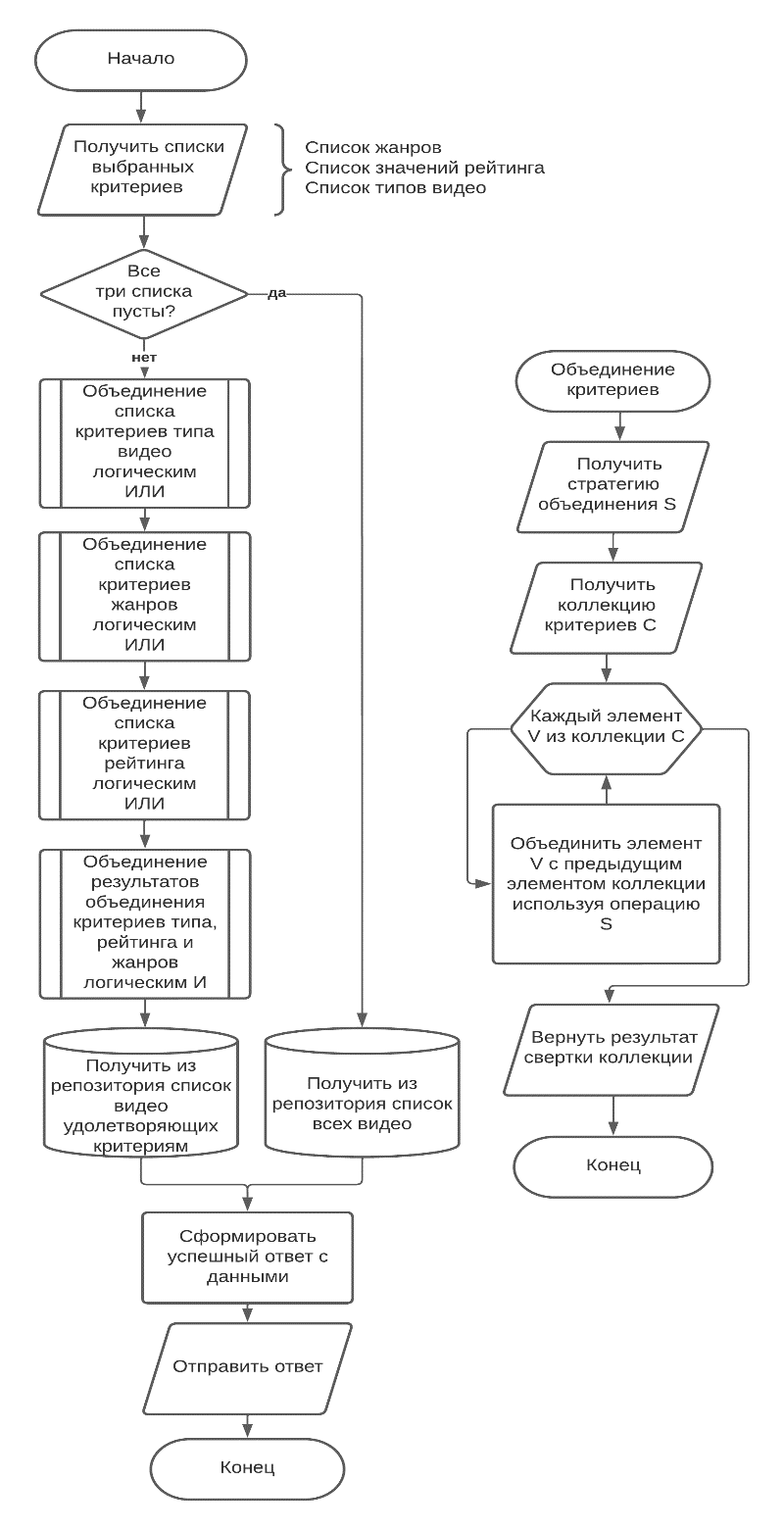
Для реализации поиска видео по критериям, таким как тип видео, рейтинг и жанры используется алгоритм, представленный на рисунке 2.3. Данный алгоритм позволяет собрать запрос к базе данных с вариативным условием поиска по спискам выбранных типов, жанров и рейтинга.

Рисунок 2.3 – Алгоритм поиска видео по критериям.

2.4 Выводы по главе

В данной главе были рассмотрены алгоритмы обработки запроса аутентификации, тривиального запроса и запроса поиска по критериям.

3. Программное конструирование

В данном разделе будут обоснованы выбор языка программирования и фреймворка, используемых для реализации программы, а также представлены основания выбора среды программирования. Будут определены и описаны основные классы и интерфейсы разрабатываемого приложения.

3.1 Обоснование выбора средств разработки

Для разработки программного средства был взят за основу язык программирования Java[7], фреймворк Spring, СУБД PostgreSQL и среда разработки IntelliJ IDEA[8].

Java — строго типизированный объектно-ориентированный язык программирования, что позволяет его использовать для создания данного проекта. Java обладает простым «си-подобным» синтаксисом, его надежность проверена временем, код на Java проще поддерживать, чем на других ЯП. Язык обладает богатой стандартной библиотекой и огромным множеством сторонних библиотек и фреймворков. Среди них есть крайне интересный фреймворк Spring. Фреймворка Spring позволяет быстро разрабатывать серверные приложения. Аналогичным фреймворком является Jakarta EE[9]. Фреймворк Spring выгодно отличается от Jakarta EE скоростью появления новых возможностей, в то же время Jakarta EE является более стабильным решением в плане совместимости со старыми версиями. Среда разработки позволяет удобно редактировать код на Java, может подключаться к СУБД PostgreSQL.

3.3 Описание основных классов серверной части

Программное средство состоит из множества классов и которые обеспечивают его работоспособность. Структура основных классов серверной части программы будет описана ниже. Связи между классами можно увидеть на UML диаграмме классов.

3.3.1 Класс «Video»

Классы модели, каким является Video, позволяют описать кортежи из реляционной базы данных объектом языка Java. Такие классы имеют закрытые поля, доступ к которым осуществляется через аксессоры, и не хранят в себе какой-либо логики. В таблице 3.1 представлены свойства класса «Video».

Таблица 3.1 – Свойства класса «Video»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Id | Long | Идентификатор видео |
| Type | Video.Type | Тип видео: фильм, сезон сериала, клип |
| Name | String | Название видео |
| description | String | Описание видео |
| Rating | Double | Рейтинг от 0 до 10 |
| Image | String | Путь до картинки |
| Episodes | Long | Количество эпизодов сезона |
| Studio | Studio | Студия производства |
| Genres | Set<Genre> | Жанры соответствующие видео |
| Contents | Set<Content> | Файлы соответствующие видео |
| comments | Set<Comment> | Комментарии соответствующие видео |
| Status | Set<Status> | Состояния соответствующие видео |

3.3.2 Класс «VideoService»

Класс «VideoService» является сервисом. Сервисы отвечают за логику предоставления данных из репозитория базы данных.

В таблице 3.2 представлены методы класса «VideoService».

Таблица 3.2 – Публичные методы класса «VideoService»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| getAll | Нет параметров | Возвращает список всех видео | List<Video> |
| getOne | Long id | Возвращет видео по его id | Video |
| Delete | Long id | Удаляет видео по id | void |
| addOrUpdate | Video video | Добавляет или обновляет видео | Long |
| getFiltered | List<Video.type> types, List<Genre> genres, List<Long> ratings | Возвращает список видео, удолетворяющих данным критериям | List<Video> |

3.3.3 Класс «VideoController»

Классы контроллеры абстрагируют логику приложения от способа передачи параметров. Они обрабатывают запросы и формируют ответы, делегируя это классам сервисам. В таблице 3.3 предоставлены методы класса «VideoController». Аналогично ему выглядят остальные контроллеры в программе.

Таблица 3.3 – Методы класса «VideoController»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| getVideos | List<Video.type> type, List<Genre> genre, List<Long> rating | Обрабатывает запросы по получению списка видео | List<Video> |
| getVideo | Long id | Обрабатывает запросы по получению видео по его идентификатору | Video |
| addOrUpdateVideo | Video video | Обрабатывает запросы на добавление и обновление видео | Long |
| deleteVideo | Long id | Удаляет видео по его идентификатору | void |
|  |  |  |  |
|  |  |  |  |

3.3.4 Интерфейс «VideoRepository»

Интерфейс «VideoRepository» является репозиторием. Репозитории описывают методы для манипуляции данными, они абстрагируют логику получения данных из источника данных. Каждый репозиторий наследует методы от интерфейса JpaRepository<T, ID>, с возможностью добавления своих собственных. В таблице 3.4 представлены методы интерфейса «VideoRepository».

Таблица 3.4 – Методы класса «VideoRepository»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| findAll | Без параметров | Получение всех видео | List<Video> |
| Save | Video video | Сохраняет видео | Video |
| deleteById | Long id | Удаляет видео по идентификатору | void |
| findById | Long id | Находит видео по идентификатору | Optional<Video> |
| findAll | Specification<Video> | Находит видео, по критериям поиска | List<Video> |
|  |  |  |  |

3.3.5 Класс «Status»

Класс «Status» описывает таблицу с состояниями видео в списке у каждого пользователя. В таблице 3.5 представлены свойства класса Status.

Таблица 3.5 – Свойства класса «Status»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Video | Video | Видео для которого выставляется статус |
| User | User | Владелец списка |
| Rating | Double | Оценка от пользователя |
| Episodes | Integer | Количество просмотренных эпизодов |
| State | Status.State | Состояние просмотра(просмотрено, смотрю, брошено, запланировано) |

3.3.6 Класс «User»

Класс «User» описывает таблицу с атрибутами пользователя. В таблице 3.6 представлены свойства класса User.

Таблица 3.6 – Свойства класса «User»

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Id | Long | Идентификатор пользователя |
| username | String | Логин пользователя |
| password | String | Пароль в захешированном виде |
| Picture | String | Путь к картинке |
| Role | Role | Роль пользователя(User, Moderator) |
| Status | Set<Status> | Состояния видео из собственного списка |
| comments | Set<Comment> | Комментарии пользователя |

Подробный листинг вышеперечисленных классов приведен в приложении Б.

3.4 Описание основных классов клиентской части

Программное средство состоит из множества классов и которые обеспечивают его работоспособность. Структура основных классов клиентской части программы будет описана ниже.

3.4.1 Класс «Structure»

Класс Structure предоставляет методы для создания основных частей страницы, определяющих ее структуру. В таблице 3.7 описаны методы класса Structure.

Таблица 3.7 — Методы класса «Structure»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| Footer | Без параметров | Возвращает html разметку для нижней части страницы | <Footer/> |
| Header | Без параметров | Возвращает html разметку для заголовочной части страницы | <Header/> |
| Main | Дочерние элементы | Возвращает html разметку для главной части страницы | <Main/> |

3.4.2 Класс «Pages»

Класс Pages предоставляет методы для создания страниц приложения. В таблице 3.8 описаны методы класса Pages.

Таблица 3.8 — Методы класса «Pages»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| LogIn | Без параметров | Возвращает разметку страницы авторизации | <LogIn/> |
| LogOut | Без параметров | Возвращает разметку страницы выхода из аккаунта | <LogOut/> |
| Registration | Без параметров | Возвращает разметку страницы регистрации | <Registration/> |
| Status | Без параметров | Возвращает разметку страницы собственного списка пользователя | <Status/> |
| UpdateImage | Номер видео | Возвращает разметку страницы обновления информации о видео | <UpdateImage/> |
| Video | Без параметров | Возвращает разметку страницы со всеми видео | <Video/> |
| VideoDetailed | Номер видео | Возвращает разметку страницы с подробным описание видео | <VideoDetailed/> |
| VideoForm | Без параметров | Возвращает страницу с формой добавления видео | <VideoForm/> |
| VideoUpdate | Номер видео | Возвращает страницу с формой обновления видео | <VideoUpdate/> |

3.4.3 Класс «VideoService»

Класс VideoService на стороне представления предоставляет методы для выполнения запросов связанных с видео. В таблице 3.9 описаны методы класса VideoService.

Таблица 3.9 — Методы класса «VideoService»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| getAll | Строка запроса с фильтрами для видео | Возвращает список всех видео, удовлетворяющих условию | Video[] |
| getOne | Номер видео | Возвращает подробную информацию о видео | Video |
| Add | Объект видео | Отправляет запрос на добавление видео в базу данных | Void |
| Delete | Номер видео | Отправляет запрос на удаления видео из базы данных | Void |
|  |  |  |  |

3.4.4 Класс «UserService»

Класс UserService на стороне представления предоставляет методы для выполнения запросов связанных с пользователем. В таблице 3.10 описаны методы класс UserService.

Таблица 3.10 — Методы класса «UserService»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Входные параметры | Описание метода | Возвращаемое значение |
| data | Без параметров | Возвращает информацию о пользователе | Номер, имя, роль пользователя |
| signIn | Имя и пароль пользователя | Отправляет запрос на авторизацию пользователя | Void |
| signUp | Имя и пароль пользователя | Отправляет запрос на регистрацию пользователя | Void |
| signOut | Без параметров | Отправляет запрос на выход из аккаунта | Void |

3.5 Вывод по главе

В данной главе был обоснован выбор языка программирования, а также среды программирования для создаваемого программного средства. Кроме того, были описаны основные классы программного средства. В таблице было приведено описание каждого из классов с указанием полей и методов.

4. Тестирование приложения

В данном разделе продемонстрированы основные этапы тестирования программного средства на основе различных входных данных.

4.1 Описание процесса тестирования

Было проведено тестирование программного средства.

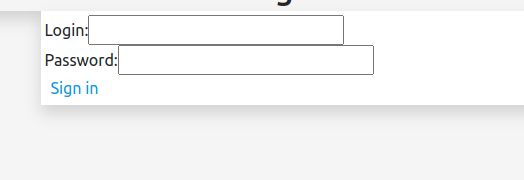
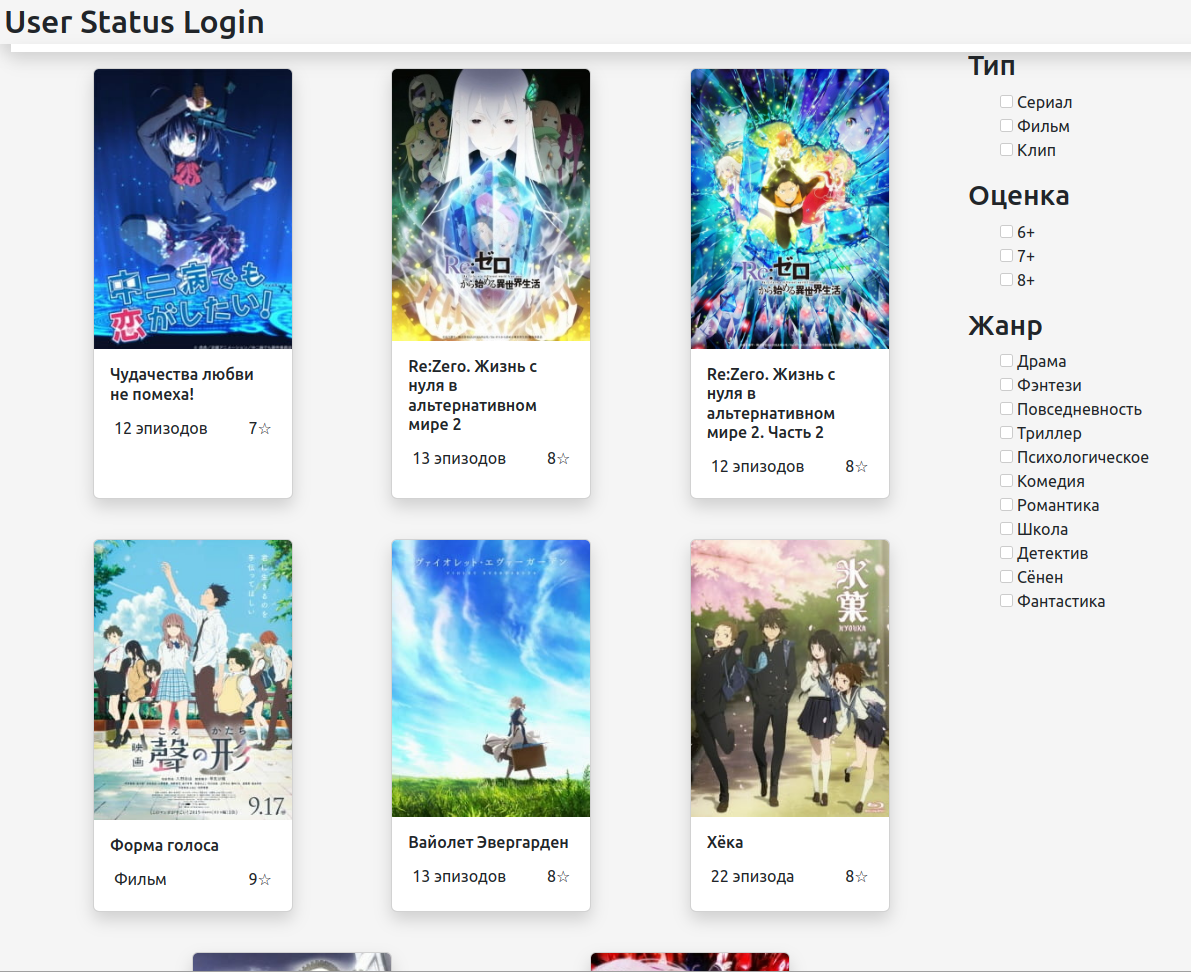
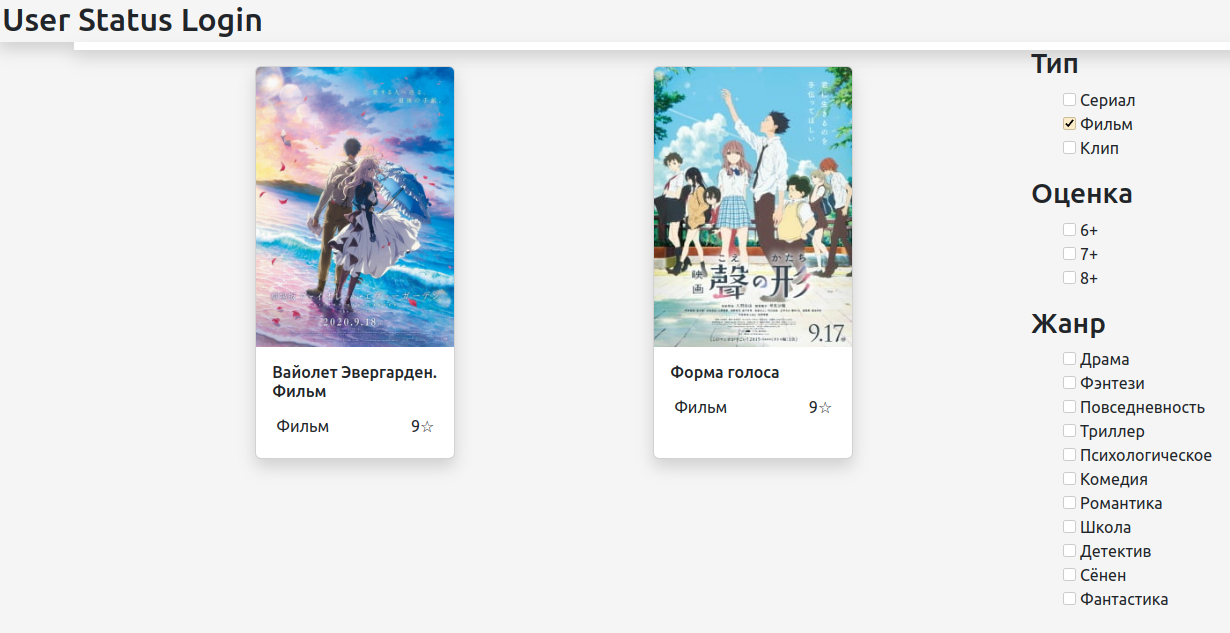
При открытии приложения появляется форма аутентификации на рисунке 4.1. Форма требует ввода логина и пароля. По нажатию кнопки происходит аутентификация пользователя.

Рисунок 4.1 – Форма аутентификации

На рисунке 4.2 представлена страница с демонстрацией всех видео. В правой части страницы находятся параметры фильтрации видео. Если они не заданы, то выводится список всех видео.

Рисунок 4.2 – Отображение всех видео

Если задать параметры фильтрации в правой стороне странице, то будут выбраны видео, удовлетворяющие этим параметрам. Возможность фильтрации видео показана на рисунке 4.3. В качестве фильтра для поиска выбран тип видео – фильм, поэтому в списке видео выведены только фильмы.

Рисунок 4.3 - Фильтрация видео

На рисунке 4.4 представлена страница со статусами видео у пользователя. Пользователь имеет возможность добавить видео в собственный список путем выставления статуса для него, кроме того пользователь может выставить количество просмотренных эпизодов и собственную оценку для видео.

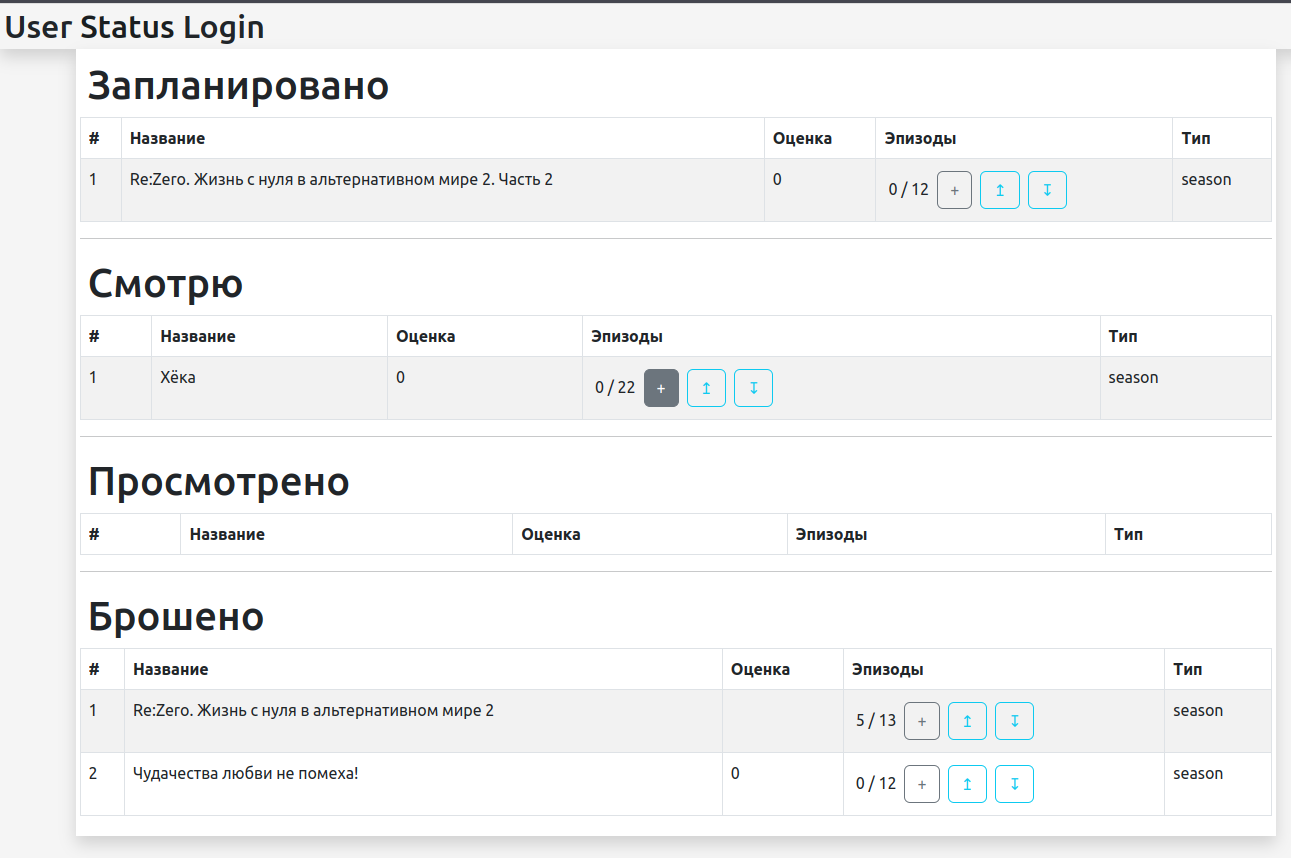


Рисунок 4.4 – Страница статусов

На рисунке 4.5 изображена страница видео. Она содержит общую информацию о видео и студии его производства. На этой странице можно выбрать состояние видео в списке пользователя, изменения отобразятся на странице статусов пользователя.



Рисунок 4.5 – Страница видео

4.2 Вывод по главе

В данной главе была показана работа программного средства. Программное средство работает стабильно.

Заключение

В рамках данной работы было разработано программное средство, выполняющее все необходимые функции информационной системы, такие как:

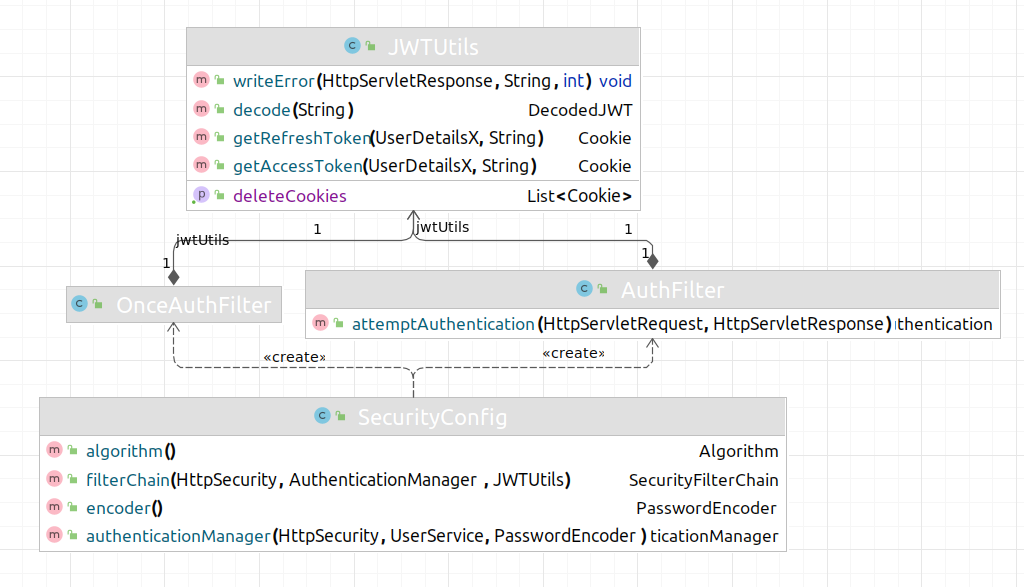
* отображение всех видео из видеотеки;
* отображение всех видео, соответствующих критериям поиска;
* отображение страницы конкретного видео;
* добавление и обновление видео через форму;
* регистрация и аутентификация пользователей;
* возможность ведения каждым пользователем собственного списка видео со статусом и прогрессом просмотра.

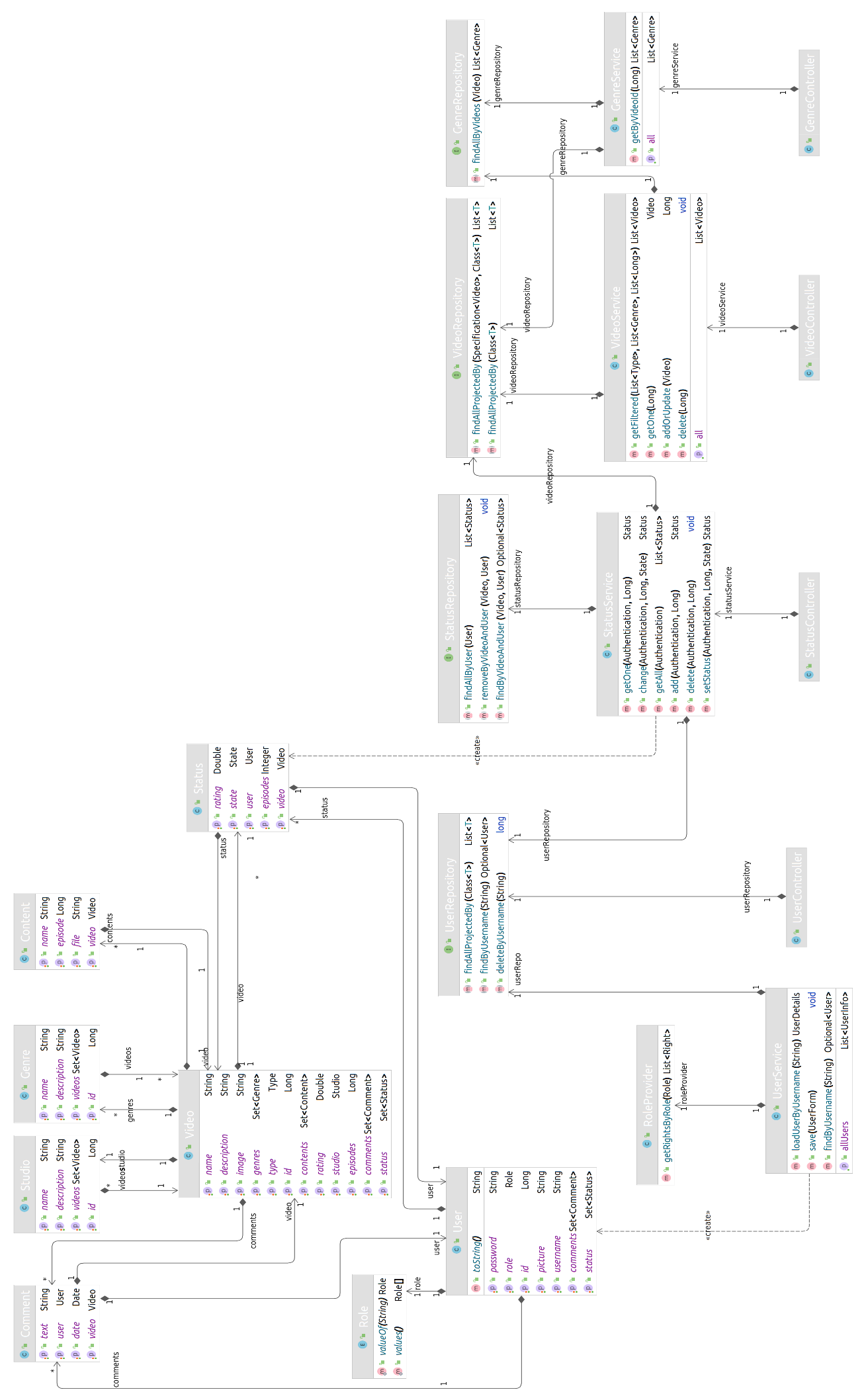
Таким образом была решена поставленная задача и выполнена проверка корректной работоспособности программного средства.

Список использованных источников

1. Почему люди смотрят фильмы [Электронный ресурс], URL: https://otvet.mail.ru/question/19816889 (дата обращения 03.12.2022).
2. Информационная система [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/Информационная\_система (дата обращения 03.12.2022).
3. Что такое видеотека [Электронный ресурс], URL: https://dic.academic.ru/dic.nsf/ogegova/277456 (дата обращения 03.12.2022).
4. Фреймворк Spring [Электронный ресурс], URL: https://spring.io/ (дата обращения 03.12.2022).
5. СУБД PostgreSQL [Электронный ресурс], URL: https://www.postgresql.org/ (дата обращения 03.12.2022).
6. Библиотека React [Электронный ресурс], URL: https://reactjs.org/ (дата обращения 03.12.2022).
7. Язык программирования Java [Электронный ресурс], URL: https://www.java.com/ru/ (дата обращения 03.12.2022).
8. Среда разработки IntelliJ IDEA [Электронный ресурс], URL: https://www.jetbrains.com/idea/ (дата обращения 03.12.2022).
9. Фреймворк Jakarta EE [Электронный ресурс], URL: https://jakarta.ee/ (дата обращения 03.12.2022).
10. Head First. Паттерны проектирования. Обновленное юбилейное издание. — СПб.: Питер, 2018. — 656 с.: ил. — (Серия «Head First O’Reilly») (дата обращения 03.12.2022).

Приложение А

UML-диаграмма классов



Приложение Б

Фрагменты кода, содержащие базовые алгоритмы

Полный код программного средства находится на съемном носителе, прилагаемом к данной работе.

@SpringBootApplication

public class VideoLibraryApplication {

public static void main(String[] args) {

SpringApplication.run(VideoLibraryApplication.class, args);

}

}

@Configuration

@RequiredArgsConstructor

@EnableMethodSecurity

@EnableWebSecurity

public class SecurityConfig {

    @Bean

    public Algorithm algorithm() {

        return Algorithm.HMAC256("No, Im not telling you a secret, mate.");

    }

    @Bean

    public PasswordEncoder encoder() {

        return new BCryptPasswordEncoder();

    }

    @Bean

    public AuthenticationManager authenticationManager(HttpSecurity httpSecurity, UserService service, PasswordEncoder encoder) throws Exception {

        return httpSecurity.getSharedObject(AuthenticationManagerBuilder.class)

                .userDetailsService(service)

                .passwordEncoder(encoder)

                .and().build();

    }

    @Bean

    public SecurityFilterChain filterChain(HttpSecurity httpSecurity, AuthenticationManager manager, JWTUtils jwtUtils) throws Exception {

        var authFilter = new AuthFilter(manager, jwtUtils);

        authFilter.setFilterProcessesUrl("/auth/login");

        return httpSecurity

                .cors(Customizer.withDefaults())

                .csrf(AbstractHttpConfigurer::disable)

                .sessionManagement()

                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)

                .and()

                .authorizeHttpRequests()

                .anyRequest()

                .permitAll()

                .and()

                .httpBasic(Customizer.withDefaults())

                .addFilter(authFilter)

                .addFilterBefore(new OnceAuthFilter(jwtUtils), UsernamePasswordAuthenticationFilter.class)

                .authenticationManager(manager)

                .build();

    }

    @Bean

    CorsConfigurationSource corsConfigurationSource() {

        CorsConfiguration configuration = new CorsConfiguration();

        configuration.setAllowedOrigins(List.of("http://localhost:3000"));

        configuration.setAllowCredentials(true);

        configuration.setAllowedMethods(List.of("\*"));

        configuration.setAllowedHeaders(List.of("\*"));

        var source = new UrlBasedCorsConfigurationSource();

        source.registerCorsConfiguration("/\*\*", configuration);

        return source;

    }

}

@Slf4j

@Service

@Transactional

@RequiredArgsConstructor

public class VideoService {

private final VideoRepository videoRepository;

private final GenreRepository genreRepository;

public List<Video> getFiltered(List<Video.Type> types,

List<Genre> genres,

List<Long> ratings) {

var spec = all();

if (types != null && !types.isEmpty())

spec = spec.and(filterType(types));

if (genres != null && !genres.isEmpty())

spec = spec.and(filterGenre(genres));

if (ratings != null && !ratings.isEmpty())

spec = spec.and(filterRating(ratings));

return videoRepository.findAll(spec);

}

private Specification<Video> filterType(List<Video.Type> types) {

if (types == null)

return null;

return Specification.anyOf(types.stream().map(this::withType).toList());

}

private Specification<Video> withType(Video.Type type) {

return (root, query, criteriaBuilder) -> criteriaBuilder.equal(root.get("type"), type);

}

private Specification<Video> filterRating(List<Long> ratings) {

if (ratings == null)

return null;

return Specification.anyOf(

ratings.stream().map(this::withRatingGreater).toList()

);

}

private Specification<Video> filterGenre(List<Genre> genres) {

if (genres == null)

return null;

/\*

ключи только для одного со сложным ключом

\*/

return (root, query, criteriaBuilder) -> {

final Expression<Set<Genre>> genres\_root = root.get("genres");

return genres.stream().reduce(

criteriaBuilder.disjunction(),

(pred, genre) -> criteriaBuilder.or(criteriaBuilder.isMember(genre, genres\_root), pred),

criteriaBuilder::or

);

};

}

private Specification<Video> withRatingGreater(Long rating) {

return (root, query, criteriaBuilder) -> criteriaBuilder.greaterThanOrEqualTo(root.get("rating"), rating);

}

private Specification<Video> all() {

return (root, query, criteriaBuilder) -> criteriaBuilder.and();

}

public List<Video> getAll() {

return videoRepository.findAll();

}

public Video getOne(Long id) throws ChangeSetPersister.NotFoundException {

return videoRepository.findById(id).orElseThrow(ChangeSetPersister.NotFoundException::new);

}

public Long addOrUpdate(Video video) {

log.info("video genres count: {}", video.getGenres().size());

return videoRepository.save(video).getId();

}

public void delete(Long id) {

videoRepository.deleteById(id);

}

public void setImagePath(Long video\_id, String image\_path) {

final var video = videoRepository.findById(video\_id).orElseThrow();

video.setImage(image\_path);

videoRepository.save(video);

}

}

@Service

@Transactional

@RequiredArgsConstructor

public class UserService implements UserDetailsService {

final UserRepository userRepository;

final RoleProvider roleProvider;

final PasswordEncoder passwordEncoder;

public UserDetailsX getPrincipal(Authentication authentication){

return (UserDetailsX)authentication.getPrincipal();

}

public User getUserByAuthentication(Authentication authentication){

final var principal = getPrincipal(authentication);

return userRepository.findById(principal.getId()).orElseThrow();

}

public List<UserInfo> getAllUsers() {

return userRepository.findAllProjectedBy(UserInfo.class);

}

public void save(String username, String password) {

if (!username.matches("^[A-Za-z0-9]{3,16}$"))

throw new RuntimeException("Логин должен быть от 3 до 16 буков или цифор");

if (!password.matches("^[A-Za-z0-9]{3,16}$"))

throw new RuntimeException("Пароль должен быть от 3 до 16 буков или цифор");

final var userOptional = findByUsername(username);

if (userOptional.isPresent())

throw new UsernameNotFoundException(username);

final var user = new User(null, username, passwordEncoder.encode(password), Role.ROLE\_USER);

userRepository.save(user);

}

public UserInfo findUserInfoByUsername(String username) {

return userRepository.getByUsername(username, UserInfo.class).orElseThrow(() -> new UsernameNotFoundException(username));

}

public Optional<User> findByUsername(String username) {

return userRepository.findByUsername(username);

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

var user = findByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException(username));

final var rights = roleProvider.getRightsByRole(user.getRole());

return new UserDetailsX(

user.getId(), user.getUsername(),

user.getPassword(), rights, user.getRole()

);

}

}

@Service

@Transactional

@RequiredArgsConstructor

public class StatusService {

final StatusRepository statusRepository;

final UserRepository userRepository;

final VideoRepository videoRepository;

final UserService userService;

public List<Status> getAll(Authentication authentication){

final var user = userService.getUserByAuthentication(authentication);

return statusRepository.findAllByUser(user);

}

public Status getOne(Authentication authentication, Long video\_id){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

return statusRepository.findByVideoAndUser(video, user).orElseThrow();

}

public Status add(Authentication authentication, Long video\_id){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

final var status = new Status(video, user);

return statusRepository.save(status);

}

public void delete(Authentication authentication, Long video\_id){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

statusRepository.removeByVideoAndUser(video, user);

}

public Status change(Authentication authentication, Long video\_id, Status.State state){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

final var status = statusRepository.findByVideoAndUser(video, user).orElseThrow();

status.setState(state);

return statusRepository.save(status);

}

public void setState(Authentication authentication, Long video\_id, Status.State state){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

final var status = new Status(video, user, 0.0, 0, state);

statusRepository.save(status);

}

public void setRating(Authentication authentication, Long video\_id, Double rating){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

final var status = statusRepository.findByVideoAndUser(video, user).orElseThrow();

status.setRating(rating);

statusRepository.save(status);

}

public void setEpisodes(Authentication authentication, Long video\_id, Integer episodes){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

final var status = statusRepository.findByVideoAndUser(video, user).orElseThrow();

status.setEpisodes(episodes);

statusRepository.save(status);

}

}

@RequiredArgsConstructor

@Service

public class ImageService {

@Value("${file.directory}")

String directory;

public String storeFile(MultipartFile file) throws IOException {

final var name = StringUtils.cleanPath(Objects.requireNonNull(file.getOriginalFilename()));

final var storage = Paths.get(directory).toAbsolutePath().normalize();

final var target\_location = storage.resolve(name);

Files.copy(file.getInputStream(), target\_location, StandardCopyOption.REPLACE\_EXISTING);

return name;

}

public Resource loadFileAsResource(String filename) throws MalformedURLException {

final var storage = Paths.get(directory).toAbsolutePath().normalize();

final var path = storage.resolve(filename).normalize();

return new UrlResource(path.toUri());

}

}

@Service

@Transactional

@RequiredArgsConstructor

public class GenreService {

final GenreRepository genreRepository;

final VideoRepository videoRepository;

public List<Genre> getAll(){

return genreRepository.findAll();

}

public List<Genre> getByVideoId(Long video\_id){

final var video = videoRepository.findById(video\_id).orElseThrow();

return genreRepository.findAllByVideos(video);

}

public Genre getReferenceById(Long genre\_id){

return genreRepository.getReferenceById(genre\_id);

}

}

@RequiredArgsConstructor

@Service

@Transactional

public class CommentService {

private final UserService userService;

private final CommentRepository commentRepository;

private final VideoRepository videoRepository;

public List<Comment> getAllVideoComments(Long video\_id) {

final var video = videoRepository.getReferenceById(video\_id);

return commentRepository.getCommentsByVideo(video);

}

public Comment addComment(Long video\_id, Authentication authentication, String text){

final var user = userService.getUserByAuthentication(authentication);

final var video = videoRepository.findById(video\_id).orElseThrow();

final var comment = new Comment(video, user, new Date(), text);

return commentRepository.save(comment);

}

}

public interface VideoRepository extends JpaRepository<Video, Long>, JpaSpecificationExecutor<Video> {

<T> List<T> findAllProjectedBy(Class<T> type);

<T> List<T> findAllProjectedBy(Specification<Video> spec, Class<T> type);

}

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username);

long deleteByUsername(String username);

<T> List<T> findAllProjectedBy(Class<T> type);

<T> Optional<T> getByUsername(String username, Class<T> type);

}

public interface StudioRepository extends JpaRepository<Studio, Long> {

}

public interface StatusRepository extends JpaRepository<Status, Long> {

List<Status> findAllByUser(User user);

void removeByVideoAndUser(Video video, User user);

Optional<Status> findByVideoAndUser(Video video, User user);

}

public interface GenreRepository extends JpaRepository<Genre, Long> {

List<Genre> findAllByVideos(Video video);

List<Genre> findAllByNameIn(Collection<String> names);

}

public interface CommentRepository extends JpaRepository<Comment, Comment.CommentId> {

List<Comment> getCommentsByVideo(Video video);

}

@Entity

@Getter @Setter @NoArgsConstructor @AllArgsConstructor

@IdClass(Status.StatusId.class)

@Table(name = "status")

public class Status {

@RequiredArgsConstructor

@EqualsAndHashCode

public static class StatusId implements Serializable {

private Long user;

private Long video;

}

public enum State {

PLANNED, WATCHING, WATCHED, ABANDONED

}

public Status(Video video, User user){

this.video = video;

this.user = user;

this.rating = null;

this.episodes = 0;

this.state = State.PLANNED;

}

@Id

@ManyToOne

@JoinColumn(name = "video\_id", nullable = false)

private Video video;

@Id

@ManyToOne

@JoinColumn(name = "user\_id", nullable = false)

private User user;

private Double rating;

private Integer episodes;

@Enumerated(EnumType.STRING)

private State state;

}

@Entity

@Getter@Setter@NoArgsConstructor@AllArgsConstructor

@Table(name = "genre")

public class Genre {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String name;

private String description;

@JsonIgnore

@ManyToMany(mappedBy = "genres", fetch = FetchType.LAZY)

private Set<Video> videos = new HashSet<>();

}

@Entity

@Getter@Setter@NoArgsConstructor@AllArgsConstructor

@IdClass(Content.ContentId.class)

@Table(name = "content")

public class Content {

@RequiredArgsConstructor

@EqualsAndHashCode

public static class ContentId implements Serializable {

private Long episode;

private Long video;

}

@Id

@Column(nullable = false)

private Long episode;

@Column(nullable = false)

private String name;

@Column(nullable = false)

private String file;

@Id

@ManyToOne(optional = false)

@JoinColumn(name = "video\_id", nullable = false)

private Video video;

}

@Entity

@IdClass(Comment.CommentId.class)

@Getter

@Setter

@NoArgsConstructor

@AllArgsConstructor

@Table(name = "comment")

public class Comment {

@RequiredArgsConstructor

@EqualsAndHashCode

public static class CommentId implements Serializable {

private Long user;

private Long video;

private Date date;

}

@Id

@ManyToOne(optional = false)

@JoinColumn(name = "video\_id", nullable = false)

private Video video;

@Id

@ManyToOne(optional = false)

@JoinColumn(name = "user\_id", nullable = false)

private User user;

@Id

@Column(nullable = false)

@Temporal(TemporalType.TIMESTAMP)

private Date date;

@Column(nullable = false)

private String text;

}

@Entity

@Getter@Setter@NoArgsConstructor@AllArgsConstructor

@Inheritance(strategy = InheritanceType.SINGLE\_TABLE)

@Table(name = "video")

public class Video {

public enum Type{

movie, season, clip

}

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

@Enumerated(EnumType.STRING)

private Type type;

@Column(nullable = false)

private String name;

@Column(length = 3000)

private String description;

@Column(nullable = false)

private Double rating;

private String image;

private Long episodes;

@ManyToOne

@JoinColumn(name = "studio\_id")

private Studio studio;

// @JsonIgnore

@ManyToMany

@JoinTable(name = "video\_to\_genre",

joinColumns = @JoinColumn(name = "video\_id"),

inverseJoinColumns = @JoinColumn(name = "genre\_id")

)

private Set<Genre> genres;

@JsonIgnore

@OneToMany(mappedBy = "video", cascade = CascadeType.ALL)

private Set<Content> contents;

@JsonIgnore

@OneToMany(mappedBy = "video", cascade = CascadeType.ALL)

private Set<Comment> comments;

@JsonIgnore

@OneToMany(mappedBy = "video", cascade = CascadeType.ALL)

private Set<Status> status;

}

@Entity

@Getter @Setter @ToString

@AllArgsConstructor @NoArgsConstructor

@Table(name = "user")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false, unique = true)

private String username;

@Column(nullable = false)

private String password;

private String picture;

@Column(nullable = false)

@Enumerated(EnumType.STRING)

private Role role;

@JsonIgnore

@OneToMany(mappedBy = "user", cascade = CascadeType.REMOVE)

private Set<Status> status;

@JsonIgnore

@OneToMany(mappedBy = "user", cascade = CascadeType.REMOVE)

private Set<Comment> comments;

public User(Long id, String username, String password, Role role) {

this.id = id;

this.username = username;

this.password = password;

this.role = role;

}

}

public enum Role {

ROLE\_USER, ROLE\_MODERATOR, ROLE\_UNAUTHORIZED

}

@Data

public class Right implements GrantedAuthority{

final private String name;

@Override

public String getAuthority() {

return name;

}

}

@Entity

@Getter @Setter @NoArgsConstructor @AllArgsConstructor

@Table(name = "studio")

public class Studio {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

private String description;

@JsonIgnore

@OneToMany(mappedBy = "studio", cascade = CascadeType.REMOVE)

private Set<Video> videos;

}

@Slf4j

@RequiredArgsConstructor

@RestController

@RequestMapping("/api/video")

public class VideoController {

final VideoService videoService;

@GetMapping

List<Video> getVideos(

@RequestParam(required = false) List<Video.Type> type,

@RequestParam(required = false) List<Genre> genre,

@RequestParam(required = false) List<Long> rating

) {

if (type == null && genre == null && rating == null)

return videoService.getAll();

else

return videoService.getFiltered(type, genre, rating);

}

@GetMapping("/{id}")

Video getVideo(@PathVariable Long id) throws ChangeSetPersister.NotFoundException {

return videoService.getOne(id);

}

@PreAuthorize("hasRole('MODERATOR')")

@PostMapping

@PutMapping

Long addOrUpdateVideo(@RequestBody Video video) {

return videoService.addOrUpdate(video);

}

@PreAuthorize("hasRole('MODERATOR')")

@DeleteMapping("/{id}")

void deleteVideo(@PathVariable Long id) {

videoService.delete(id);

}

}

@RequiredArgsConstructor

@RestController

@RequestMapping("/api/user")

public class UserController {

private final UserRepository userRepository;

@PreAuthorize("isAuthenticated()")

@GetMapping

UserInfoDTO getInfo(Authentication authentication){

final var user = (UserDetailsX)authentication.getPrincipal();

return new UserInfoDTO(user.getId(), user.getUsername(), user.getRole());

}

@PreAuthorize("hasRole('MODERATOR')")

@GetMapping("/all")

List<UserInfo> getAllUsers() {

return userRepository.findAllProjectedBy(UserInfo.class);

}

@PreAuthorize("hasRole('MODERATOR')")

@DeleteMapping("/{id}")

void deleteUserById(@PathVariable Long id){

final var user = userRepository.findById(id).orElseThrow();

userRepository.delete(user);

}

}

@RequiredArgsConstructor

@RestController

@RequestMapping("/api/studio")

public class StudioController {

final StudioRepository studioRepository;

@GetMapping

List<Studio> getAllStudios(){

return studioRepository.findAll();

}

}

@Slf4j

@RequiredArgsConstructor

@RestController

@RequestMapping("/api/status")

public class StatusController {

final StatusService statusService;

@PreAuthorize("isAuthenticated()")

@GetMapping("/all")

List<Status> getAllStatus(Authentication authentication) {

return statusService.getAll(authentication);

}

@PreAuthorize("isAuthenticated()")

@GetMapping

Status getVideoStatus(Authentication authentication, @RequestParam Long video\_id) {

final var status = statusService.getOne(authentication, video\_id);

log.info("video state = {}", status.getState());

return status;

}

@PreAuthorize("isAuthenticated()")

@DeleteMapping

void deleteStatus(Authentication authentication, @RequestParam Long video\_id) {

statusService.delete(authentication, video\_id);

}

@PreAuthorize("isAuthenticated()")

@PatchMapping

void setState(Authentication authentication, @RequestParam Long video\_id, @RequestParam Status.State state) {

statusService.setState(authentication, video\_id, state);

}

@PreAuthorize("isAuthenticated()")

@PostMapping("/rating")

void setRating(Authentication authentication, @RequestParam Long video\_id, @RequestParam Double rating){

statusService.setRating(authentication, video\_id, rating);

}

@PreAuthorize("isAuthenticated()")

@PostMapping("/episode")

void setEpisode(Authentication authentication, @RequestParam Long video\_id, @RequestParam Integer episodes){

statusService.setEpisodes(authentication, video\_id, episodes);

}

}

@RequiredArgsConstructor

@RestController

@RequestMapping("/auth")

public class LoginController {

private final UserService userService;

private final JWTUtils jwtUtils;

@PostMapping("/registration")

private UserInfo register(@RequestParam String username, @RequestParam String password) throws Exception {

userService.save(username, password);

return userService.findUserInfoByUsername(username);

}

@DeleteMapping("/logout")

private void logout(HttpServletResponse response) {

jwtUtils.getDeleteCookies().forEach(response::addCookie);

}

}

@RequiredArgsConstructor

@RestController

@RequestMapping("/image")

public class ImageController {

private final ImageService imageService;

private final VideoService videoService;

@GetMapping("/{name}")

public ResponseEntity<Resource> getImage(@PathVariable String name, HttpServletRequest request) throws IOException {

final var resource = imageService.loadFileAsResource(name);

String content\_type = request.getServletContext().getMimeType(resource.getFile().getAbsolutePath());

return ResponseEntity.ok()

.contentType(MediaType.parseMediaType(content\_type))

.header(HttpHeaders.CONTENT\_DISPOSITION, "attachment; filename=\"" + resource.getFilename() + "\"")

.body(resource);

}

@PreAuthorize("isAuthenticated() && hasRole('MODERATOR')")

@PostMapping

public void uploadImage(@RequestParam("file") MultipartFile file, @RequestParam Long video\_id) throws IOException {

final var filename = imageService.storeFile(file);

videoService.setImagePath(video\_id, filename);

}

}

@RequiredArgsConstructor

@RestController

@RequestMapping("/api/genre")

public class GenreController {

final GenreService genreService;

@GetMapping

List<Genre> getAllGenres(@RequestParam(required = false) Long video\_id) {

if (video\_id == null)

return genreService.getAll();

else

return genreService.getByVideoId(video\_id);

}

}

public class UserDetailsX extends User {

@Getter

final Role role;

@Getter

final Long id;

public UserDetailsX(Long id, String username, String password, Collection<? extends GrantedAuthority> authorities, Role role) {

super(username, password, authorities);

this.role = role;

this.id = id;

}

}

@Component

@RequiredArgsConstructor

public class RoleProvider {

@Value("#{'${rights.user}'.split(' ')}")

private final List<String> userStringRights;

@Value("#{'${rights.moderator}'.split(' ')}")

private final List<String> moderatorStringRights;

public List<Right> getRightsByRole(Role role){

final var rights = switch (role){

case ROLE\_USER -> userStringRights;

case ROLE\_MODERATOR -> moderatorStringRights;

case ROLE\_UNAUTHORIZED -> null;

};

assert rights != null;

return rights.stream().map(Right::new).toList();

}

}

@Slf4j

@Component

@RequiredArgsConstructor

public class OnceAuthFilter extends OncePerRequestFilter {

private final JWTUtils jwtUtils;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws IOException, ServletException {

// || request.getServletPath().startsWith("/shop/")

if (!request.getServletPath().startsWith("/auth/")) {

final var cookie = WebUtils.getCookie(request, jwtUtils.access\_cookie\_name);

if (cookie != null) {

String accessToken = cookie.getValue();

var decodedJWT = jwtUtils.decode(accessToken);

final var rights = Arrays.stream(decodedJWT.getClaim("rights").asArray(String.class)).map(SimpleGrantedAuthority::new).toList();

final var user = new UserDetailsX(

decodedJWT.getClaim("userId").asLong(),

decodedJWT.getSubject(), "",

rights,

Role.valueOf(decodedJWT.getClaim("role").asString())

);

final var authToken = new UsernamePasswordAuthenticationToken(user, "", rights);

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}

@Service

@Slf4j

@RequiredArgsConstructor

public class JWTUtils {

private final Algorithm algorithm;

@Value("${jwt.access\_token\_expiration}")

public long access\_token\_expiration;

@Value("${jwt.refresh\_token\_expiration}")

public long refresh\_token\_expiration;

// @Value("${spring.web.resources.static-locations}")

// List<String> locations;

@Value("${jwt.access\_cookie\_name}")

public String access\_cookie\_name;

@Value("${jwt.refresh\_cookie\_name}")

public String refresh\_cookie\_name;

public Cookie getAccessToken(UserDetailsX user, String issuer) {

final var token = JWT.create()

.withSubject(user.getUsername())

.withExpiresAt(new Date().toInstant().plusSeconds(access\_token\_expiration))

.withIssuer(issuer)

.withClaim("rights", user.getAuthorities().stream().map(GrantedAuthority::getAuthority).toList())

.withClaim("role", user.getRole().toString())

.withClaim("userId", user.getId())

.sign(algorithm);

final var cookie = new Cookie(access\_cookie\_name, "Bearer\_" + token);

cookie.setMaxAge((int) access\_token\_expiration);

cookie.setHttpOnly(true);

cookie.setPath("/");

return cookie;

}

public Cookie getRefreshToken(UserDetailsX user, String issuer) {

final var token = JWT.create()

.withSubject(user.getUsername())

.withExpiresAt(new Date().toInstant().plusSeconds(refresh\_token\_expiration))

.withIssuer(issuer)

.sign(algorithm);

final var cookie = new Cookie(refresh\_cookie\_name, "Bearer\_" + token);

cookie.setMaxAge((int) refresh\_token\_expiration);

cookie.setHttpOnly(true);

cookie.setPath("/");

return cookie;

}

public DecodedJWT decode(String token) {

if (token != null && token.startsWith("Bearer\_")) {

String trimmed = token.substring("Bearer\_".length());

JWTVerifier verifier = JWT.require(algorithm).build();

return verifier.verify(trimmed);

}

else throw new RuntimeException("No token is present.");

}

public List<Cookie> getDeleteCookies() {

var cookies = new ArrayList<Cookie>(2);

for (var cookieName: List.of(access\_cookie\_name, refresh\_cookie\_name)) {

var cookie = new Cookie(cookieName, "");

cookie.setHttpOnly(true);

cookie.setMaxAge(0);

cookie.setPath("/");

cookies.add(cookie);

}

return cookies;

}

public void writeError(HttpServletResponse response, String message, int error) {

log.error("Error on token processing attempt: {}", message);

response.setContentType(MediaType.APPLICATION\_JSON\_VALUE);

response.setHeader("error", message);

response.setStatus(error);

}

}

@RequiredArgsConstructor

@Slf4j

public class AuthFilter extends UsernamePasswordAuthenticationFilter {

private final AuthenticationManager manager;

private final JWTUtils jwtUtils;

@Override

public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException {

String username = request.getParameter("username");

String password = request.getParameter("password");

log.info("Username: {}, password: {}", username, password);

return manager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

}

@Override

protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain, Authentication authResult) throws IOException{

final var user = (UserDetailsX) authResult.getPrincipal();

log.info("Username: {} successfully logged in.", user.getUsername());

response.addCookie(jwtUtils.getAccessToken(user, request.getRequestURL().toString()));

response.addCookie(jwtUtils.getRefreshToken(user, request.getRequestURL().toString()));

}

}