

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**ПРОГРАММНЫЙ АГРЕГАТОР СИСТЕМЫ
МГНОВЕННОГО ОБМЕНА
СООБЩЕНИЯМИ (TELEGRAM)**

Студент

Вожов Вожов Илья Владимирович
(подпись, дата) 19.06.2023 (ФИО полностью)

Руководитель

А.В. А.В. Дубравин
(подпись, дата) 19.06.2023 (фамилия, инициалы)

Нормоконтролёр

А.С. А.С. Бычков
(подпись, дата) 19.06.2023 (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 19.06.2023 г. № 4)

Заведующий кафедрой

М.А. М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 20.06.2023 г. № 24)

Секретарь ГЭК

Е.И. Е.И. Гуляев
(подпись) (фамилия, инициалы)

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» 04 2023 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Вожжов Илья Владимирович гр. 19ВВП1 факультета вычислительной техники.
2. Тема работы «Программный агрегатор системы мгновенного обмена сообщениями (telegram)»

Тема утверждена приказом ПГУ № 458/0 от "26" 04 2023 г.

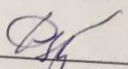
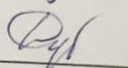
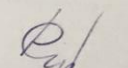
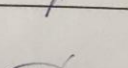
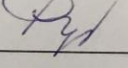
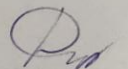
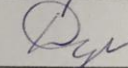
3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ Дубравин А.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать приложение «Программный агрегатор системы мгновенного обмена сообщениями (telegram)», подбирающее кинокартины по определенным критериям пользователя и позволяющее пользователю производить обмен информации с тематическим сообществом в формате мгновенного сообщения. Приложение должно быть спроектировано с применением клиент-серверной архитектуры. Клиентской частью программы должен являться чат-бот в мессенджере Telegram. Система должна включать в себя функционал подбора кинофильмов в соответствии с предпочтениями пользователя.

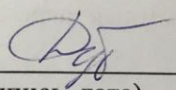
5. Перечень подлежащих разработке вопросов

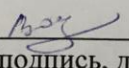
Требуется произвести анализ предметной области, изучить аналоги существующих систем подбора фильмов, изучить актуальные способы анализа текстовых и графических данных, изучить способы проектирования и разработки клиент-серверного приложения с применением REST API методологии. Необходимо изучить функционирование фреймворка Spring и языка программирования Java.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований к реализации чат-ботов	10	31.04.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке клиент-серверных приложений	50	03.05.2023-07.05.2023	
4.	Описание алгоритмов получения и анализа данных из сторонних источников в сети интернет	40	08.05.2023-11.05.2023	
5.	Разработка приложения «Программный агрегатор системы мгновенного обмена сообщениями (telegram)»	130	12.05.2023-27.05.2023	
6.	Отладка разработанного приложения	20	27.05.2023-29.05.2023	
7.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы  Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент  Вожжов И.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)
20.06.2023

Реферат

Работа содержит 87 страницы формата А4, 22 рисунка, 1 таблицу, 3 источника

БАЗА ДАННЫХ, МГНОВЕННЫЕ СООБЩЕНИЯ, ПАТТЕРН MVC, POSTGRESQL, JAVA, SPRING

Объект разработки – Программный агрегатор системы мгновенного обмена сообщениями.

Целью дипломной работы является разработка приложения для мгновенного обмена информацией с тематическим сообществом о фильмах, которые разработанный телеграм – бот нашел в сети интернет и рекомендовал пользователю. Цель данного проекта достигается путем использования специального API и базы данных.

В процессе дипломного проектирования были решены задачи сбора информации, структурирования и обработки информации о фильмах с последующей передачей информации тематическому сообществу.

В результате проведенной работы создано и протестировано клиент-серверное приложение, соответствующее заданию на дипломное проектирование.

В пояснительной записке представлены: анализ предметной области, проектирование и реализация приложения, технология разработки и отладки программы, раздел БЖД.

Графическая часть содержит модели БД и UML-диаграммы приложения.

ПГУ1.090301.2.001 ПЗ				
Изм.	Лист	№ докум	Подпись	Дата
Разраб.		Вожжов И.В.		
Провер.		Дубравин А.В.		
Н. Контр.		Бычков А.С.		
Утверд.		Митрохин М.А.		

Лит.	Лист	Листов
	2	129

Программный агрегатор
системы мгновенного обмена
сообщениями (telegram)

ФВТ, каф. ВТ
гр. 19ВВП1

Содержание

Перечень сокращений и обозначений.....	4
Введение	5
1 Программный агрегатор системы мгновенного обмена сообщениями.....	7
1.1 Предпроектный анализ.....	7
1.1.1 История развития чат-ботов	7
1.1.2 Анализ предметной области и аналогов.....	8
1.1.3 Анализ стека технологий	8
1.1.4 Выбор СУБД.....	13
1.1.5 Выбор среды разработки	16
1.2 Проектирование и разработка	17
1.2.1 Реализация сбора данных	18
1.2.2 Реализация серверной части	19
1.2.3 Разработка БД.....	23
1.2.4 Разработка клиентской части.....	25
1.3 Анализ результатов	27
1.3.1 Тестирование приложения	28
2 Охрана труда.....	33
2.1 Характеристики потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ	34
2.2 Классификация и категорирование помещения, в котором производится разработка	35
2.2.1 Классификация по взрывопожарной и пожарной опасности.....	36
2.2.2 Классификация по опасности поражения электрическим током.....	37
2.3 Санитарно-гигиеническая характеристика производства	38
Приложение А – <i>UML</i> диаграммы.....	43
Приложение Б – Листинг программы	47
Приложение В - Презентация.....	82

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

Паттерн – шаблон проектирования

СУБД – система управления базами данных

ЭВМ – электронно-вычислительная машина

API (Application Programming Interface) – программный интерфейс приложения

CRUD – акроним, обозначающий четыре базовые функции, используемые при работе с базами данных: создание (англ. *create*), чтение (*read*), модификация (*update*), удаление (*delete*).

MVC (Model-View-Controller) – «Модель-Представление-Контроллер» или «Модель-Вид-Контроллер»

				ПГУ1.090301.2.001 ПЗ	Лист
					4
Изм.	Лист	№ докум	Подпись Дата		

Введение

В современном мире стремительно развивающихся технологий люди сталкиваются с информационным перенасыщением. Одной из сфер, где поиск необходимой информации занимает большую часть времени, является кинематография. При этом, для многих любителей кино поиск нужного фильма может быть трудным и затратным процессом. Для упрощения этой задачи, было принято решение разработать телеграм бота для поиска кино.

Телеграм бот представляет из себя программу, которая использует искусственный интеллект для поиска информации и общения с пользователями. Разработка бота для поиска кино может повысить удобство и эффективность поиска необходимой информации и таким образом увеличить удовлетворенность пользователей. Основная задача бота - помочь любителям кино находить фильмы по заданным параметрам быстро и эффективно, а также предоставить дополнительную информацию о фильме, например, дату выпуска, актерский состав и рейтинг.

Чат - боты являются эффективным инструментом построения коммуникаций с клиентами, являясь новым каналом таких коммуникаций. По прогнозам в 2022 году за счет использования чат ботов взаимодействие с клиентами будет на 90% автоматизировано.

Согласно статистике, 2 млрд. человек используют мессенджеры, а за период 2016–2023 гг. число пользователей мессенджеров вырастет почти на миллиард человек. Кроме того, активность пользователей очень высока: средний пользователь *WhatsApp* проводит в мессенджере 195 минут. «ВКонтакте» является одной из самых популярных социальных сетей в России. 38,2 млн пользователей из РФ заходят в данную сеть ежемесячно, а 23 млн - ежедневно.

Аудитория мессенджеров в России составляет более 50 млн человек. Более половины из них, являются и пользователями *WhatsApp* и *Viber*.

				ПГУ1.090301.2.001 ПЗ	Лист
					5
Изм.	Лист	№ докум	Подпись Дата		

Далее идет Skype и приложения социальных сетей «ВКонтакте», *Facebook Messenger* и Одноклассники. Аудитория *Telegram* составляет 24 млн пользователей. При этом именно *Telegram* обладает наибольшим потенциалом в сфере чат-бот разработок.

Выше сказанное показывает актуальность тематики реализуемого в дипломной работе приложения.

С точки зрения технической реализации устойчивого к трафику приложения для необходимо реализовать программное приложение с архитектурой, состоящей из следующих частей: клиентская часть, серверная часть с использованием СУБД, а также внешний *API* с базой данных о различных кинокартинах.

				ПГУ1.090301.2.001 ПЗ	Лист
					6
Изм.	Лист	№ докум	Подпись Дата		

1 Программный агрегатор системы мгновенного обмена сообщениями

1.1 Предпроектный анализ

1.1.1 История развития чат-ботов

Термин «чат-бот» придумал Майкл Молдинг в 1994 г. для описания разговорных программ.

Само понятие «чат-бот» вошло в массовый обиход в начале 1990-х., однако концепция и технологии для его реализации восходят к 1950-м, когда о подобного рода решениях писали Алан Тьюринг и Джозеф Визенбаум.

Чат-боты работают в ставшей уже привычной для многих людей среде виртуального общения — это мессенджеры и чаты социальных сетей.

В 1995 году компания *A.L.I.C.E. (Artificial Linguistic Internet Computer Entity)* создала новый чат-бот, который был способен общаться с пользователями на естественном языке. Этот чат-бот получил название *ALICE* и стал первым чат-ботом, который использовал искусственный интеллект для общения с людьми.

С развитием интернета и мобильных устройств чат-боты стали все более популярными. В 2011 году компания *Apple* запустила *Siri* - первый голосовой помощник, который был встроен в операционную систему *iOS*. В 2016 году *Facebook* запустил свою платформу для создания чат-ботов - *Messenger Platform*, которая позволяет разработчикам создавать ботов для общения с пользователями через *Facebook Messenger*.

Сегодня чат-боты используются в различных сферах, включая бизнес, медицину, образование и многое другое. Они помогают автоматизировать процессы, улучшать обслуживание клиентов и повышать эффективность работы компаний. С развитием технологий искусственного интеллекта и машинного обучения чат-боты становятся все более умными и способными общаться с пользователями на более высоком уровне.

				ПГУ1.090301.2.001 ПЗ	Лист
					7
Изм.	Лист	№ докум	Подпись Дата		

1.1.2 Анализ предметной области и аналогов

Для разработки телеграм бота, который будет искать кино, необходимо знать, как находятся и рекомендуются фильмы, какие источники информации использовать и какие алгоритмы поиска и рекомендаций применять.

Важными факторами для анализа предметной области являются:

- какие источники информации о кино существуют (например, IMDb, Kinopoisk) и как они работают;
- какие критерии поиска могут использоваться (например, жанр, год, страна производства);
- какие алгоритмы рекомендаций существуют (например, коллаборативная фильтрация, контентная рекомендация);
- как проводят тестирование и оценку качества рекомендаций.

Перед разработкой телеграм бота, который будет искать кино, необходимо изучить уже существующие аналоги и выяснить, что они уже делают и что нужно улучшить или добавить. Несколько известных аналогов, которые стоит проанализировать, включают:

- @hd_kinobot - бот в *Telegram*, который ищет *HD* фильмы;
- @IMDbBot - бот в *Telegram*, который анализирует фильмы на основе базы данных *IMDb*;
- @kinopoisk_bot - бот в *Telegram*, который позволяет искать фильмы и получать информацию о них.

При анализе аналогов необходимо обратить внимание на функциональность, качество рекомендаций и удобство использования. Также стоит узнать, какие ограничения имеют эти боты, например, на доступные базы данных или жанры, и каким образом они могут быть улучшены.

1.1.3 Анализ стека технологий

На текущий момент есть широкий выбор языков программирования, как правило высокоуровневых, но бывают и исключения, для написания клиент-серверных программ: *Java*, *C#*, *Python*, *C++* и др.

				ПГУ1.090301.2.001 ПЗ	Лист
					8
Изм.	Лист	№ докум	Подпись Дата		

C++ - компилируемый язык программирования со статической типизацией. В данном языке разрешена работа с памятью, что при правильном использовании позволяет писать эффективный код. Данный язык является сложным в освоении и клиент-серверное приложение требует от разработчика много времени и профессионального опыта. Также имеется возможность создания обобщенных алгоритмов для разных типов данных, их специализация, и вычисления на этапе компиляции, используя шаблоны. Сложно находить ошибки в коде. Если, например, для C# есть такие инструменты как *ReSharper*, которые ищет ошибки за вас, то в C++ отсутствуют подобные аналоги. Для портируемости приложений нужно соблюдать требования стандарта, чтобы не возникло ошибок при работе с другим компилятором.

C# - как и C++, компилируемый язык программирования со статической типизацией. Развивающийся язык с большим сообществом программистов. Не такой высокий порог вхождения как у C++, поддерживает функциональное программирование. Благодаря покупке *Xamarin* на C# теперь можно писать под *Android* и *iOS*. Из недостатков можно выделить: ориентированность на *.NET* платформу.

Python – объектно-ориентированный интерпретируемый язык программирования высокого уровня с динамической типизацией. На *Python* есть возможность в короткие сроки разработать приложение в силу большого количества различных готовых инструментов. Так как язык не был разработан для какой-то конкретной цели, из-за своих преимуществ применяется в разных областях, от тестирования до научных исследований. В свою очередь этот язык программирования работает медленнее чем вышеперечисленные, а также трудно переносится на другие платформы из-за зависимости языка от библиотек.

PHP – язык программирования с динамической типизацией. Изучение языка занимает меньше времени чем на аналогах, но в то же время данный

				ПГУ1.090301.2.001 ПЗ	Лист
					9
Изм.	Лист	№ докум	Подпись Дата		

язык имеет достаточно узкое направление – веб разработка. Также у данного языка есть проблемы с безопасностью, несмотря на активную поддержку языка.

Java – язык программирования со статической типизацией. Это язык высокого уровня, который явно следует основным принципам ООП. Этот язык отличается своей безопасностью, а также является кроссплатформенным за счет того, что код запускается на виртуальной *Java machine*. Также язык имеет очень большое сообщество программистов. Из недостатков языка выделяют низкую скорость работы и многословность.

Go - компилируемый многопоточный язык программирования. Разработан в *Google*. Молодой и активно растущий язык программирования. Идеально подходит для реализации серверных приложений, однако библиотек для языка не так и много, что затрудняет его использование при реализации графического интерфейса клиентских приложений, а также нейронных сетей.

JavaScript – язык программирования с динамической типизацией. В начале своего существования использовался преимущественно как язык сценариев для интерактивных веб-страниц, однако, со временем, язык обрел популярность и для него появилось средство транслирования в машинный код – *NodeJS*. Это позволило использовать язык в самых различных сценариях. Язык поддерживает большое количество библиотек и фреймворков. Также преимуществом языка является автоматическое управление памятью.

Учитывая все преимущества и недостатки представленных языков программирования – был выбран язык *Java*.

В качестве фреймворка для проекта – был выбран *Spring Framework*.

Spring - один из самых популярных фреймворков для создания веб-приложений на *Java*. Фреймворк — готовая модель в *IT* для быстрой разработки, на основе которой можно дописать собственный код. Он задает

				ПГУ1.090301.2.001 ПЗ	Лист
					10
Изм.	Лист	№ докум	Подпись Дата		

структуру, определяет правила и предоставляет необходимый набор инструментов для создания проекта. Но не обязательно именно так. В *Spring* например стараются по максимуму отойти от такой жесткой связности (когда ваши классы напрямую зависят от каких-то классов/интерфейсов из этого фреймворка), и используют для этой цели аннотации. Фреймворки подходят для создания как крупных проектов, так и простых сайтов и приложений, которые планируется развивать в будущем. Они позволяют правильно выстроить бизнес-логику. В основном фреймворки используются для создания калькуляторов, интернет-магазинов с нестандартным функционалом, собственных *CRM*, которые нельзя создать на *CMS*, десктопных и мобильных приложений. В отличие от *CMS*, *frameworks* — это низкоуровневое решение, которое обладает большей производительностью и гибкостью. Позволяет получить готовый каркас для проекта без потери гибкости в плане функционала.

По сути *Spring Framework* представляет собой просто контейнер, служащий для внедрения зависимостей, с несколькими удобными слоями: доступ к базе данных, прокси, аспектно-ориентированное программирование, *RPC*, веб-инфраструктура *MVC*. Это все позволяет гораздо быстрее и удобнее создавать *Java*-приложения.

Spring Boot - это фреймворк для разработки веб-приложений на языке *Java*. Он основан на фреймворке *Spring* и предоставляет удобный способ создания приложений, которые можно запускать как обычные *Java*-приложения.

Основными преимуществами *Spring Boot* являются:

- простота использования: *Spring Boot* предоставляет удобный способ создания приложений без необходимости настройки сложной конфигурации;
- быстрый запуск: *Spring Boot* позволяет запускать приложения быстрее, чем традиционные приложения на основе *Spring*;

- гибкость: *Spring Boot* поддерживает различные типы приложений, включая веб-приложения, микросервисы;
- удобство тестирования: *Spring Boot* предоставляет удобный способ тестирования приложений с помощью встроенных инструментов;
- поддержка множества баз данных: *Spring Boot* поддерживает множество баз данных, включая *MySQL*, *PostgreSQL*, *Oracle* и других;
- интеграция с другими инструментами: *Spring Boot* интегрируется с другими инструментами, такими как *Maven*, *Gradle*, *Eclipse*.

Spring Boot также предоставляет множество других функций, таких как автоматическая конфигурация, управление зависимостями и т.д. В целом, *Spring Boot* является мощным инструментом для разработки веб-приложений на языке *Java*.

MVC – набор архитектурных решений для проектирования сложных информационных систем с пользовательским интерфейсом. Данная архитектура состоит из трех составляющих компонентов: модели, вида, контроллера.

Модель – содержит всю основную бизнес логику приложения.

Вид – данный модуль предназначен для отображения данных конечному пользователю.

Контроллер – данное звено отвечает за обработку действий пользователя.

На рисунке 1.1 изображена архитектура *MVC*

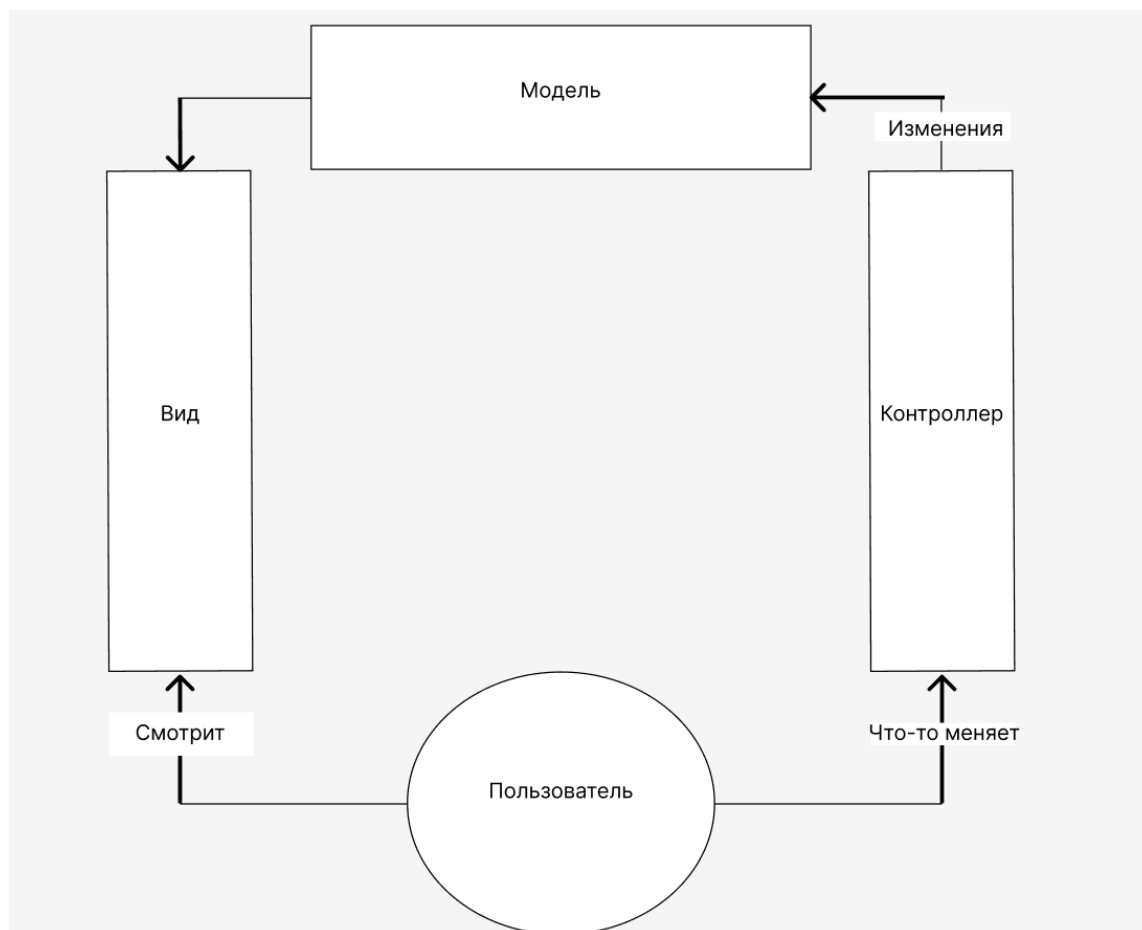


Рисунок 1.1 – Схема архитектуры MVC

1.1.4 Выбор СУБД

В данное время существует огромное множество систем управления базами данных. Их подразделяют на два основных вида: реляционные (*SQL*) и нереляционные (*NoSQL*). Основное отличие данных СУБД в способе хранения данных.

Реляционные базы данных - это базы данных, которые используют реляционную модель данных для организации и хранения информации. В реляционной модели данные представлены в виде таблиц (реляций) с определенными правилами и связями между ними. Реляционные базы данных используют язык *SQL* для управления данными. Они обычно хранят информацию о связях между различными сущностями внутри базы данных, которые могут быть связаны через первичные и вторичные ключи. Примеры реляционных СУБД: *PostgreSQL*, *MySQL*, *Oracle*, *Microsoft SQL Server*.

Суть реляционной модели заключается в том, что основанная на математической теории множеств она рассматривает таблицы как отдельные множества, объединённые по определённому признаку (рисунок 1.2).

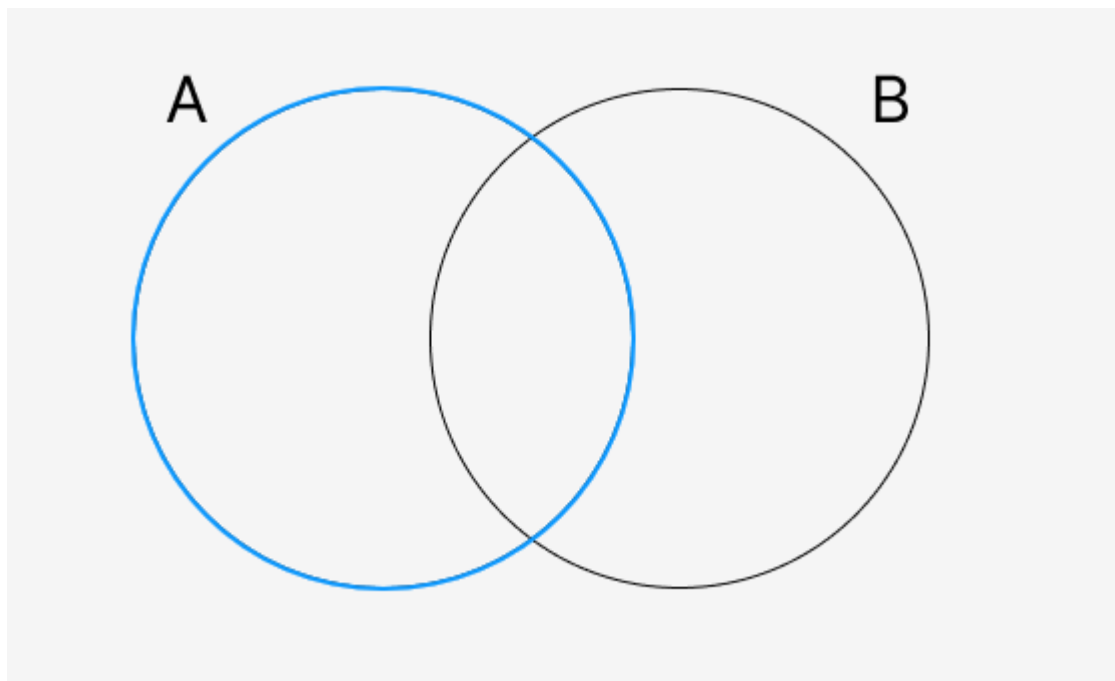


Рисунок 1.2 – Операция над таблицами (реляционная модель)

Нереляционные базы данных (*NoSQL*) - это базы данных, которые не используют реляционную модель данных. Нереляционные базы данных обычно используются там, где реляционные базы данных не могут обработать такое объем информации или не могут быть оптимизированы для различных типов запросов. *NoSQL* базы данных сохраняют данные не в таблицах, а в структурированных документах, например, в *JSON* и *XML* форматах. Нереляционные базы данных могут быть колоночными, ключ-значение, документными и графовыми. Примеры нереляционных СУБД: *Redis*, *MongoDB*, *Cassandra*, *Couchbase*. Пример организации нереляционной базы данных представлен на рисунке 1.3.

Ключ	Документ
100	{ "filmId": 435, "nameRu": "Зеленая миля", "nameEn": "The Green Mile", "type": "FILM", "year": "1999", "description": "Пол Эджкомб ...", "filmLength": "03:09" }
101	{ "filmId": 198891, "nameRu": "Лесной воин", "nameEn": "Forest Warrior", "type": "FILM", "year": "1996", "description": "Живописные...", "filmLength": "01:33" }

Рисунок 1.3 – Пример хранения данных в нереляционной СУБД

Нереляционные базы данных имеют множество преимуществ. Они могут обрабатывать большие объемы неструктурированных данных и могут масштабироваться гораздо проще, поскольку они не требуют такого жесткого определения схемы и связей между различными типами данных. Реляционные базы данных имеют более сильное внутреннее противодействие, поскольку они основаны на формализованном представлении данных в форме таблиц с явно определенными связями между ними.

Для реализации проекта была выбрана реляционная СУБД – *PostgreSQL*.

PostgreSQL - это мощная система управления реляционными базами данных, нацеленная на достижение высокой производительности, надежности и расширяемости. Вот несколько основных преимуществ *PostgreSQL*:

- открытый исходный код. *PostgreSQL* распространяется на условиях лицензии *MIT*, что позволяет пользователям свободно использовать и модифицировать его;
- масштабируемость. *PostgreSQL* может обрабатывать большие объемы данных без каких-либо проблем в производительности или безопасности;
- надежность. Она имеет многолетний трек-рекорд надежности и проводится полное тестирование на устойчивость к сбоям и отказам;
- *ACID*-совместимость. *PostgreSQL* использует транзакционную модель, что обеспечивает надежность данных и поддерживает операции *ACID* (атомарность, согласованность, изолированность и долговечность);
- расширяемость. Пользователи могут легко добавлять собственные функции и процедуры на *SQL*, а также необходимые модули по мере необходимости;
- гибкая архитектура. *PostgreSQL* поддерживает различные типы данных, включая *JSON*, *XML*, графы и текстовые данные, что делает его более универсальным и применимым для многих сценариев.

1.1.5 Выбор среды разработки

Для разработки на языке программирования *Java* можно использовать любой текстовый редактор, однако есть возможность использовать множество разных сред разработки, которые включают в себя средства для отладки кода, сборщики проектов, плагины, системы контроля версий. Самый популярные среды разработки: *Eclipse*, *IntelliJ IDEA*, *NetBeans*, *JDeveloper*, *BlueJ*.

Eclipse - это лучшая бесплатная *IDE* для разработчиков *JAVA* и в основном написанная на *Java*. Она используется в различных платформах для создания мобильных приложений, настольных программ, веб-сайтов. Многие разработчики предпочитают эту *IDE* для программирования на *Java*.

				ПГУ1.090301.2.001 ПЗ	Лист
					16
Изм.	Лист	№ докум	Подпись Дата		

IDEA - еще одна лучшая бесплатная среда разработки *Java IDE*. Эта *IDE* в основном используется для разработки приложений для *Android*, программирования на *Java*, *Java SE*, *Groovy* и *Scala*. Это легкая *IDE*.

NetBeans - еще одна популярная *Java IDE*. *NetBeans* - это официальная *IDE* для *Java 8*. Она позволяет разрабатывать приложения из набора программных компонентов, называемых модулями. Работает под управлением *Windows*, *macOS*, *Linux* и *Solaris*. Немного похоже на другие *IDE*.

JDeveloper - это бесплатная *Java IDE*, разработанная компанией *Oracle*. Она предлагает интегрированную структуру разработки с несколькими визуальными инструментами разработки. *JDeveloper* охватывает весь жизненный цикл разработки, от кодирования, проектирования, отладки, оптимизации и развертывания.

Oracle JDeveloper может интегрироваться с приложением *Oracle (Oracle ADF)* для дальнейшего упрощения разработки приложений. Помимо *Java*, *JDeveloper* можно использовать для разработки приложений на языках *GTML*, *JavaScript*, *PHP*, *SQL* и *XML*.

После анализа различных сред разработки выбор был сделан в пользу *IDEA*.

1.2 Проектирование и разработка

Разрабатываемое приложение состоит из серверной и клиентской частей. Сервер получает данные от стороннего *API*, а также работает с базой данных. Клиентской частью, в разрабатываемом приложении, будет выступать *Telegram*, так как в нем есть возможность достаточно просто реализовать интуитивно понятный и простой интерфейс для пользователя.

Пользователь будет обладать возможностями поиска определенной кинокартины по конкретным критериям, получения информации о фильме, а также просмотра или сохранения выбранного кинофильма.

				ПГУ1.090301.2.001 ПЗ	Лист
					17
Изм.	Лист	№ докум	Подпись Дата		

Перед началом работы сервера производится инициализация всех модулей и компонентов, внедрение зависимостей и подключение к базе данных.

1.2.1 Реализация сбора данных

Для сбора данных о кинофильмах в проекте используется открытое *Kinopoisk API*. Выбранное *API* позволяет получать данные в формате *JSON* с помощью *HTTP* запросов. Данное *API* предоставляет широкий спектр данных о кинофильме, а также позволяет выполнить гибкий поиск по выбранным критериям.

Для взаимодействия с сервисом *Kinopoisk API* используется библиотека *OpenFeign* из фреймворка *Spring Cloud*.

Spring Cloud OpenFeign – представляет из себя реализацию *feign* клиента для работы с модулем *Spring Boot*. *Feign* – это декларативный *HTTP* клиент, для более простого написания интеграций с различными *API*. Основными причинами для выбора данной технологии это:

- автоматическое управление и реализация *HTTP* клиента;
- автоматическая настройка обработки ошибок в ответе на запрос;
- легкость и простота использования.

В проекте был создан интерфейс *FeignKinopoiskApi* для взаимодействия с *Kinopoisk API* с методами *getByGenre*, *getTop*, *getByRating*, *getByName*, реализованный при помощи *OpenFeign*. Данные методы предназначены для получения данных по определенным параметрам: по жанру, по рейтингу, по имени и получение топа лучших кинофильмов.

На рисунке 1.4 показана *UML* диаграмма реализованного *FeignClient*

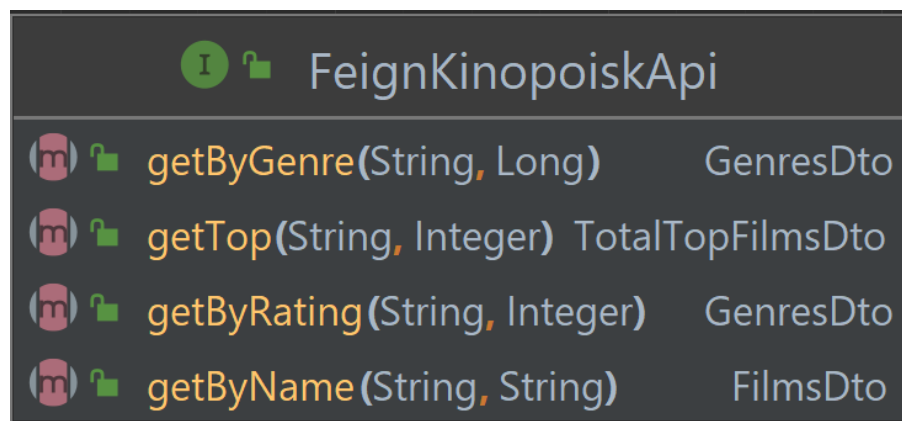


Рисунок 1.4 – UML диаграмма интерфейса *FeignKinopoiskApi*

1.2.2 Реализация серверной части

Разработка серверной части чат-бота включает в себя несколько этапов:

- выбор платформы для разработки. Это может быть любой язык программирования, который поддерживает работу с API и базами данных;
- создание API. это интерфейс, который позволяет чат-боту общаться с другими сервисами и получать от них информацию;
- написание логики работы чат-бота. Это может быть набор правил, которые определяют, как бот будет отвечать на определенные запросы, или алгоритмы машинного обучения, которые позволяют боту самостоятельно принимать решения;
- интеграция с базой данных. Это позволяет хранить информацию о пользователях и их запросах, а также использовать эту информацию для улучшения работы чат-бота;
- тестирование и отладка. Это важный этап, который позволяет проверить работу чат-бота на различных сценариях и исправить возможные ошибки;
- развертывание на сервере. Это позволяет запустить чат-бота на удаленном сервере и обеспечить его доступность для пользователей.

Серверная часть была разработана с использованием фреймворка *Spring*. В качестве сборщика проекта был выбран *Gradle*, с его помощью в проект добавлялись необходимые зависимости.

Для использования самого бота нужно создать его в *Telegram*. Следуя инструкциям, в результате получаем ссылку на созданного бота и токен для доступа к нему, как показано на рисунке 1.5.

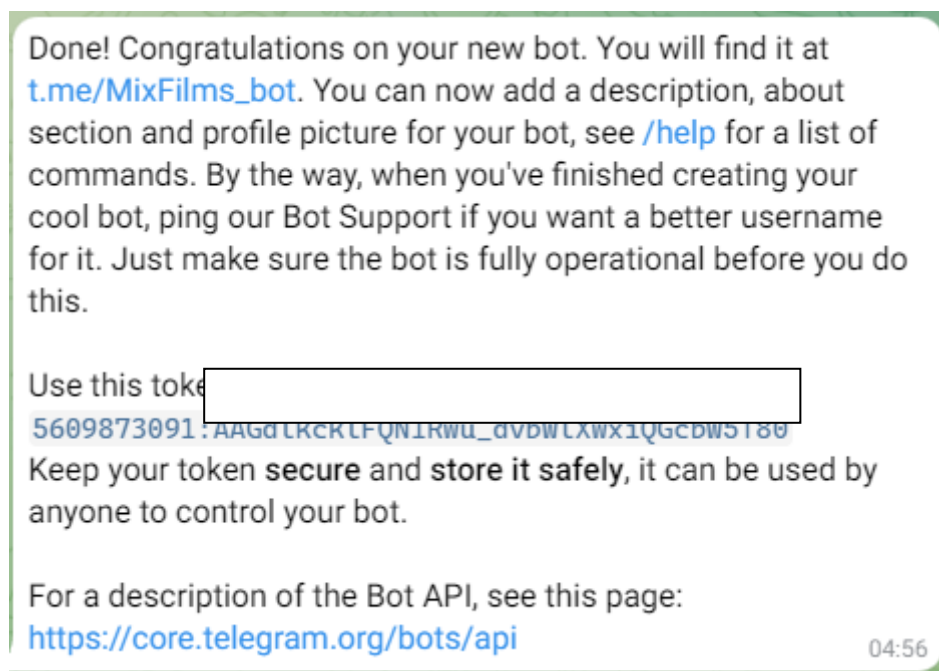


Рисунок 1.5 – Создание бота

Также в файле *application.yaml* были прописаны основные настройки проекта, такие как:

- порт, на котором будет работать приложение;
- поля для организации связи с ботом(токен, имя);
- токен для *API*;
- ссылка на базу данных;
- конфигурация *JPA*.

На рисунке 1.6 видно конечный вариант конфигурационного файла

```

bot:
  name: MixFilms_bot
  token: 5609873091: [REDACTED]
API:
  token: ce [REDACTED]
server:
  port: 8080
feign:
  httpclient:
    enabled: true
  okhttp:
    enabled: true
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/films
    username: postgres
    password: [REDACTED]
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: create-drop
      open-in-view: false
    properties:
      hibernate:
        enable_lazy_load_no_trans: true

```

Рисунок 1.6 – Конфигурация проекта

Для запуска приложения, использован класс *SpringApplication* с методом *run*, который вызывается из метода *main* разрабатываемого приложения.

После был реализован класс *Bot.java*, который содержит основную логику работы бота. Внутри реализована обработка команд, реакция нажатия на кнопки, получение данных *Feign Client*. Слои для работы с *Kinopoisk API* и для работы с базой данных внедрены с помощью контейнера *Spring IoC*, который позволяет автоматически определять зависимости для компонента и внедрять их. На рисунках 1.7 – 1.8 представлены *UML* диаграммы файлов проекта.

				ПГУ1.090301.2.001 ПЗ	Лист
					21
Изм.	Лист	№ докум	Подпись Дата		

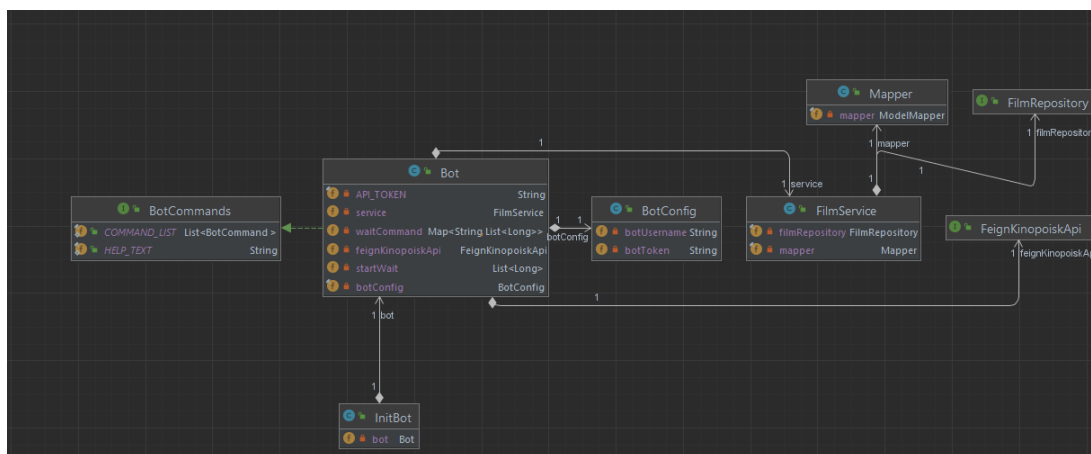


Рисунок 1.7 – UML диаграмма основных классов проекта

Помимо всего были написаны файлы *DTO*. *DTO*(data transfer object) - паттерн, используемый для передачи данных между подсистемами приложения.

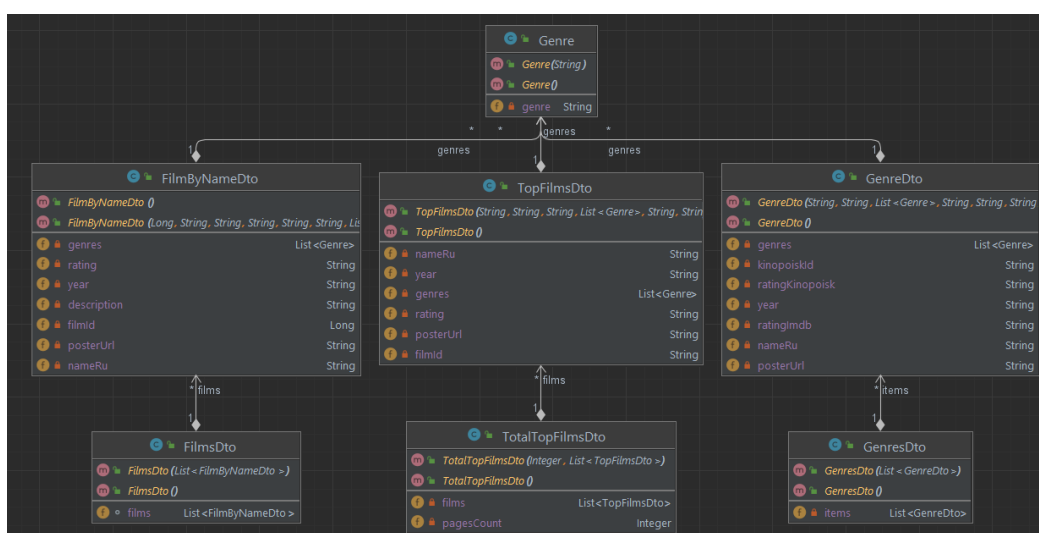


Рисунок 1.8 – UML диаграмма файлов *DTO*

Архитектура приложения изображена на рисунке 1.9.

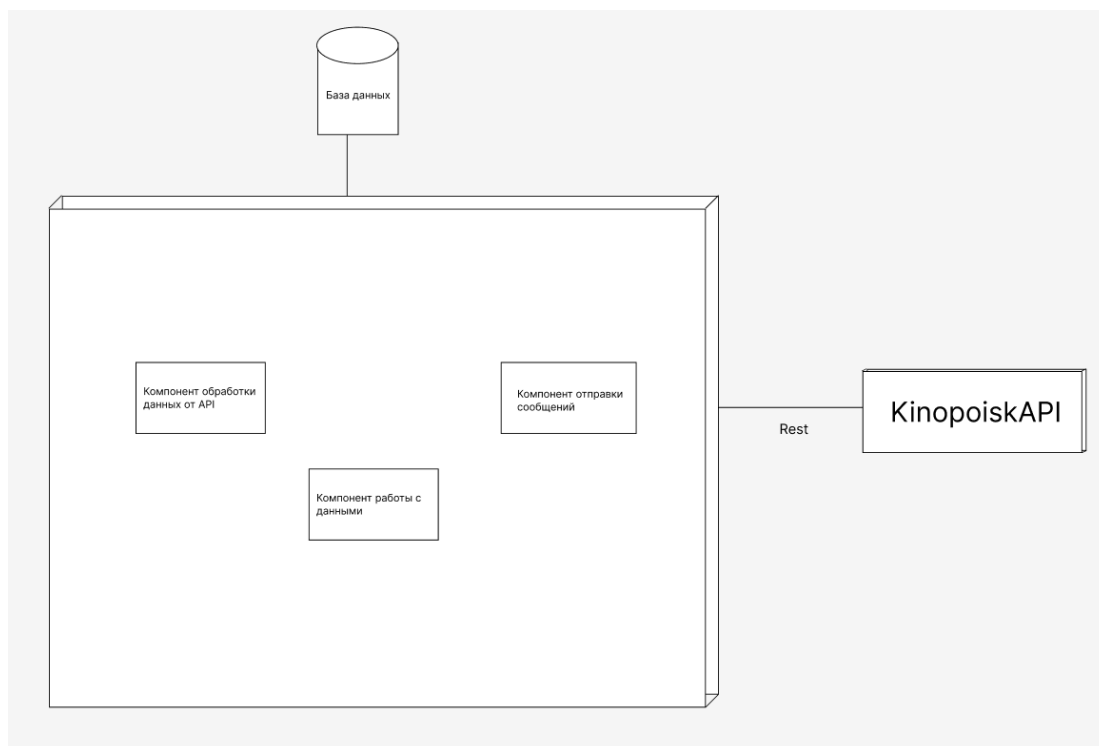


Рисунок 1.9 – Схема архитектуры приложения

1.2.3 Разработка БД

Spring Data JPA позволяет создавать, получать, изменять данные без прямых *SQL* запросов. В разрабатываемом приложении база данных используется для предоставления пользователю непосредственно кинофильма. Для этого была создана таблица *Film*, с помощью аннотации *Entity*, написанной внутри класса модели *Film*. Название полей таблицы перечислены в таблице 1.1., также на рисунке 1.10 представлены поля класса модели *Film*.



Рисунок 1.10 – UML диаграмма для работы с таблицей

Таблица 1.1 – Атрибуты сущности «Film»

№	Наименование атрибута	Тип	Обязательный
1	Name	Строка	да
2	Id	Целое число	да
3	Reference	Строка	да
4	FilmId	Целое число	да

Все взаимодействие с базой данных будет производится через паттерн репозиторий. *Spring JPA* позволяет работать с базой данных очень абстрактно, создавая сигнатуры методов внутри интерфейса репозитория, без конкретной реализации.



Рисунок 1.11 – UML диаграмма репозитория

1.2.4 Разработка клиентской части

В качестве клиентского приложения в проекте используется *Telegram*, так как большая часть реализована, остается только настроить приложение в соответствии с требованиями.

Для взаимодействия с пользователем бот использует кнопки и команды. В классах *MainButtons*, *GenreButtons*, *BackButtons* реализованы различные наборы клавиатур для удобного использования бота. В интерфейсе *BotCommands* находятся основные команды для взаимодействия с ботом. Диаграмма классов изображена на рисунке 1.12.

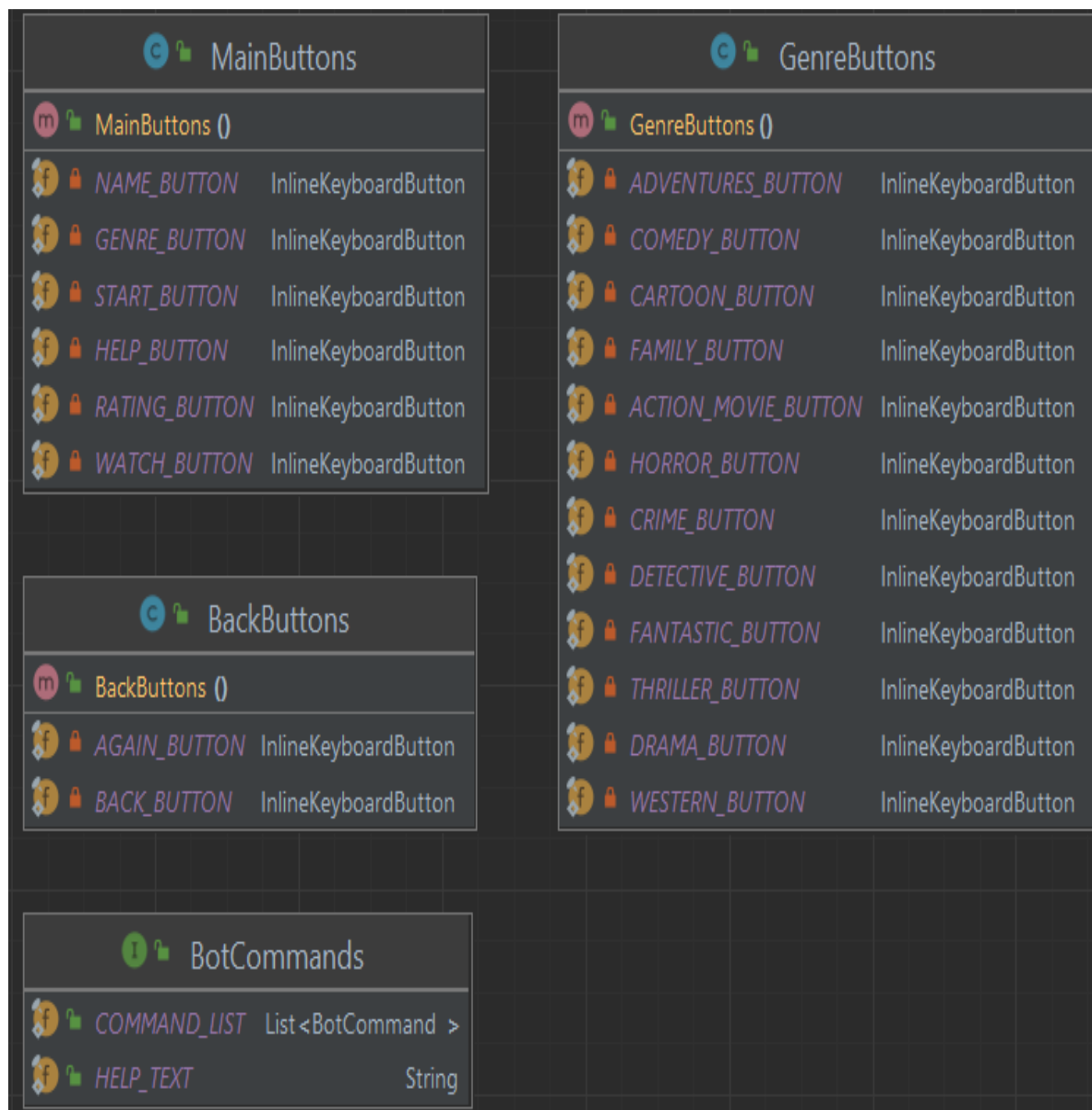


Рисунок 1.12 – UML диаграмма классов кнопок

Диаграмма всех классов проекта, с их полями, представлена на рисунке 1.13.

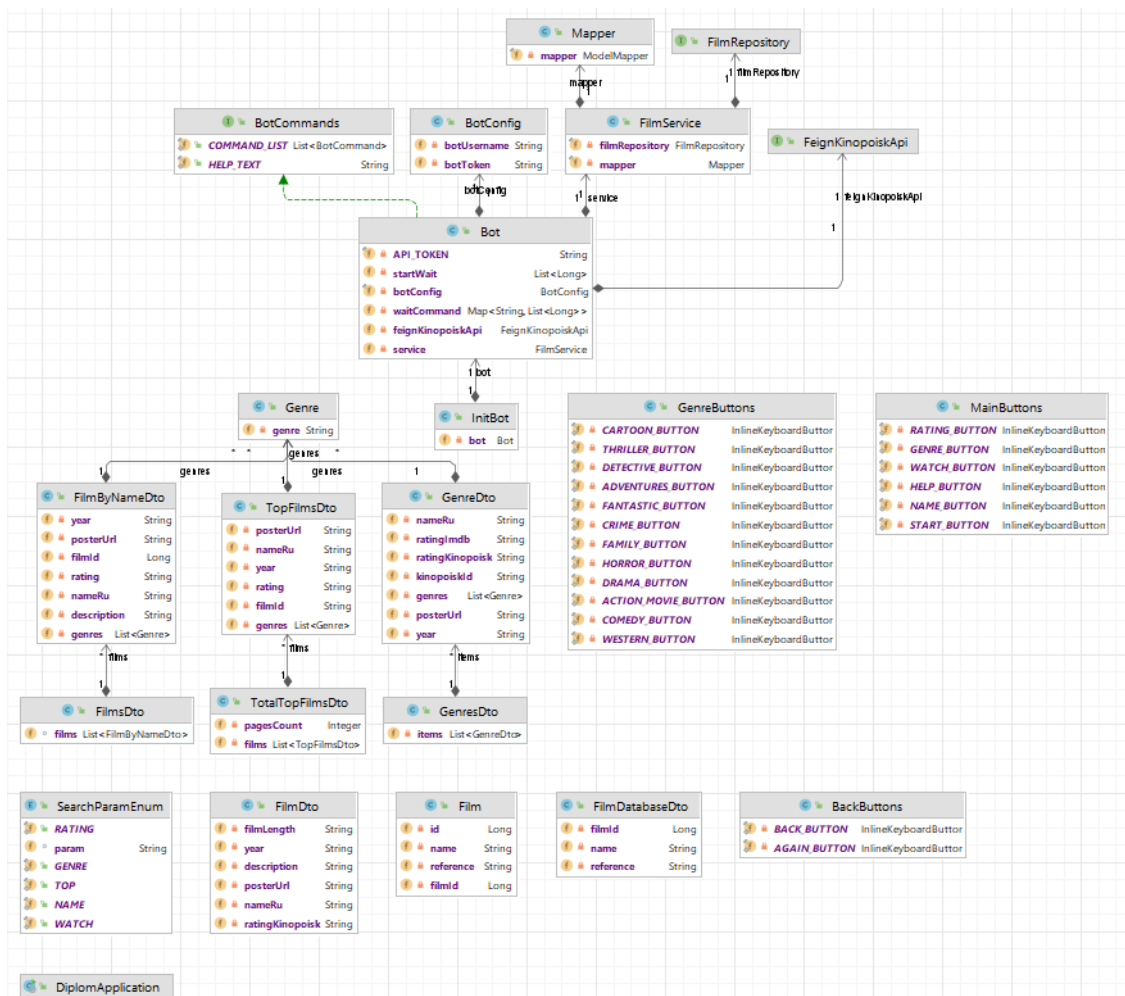


Рисунок 1.13 – UML диаграмма классов проекта

1.3 Анализ результатов

Было разработан бот для подбора кинофильмов на языке *Java* с применением *Spring Framework*.

Пользователю предоставляется следующий набор возможностей (рисунок 1.14):

- поиск кинофильмов по названию;
- поиск по рейтингу;
- поиск по жанру;
- получение информации о лучших фильмах;
- получение кинофильма для просмотра или скачивания.

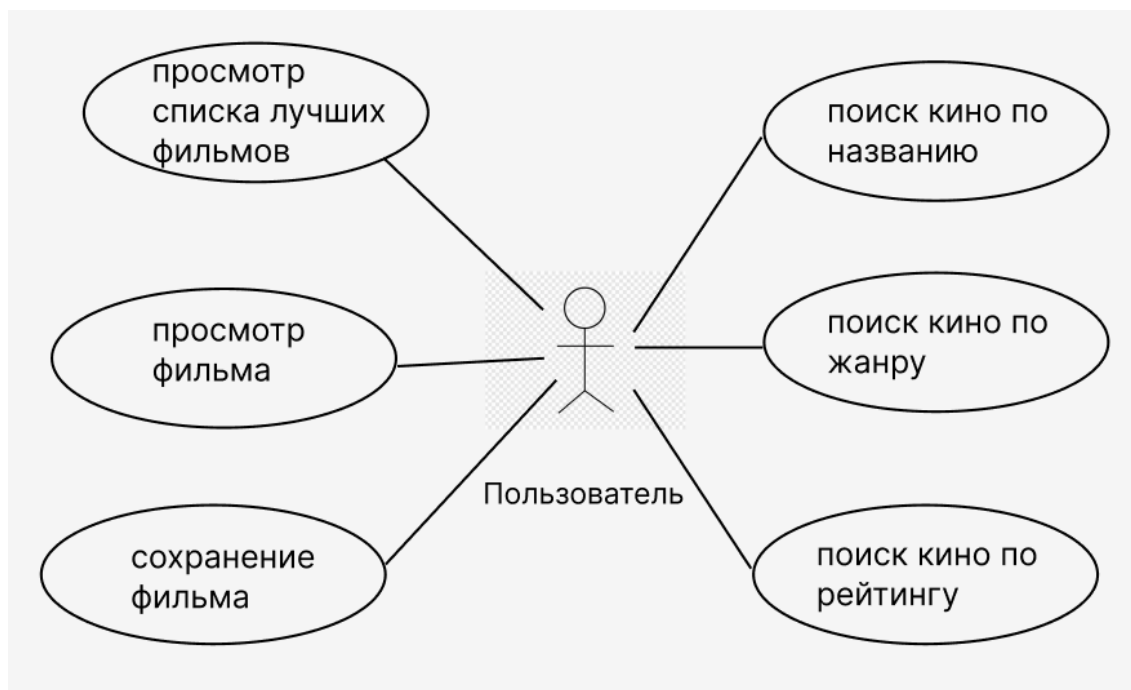


Рисунок 1.14 - UML-диаграмма вариантов использования приложения

1.3.1 Тестирование приложения

Для интеграционного тестирования использовались библиотеки *Junit* и *Mockito*.

Интеграционное тестирование в *Spring Boot* используется для проверки взаимодействия между различными компонентами приложения, такими как базы данных, веб-сервисы и так далее. Это позволяет убедиться, что все компоненты работают корректно вместе и обеспечивают правильную функциональность приложения.

Для проведения интеграционного тестирования в *Spring Boot* можно использовать различные инструменты, такие как *JUnit*, *TestNG* или *Mockito*. Тесты могут быть написаны как вручную, так и автоматически с помощью инструментов, таких как *Spring Test* или *Spring Boot Test*.

В процессе интеграционного тестирования в *Spring Boot* можно проверять различные аспекты приложения, такие как:

- проверка работы базы данных и соответствующих запросов;
- проверка работы веб-сервисов и *REST API*;

- проверка работы механизмов безопасности и аутентификации;
- проверка работы механизмов кэширования и оптимизации.

Интеграционное тестирование является важной частью процесса разработки веб-приложений на *Spring Boot* и позволяет обеспечить высокое качество и надежность приложения.

Протестировали слой репозитория, для этого внедрили в класс с тестами интерфейс *FilmRepository* и проверили правильность работы данного слоя. Пример интеграционного теста и результат тестирования приведен на рисунках 1.15 – 1.16.

```
@Test
@Transactional
void findById() {
    Film persist = entityManager.persist(new Inspection());
    assertEquals(filmRepository.findById(435L).orElse(persist), persist);
}
```

Рисунок 1.15 – Интеграционный тест слоя репозитория

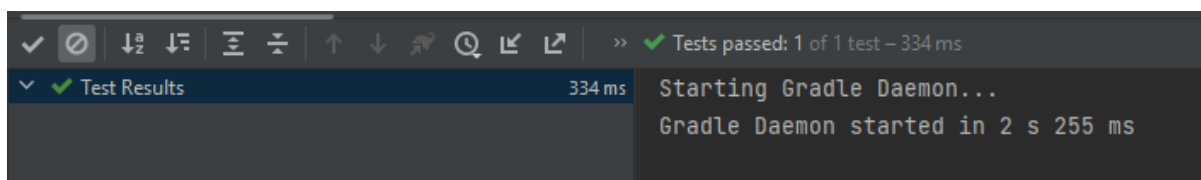


Рисунок 1.16 – Результат тестирования

Также было проведено ручное тестирование приложения. Для начала работы следует отправить боту команду */start* (рисунок 1.17). Команда обработается на сервере, после чего бот поприветствует пользователя и предложит ему выбрать доступные действия.

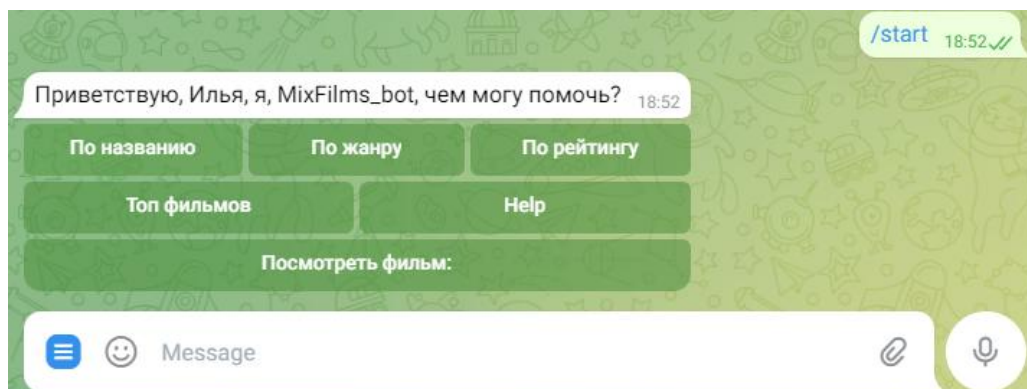


Рисунок 1.17 – Начало работы бота

Далее можно попробовать получить информацию о фильме по названию. Для этого нужно нажать соответствующую кнопку на представленной ботом клавиатуре. Бот попросит ввести название искомого фильма. Пример поиска по имени показан на рисунке 1.18.

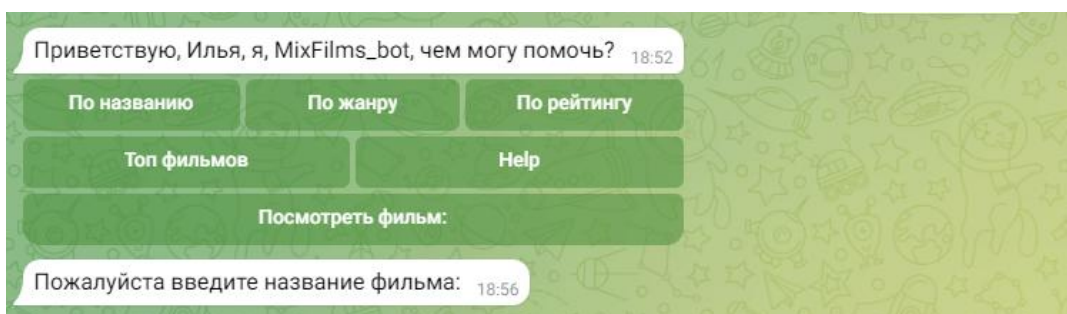


Рисунок 1.18 – Тестирование поиска по имени

После ввода названия, сервер отправляет соответствующий запрос к *API* с названием фильма которое ввел пользователь через *FeignClient*. Далее из ответа, полученного в формате *JSON*, бот отбирает основную информацию и присылает ее пользователю. Результат поиска по названию изображен на рисунке 1.19, также на рисунке 1.20 показаны кнопки для дальнейшей навигации.

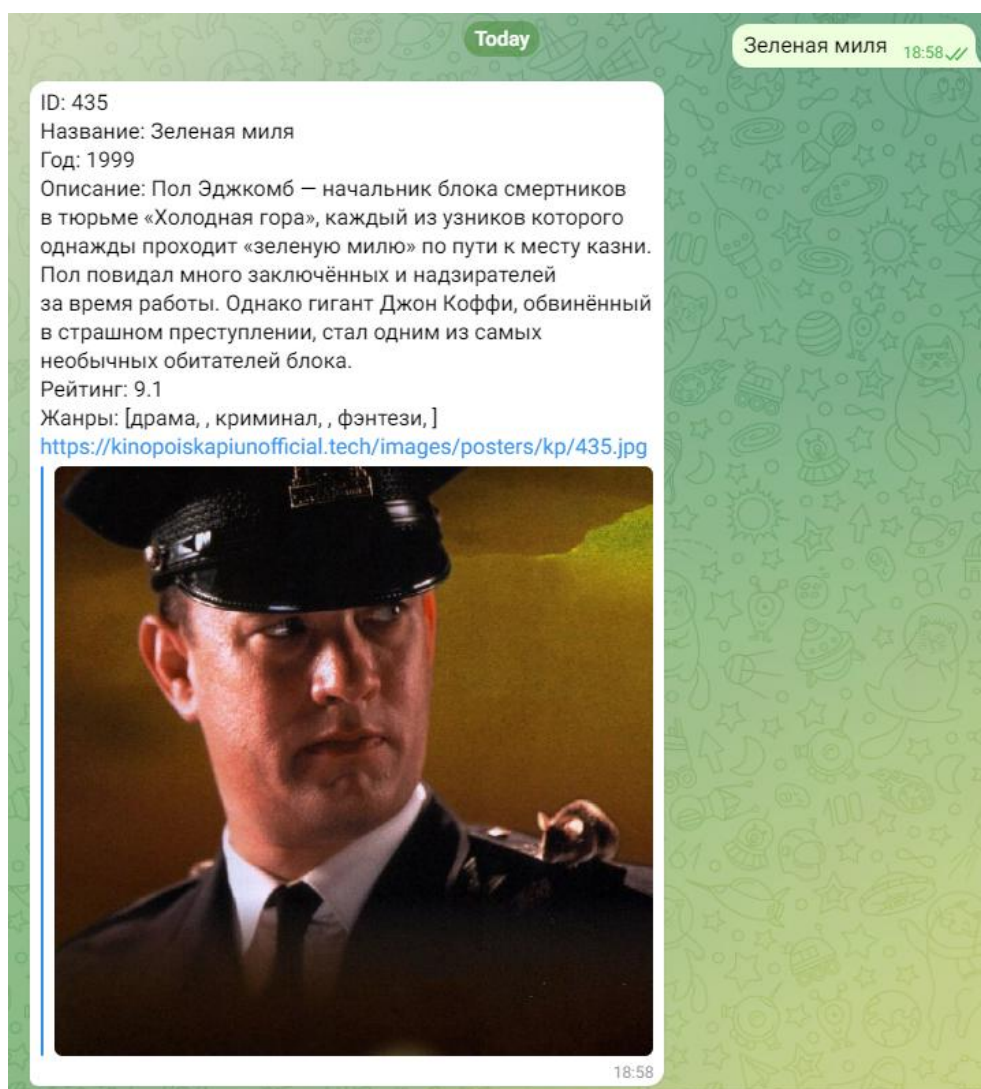


Рисунок 1.19 – Результат поиска по названию

В конце сообщения присутствуют кнопки для дальнейшей навигации

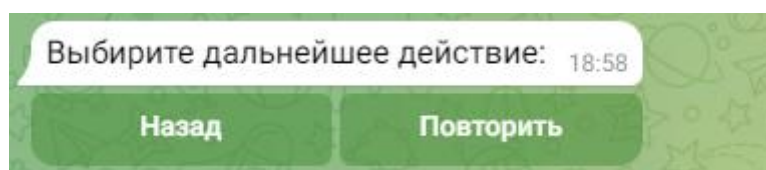


Рисунок 1.20 – Кнопки навигации

Теперь есть возможность получить фильм для просмотра или скачивания. Для этого нужно вернуться к начальной клавиатуре бота, нажав на кнопку «Назад». И далее следует нажать на кнопку «Посмотреть фильм» и ввести ID желаемого кинофильма. При правильном выполнении выше

описанной последовательности действий пользователь получает ссылку по которой он может посмотреть кино или скачать его (рисунок 1.21).

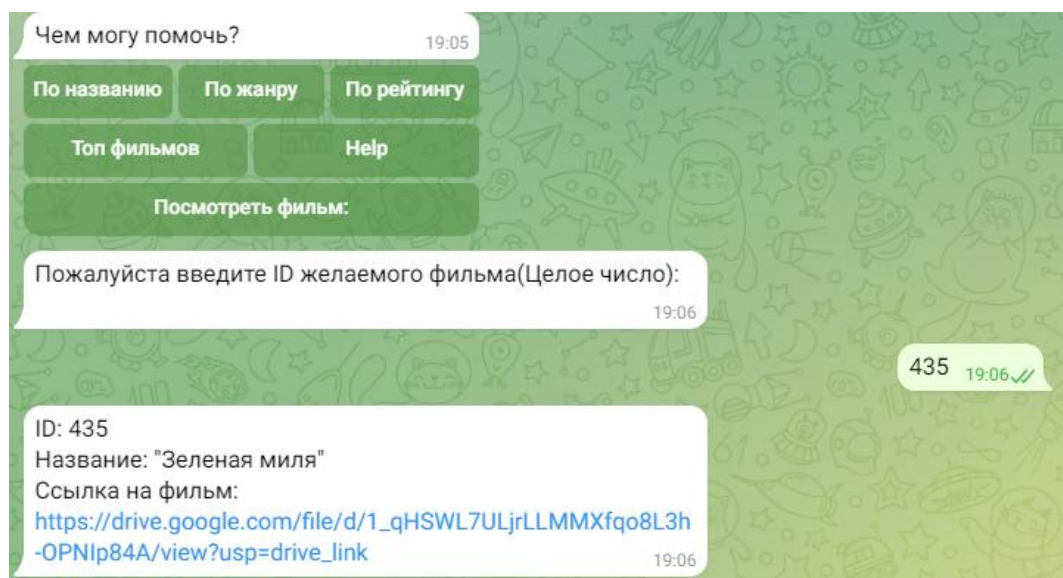


Рисунок 1.21 – Получение ссылки на фильм

При переходе по ссылке пользователь видит плеер для воспроизведения фильма и кнопку для скачивания, который изображен на рисунке 1.22.

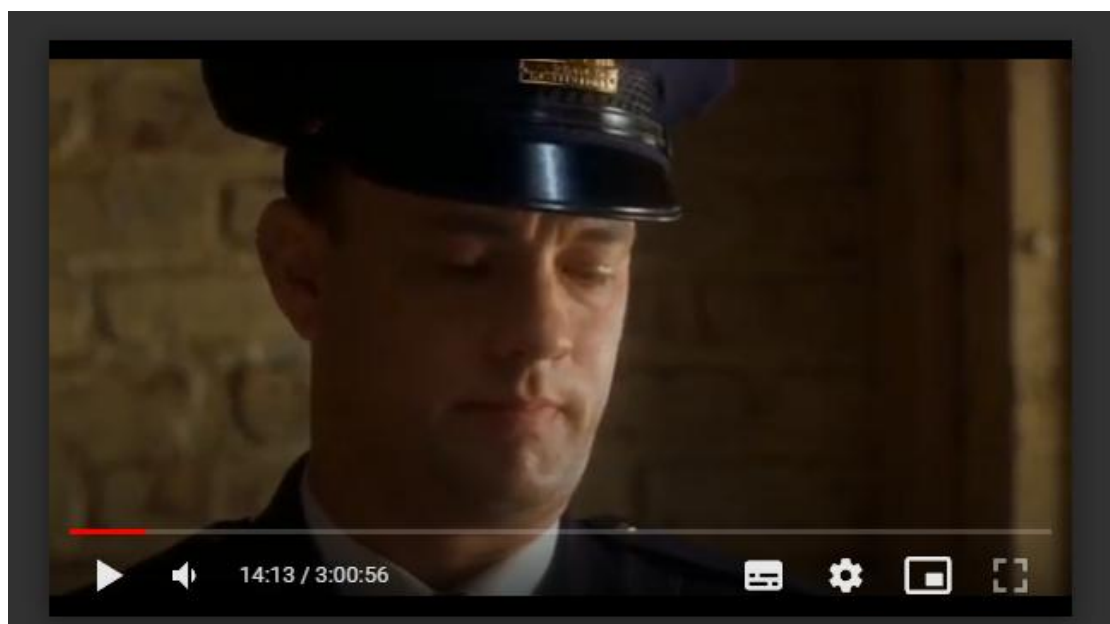


Рисунок 1.22 – Просмотр фильма

2 Охрана труда

В данном разделе рассматриваются требования по обеспечению безопасности жизнедеятельности и охраны труда пользователей ПЭВМ, участвующих в разработке и эксплуатации программного обеспечения, а также способы минимизации пагубных факторов на здоровье. Охрана труда - это область, которая занимается созданием безопасных и здоровых условий труда, а также предотвращением производственных несчастных случаев и профессиональных заболеваний. Ниже перечислены некоторые важные аспекты охраны труда:

- соблюдение правил и инструкций по безопасности на рабочем месте;
- обеспечение необходимой защитной экипировки, если таковая требуется для выполнения работы;
- прохождение медицинского осмотра перед началом работы, а также периодические медицинские осмотры для выявления заболеваний, связанных с трудом;
- обучение сотрудников навыкам безопасного поведения на рабочем месте и проведение регулярных тренингов;
- обеспечение правильной организации и эргономики рабочего места;
- проведение регулярных проверок оборудования на безопасность и проведение ремонта в случае необходимости;
- систематический анализ и контроль производственных процессов на предмет выявления возможных рисков и их устранения;
- своевременное сообщение администрации об осложнениях состояния здоровья, связанных с работой.

Эти меры помогут обеспечить безопасное и здоровое рабочее место, а также повышение производительности и качества производства.

				ПГУ1.090301.2.001 ПЗ	Лист
					33
Изм.	Лист	№ докум	Подпись Дата		

2.1 Характеристики потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Потенциальные опасности и вредности, связанные с работой с ЭВМ, могут включать в себя следующее:

- утомление глаз и ухудшение зрения. Длительная работа перед монитором может вызвать утомление глаз, сухость и раздражение. Это может привести к ухудшению зрения и развитию проблем со здоровьем глаз;

- болезни опорно-двигательного аппарата. Длительная стационарная работа может привести к нарушению осанки, развитию болей в спине, шее, руках и ногах. Отсутствие физической активности может усугубить эти проблемы;

- инфекционными заболеваниями. Клавиатуры и мыши могут стать носителями бактерий и вирусов, которые могут вызвать разные инфекционные заболевания;

- радиационным излучением. Мониторы могут выделять небольшое количество ионизирующего излучения, что может оказывать влияние на здоровье работающих. Однако, современные мониторы существенно снизили этот риск;

- акустическими рисками. Частая работа с наушниками может привести к появлению проблем со слухом;

- психическими проблемами. Длительная работа перед компьютером может привести к раздражительности, бессоннице, уменьшению концентрации и депрессии;

- социальными проблемами. Сидячий образ жизни, связанный с работой на компьютере, может привести к социальной изоляции и ограничению общения с окружающими людьми.

В целом, работа на компьютере может повлечь за собой большое количество проблем со здоровьем, поэтому важно следить за своим

				ПГУ1.090301.2.001 ПЗ	Лист
					34
Изм.	Лист	№ докум	Подпись Дата		

здоровьем и соблюдать профилактические меры, такие как регулярные перерывы, физические упражнения и правильное положение тела при работе.

2.2 Классификация и категорирование помещения, в котором производится разработка

Для классификации помещения, в котором производится разработка, нужно учитывать ряд параметров, таких как:

- назначение помещения. Офис, лаборатория, цех, мастерская и так далее;
- размеры помещения. Малое, среднее, большое;
- характер работы. Ручная работа, использование техники, высокоточная работа и так далее;
- число сотрудников, которые будут работать в помещении;
- требования к безопасности. Наличие специального оборудования, вентиляции, освещения и прочее.

На основе этих параметров помещения могут быть классифицированы по категориям:

- офисные помещения. Используются для разработки проектов, проведения совещаний, оформления документов;
- лаборатории. Используются для исследований, анализа и тестирования;
- цеха. Используются для производства и сборки изделий или их компонентов;
- мастерские. Используются для ручной работы, например, для создания прототипов или макетов.

Кроме того, помещения могут быть классифицированы по виду деятельности или отрасли:

- IT-разработка. Помещение, где разрабатывается и тестируется программное обеспечение;

				ПГУ1.090301.2.001 ПЗ	Лист
					35
Изм.	Лист	№ докум	Подпись Дата		

– производственные помещения. Используются для производства электронных, механических или химических компонентов или изделий;

– дизайнерские студии. Используются для разработки дизайна и создания макетов.

Категорирование помещений в зависимости от размера и характера работы может быть следующим:

– малые помещения. Используются для небольшой команды (до 5 человек) и ручной работы;

– средние помещения. Используются для команды от 5 до 20 человек и работы с использованием техники;

– большие помещения. Используются для команды более 20 человек и/или высокоточной работы.

2.2.1 Классификация по взрывопожарной и пожарной опасности

По взрывопожарной и пожарной опасности помещения подразделяются на категории А, Б, В1-В4, Г и Д, а здания - на категории А, Б, В, Г и Д.

Горючие вещества и материалы их пожароопасные свойства и количество определяют категорию помещений и зданий, а также на основе планировочных решений помещений и характеристик проводимых в них процессов. Ниже приведены категории помещения и характеристик веществ и материалов, находящихся в помещении:

– «А» повышенная взрывопожароопасность. Горючие газы, легковоспламеняющиеся жидкости с температурой вспышки не более 28 °С в таком количестве, что могут образовывать взрывоопасные парогазовоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 кПа, и вещества и материалы, которые способны взрываться и гореть при взаимодействии с водой, кислородом воздуха или друг с другом, в количестве, когда расчетное избыточное давление взрыва в помещении превышает 5 кПа;

				ПГУ1.090301.2.001 ПЗ	Лист
					36
Изм.	Лист	№ докум	Подпись Дата		

– «Б» взрывопожароопасность. Горючие пыли и волокна, легковоспламеняющиеся жидкости с температурой воздуха более 28 °С, горючие жидкости, способные образовать взрывоопасные пылевоздушные или паровоздушные смеси, при воспламенении которых развивается расчетное избыточное давление взрыва в помещении, превышающее 5 кПа;

– «В1-В4» пожароопасность. Горючие и трудногорючие жидкости, вещества и материалы, способные при взаимодействии с водой, кислородом воздуха или друг с другом только гореть, при условии, что помещения, в которых они находятся, не относятся к категории А или Б;

– «Г» умеренная пожароопасность. Негорючие вещества и материалы в горячем, раскаленном или расплавленном состоянии, при обработке которых выделяется лучистое тепло, искры и пламя;

– «Д» пониженная пожароопасность. Негорючие материалы в холодном состоянии, а также негорючие вещества в холодном состоянии.

2.2.2 Классификация по опасности поражения электрическим током

Условия повышенной опасности поражения человека электрическим током:

– высокая влажность, пары или конденсирующаяся влага выделяется в виде мелких капель, и относительная влажность воздуха превышает 75%;

– наличие проводящей пыли, технологическая пыль, оседая на проводах, проникая внутри аппаратов или машин ухудшает условия изоляции и охлаждения, но не вызывает опасности пожара или взрыва;

– отсутствие заземления в электрооборудовании;

– наличие повышенной температуры;

– неправильный монтаж или установка электрооборудования;

– работа в условиях ограниченной видимости, например, внутри труб, шахт или под землей;

				ПГУ1.090301.2.001 ПЗ	Лист
					37
Изм.	Лист	№ докум	Подпись Дата		

– несоблюдение правил безопасности при работе с электрооборудованием.

Таким образом можно выделить три категории опасности помещений: безопасные, с повышенной опасностью и особо опасные.

Для минимизации рисков поражения электрическим током рекомендуется работать в помещениях, соответствующих следующим критериям:

- температура воздуха не ниже 30°C, рекомендованный диапазон 18-20 °C;
- полы должны быть выполнены из диэлектрического материала;
- помещение должно проветриваться;
- влажность воздуха не превышает 60%.

2.3 Санитарно-гигиеническая характеристика производства

Для проведения санитарно-гигиенической характеристики производства необходимо изучить следующие факторы:

– месторасположение производства и окружающая среда. Подвергается ли производство влиянию окружающей среды, такой как шум, вибрации, загрязнение воздуха или воды? Насколько далеко находится от жилых зон? Существует ли возможность дальнейшего улучшения качества окружающей среды;

– оборудование и производственные процессы. Какова эффективность системы очистки и фильтрации воздуха? Как правильно организованы процессы работы с химическими веществами и опасными отходами? Соответствует ли использование оборудования и технологий санитарно-гигиеническим стандартам;

– компетенция персонала. Сотрудники должны быть обучены соблюдению стандартов санитарного контроля. Они должны знать, как правильно использовать и обрабатывать химические вещества и

				ПГУ1.090301.2.001 ПЗ	Лист
					38
Изм.	Лист	№ докум	Подпись Дата		

оборудование, а также должны проходить регулярные медицинские обследования;

– санитарные условия территории производства. Регулярное обслуживание и чистка оборудования, а также регулярное поддержание санитарных условий помещений - это необходимые условия для предотвращения заболеваний и защиты здоровья персонала.

Таким образом к обязательным санитарно-гигиеническим критериями можно отнести: сухость помещения, температура внутри него и количество света. Также перечень основных требований:

- каждое рабочее место должно быть в объёме не менее 15 м³;
- высота помещения не менее 3,2 м;
- ширина дверей не менее 0,8 м;
- ширина коридоров не менее 1,4 м.

Окна должны освещать каждое рабочее место, каждое окно должно открываться и регулярно очищаться от пыли и грязи для обеспечения максимальной освещенности. В условиях недостаточной освещенности естественным светом в помещении должны быть установлены электроосветительные приборы.

Руководители производства обязаны обеспечить абсолютно каждое рабочее помещение чистой питьевой водой. Настоятельно рекомендуется заменять воду не более чем через сутки.

Помещения с уровнем шума превышающем норму должны быть маркированы специальными знаками. Каждый работник должен быть проинструктирован и обязан знать правила безопасности при работе в шумных условиях.

				ПГУ1.090301.2.001 ПЗ	Лист
					39
Изм.	Лист	№ докум	Подпись Дата		

Заключение

Рост числа российских интернет-пользователей, доступность смартфонов и развитие мобильного интернета формируют новые привычки и модели поведения людей. Пользователи интернета хотят иметь интегрированную информацию из различных источников, в этом им помогают чат-боты.

Технология чат-ботов позволяет автоматизировать процесс сбора информации и передать ее конечному пользователю в виде мгновенного сообщения.

В процессе выполнения дипломной работы были изучены: *Telegram API*, актуальные технологии для разработки веб-приложений и актуальные паттерны проектирования веб-приложений. Были приобретены навыки интеграции *Spring Framework* и СУБД *PostgreSQL*. Получены навыки взаимодействия с *API* при помощи *FeignClient*.

Во время выполнения работы были разработаны: класс для извлечения данных из *KinopoiskAPI*, база данных для хранения кинофильмов, интерфейс для взаимодействия пользователя с телеграм-ботом.

Следующим шагом реализации программы является увеличение количества фильмов доступных конечному пользователю, путем расширения облачного хранилища, сотрудничества с другими площадками, посвященными тематике кино.

				ПГУ1.090301.2.001 ПЗ	Лист
					40
Изм.	Лист	№ докум	Подпись Дата		

Список использованных источников

1. Эккель Б. Философия Java. 4-е издание; Питер - 1168 с.
2. Хо К., Шефер К., Козмина Ю. Spring 5 для профессионалов; Диалектика-Вильямс - 1120 с.
3. Документация к фреймворку Spring. [Электронный ресурс] / Режим доступа: <https://docs.spring.io/spring-framework/reference/> свободный. (дата обращения 25.04.2023).

				ПГУ1.090301.2.001 ПЗ	Лист
					41
Изм.	Лист	№ докум	Подпись Дата		

Приложение А

(обязательное)

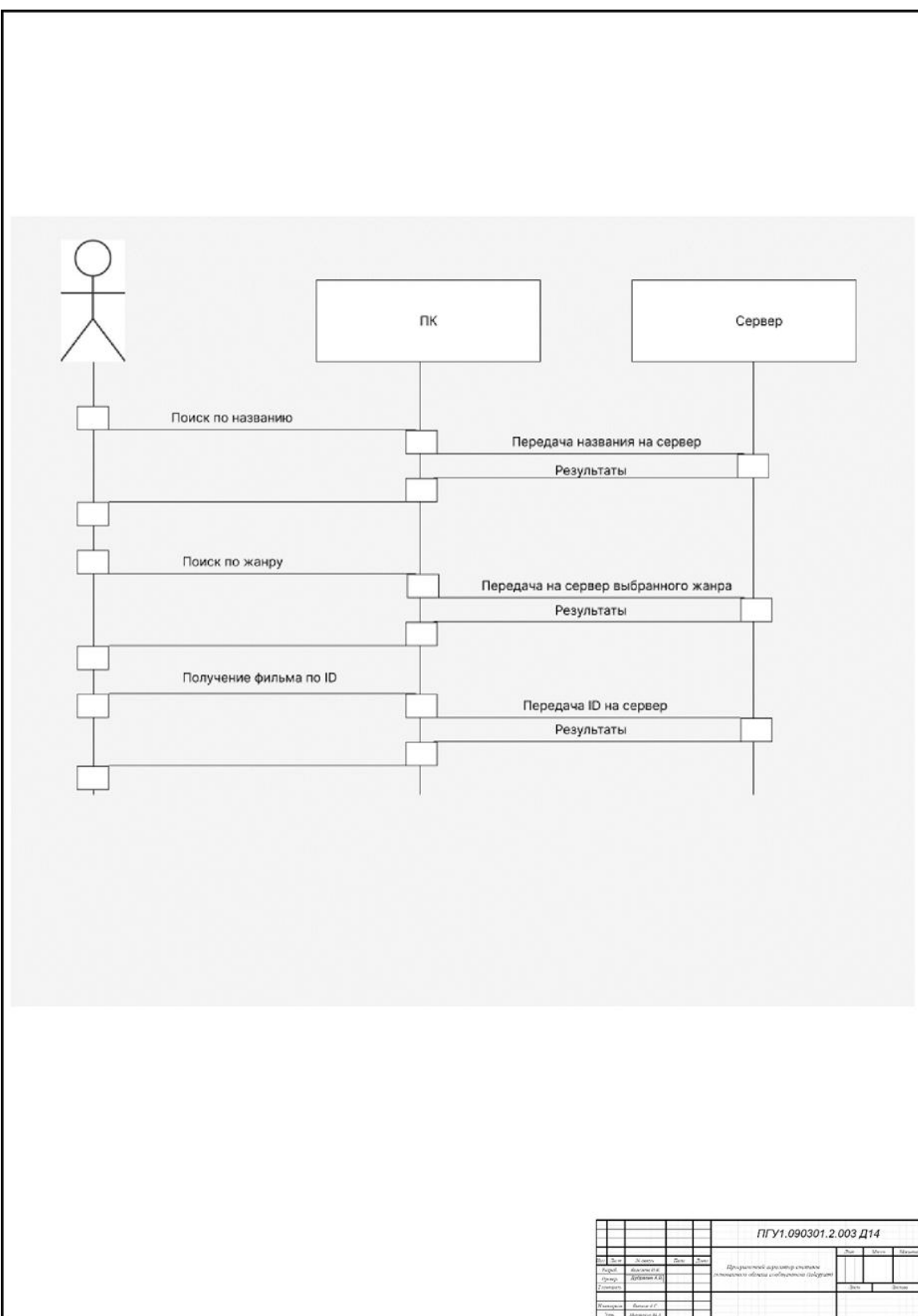
Программный агрегатор системы мгновенного обмена

сообщениями (telegram)

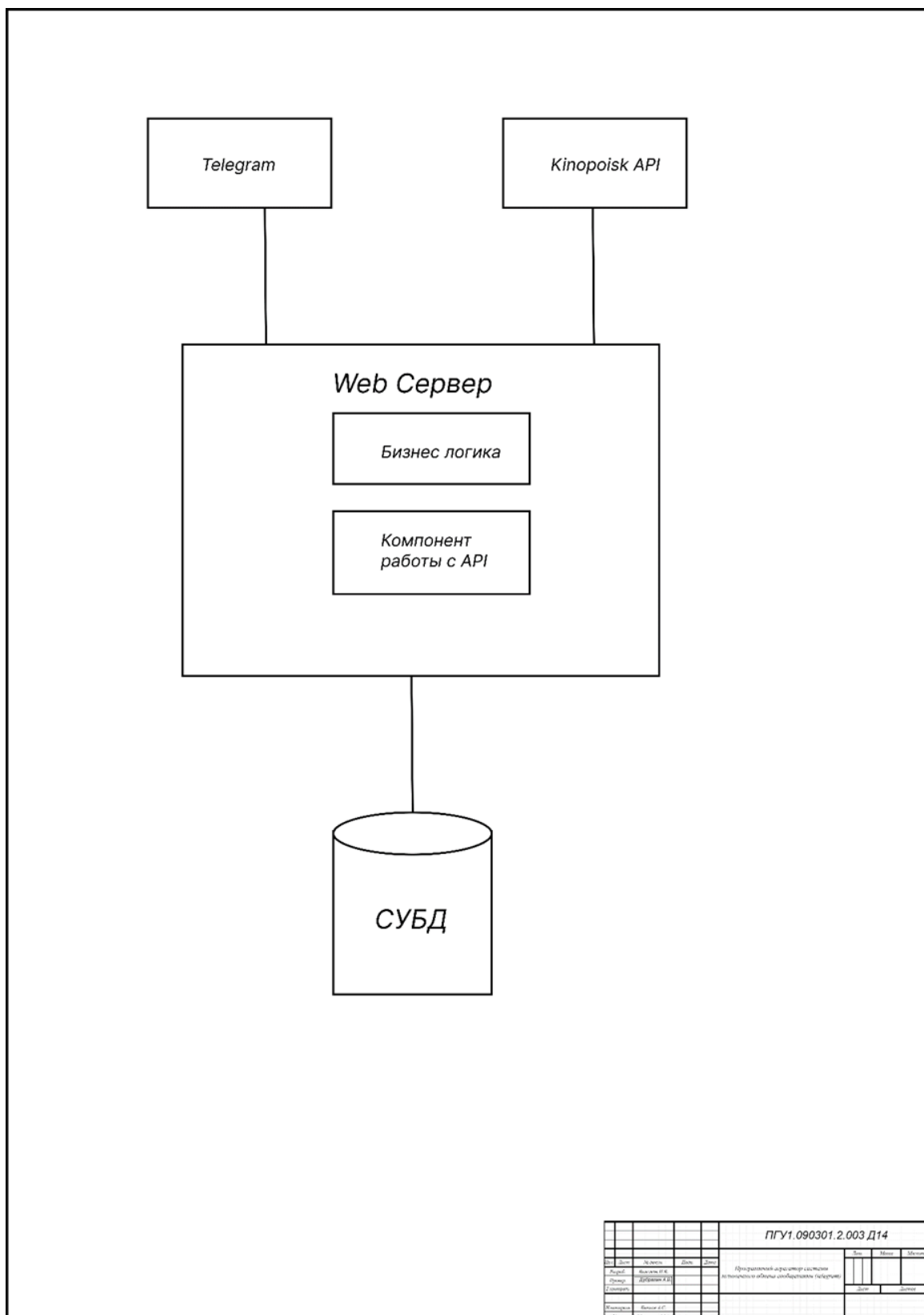
UML - диаграммы

				ПГУ1.090301.2.001 ПЗ	Лист
					42
Изм.	Лист	№ докум	Подпись Дата		

Приложение А.1 – Диаграмма последовательностей



Приложение А.2 – Диаграмма развертывания



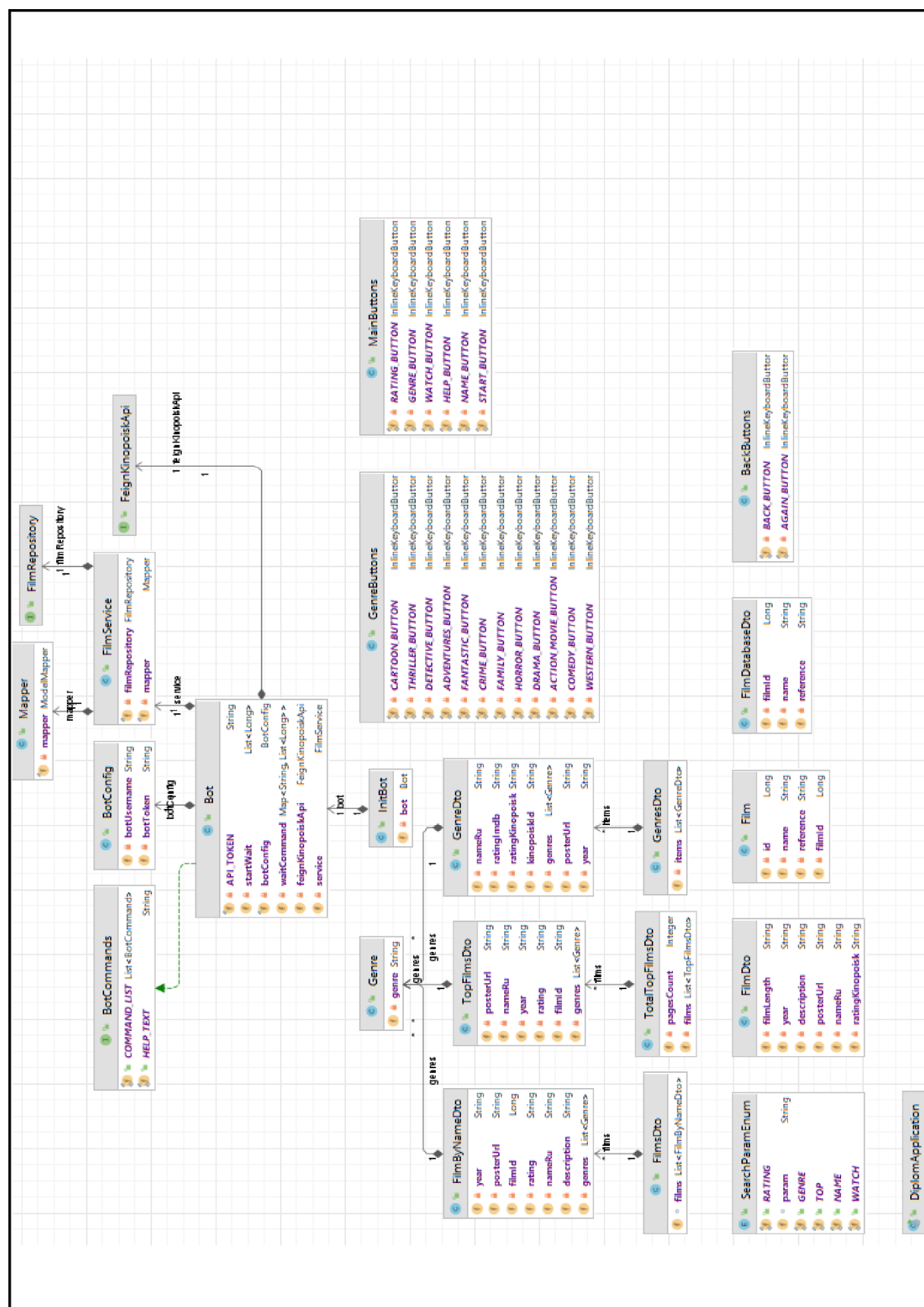
Изм.	Лист	№ докум	Подпись	Дата

ПГУ1.090301.2.001 ПЗ

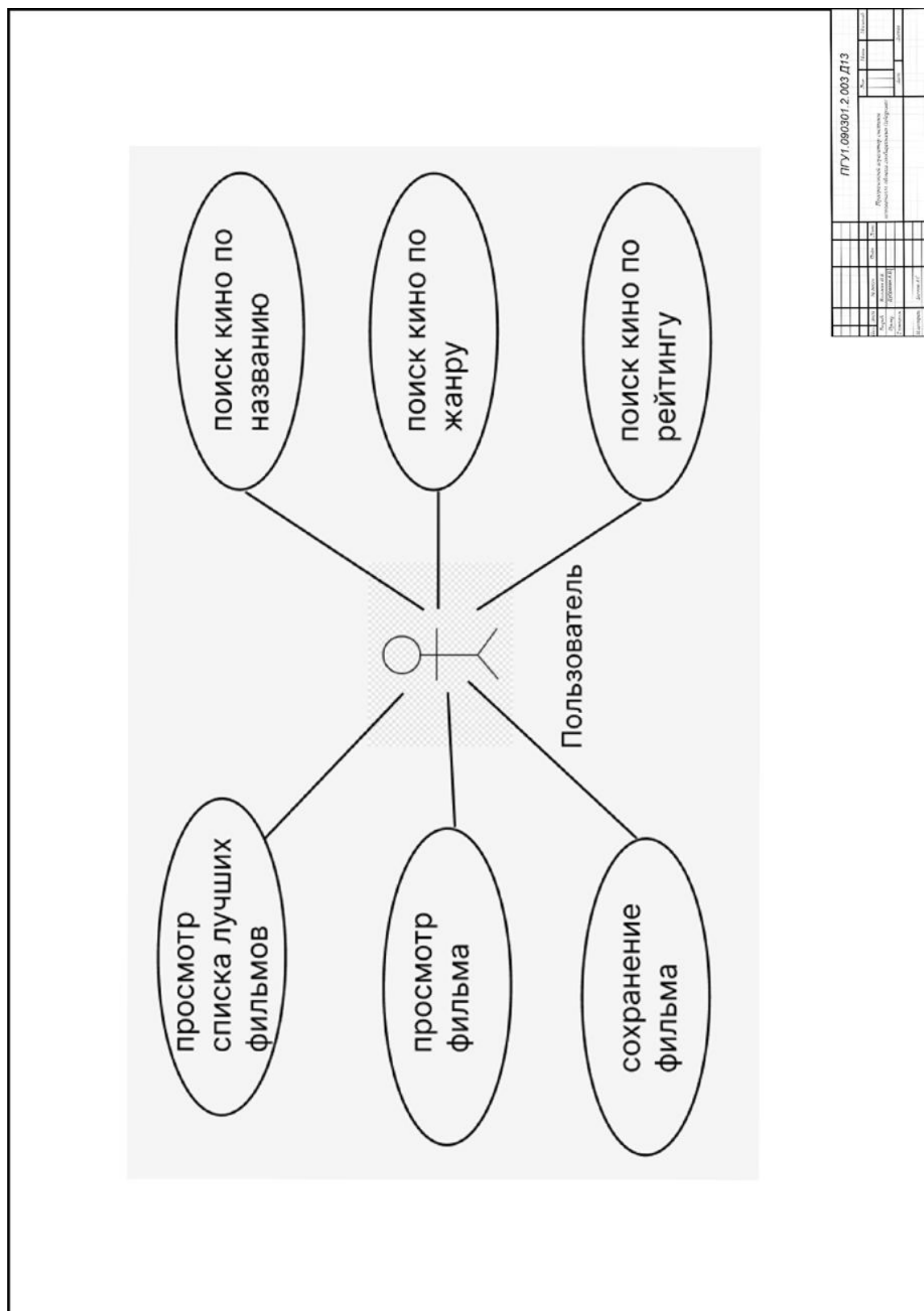
Лист

44

Приложение А.3 – Диаграмма классов



Приложение А.4 – Диаграмма прецедентов



ПГУ1.090301.2.003 Д13									
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
1	Иванов	Иван	Иванов	Иван	Иванов	Иван	Иванов	Иван	Иванов
2	Петров	Петр	Петров	Петр	Петров	Петр	Петров	Петр	Петров
3	Сидоров	Сидор	Сидоров	Сидор	Сидоров	Сидор	Сидоров	Сидор	Сидоров
4	Смирнов	Смирнов	Смирнов	Смирнов	Смирнов	Смирнов	Смирнов	Смирнов	Смирнов
5	Тихонов	Тихонов	Тихонов	Тихонов	Тихонов	Тихонов	Тихонов	Тихонов	Тихонов
6	Васильев	Васильев	Васильев	Васильев	Васильев	Васильев	Васильев	Васильев	Васильев
7	Попов	Попов	Попов	Попов	Попов	Попов	Попов	Попов	Попов
8	Соколов	Соколов	Соколов	Соколов	Соколов	Соколов	Соколов	Соколов	Соколов
9	Кузнецов	Кузнецов	Кузнецов	Кузнецов	Кузнецов	Кузнецов	Кузнецов	Кузнецов	Кузнецов
10	Лебедев	Лебедев	Лебедев	Лебедев	Лебедев	Лебедев	Лебедев	Лебедев	Лебедев

Изм.	Лист	№ докум	Подпись	Дата

ПГУ1.090301.2.001 ПЗ

Лист

46

Приложение Б

(Обязательное)

Программный агрегатор системы мгновенного обмена сообщениями

(telegram)

Листинг программы

				ПГУ1.090301.2.001 ПЗ	Лист
					47
Изм.	Лист	№ докум	Подпись Дата		

Файл DiplomApplication.java

```
package com.example.diplom;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
public class DiplomApplication {

    public static void main(String[] args) {
        SpringApplication.run(DiplomApplication.class, args);
    }

}
```

Файл Bot.java

```
package com.example.diplom.bot.service;

import com.example.diplom.bot.buttons.BackButtons;
import com.example.diplom.bot.buttons.BotCommands;
import com.example.diplom.bot.buttons.GenreButtons;
import com.example.diplom.bot.buttons.MainButtons;
import com.example.diplom.bot.config.BotConfig;
import com.example.diplom.bot.enums.SearchParamEnum;
import com.example.diplom.dto.db.FilmDatabaseDto;
import com.example.diplom.dto.genre.GenresDto;
import com.example.diplom.dto.name.FilmsDto;
import com.example.diplom.dto.top.TotalTopFilmsDto;
import com.example.diplom.proxy.FeignKinopoiskApi;
import com.example.diplom.service.FilmService;
import lombok.Data;
import org.springframework.beans.factory.annotation.Autowired;
```

				ПГУ1.090301.2.001 ПЗ	Лист
					48
Изм.	Лист	№ докум	Подпись Дата		

```

import org.springframework.stereotype.Component;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import
org.telegram.telegrambots.meta.api.methods.commands.SetMyCommand
S;
import
org.telegram.telegrambots.meta.api.methods.send.SendDocument;
import
org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import
org.telegram.telegrambots.meta.api.methods.send.SendPhoto;
import
org.telegram.telegrambots.meta.api.methods.send.SendVideo;
import org.telegram.telegrambots.meta.api.objects.InputFile;
import org.telegram.telegrambots.meta.api.objects.Update;
import
org.telegram.telegrambots.meta.api.objects.commands.scope.BotCom
mandScopeDefault;
import
org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.*;

@Component
@Data
public class Bot extends TelegramLongPollingBot implements
BotCommands {

    private final String API_TOKEN = "ce25666c-fbd9-45cb-8466-
071170dad302";

    @Autowired

```

				ПГУ1.090301.2.001 ПЗ	Лист
					49
Изм.	Лист	№ докум	Подпись Дата		

```

private final BotConfig botConfig;
@Autowired
private FeignKinopoiskApi feignKinopoiskApi;
@Autowired
private FilmService service;
private List<Long> startWait;

private Map<String, List<Long>> waitCommand;

public Bot(BotConfig botConfig, FeignKinopoiskApi
feignKinopoiskApi) {
    this.botConfig = botConfig;
    waitCommand = new HashMap<>();
    waitCommand.put(SearchParamEnum.RATING.name(), new
ArrayList<>());
    waitCommand.put(SearchParamEnum.NAME.name(), new
ArrayList<>());
    waitCommand.put(SearchParamEnum.GENRE.name(), new
ArrayList<>());
    waitCommand.put(SearchParamEnum.TOP.name(), new
ArrayList<>());
    waitCommand.put(SearchParamEnum.WATCH.name(), new
ArrayList<>());
    try {
        execute(new SetMyCommands(COMMAND_LIST, new
BotCommandScopeDefault(), null));
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
}

@Override
public String getBotUsername() {
    return botConfig.getBotUsername();
}

```

				ПГУ1.090301.2.001 ПЗ	Лист
					50
Изм.	Лист	№ докум	Подпись Дата		

```

    }

    @Override
    public String getBotToken() {
        return botConfig.getBotToken();
    }

    @Override
    public void onUpdateReceived(Update update) {
        long chatId = 0;
        long userId = 0; //это нам понадобится позже
        String userName = null;
        String receivedMessage;

        //если получено сообщение текстом
        if (update.hasMessage()) {
            chatId = update.getMessage().getChatId();
            userId = update.getMessage().getFrom().getId();
            userName =
update.getMessage().getFrom().getFirstName();
            if
(waitCommand.get(SearchParamEnum.NAME.name()).contains(update.ge
tMessage().getChatId()) && update.getMessage().hasText()) {
                searchByName(update.getMessage().getChatId(),
update.getMessage().getText());
                try {
                    execute(SendMessage.builder()

.chatId(String.valueOf(update.getMessage().getChatId()))
                    .text("Выберите дальнейшее
действие:")

.replyMarkup(BackButtons.inlineMarkup())
                    .build());
                } catch (TelegramApiException e) {

```

				ПГУ1.090301.2.001 ПЗ	Лист
					51
Изм.	Лист	№ докум	Подпись Дата		

```

        throw new RuntimeException(e);
    }
    } else if
    (waitCommand.get(SearchParamEnum.RATING.name()).contains(update.
getMessage().getChatId()) && update.getMessage().hasText()) {
        searchByRating(update.getMessage().getChatId(),
Integer.valueOf(update.getMessage().getText()));
        try {
            execute(SendMessage.builder()

.chatId(String.valueOf(update.getMessage().getChatId()))
                .text("Выберите дальнейшее
действие:")

.replyMarkup(BackButtons.inlineMarkup())
                .build());
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    } else if
    (waitCommand.get(SearchParamEnum.WATCH.name()).contains(update.g
etMessage().getChatId()) && update.getMessage().hasText()) {
        watch(chatId,
Long.valueOf(update.getMessage().getText()));
        try {
            execute(SendMessage.builder()

.chatId(String.valueOf(update.getMessage().getChatId()))
                .text("Выберите дальнейшее
действие:")

.replyMarkup(BackButtons.inlineMarkup())
                .build());
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);

```

				ПГУ1.090301.2.001 ПЗ	Лист
					52
Изм.	Лист	№ докум	Подпись Дата		

```

        }
    }

    if (update.getMessage().hasText()) {
        receivedMessage = update.getMessage().getText();
        try {
            botAnswerUtils(receivedMessage, chatId,
userName);
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    }

    //если нажата одна из кнопок бота
    } else if (update.hasCallbackQuery()) {
        chatId =
update.getCallbackQuery().getMessage().getChatId();
        userId =
update.getCallbackQuery().getFrom().getId();
        userName =
update.getCallbackQuery().getFrom().getFirstName();
        receivedMessage =
update.getCallbackQuery().getData();

        try {
            botAnswerUtils(receivedMessage, chatId,
userName);
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    }
}

private void botAnswerUtils(String receivedMessage, long
chatId, String userName) throws TelegramApiException {

```

				ПГУ1.090301.2.001 ПЗ	Лист
					53
Изм.	Лист	№ докум	Подпись Дата		

```

switch (receivedMessage) {
    case "/start":
        startBot(chatId, userName);
        break;
    case "/help":
        sendHelpText(chatId, HELP_TEXT);
        break;
    case "/name":
        preSearchByName(chatId);
        break;
    case "/back":
        goBack(chatId);
        break;
    case "/repead":
        goRepead(chatId);
        break;
    case "/genre":
        preSearchByGenre(chatId);
        break;
    case "/rating":
        preSearchByRating(chatId);
        break;
    case "/top":
        getTop(chatId);
        break;
    case "/watch":
        preWatch(chatId);
        break;
    case "/drama":
        searchByGenre(chatId, 2L);
        break;
    case "/thriller":
        searchByGenre(chatId, 1L);
        break;
    case "/crime":

```

				ПГУ1.090301.2.001 ПЗ	Лист
					54
Изм.	Лист	№ докум	Подпись Дата		

```

        searchByGenre(chatId, 3L);
        break;
    case "/detective":
        searchByGenre(chatId, 5L);
        break;
    case "/fantastic":
        searchByGenre(chatId, 6L);
        break;
    case "/adventure":
        searchByGenre(chatId, 7L);
        break;
    case "/action_move":
        searchByGenre(chatId, 11L);
        break;
    case "/horror":
        searchByGenre(chatId, 17L);
        break;
    case "/western":
        searchByGenre(chatId, 10L);
        break;
    case "/cartoon":
        searchByGenre(chatId, 18L);
        break;
    case "/family":
        searchByGenre(chatId, 19L);
        break;
    case "/comedy":
        searchByGenre(chatId, 13L);
        break;
    default:
        break;
    }
}

private void preWatch(Long chatId) {

```

				ПГУ1.090301.2.001 ПЗ	Лист
					55
Изм.	Лист	№ докум	Подпись Дата		

```

waitCommand.get(SearchParamEnum.WATCH.name()).add(chatId);

SendMessage searchWord = SendMessage.builder()
    .chatId(String.valueOf(chatId))
    .text("Пожалуйста введите ID желаемого
фильма(Целое число): ")
    .build();

try {
    execute(searchWord);
} catch (TelegramApiException e) {
    throw new RuntimeException(e);
}

}

private void watch(Long chatID, Long id) {
    FilmDatabaseDto filmById = service.getFilmById(435L);
    try {
        execute(SendMessage.builder()
            .chatId(String.valueOf(chatID))
            .text(filmById.toString())
            .build());
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
}

private void startBot(long chatId, String userName) {

    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText("Приветствую, " + userName + ", я, " +
getBotUsername() + ", чем могу помочь?");
    message.setReplyMarkup(MainButtons.inlineMarkup());

    try {

```

				ПГУ1.090301.2.001 ПЗ	Лист
					56
Изм.	Лист	№ докум	Подпись Дата		

```

        execute(message);
    } catch (TelegramApiException e) {
    }
}

private void goBack(Long chatId) {
    if
(waitCommand.get(SearchParamEnum.NAME.name()).contains(chatId))
{

waitCommand.get(SearchParamEnum.NAME.name()).remove(chatId);
    } else if
(waitCommand.get(SearchParamEnum.RATING.name()).contains(chatId)
) {

waitCommand.get(SearchParamEnum.RATING.name()).remove(chatId);
    } else
waitCommand.get(SearchParamEnum.GENRE.name()).remove(chatId);
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText("Чем могу помочь?");
    message.setReplyMarkup(MainButtons.inlineMarkup());

    try {
        execute(message);
    } catch (TelegramApiException e) {
    }

}

private void goRepead(Long chatId) {
    if
(waitCommand.get(SearchParamEnum.NAME.name()).contains(chatId))
{

        preSearchByName(chatId);

```

				ПГУ1.090301.2.001 ПЗ	Лист
					57
Изм.	Лист	№ докум	Подпись Дата		

```

        }
        if
(waitCommand.get(SearchParamEnum.GENRE.name()).contains(chatId))
{
            preSearchByGenre(chatId);
        }
    }

private void sendHelpText(long chatId, String textToSend) {
    SendMessage message = new SendMessage();
    message.setChatId(String.valueOf(chatId));
    message.setText(textToSend);
    message.setReplyMarkup(MainButtons.inlineMarkup());

    try {
        execute(message);
    } catch (TelegramApiException e) {
    }
}

private void getTop(Long chatId) {
    waitCommand.get(SearchParamEnum.TOP.name()).add(chatId);
    TotalTopFilmsDto filmDto =
feignKinopoiskApi.getTop(API_TOKEN, (int) (Math.random() * 11));
    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(String.valueOf(chatId));

    System.out.println(filmDto.getFilms().get(0).getNameRu());
    sendMessage.setText(filmDto.toString());
    if (filmDto.getFilms().size() >= 5) {
        filmDto.getFilms().stream().limit(5).map(x -> {
            try {
                return execute(SendMessage.builder()
                    .chatId(String.valueOf(chatId))

```

```

        .text(x.toString())
        .build());
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
    })).forEach(System.out::println);
} else {
    filmDto.getFilms().stream().map(x -> {
        try {
            return execute(SendMessage.builder()
                .chatId(String.valueOf(chatId))
                .text(x.toString())
                .build());
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    })).forEach(System.out::println);
}
try {
    execute(SendMessage.builder()
        .chatId(String.valueOf(chatId))
        .text("Выберите дальнейшее действие:")
        .replyMarkup(BackButtons.inlineMarkup())
        .build());
} catch (TelegramApiException e) {
    throw new RuntimeException(e);
}
}

```

```

private void preSearchByRating(Long chatId) {

waitCommand.get(SearchParamEnum.RATING.name()).add(chatId);
    SendMessage searchWord = SendMessage.builder()
        .chatId(String.valueOf(chatId))
        .text("Пожалуйста введите желаемый рейтинг

```

				ПГУ1.090301.2.001 ПЗ	Лист
					59
Изм.	Лист	№ докум	Подпись Дата		

```

фильма(Целое число): ")
        .build();

    try {
        execute(searchWord);
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
}

private void searchByRating(Long chatId, Integer rating) {
    GenresDto ratingDto =
feignKinopoiskApi.getByRating(API_TOKEN, rating);
    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(String.valueOf(chatId));

    System.out.println(ratingDto.getItems().get(0).getNameRu());
    sendMessage.setText(rating.toString());
    if (ratingDto.getItems().size() >= 5) {
        ratingDto.getItems().stream().limit(5).map(x -> {
            try {
                return execute(SendMessage.builder()
                    .chatId(String.valueOf(chatId))
                    .text(x.toString())
                    .build());
            } catch (TelegramApiException e) {
                throw new RuntimeException(e);
            }
        }).forEach(System.out::println);
    } else {
        ratingDto.getItems().stream().map(x -> {
            try {
                return execute(SendMessage.builder()
                    .chatId(String.valueOf(chatId))
                    .text(x.toString())
                    .build());
            }

```

				ПГУ1.090301.2.001 ПЗ	Лист
					60
Изм.	Лист	№ докум	Подпись Дата		

```

        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    }).forEach(System.out::println);
}

}

private void preSearchByGenre(Long chatId) {

waitCommand.get(SearchParamEnum.GENRE.name()).add(chatId);
    SendMessage searchGenre = SendMessage.builder()
        .chatId(String.valueOf(chatId))
        .text("Пожалуйста выберите жанр:")
        .replyMarkup(GenreButtons.inlineMarkup())
        .build();

    try {
        execute(searchGenre);
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
}

private void searchByGenre(Long chatId, Long id) {
    GenresDto filmDto =
feignKinopoiskApi.getByGenre(API_TOKEN, id);
    System.out.println(filmDto.toString());
    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(String.valueOf(chatId));

    System.out.println(filmDto.getItems().get(0).getNameRu());
    sendMessage.setText(filmDto.toString());
    if (filmDto.getItems().size() >= 5) {
        filmDto.getItems().stream().limit(5).map(x -> {
            try {

```

				ПГУ1.090301.2.001 ПЗ	Лист
					61
Изм.	Лист	№ докум	Подпись Дата		

```

        return execute(SendMessage.builder()
            .chatId(String.valueOf(chatId))
            .text(x.toString())
            .build());
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
    }).forEach(System.out::println);
} else {
    filmDto.getItems().stream().map(x -> {
        try {
            return execute(SendMessage.builder()
                .chatId(String.valueOf(chatId))
                .text(x.toString())
                .build());
        } catch (TelegramApiException e) {
            throw new RuntimeException(e);
        }
    }).forEach(System.out::println);
}
try {
    execute(SendMessage.builder()
        .chatId(String.valueOf(chatId))
        .text("Выберите дальнейшее действие:")
        .replyMarkup(BackButtons.inlineMarkup())
        .build());
} catch (TelegramApiException e) {
    throw new RuntimeException(e);
}
}

```

```

private void preSearchByName(Long chatId) {

```

```

    waitCommand.get(SearchParamEnum.NAME.name()).add(chatId);
    SendMessage searchWord = SendMessage.builder()

```

				ПГУ1.090301.2.001 ПЗ	Лист
					62
Изм.	Лист	№ докум	Подпись Дата		

```

        .chatId(String.valueOf(chatId))
        .text("Пожалуйста введите название фильма:")
        .build();

    try {
        execute(searchWord);
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
}

private void searchByName(Long chatId, String name) {
    FilmsDto filmDto =
feignKinopoiskApi.getBy_name(API_TOKEN, name);
    SendMessage sendMessage = new SendMessage();
    sendMessage.setChatId(String.valueOf(chatId));

    System.out.println(filmDto.getFilms().get(0).getNameRu());
    sendMessage.setText(filmDto.toString());
    if (filmDto.getFilms().size() >= 5) {
        filmDto.getFilms().stream().limit(5).map(x -> {
            try {
                return execute(SendMessage.builder()
                    .chatId(String.valueOf(chatId))
                    .text(x.toString())
                    .build());
            } catch (TelegramApiException e) {
                throw new RuntimeException(e);
            }
        }).forEach(System.out::println);
    } else {
        filmDto.getFilms().stream().map(x -> {
            try {
                return execute(SendMessage.builder()
                    .chatId(String.valueOf(chatId))

```

```

        .text(x.toString())
        .build());
    } catch (TelegramApiException e) {
        throw new RuntimeException(e);
    }
    }).forEach(System.out::println);
}
}
}

```

Файл InitBot.java

```

package com.example.diplom.bot.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import
org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import
org.telegram.telegrambots.updatesreceivers.DefaultBotSession;

@Component
public class InitBot {
    @Autowired
    private Bot bot;

    @EventListener({ContextRefreshedEvent.class})
    public void init() throws TelegramApiException {
        TelegramBotsApi telegramBotsApi = new
TelegramBotsApi(DefaultBotSession.class);
        telegramBotsApi.registerBot(bot);
    }
}

```

				ПГУ1.090301.2.001 ПЗ	Лист
					64
Изм.	Лист	№ докум	Подпись Дата		

Файл BotConfig.java

```
package com.example.diplom.bot.config;

import lombok.Getter;
import org.springframework.beans.factory.annotation.Value;
import
org.springframework.boot.context.properties.ConfigurationPropert
ies;
import org.springframework.context.annotation.Configuration;

@Configuration
@Getter
public class BotConfig {

    @Value("${bot.name}")
    private String botUsername;          //TODO refactor String on
prop variable
    @Value("${bot.token}")
    private String botToken;
}
```

Файл SearchParamEnum.java

```
package com.example.diplom.bot.enums;

public enum SearchParamEnum {
    NAME("name"), RATING("rating"), GENRE("genre"),
    TOP("top"), WATCH("watch");
    String param;
    SearchParamEnum(String param) {

    }
}
```

Файл BackButtons.java

```
package com.example.diplom.bot.buttons;
```

				ПГУ1.090301.2.001 ПЗ	Лист
					65
Изм.	Лист	№ докум	Подпись Дата		

```

import
org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineK
eyboardMarkup;
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons
.InlineKeyboardButton;

import java.util.List;

public class BackButtons {

    private static final InlineKeyboardButton BACK_BUTTON = new
InlineKeyboardButton("Назад");

    private static final InlineKeyboardButton AGAIN_BUTTON = new
InlineKeyboardButton("Повторить");

    public static InlineKeyboardMarkup inlineMarkup() {
        BACK_BUTTON.setCallbackData("/back");
        AGAIN_BUTTON.setCallbackData("/repead");

        List<InlineKeyboardButton> rowNavigate =
List.of(BACK_BUTTON, AGAIN_BUTTON);
        List<List<InlineKeyboardButton>> rowsInLine =
List.of(rowNavigate);

        InlineKeyboardMarkup markupInline = new
InlineKeyboardMarkup();
        markupInline.setKeyboard(rowsInLine);

        return markupInline;
    }
}

```

				ПГУ1.090301.2.001 ПЗ	Лист
					66
Изм.	Лист	№ докум	Подпись Дата		

Файл GenreButtons.java

```
package com.example.diplom.bot.buttons;

import
org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineK
eyboardMarkup
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons
.InlineKeyboardButton;

import java.util.List;

public class GenreButtons {

    private static final InlineKeyboardButton DRAMA_BUTTON = new
InlineKeyboardButton("Драма");
    private static final InlineKeyboardButton THRILLER_BUTTON =
new InlineKeyboardButton("Триллер");
    private static final InlineKeyboardButton CRIME_BUTTON = new
InlineKeyboardButton("Криминал");
    private static final InlineKeyboardButton DETECTIVE_BUTTON =
new InlineKeyboardButton("Детектив");
    private static final InlineKeyboardButton FANTASTIC_BUTTON =
new InlineKeyboardButton("Фантастика");
    private static final InlineKeyboardButton ADVENTURES_BUTTON
= new InlineKeyboardButton("Приключения");
    private static final InlineKeyboardButton
ACTION_MOVIE_BUTTON = new InlineKeyboardButton("Боевик");
    private static final InlineKeyboardButton COMEDY_BUTTON =
new InlineKeyboardButton("Комедия");
    private static final InlineKeyboardButton HORROR_BUTTON =
new InlineKeyboardButton("Ужасы");
    private static final InlineKeyboardButton WESTERN_BUTTON =
new InlineKeyboardButton("Вестерн");
    private static final InlineKeyboardButton CARTOON_BUTTON =
new InlineKeyboardButton("Мультфильм");
```

				ПГУ1.090301.2.001 ПЗ	Лист
					67
Изм.	Лист	№ докум	Подпись Дата		

```
private static final InlineKeyboardButton FAMILY_BUTTON =
new InlineKeyboardButton("Семейный");
```

```
public static InlineKeyboardMarkup inlineMarkup() {
    DRAMA_BUTTON.setCallbackData("/drama");
    THRILLER_BUTTON.setCallbackData("/thriller");
    CRIME_BUTTON.setCallbackData("/crime");
    DETECTIVE_BUTTON.setCallbackData("/detective");
    FANTASTIC_BUTTON.setCallbackData("/fantastic");
    ADVENTURES_BUTTON.setCallbackData("/adventure");
    ACTION_MOVIE_BUTTON.setCallbackData("/action_move");
    HORROR_BUTTON.setCallbackData("/horror");
    WESTERN_BUTTON.setCallbackData("/western");
    CARTOON_BUTTON.setCallbackData("/cartoon");
    FAMILY_BUTTON.setCallbackData("/family");
    COMEDY_BUTTON.setCallbackData("/comedy");
```

```
List<InlineKeyboardButton> rowNavigate =
List.of(DRAMA_BUTTON, THRILLER_BUTTON, CRIME_BUTTON,
DETECTIVE_BUTTON);
```

```
List<InlineKeyboardButton> rowNavigate2 =
List.of(FANTASTIC_BUTTON, ADVENTURES_BUTTON,
ACTION_MOVIE_BUTTON, HORROR_BUTTON);
```

```
List<InlineKeyboardButton> rowNavigate3 =
List.of(WESTERN_BUTTON, CARTOON_BUTTON, FAMILY_BUTTON,
COMEDY_BUTTON);
```

```
List<List<InlineKeyboardButton>> rowsInLine =
List.of(rowNavigate, rowNavigate2, rowNavigate3);
```

```
InlineKeyboardMarkup markupInline = new
InlineKeyboardMarkup();
markupInline.setKeyboard(rowsInLine);
```

				ПГУ1.090301.2.001 ПЗ	Лист
					68
Изм.	Лист	№ докум	Подпись Дата		

```

        return markupInline;
    }
}

```

Файл MainButtons.java

```

package com.example.diplom.bot.buttons;

import
org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineK
eyboardMarkup
import
org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons
.InlineKeyboardButton;

import java.util.List;

public class MainButtons {
    private static final InlineKeyboardButton START_BUTTON = new
InlineKeyboardButton("Топ фильмов");
    private static final InlineKeyboardButton HELP_BUTTON = new
InlineKeyboardButton("Help");
    private static final InlineKeyboardButton NAME_BUTTON = new
InlineKeyboardButton("По названию");
    private static final InlineKeyboardButton GENRE_BUTTON = new
InlineKeyboardButton("По жанру");
    private static final InlineKeyboardButton RATING_BUTTON =
new InlineKeyboardButton("По рейтингу");
    private static final InlineKeyboardButton WATCH_BUTTON = new
InlineKeyboardButton("Посмотреть фильм:");

    public static InlineKeyboardMarkup inlineMarkup() {
        START_BUTTON.setCallbackData("/top");
        HELP_BUTTON.setCallbackData("/help");
        NAME_BUTTON.setCallbackData("/name");
    }
}

```

				ПГУ1.090301.2.001 ПЗ	Лист
					69
Изм.	Лист	№ докум	Подпись Дата		

```

        GENRE_BUTTON.setCallbackData("/genre");
        RATING_BUTTON.setCallbackData("/rating");
        WATCH_BUTTON.setCallbackData("/watch");

        List<InlineKeyboardButton> rowHelp =
List.of(START_BUTTON, HELP_BUTTON);
        List<InlineKeyboardButton> rowInline =
List.of(NAME_BUTTON, GENRE_BUTTON, RATING_BUTTON);
        List<InlineKeyboardButton> rowWatch =
List.of(WATCH_BUTTON);

        List<List<InlineKeyboardButton>> rowsInLine =
List.of(rowInline, rowHelp, rowWatch);

        InlineKeyboardMarkup markupInline = new
InlineKeyboardMarkup();
        markupInline.setKeyboard(rowsInLine);

        return markupInline;
    }
}

```

Файл BotCommands.java

```

package com.example.diplom.bot.buttons;
import
org.telegram.telegrambots.meta.api.objects.commands.BotCommand;
import java.util.List;

public interface BotCommands {
    List<BotCommand> COMMAND_LIST = List.of(
        new BotCommand("/start", "start bot"),
        new BotCommand("/help", "bot info"),
        new BotCommand("/name", "name search"),
        new BotCommand("/rating", "rating search"),
        new BotCommand("/genre", "genre search"),

```

				ПГУ1.090301.2.001 ПЗ	Лист
					70
Изм.	Лист	№ докум	Подпись Дата		

```

        new BotCommand("/back", "back"),
        new BotCommand("/repead", "repead of search")
    );

    String HELP_TEXT = "Я помогу выбрать фильм для прекрасного
времяпрепровождения:)";
}

```

Файл FilmDto.java

```

package com.example.diplom.dto;
import lombok.*;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class FilmDto {
    private String nameRu;
    private String year;
    private String description;
    private String filmLength;
    private String ratingKinopoisk;
    private String posterUrl;

    @Override
    public String toString() {

        return
            "Название: " + nameRu + "\n"
            + "Год: " + year + "\n"
            + "Описание: " + description + "\n"
            + "Продолжительность: " + filmLength + "\n"
            + "Рейтинг: " + ratingKinopoisk + "\n"
            + posterUrl;
    }
}

```

```
}  
}
```

Файл FilmByNameDto.java

```
package com.example.diplom.dto.name;  
  
import lombok.*;  
import java.util.List;  
  
@Getter  
@Setter  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
public class FilmByNameDto {  
    private Long filmId;  
    private String nameRu;  
    private String year;  
    private String description;  
    private String rating;  
    private String posterUrl;  
    private List<Genre> genres;  
  
    @Override  
    public String toString() {  
  
        return  
            "ID: " + filmId + "\n"  
            + "Название: " + nameRu + "\n"  
            + "Год: " + year + "\n"  
            + "Описание: " + description + "\n"  
            + "Рейтинг: " + rating + "\n"  
            + "Жанры: " + genres + "\n"  
            + posterUrl;  
    }  
}
```

				ПГУ1.090301.2.001 ПЗ	Лист
					72
Изм.	Лист	№ докум	Подпись Дата		

Файл FilmsDto.java

```
package com.example.diplom.dto.name;
import lombok.*;
import java.util.List;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class FilmsDto {
    List<FilmByNameDto> films;

    @Override
    public String toString() {

        return
            "Название: " + films.get(0).getNameRu() + "\n"
            + "Год: " + films.get(0).getYear() +
            "\n"
            + "Описание: " +
            films.get(0).getDescription() + "\n"
            + "Рейтинг: " + films.get(0).getRating()
            + "\n"
            + films.get(0).getPosterUrl();
    }
}
```

Файл Genre.java

```
package com.example.diplom.dto.name;
import lombok.*;

@Getter
@Setter
@Builder
@AllArgsConstructor
```

				ПГУ1.090301.2.001 ПЗ	Лист
					73
Изм.	Лист	№ докум	Подпись Дата		

```

@NoArgsConstructor
public class Genre {
    private String genre;

    @Override
    public String toString() {
        return genre + ", ";
    }
}

```

Файл TopFilmsDto.java

```

package com.example.diplom.dto.top;
import com.example.diplom.dto.name.Genre;
import lombok.*;
import java.util.List;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class TopFilmsDto {
    private String filmId;
    private String nameRu;
    private String year;
    private List<Genre> genres;
    private String rating;
    private String posterUrl;

    @Override
    public String toString() {

        return
            "ID: " + filmId + "\n"
            + "Название: " + nameRu + "\n"
            + "Год: " + year + "\n"

```

				ПГУ1.090301.2.001 ПЗ	Лист
					74
Изм.	Лист	№ докум	Подпись Дата		

```

        + "Рейтинг: " + rating + "\n"
        + "Жанры: " + genres + "\n"
        + posterUrl;
    }
}

```

Файл TotalTopFilmsDto.java

```

package com.example.diplom.dto.top;

import lombok.*;
import java.util.List;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class TotalTopFilmsDto {
    private Integer pageCount;
    private List<TopFilmsDto> films;
}

```

Файл GenreDto.java

```

package com.example.diplom.dto.genre;

import com.example.diplom.dto.name.Genre;
import lombok.*;
import java.util.List;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class GenreDto {
    private String kinopoiskId;
    private String nameRu;
    private List<Genre> genres;
    private String ratingKinopoisk;
}

```

				ПГУ1.090301.2.001 ПЗ	Лист
					75
Изм.	Лист	№ докум	Подпись Дата		

```

private String ratingImdb;
private String year;
private String posterUrl;

@Override
public String toString() {

    return
        "ID: " + kinopoiskId + "\n"
        + "Название: " + nameRu + "\n"
        + "Год: " + year + "\n"
        + "Рейтинг кинопоиск: " +
ratingKinopoisk + "\n"
        + "Рейтинг IMdb: " + ratingImdb + "\n"
        + "Жанры: " + genres + "\n"
        + posterUrl;

}
}

```

Файл GenresDto.java

```

package com.example.diplom.dto.genre;

import lombok.*;
import java.util.List;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class GenresDto {
    private List<GenreDto> items;

    @Override
    public String toString() {

        return

```

```

        "Название: " + items.get(0).getNameRu() + "\n"
        + "Год: " + items.get(0).getYear() +
        "\n"
        + "Рейтинг: " +
        items.get(0).getRatingKinopoisk() + "\n"
        + items.get(0).getPosterUrl();
    }
}

```

Файл FilmDatabaseDto.java

```

package com.example.diplom.dto.db;

import lombok.*;

@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class FilmDatabaseDto {
    private Long filmId;
    private String name;
    private String reference;

    @Override
    public String toString() {

        return
            "ID: " + filmId + "\n"
            + "Название: \"" + name + "\"\n"
            + "Ссылка на фильм: " + reference;
    }
}

```

Файл Mapper.java

```

package com.example.diplom.mapper;

import com.example.diplom.dto.db.FilmDatabaseDto;

```

				ПГУ1.090301.2.001 ПЗ	Лист
					77
Изм.	Лист	№ докум	Подпись Дата		

```

import com.example.diplom.model.Film;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

@Component
public class Mapper {
    private final ModelMapper mapper;

    public Mapper() {
        this.mapper = new ModelMapper();
    }

    public FilmDatabaseDto toDto(Film film) {
        return mapper.map(film, FilmDatabaseDto.class);
    }

    public Film toEntity(FilmDatabaseDto dto) {
        return mapper.map(dto, Film.class);
    }
}

```

Файл FilmService.java

```

package com.example.diplom.service;

import com.example.diplom.dto.db.FilmDatabaseDto;
import com.example.diplom.mapper.Mapper;
import com.example.diplom.model.Film;
import com.example.diplom.repository.FilmRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class FilmService {

    @Autowired
    private final FilmRepository filmRepository;

    @Autowired

```

				ПГУ1.090301.2.001 ПЗ	Лист
					78
Изм.	Лист	№ докум	Подпись Дата		

```

private final Mapper mapper;

public FilmService(FilmRepository filmRepository, Mapper
mapper) {
    this.filmRepository = filmRepository;
    this.mapper = mapper;
    init();
}

private void init(){
    if (filmRepository.findAll().size() == 0) {
        filmRepository.save(Film.builder()
            .filmId(435L)
            .name("Зеленая миля")

            .reference("https://drive.google.com/file/d/1_qHswL7ULjrLLMMXfqo
8L3h-OPNIp84A/view?usp=drive_link")
            .build());
    }
}

public FilmDatabaseDto getFilmById(Long id) {
    return mapper.toDto(filmRepository.findById(id));
}
}

```

Файл Film.java

```

package com.example.diplom.model;
import lombok.*;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

```

				ПГУ1.090301.2.001 ПЗ	Лист
					79
Изм.	Лист	№ докум	Подпись Дата		

```

@Entity
@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Film {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Long filmId;
    private String name;
    private String reference;
}

```

Файл FeignKinopoiskApi.java

```

package com.example.diplom.proxy;

import com.example.diplom.dto.genre.GenresDto;
import com.example.diplom.dto.name.FilmsDto;
import com.example.diplom.dto.top.TotalTopFilmsDto;
import feign.Headers;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestParam;

@Component
@FeignClient(name = "kinoApi", url =
    "https://kinopoiskapiunofficial.tech/api")
@Headers("Content-Type:application/json")
public interface FeignKinopoiskApi {

```

				ПГУ1.090301.2.001 ПЗ	Лист
					80
Изм.	Лист	№ докум	Подпись Дата		

```

    @GetMapping("v2.1/films/search-by-keyword")
    FilmsDto getByName(@RequestHeader("X-API-KEY") String token,
        @RequestParam("keyword") String keyword);

    @GetMapping("/v2.2/films?genres={genres}")
    GenresDto getByGenre(@RequestHeader("X-API-KEY") String
        token, @PathVariable Long genres);

    @GetMapping("/v2.2/films")
    GenresDto getByRating(@RequestHeader("X-API-KEY") String
        token, @RequestParam("ratingFrom") Integer ratingFrom);

    @GetMapping("/v2.2/films/top")
    TotalTopFilmsDto getTop(@RequestHeader("X-API-KEY") String
        token, @RequestParam("page") Integer page);
}

```

Файл FilmRepository.java

```

package com.example.diplom.repository;

import com.example.diplom.model.Film;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface FilmRepository extends JpaRepository<Film,
    Long> {
    Film findByFilmId(Long id);
}

```

				ПГУ1.090301.2.001 ПЗ	Лист
					81
Изм.	Лист	№ докум	Подпись Дата		

Приложение В

(Обязательное)

Программный агрегатор системы мгновенного обмена сообщениями

(telegram)

Презентация

				ПГУ1.090301.2.001 ПЗ	Лист
					82
Изм.	Лист	№ докум	Подпись Дата		

ПРОГРАММНЫЙ АГРЕГАТОР СЕСТЕМЫ МГНОВЕННОГО ОБМЕНА СООБЩЕНИЯМИ (TELEGRAM)

Автор работы:

студент гр. 19ВВП1 Вожжов Илья Владимирович

Научный руководитель:

к.т.н., доцент кафедры ВТ Дубравин Алексей Викторович

Пенза, 2023

Цель работы и объекты

Цель работы – приложение предназначено для подбора кинофильма пользователю по определенным критериям.

Объектом проекта является клиент-серверное приложение, выполняющие следующие функции:

- предоставлении данных о кино;
- предоставление возможности поиска и просмотра кино.

				ПГУ1.090301.2.001 ПЗ	Лист
					83
Изм.	Лист	№ докум	Подпись Дата		

Задачи приложения

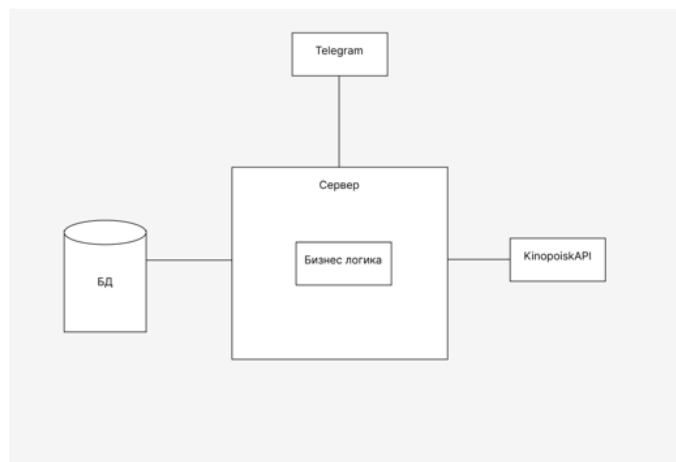
предоставить пользователю возможность

- поиска кинофильма по определенным критериям;
- получение основной информации о кинофильме;
- посмотреть или скачать выбранный фильм.

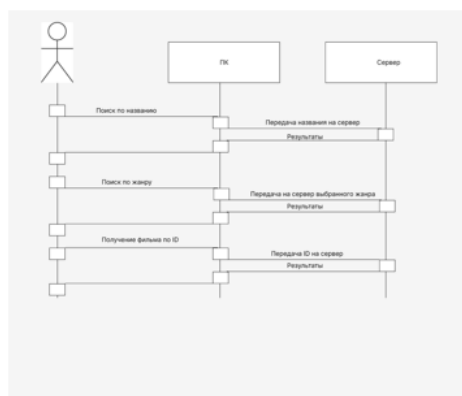
Стек технологий



Архитектура приложения

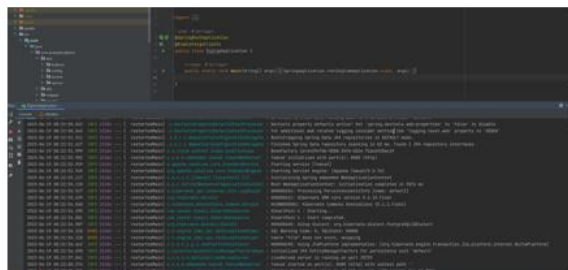
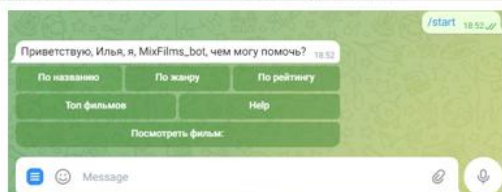
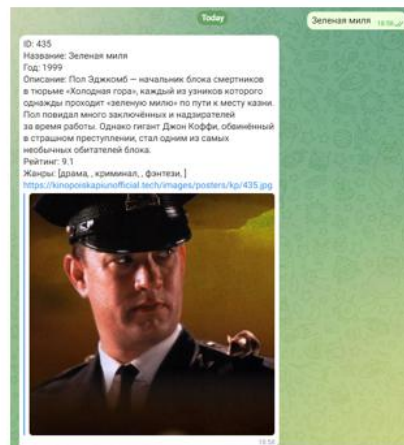


UML – диаграмма последовательности



Изм.	Лист	№ докум	Подпись Дата

Запуск приложения на исполнение



Результаты работы

- Разработано приложение полностью соответствующее техническому заданию;
- Проведено тестирование (успешно);
- Спланированы задачи по улучшению функциональности приложения.

Изм.	Лист	№ докум	Подпись Дата

ПГУ1.090301.2.001 ПЗ

Лист

86

Спасибо за внимание!

				ПГУ1.090301.2.001 ПЗ	Лист
					87
Изм.	Лист	№ докум	Подпись Дата		