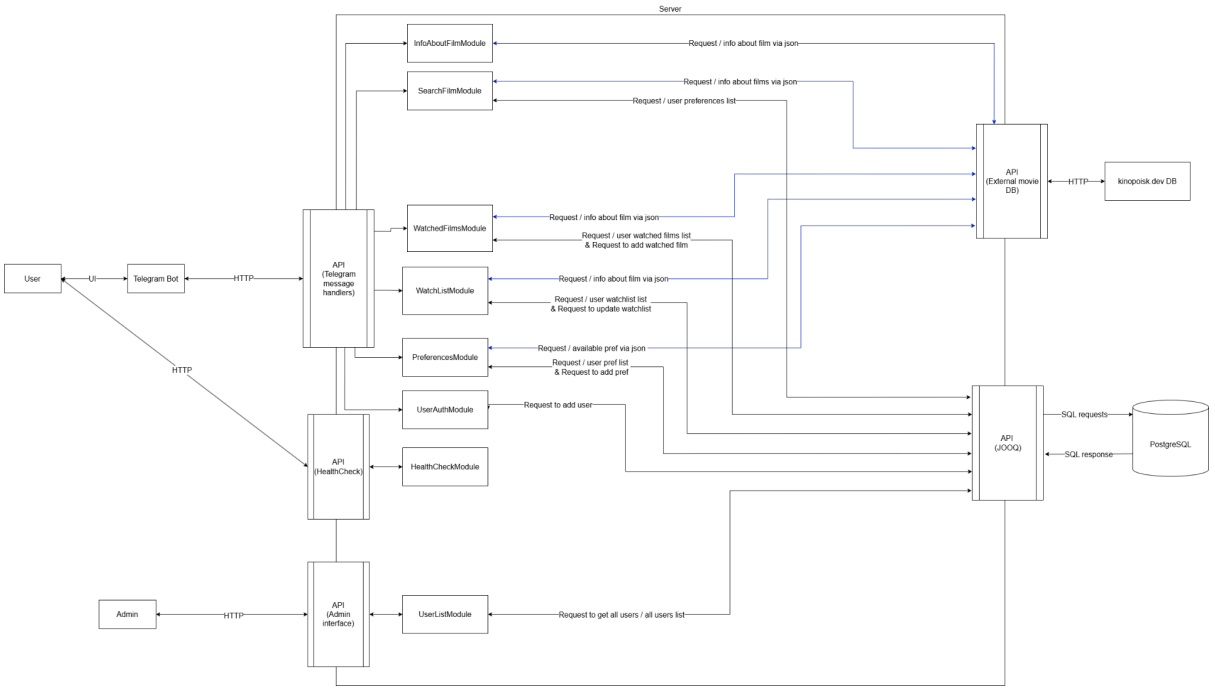
1. Диаграммы

1.1. Компонентная диаграмма



1. Server

- Основной компонент: сервер, который обрабатывает все запросы от Telegram бота и администратора.
- Функции: обеспечивает взаимодействие между Telegram ботом, администратором и различными модулями.

---

2. Telegram Bot

- Компонент: интерфейс для взаимодействия с пользователями через Telegram.
- Функции:
    - Передает команды и запросы от пользователя на сервер.
    - Отображает ответы сервера в чате Telegram.

---

3. User

- Компонент: пользователь Telegram.
- Функции:
    - Инициирует запросы через Telegram бота (например, поиск фильмов, добавление фильмов в список "Буду смотреть" и др. команды).
    - Получает ответы от бота.
    - Имеет возможность сделать `/healthcheck` (доступно без авторизации)

---

4. Admin

- Компонент: администратор системы.
- Функции:
    - Использует HTTP API для управления пользователями (получение списка всех пользователей).

---

5. API (Telegram message handlers)

- Компонент: модуль обработки сообщений Telegram.
- Функции:
    - Обрабатывает команды и запросы от Telegram бота.
    - Передает данные между Telegram ботом и соответствующими модулями.

---

6. API (HealthCheck)

- Компонент: модуль проверки состояния сервера.
- Функции:
    - Реализует команду /healthcheck для проверки работоспособности сервера.
    - Доступен без авторизации.

---

7. InfoAboutFilmModule

- Компонент: модуль получения информации о фильме.
- Функции:
    - Обрабатывает запросы на получение подробной информации о фильме.
    - Запрашивает данные из внешней базы данных фильмов (Kinopoisk.dev DB).

---

8. SearchFilmModule

- Компонент: модуль поиска фильмов.
- Функции:
    - Обрабатывает запросы на поиск фильмов с использованием фильтров, предпочтений пользователя или поиск случайного фильма.
    - Запрашивает данные из внешней базы данных фильмов (Kinopoisk.dev DB).
    - Для поиска по предпочтениям делается запрос в PostgreSQL БД.

---

9. WatchedFilmsModule

- Компонент: модуль управления списком просмотренных фильмов.
- Функции:
    - Обрабатывает запросы на добавление фильмов в список просмотренных.
    - Возвращает список просмотренных фильмов за определенный период пользователю.
    - Сохраняет информацию о фильмах в PostgreSQL БД.
    - При добавлении фильма в список просмотренных фильмов пользователя делается запрос во внешнюю БД фильмов (Kinopoisk.dev DB) для проверки наличия фильма и получения данных о нем.

---

10. WatchListModule

- Компонент: модуль управления списком "Буду смотреть".
- Функции:
    - Обрабатывает запросы на добавление/удаление фильмов из списка "Буду смотреть".
    - Возвращает текущий список "Буду смотреть" пользователю.
    - Сохраняет информацию о фильмах в PostgreSQL.
    - При добавлении фильма в список "Буду смотреть" пользователя делается запрос в внешнюю БД фильмов (Kinopoisk.dev DB) для проверки наличия фильма и получения данных о нем.

---

11. PreferencesModule

- Компонент: модуль управления предпочтениями пользователя.
- Функции:
    - Обрабатывает запросы на добавление/удаление/просмотр предпочтений пользователя.
    - Сохраняет предпочтения в PostgreSQL.
    - При добавлении новых предпочтений пользователя делается запрос во внешнюю БД фильмов (Kinopoisk.dev DB) для проверки корректности и получения данных о них (например, при добавлении предпочтений в актерах).

---

12. UserAuthModule

- Компонент: модуль аутентификации пользователей.
- Функции:
    - Обрабатывает автоматическую регистрацию новых пользователей.
    - Сохраняет метаданные пользователей в PostgreSQL.

---

13. HealthCheckModule

- Компонент: модуль проверки состояния сервера.
- Функции:
    - Реализует команду /healthcheck для проверки работоспособности сервера.
    - Возвращает состояние сервера и список студентов (авторов проекта).

---

14. UserListModule

- Компонент: модуль управления списком пользователей.
- Функции:
    - Обрабатывает запросы администратора на получение списка всех пользователей.
    - Возвращает список всех пользователей с их telegramId.

---

15. API (External movie DB)

- Компонент: внешняя база данных фильмов (Kinopoisk.dev DB).
- Функции:
    - Предоставляет информацию о фильмах (название, жанр, актеры, рейтинг и т.д.).
    - Используется только как источник данных.

---

16. API (JOOQ)

- Компонент: модуль работы с PostgreSQL.
- Функции:
    - Обрабатывает SQL запросы для взаимодействия с PostgreSQL.
    - Сохраняет/извлекает данные о пользователях, фильмах, предпочтениях и т.д.

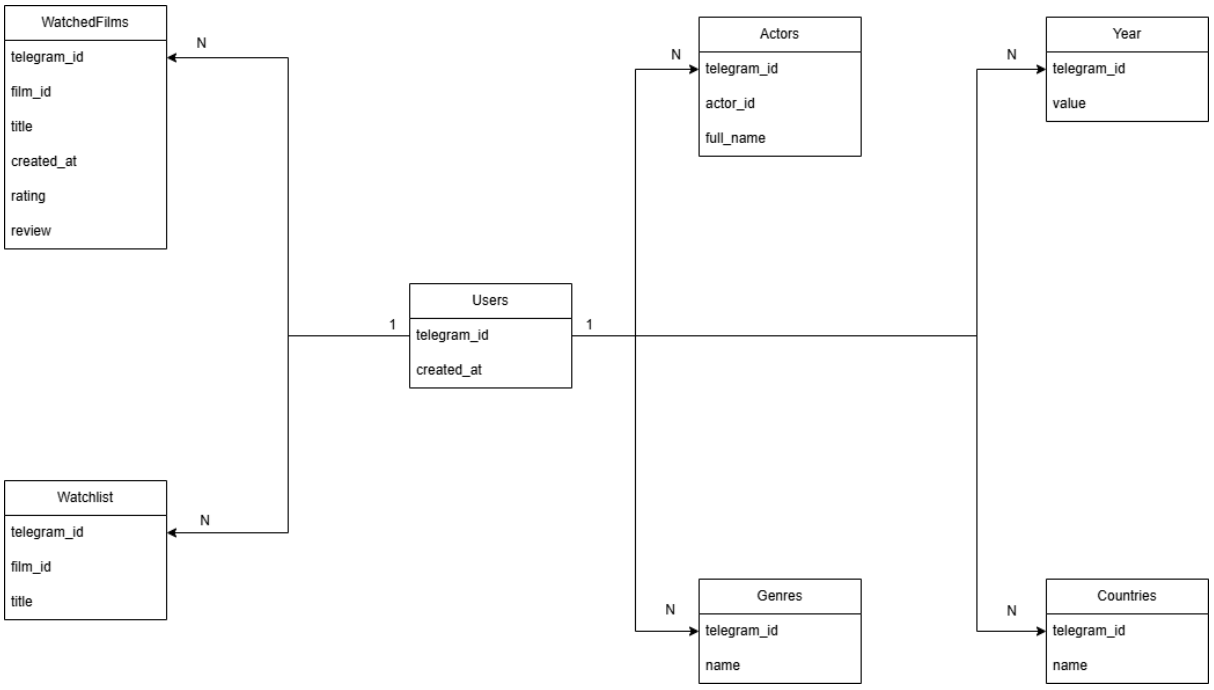
---

17. PostgreSQL

- Компонент: локальная реляционная база данных.

- Функции: хранит метаданные пользователей, такие как: список "Буду смотреть", просмотренные фильмы, оценки, отзывы, декомпозированные предпочтения пользователей.

1.2. База данных пользователей



2. Обоснование технологического стека

1. Язык программирования: Java 23
    a. Java является одним из самых надежных и производительных языков программирования, что критично для приложения, которое должно обрабатывать большое количество одновременных пользователей.
    b. Версия Java 23 предоставляет доступ к новым языковым конструкциям и улучшениям.
    c. Интеграция с Spring: Java сочетается с фреймворком Spring.
2. Фреймворк: Spring 6.2.3 (WebFlux, Spring Rest Docs)
    a. WebFlux:
        - Spring WebFlux позволяет создавать асинхронные и неблокирующие приложения, что важно для обеспечения высокой производительности и масштабируемости.
        - Реактивный подход позволяет эффективно обрабатывать большое количество запросов.
        - Подходит для интеграции с внешними API, минимизируя блокировку потоков.
    b. Spring Rest Docs:
        - Spring Rest Docs автоматизирует процесс создания документации для RESTful API, что упрощает поддержку и взаимодействие с приложением.
3. JOOQ
    a. JOOQ позволяет писать SQL-запросы в типизированном виде, что уменьшает вероятность ошибок и упрощает работу с базой данных.
    b. JOOQ предоставляет гибкость в работе с SQL, что позволяет оптимизировать запросы для повышения производительности.
4. База данных: PostgreSQL
    a. Поддерживает реляционную модель данных, что позволяет эффективно хранить и обрабатывать сложные связи между данными.
    b. Производительность и масштабируемость PostgreSQL
    c. Поддерживает стандартный SQL, что упрощает написание и оптимизацию запросов.
    d. Совместима с JOOQ
5. Инструмент сборки: Maven

    a. Maven упрощает управление зависимостями проекта, что особенно важно для проекта с большим количеством библиотек.
    b. Maven автоматизирует процесс сборки, тестирования и упаковки приложения, что ускоряет разработку и упрощает развертывание.
    c. Maven является стандартом для управления проектами на Java, что обеспечивает совместимость с другими инструментами и библиотеками.
6. Контейнеризация: Docker
    a. Docker позволяет изолировать приложение от окружения, что упрощает развертывание и тестирование на различных платформах.
    b. Docker упрощает масштабирование приложения, так как контейнеры могут быть легко развернуты на различных серверах или в облаке.
    c. Docker гарантирует, что приложение будет работать одинаково в любом окружении, что упрощает разработку и развертывание.
7. Тестирование: JUnit
    a. JUnit является стандартом для модульного тестирования в Java, что обеспечивает совместимость с другими инструментами и библиотеками.
    b. JUnit позволяет автоматизировать процесс тестирования, что ускоряет разработку и повышает качество кода.
    c. JUnit интегрируется с Maven, что упрощает запуск тестов в процессе сборки.
8. Интеграция с Telegram: Telegram API (+TelegramBots)
    a. Telegram API предоставляет все необходимые инструменты для взаимодействия с Telegram-ботом, что обеспечивает надежность и безопасность.
    b. TelegramBots – специализированный интерфейс, построенный на основе Telegram API, который упрощает создание и управление ботами. Он предоставляет готовые методы для работы с ботами, такие как отправка сообщений, обработка команд и взаимодействие с пользователями.
9. Сериализация и десериализация json-объектов: Jackson
    a. Jackson позволяет легко преобразовывать Java-объекты в JSON и обратно. Это важно для взаимодействия с API Кинопоиска, который возвращает данные в формате JSON.

b. Jackson является одной из самых быстрых библиотек для работы с JSON в Java.
c. Интегрируется с Spring WebFlux: автоматическое преобразование DTO в JSON через аннотации @RequestBody и @ResponseBody.

3. Описание развертывания приложения

3.1. Сборка приложения

Сборка приложения включает подготовку исполняемого файла (FatJAR) и Docker-образа

1. Убедитесь, что установлены следующие инструменты:
   - Java 23
   - Maven
   - Docker
2. Соберите проект при помощи Maven:

Пример команды для сборки проекта:

```
mvn clean package
```

3. Создайте Dockerfile, пример содержимого Dockerfile:

В pom.xml укажите конфигурацию для создания FatJAR:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.2.0</version>
            <configuration>
                <archive>
                    <manifest>

<addDefaultImplementationEntries>true</addDefaultImplementationEnt
ries>

<mainClass>com.example.MainClass</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
```

4. Создайте Dockerfile, пример содержимого Dockerfile:

```
FROM openjdk:23-jdk-alpine
COPY target/*.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

5. Создайте Docker-образ, пример команды для сборки Docker-образа:

```
docker build -t movie-recommendation-bot .
```

3.2. Описание деплоя

Деплой включает настройку инфраструктуры и запуск сервисов через Docker Compose.

1. Создайте файл docker-compose.yml.

Пример содержимого файла

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    image: dockerhub-username/telegram-bot-app:latest
    container_name: telegram-bot-app
    restart: unless-stopped
    depends_on:
      - postgres
    environment:
      - SPRING_PROFILES_ACTIVE=dev
      -
   DATABASE_URL=jdbc:postgresql://postgres:5432/telegram_bot_db
      - DATABASE_USERNAME=postgres
      - DATABASE_PASSWORD=postgres
      - TELEGRAM_BOT_TOKEN={telegram_bot_token}
      - TELEGRAM_BOT_USERNAME={bot_username}
    ports:
      - "8110:8110"
    volumes:
      - ./logs:/app/logs

  postgres:
    image: postgres:15-alpine
    container_name: telegram-bot-postgres
    environment:
      - POSTGRES_DB=telegram_bot_db
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  pgadmin:
    image: dpage/pgadmin4
```

```
    container_name: pgadmin
    environment:
      - PGADMIN_DEFAULT_EMAIL=admin@default.com
      - PGADMIN_DEFAULT_PASSWORD=admin
    ports:
      - "5050:80"
    depends_on:
      - postgres

volumes:
  postgres_data:
```

3.3. Описание запуска

После деплоя приложение спустя некоторое время будет готово к работе.

1. Запустите контейнеры: в терминале выполните команду:
   docker-compose up -d
2. Проверьте статус контейнеров: выполните команду:
   docker-compose ps
3. Отправьте запрос на healthcheck эндпоинт:
   /healthcheck.