### Анализ аналогичных платформ для управления спортивными мероприятиями и соревнованиями

#### Введение

В процессе проектирования и разработки социальной платформы для организации активных спортивных игр был проведен анализ существующих цифровых решений, ориентированных на управление спортивными организациями, турнирами и соревнованиями. Основной целью исследования было выявление ключевых функциональных возможностей, преимуществ и недостатков аналогичных платформ, а также определение направлений для создания уникального предложения, которое выделит разрабатываемое приложение среди существующих решений.

#### Описание и функционал аналогичных платформ

1. **e-Champs**  
   Цифровая платформа для управления спортивными организациями и соревнованиями.  
   **Основные функции:**
   * Прием заявок и онлайн-оплат.
   * Автоматическое создание турнирных сеток.
   * Публикация результатов.  
     **Преимущества:**
   * Упрощает процесс организации турниров.
   * Поддерживает интеграцию с платежными системами.  
     **Недостатки:**
   * Ориентирована преимущественно на турниры, может быть неприменима для регулярных тренировок или неформальных спортивных игр.
2. **MTGame**  
   Платформа для создания спортивных турниров, лиг, сбора статистики и проведения трансляций матчей.  
   **Основные функции:**
   * Гибкий конструктор лиг и турниров.
   * Цифровая заявочная кампания.
   * Управление площадками и расписанием.  
     **Преимущества:**
   * Поддерживает различные виды спорта.
   * Имеет инструменты для сбора статистики и трансляций.  
     **Недостатки:**
   * Может быть избыточной для небольших сообществ и любительских спортивных игр.
3. **Юнибор**  
   Платформа для организации спортивных турниров и учета профессионального роста спортсменов.  
   **Основные функции:**
   * Автоматизация подачи заявок и регистрации участников.
   * Учет результатов и достижений спортсменов.  
     **Преимущества:**
   * Объединяет федерации, спортивные школы и тренеров.
   * Поддерживает проведение онлайн-турниров.  
     **Недостатки:**
   * Ограниченные возможности для организации разовых или неформальных спортивных мероприятий.
4. **Мой Спорт**  
   Платформа для управления спортивными организациями, тренировочным процессом и соревнованиями.  
   **Основные функции:**
   * Ведение электронных дневников спортсменов.
   * Управление расписанием тренировок и соревнований.
   * Аналитика и отчеты.  
     **Преимущества:**
   * Интеграция с государственными информационными системами.
   * Поддержка различных ролей: спортсмены, тренеры, администраторы.  
     **Недостатки:**
   * Сложный интерфейс и функционал, ориентированный на крупные спортивные организации, что может быть неудобным для небольших сообществ.
5. **Orgeo.ru**  
   Сервис для организации спортивных мероприятий с функцией онлайн-заявок и трансляции результатов.  
   **Основные функции:**
   * Подача онлайн-заявок на участие в соревнованиях.
   * Трансляция результатов в реальном времени.  
     **Преимущества:**
   * Удобен для организаторов и участников.
   * Поддерживает различные виды спорта и охватывает широкую региональную аудиторию.  
     **Недостатки:**
   * Основной фокус сделан на соревнованиях, что снижает его применимость для регулярных тренировок и игр любительских команд.

#### Выводы для разрабатываемого проекта

Анализ аналогичных платформ показывает, что большинство существующих решений ориентированы на:

* Управление турнирами и соревнованиями, преимущественно для крупных или профессиональных сообществ.
* Поддержку организаций и федераций, что делает их избыточными для любительских или разовых мероприятий.

На основании проведенного анализа было выделено несколько ключевых направлений, которые помогут разрабатываемому приложению выделиться среди конкурентов:

1. **Фокус на неформальных или любительских спортивных играх.**  
   Пользователи смогут быстро создавать разовые мероприятия, такие как игры на выходных или тренировки, с минимальными требованиями к настройке.
2. **Простота и удобство использования.**  
   Основной акцент будет сделан на интуитивно понятный интерфейс и минимальное время, необходимое для создания или управления событиями.
3. **Интеграция с картографическими сервисами.**  
   Важно предоставить пользователям возможность выбирать и отмечать места проведения игр на карте, а также просматривать доступные события поблизости.
4. **Гибкая система оповещений и приглашений.**  
   Возможность приглашать участников, а также получать уведомления о предстоящих событиях.
5. **Минимизация избыточной функциональности.**  
   В отличие от платформ, ориентированных на крупные турниры, данное решение будет подходить как для небольших групп, так и для индивидуальных пользователей.

Таким образом, новая социальная платформа для активных игр станет удобным и эффективным инструментом для организации неформальных спортивных мероприятий, облегчая коммуникацию между участниками и улучшая доступ к информации о предстоящих играх.

**Описание архитектуры и функционала микросервисов**

**1. UserService**

**Назначение:**  
Сервис управления пользователями, их регистрацией, аутентификацией и профилями. Отвечает за создание и хранение профилей пользователей, включая их данные и взаимодействие с другими пользователями.

**Функционал:**

* **Регистрация и вход пользователей:** Используется OAuth2 или JWT для обеспечения безопасной аутентификации.
* **Хранение данных профилей:** Информация о пользователях (имя, фото, интересы, предпочитаемые виды спорта) хранится в базе данных.
* **Управление друзьями и участниками игр:** Пользователи могут добавлять других пользователей в друзья или пригласить их на участие в играх.

**Технологии:**

* **Spring Security** для реализации аутентификации и авторизации.
* **PostgreSQL** для хранения данных пользователей (имя, фото, интересы).

**2. EventService**

**Назначение:**  
Сервис для создания и управления спортивными событиями. Обеспечивает организацию мероприятий, управление участниками и оповещение о новых событиях.

**Функционал:**

* **Создание событий:** Включает информацию о дате, времени, месте, виде спорта и количестве участников.
* **Уведомления о новых событиях:** Подписчики получают уведомления о создании новых событий.
* **Управление участниками:** Пользователи могут записываться на участие или удаляться из списка участников.

**Технологии:**

* **PostgreSQL** для хранения данных о событиях (время, место, вид спорта).
* **REST API** для взаимодействия с другими сервисами платформы.

**3. NotificationService**

**Назначение:**  
Сервис отправки уведомлений пользователям о новых событиях, приглашениях и изменениях. Уведомления могут быть в формате email или push.

**Функционал:**

* **Отправка уведомлений:** Используется SMTP для email-уведомлений или Firebase для push-уведомлений.
* **Оповещение о новых играх и изменениях:** Пользователи получают уведомления о новых играх, приглашениях, а также изменениях в статусах событий.

**Технологии:**

* **Spring Boot** с интеграцией SMTP или Firebase для push-уведомлений.

**4. RecommendationService**

**Назначение:**  
Сервис для рекомендаций пользователей относительно событий и участников на основе их интересов и данных профилей. Использует алгоритмы анализа интересов пользователей для подбора наиболее релевантных событий.

**Функционал:**

* **Анализ интересов пользователей:** Сервис анализирует предпочтения пользователей (например, предпочтительные виды спорта и местоположение).
* **Алгоритм подбора событий:** Рекомендует подходящие события и участников на основе локации, вида спорта и уровня участников.

**Технологии:**

* **Машинное обучение или простая фильтрация** на основе базы данных для реализации алгоритмов рекомендаций.

**5. LocationService**

**Назначение:**  
Сервис для работы с геолокацией и выбора мест проведения спортивных игр. Он помогает пользователям найти подходящие места для игр и строить маршруты до них.

**Функционал:**

* **Интеграция с Google Maps API:** Для отображения карт и поиска ближайших локаций.
* **Сохранение популярных площадок:** Поддерживает список популярных мест для проведения игр.
* **Расчет маршрутов:** Сервис позволяет рассчитывать маршруты до места проведения игры.

**Технологии:**

* **API Google Maps** или аналогичные сервисы для работы с картами и геолокацией.

**6. ChatService**

**Назначение:**  
Сервис для обмена сообщениями между участниками игр. Он позволяет участникам обсуждать детали игр, делиться информацией и получать уведомления о новых сообщениях.

**Функционал:**

* **Групповые чаты:** Участники могут обмениваться сообщениями в реальном времени.
* **Уведомления о новых сообщениях:** Пользователи получают уведомления о новых сообщениях в чате.

**Технологии:**

* **WebSocket** для обмена сообщениями в реальном времени.
* **Redis** для хранения сообщений и поддержания состояния чатов.

**7. PaymentService (Опционально)**

**Назначение:**  
Сервис для работы с оплатой участия в платных мероприятиях. Обрабатывает транзакции и возвраты средств.

**Функционал:**

* **Интеграция с платежными системами:** Поддержка популярных платежных сервисов, таких как Stripe или PayPal.
* **Хранение информации о транзакциях:** Управление данными о проведенных оплатах и возвратах.
* **Возвраты средств:** Обработка запросов на возврат средств в случае отмены мероприятий.

**Технологии:**

* **Spring Boot** с интеграцией внешних платежных API.

**8. AnalyticsService**

**Назначение:**  
Сервис для анализа данных и генерации отчетов для администраторов платформы. Предоставляет информацию о популярных событиях, вовлеченности пользователей и других метриках.

**Функционал:**

* **Отчеты о популярности событий:** Оценка самых популярных событий.
* **Метрики вовлеченности пользователей:** Анализ активности пользователей на платформе.
* **Статистика по видам спорта и локациям:** Отчеты о наиболее популярных видах спорта и локациях.

**Технологии:**

* **Apache Kafka** для передачи событий в режиме реального времени.
* **PostgreSQL или ElasticSearch** для хранения и обработки данных.

**9. ApiGatewayService**

**Назначение:**  
Сервис, предоставляющий единую точку входа для всех пользователей и маршрутизирующий запросы к соответствующим микросервисам. Обеспечивает глобальную обработку ошибок и логирование.

**Функционал:**

* **Маршрутизация запросов:** Направление запросов от пользователей к нужным сервисам.
* **Обработка ошибок:** Централизованная обработка ошибок и логирование всех действий на платформе.

**Технологии:**

* **Spring Cloud Gateway** для реализации API Gateway.

**10. EurekaServer**

**Назначение:**  
Сервис для регистрации и обнаружения микросервисов, обеспечивающий автоматическое масштабирование и доступность всех сервисов.

**Функционал:**

* **Регистрация микросервисов:** Каждое приложение может зарегистрироваться в Eureka Server для автоматического обнаружения.
* **Масштабирование сервисов:** Поддержка динамического масштабирования сервисов.

**Технологии:**

* **Spring Cloud Netflix Eureka** для реализации обнаружения микросервисов.

**Общая архитектура**

1. **Backend:**
   * **Java + Spring Boot**: Основной стек для разработки микросервисов.
2. **База данных:**
   * **PostgreSQL**: Для хранения основной информации о пользователях, событиях и т.д.
   * **Redis**: Для кеширования данных, например, сообщений в чатах.
3. **Координация сервисов:**
   * **Eureka Server** и **API Gateway**: Обеспечивают доступность и маршрутизацию запросов.
4. **Обмен событиями:**
   * **Kafka**: Для асинхронного обмена событиями между сервисами.
5. **Облачное хранилище:**
   * **Amazon S3**: Для хранения файлов, таких как фотографии пользователей.

Эта архитектура предоставляет гибкую и масштабируемую платформу для управления спортивными событиями и взаимодействия пользователей в реальном времени.

**Описание таблиц для базы данных и ERD-диаграмма.**

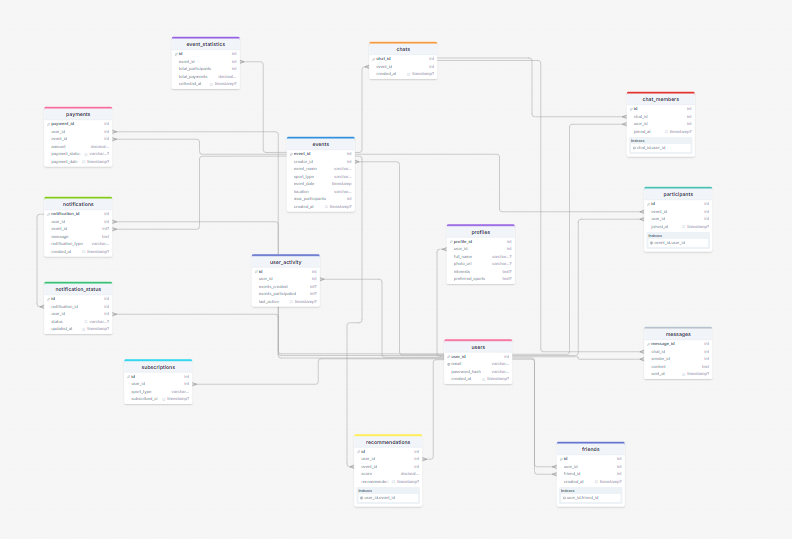


Рисунок 1 – ERD-диаграмма базы данных

#### Краткое пояснение каждой из таблиц. **1. Таблица** users

Хранит данные о зарегистрированных пользователях платформы.

* user\_id — уникальный идентификатор пользователя.
* email — электронная почта пользователя (уникальная).
* password\_hash — хэшированный пароль пользователя.
* created\_at — дата и время регистрации пользователя.

#### **2. Таблица** profiles

Хранит дополнительные данные профиля пользователя.

* profile\_id — уникальный идентификатор профиля.
* user\_id — идентификатор пользователя (внешний ключ к таблице users).
* full\_name — полное имя пользователя.
* photo\_url — URL-адрес фотографии профиля.
* interests — текстовое описание интересов пользователя.
* preferred\_sports — предпочитаемые виды спорта пользователя.

#### **3. Таблица** friends

Хранит информацию о дружеских связях между пользователями.

* id — уникальный идентификатор записи.
* user\_id — идентификатор пользователя (внешний ключ к таблице users).
* friend\_id — идентификатор друга (внешний ключ к таблице users).
* created\_at — дата и время добавления друга.
* **Уникальная пара** user\_id, friend\_id предотвращает дублирование связей.

#### **4. Таблица** events

Хранит информацию о созданных спортивных мероприятиях.

* event\_id — уникальный идентификатор мероприятия.
* creator\_id — идентификатор создателя мероприятия (внешний ключ к таблице users).
* event\_name — название мероприятия.
* sport\_type — тип спортивного мероприятия.
* event\_date — дата и время проведения мероприятия.
* location — место проведения мероприятия.
* max\_participants — максимальное количество участников.
* created\_at — дата и время создания мероприятия.

#### **5. Таблица** participants

Хранит информацию об участниках мероприятий.

* id — уникальный идентификатор записи.
* event\_id — идентификатор мероприятия (внешний ключ к таблице events).
* user\_id — идентификатор участника (внешний ключ к таблице users).
* joined\_at — дата и время присоединения к мероприятию.
* **Уникальная пара** event\_id, user\_id предотвращает дублирование участников одного мероприятия.

#### **6. Таблица** subscriptions

Хранит информацию о подписках пользователей на определенные виды спорта.

* id — уникальный идентификатор записи.
* user\_id — идентификатор пользователя (внешний ключ к таблице users).
* sport\_type — тип спорта, на который подписан пользователь.
* subscribed\_at — дата и время подписки.

#### **7. Таблица** notifications

Хранит уведомления для пользователей о событиях и действиях.

* notification\_id — уникальный идентификатор уведомления.
* user\_id — идентификатор пользователя, получившего уведомление (внешний ключ к таблице users).
* event\_id — идентификатор мероприятия, связанного с уведомлением (внешний ключ к таблице events).
* message — текст уведомления.
* notification\_type — тип уведомления (например, "новое событие", "приглашение").
* created\_at — дата и время создания уведомления.

#### **8. Таблица** notification\_status

Хранит статусы отправленных уведомлений.

* id — уникальный идентификатор записи.
* notification\_id — идентификатор уведомления (внешний ключ к таблице notifications).
* user\_id — идентификатор пользователя (внешний ключ к таблице users).
* status — статус уведомления (например, "отправлено", "просмотрено").
* updated\_at — дата и время последнего обновления статуса.

#### **9. Таблица** recommendations

Хранит рекомендации мероприятий для пользователей.

* id — уникальный идентификатор записи.
* user\_id — идентификатор пользователя, для которого создана рекомендация (внешний ключ к таблице users).
* event\_id — идентификатор рекомендуемого мероприятия (внешний ключ к таблице events).
* score — оценка или приоритет рекомендации.
* recommended\_at — дата и время создания рекомендации.

#### **10. Таблица** chats

Хранит информацию о чатах для мероприятий.

* chat\_id — уникальный идентификатор чата.
* event\_id — идентификатор мероприятия, к которому относится чат (внешний ключ к таблице events).
* created\_at — дата и время создания чата.

#### **11. Таблица** chat\_members

Хранит информацию об участниках чатов.

* id — уникальный идентификатор записи.
* chat\_id — идентификатор чата (внешний ключ к таблице chats).
* user\_id — идентификатор участника (внешний ключ к таблице users).
* joined\_at — дата и время присоединения к чату.
* **Уникальная пара** chat\_id, user\_id предотвращает дублирование участников в одном чате.

#### **12. Таблица** messages

Хранит сообщения в чатах.

* message\_id — уникальный идентификатор сообщения.
* chat\_id — идентификатор чата (внешний ключ к таблице chats).
* sender\_id — идентификатор отправителя (внешний ключ к таблице users).
* content — текст сообщения.
* sent\_at — дата и время отправки сообщения.

#### **13. Таблица** payments

Хранит информацию о платежах за участие в мероприятиях.

* payment\_id — уникальный идентификатор платежа.
* user\_id — идентификатор пользователя (внешний ключ к таблице users).
* event\_id — идентификатор мероприятия (внешний ключ к таблице events).
* amount — сумма платежа.
* payment\_status — статус платежа (например, "в ожидании", "завершен", "отменен").
* payment\_date — дата и время платежа.

#### **14. Таблица** event\_statistics

Хранит статистику мероприятий.

* id — уникальный идентификатор записи.
* event\_id — идентификатор мероприятия (внешний ключ к таблице events).
* total\_participants — общее количество участников мероприятия.
* total\_payments — общая сумма платежей за мероприятие.
* collected\_at — дата и время сбора статистики.

#### **15. Таблица** user\_activity

Хранит информацию о пользовательской активности.

* id — уникальный идентификатор записи.
* user\_id — идентификатор пользователя (внешний ключ к таблице users).
* events\_created — количество созданных пользователем мероприятий.
* events\_participated — количество мероприятий, в которых пользователь принял участие.
* last\_active — дата и время последней активности пользователя.

### ****1. Микросервис "Пользователи"****

Этот микросервис будет отвечать за управление пользователями, их ролями и аутентификацией.

#### **Функционал:**

* **Регистрация**: создание нового пользователя.
* **Авторизация**: вход в систему (JWT или OAuth2).
* **Профили**: просмотр и редактирование данных пользователя.
* **Роли**: разделение на организаторов и участников.

#### **Технологии:**

* Spring Boot, Spring Security, JWT.
* PostgreSQL для хранения данных пользователей.

#### **API:**

* POST /api/auth/register – регистрация.
* POST /api/auth/login – авторизация.
* GET /api/users/{id} – получение профиля.
* PUT /api/users/{id} – обновление профиля.

### ****2. Микросервис "Игры"****

Этот микросервис будет отвечать за создание, редактирование и поиск игр.

#### **Функционал:**

* **Создание игры**: организатор создает новую игру.
* **Редактирование игры**: изменение параметров игры.
* **Поиск игр**: фильтрация по дате, локации, типу игры.
* **Запись на игру**: участники могут записаться на игру.

#### **Технологии:**

* Spring Boot, Spring Data JPA.
* PostgreSQL для хранения данных об играх.

#### **API:**

* POST /api/games – создание игры.
* PUT /api/games/{id} – редактирование игры.
* GET /api/games – поиск игр (с фильтрами).
* POST /api/games/{id}/join – запись на игру.

### ****3. Микросервис "Мероприятия"****

Этот микросервис будет отвечать за управление мероприятиями, их расписанием и статусом.

#### **Функционал:**

* **Расписание**: отображение предстоящих и завершенных мероприятий.
* **Геолокация**: привязка мероприятия к локации (интеграция с картами).
* **Статус**: управление статусом мероприятия (например, "предстоящее", "завершенное").

#### **Технологии:**

* Spring Boot, Spring Data JPA.
* PostgreSQL для хранения данных о мероприятиях.
* Интеграция с Google Maps API или OpenStreetMap.

#### **API:**

* POST /api/events – создание мероприятия.
* PUT /api/events/{id} – редактирование мероприятия.
* GET /api/events – получение списка мероприятий.
* GET /api/events/{id}/location – получение геолокации.

### ****4. Микросервис "Социальная составляющая"****

Этот микросервис будет отвечать за взаимодействие пользователей между собой.

#### **Функционал:**

* **Чаты**: общение между участниками и организаторами.
* **Комментарии**: обсуждение игр и мероприятий.
* **Рейтинги**: оценка игр и организаторов.
* **Друзья**: добавление в друзья, список друзей.

#### **Технологии:**

* Spring Boot, WebSocket для чатов.
* PostgreSQL для хранения сообщений, комментариев и рейтингов.

#### **API:**

* POST /api/chat/messages – отправка сообщения.
* GET /api/chat/messages – получение сообщений.
* POST /api/comments – добавление комментария.
* POST /api/ratings – добавление рейтинга.
* POST /api/friends – добавление в друзья.

### ****5. Микросервис "Администрирование"****

Этот микросервис будет отвечать за управление платформой.

#### **Функционал:**

* **Управление пользователями**: блокировка, удаление, изменение ролей.
* **Управление играми**: модерация созданных игр.
* **Модерация контента**: удаление нежелательных комментариев или чатов.

#### **Технологии:**

* Spring Boot, Spring Security.
* PostgreSQL для хранения данных.

#### **API:**

* PUT /api/admin/users/{id} – изменение роли пользователя.
* DELETE /api/admin/users/{id} – удаление пользователя.
* DELETE /api/admin/games/{id} – удаление игры.
* DELETE /api/admin/comments/{id} – удаление комментария.

### ****6. Взаимодействие микросервисов****

Для взаимодействия между микросервисами можно использовать:

* **REST API**: простой и понятный способ.
* **Message Broker (Kafka, RabbitMQ)**: для асинхронного взаимодействия (например, уведомления).
* **Service Discovery (Eureka, Consul)**: для автоматического обнаружения сервисов.

### ****7. База данных****

Каждый микросервис может иметь свою базу данных (это принцип микросервисов). Для этого:

* Используйте отдельные схемы или базы данных в PostgreSQL для каждого сервиса.
* Настройте связи между данными через API (например, микросервис "Игры" запрашивает данные о пользователях через микросервис "Пользователи").

### ****8. Деплой****

Для деплоя микросервисов можно использовать:

* **Docker**: упакуйте каждый микросервис в контейнер.
* **Kubernetes**: для оркестрации контейнеров.
* **CI/CD**: автоматизируйте сборку и деплой (GitHub Actions, GitLab CI).

### ****9. Пример взаимодействия****

1. Пользователь регистрируется через микросервис "Пользователи".
2. Организатор создает игру через микросервис "Игры".
3. Участник ищет игру и записывается на нее.
4. Участник общается с другими пользователями через микросервис "Социальная составляющая".
5. Администратор управляет контентом через микросервис "Администрирование".
6. <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-data-jpa</artifactId>  
   </dependency>  
   <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
   </dependency>  
     
   *<!-- Spring Security -->*<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-security</artifactId>  
   </dependency>  
     
   *<!-- Database -->*<dependency>  
    <groupId>org.postgresql</groupId>  
    <artifactId>postgresql</artifactId>  
    <scope>runtime</scope>  
   </dependency>  
     
   *<!-- Lombok -->*<dependency>  
    <groupId>org.projectlombok</groupId>  
    <artifactId>lombok</artifactId>  
    <version>1.18.30</version>  
    <scope>provided</scope>  
   </dependency>  
     
   *<!-- MapStruct -->*<dependency>  
    <groupId>org.mapstruct</groupId>  
    <artifactId>mapstruct</artifactId>  
    <version>1.5.5.Final</version>  
   </dependency>  
   <dependency>  
    <groupId>org.mapstruct</groupId>  
    <artifactId>mapstruct-processor</artifactId>  
    <version>1.5.5.Final</version>  
    <scope>provided</scope>  
   </dependency>  
     
   *<!-- Spring Boot Configuration Processor -->*<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-configuration-processor</artifactId>  
    <optional>true</optional>  
   </dependency>  
     
   *<!-- Security -->*<dependency>  
    <groupId>org.springframework.security</groupId>  
    <artifactId>spring-security-crypto</artifactId>  
   </dependency>  
     
   <dependency>  
    <groupId>jakarta.servlet</groupId>  
    <artifactId>jakarta.servlet-api</artifactId>  
    <version>5.0.0</version>  
    <scope>provided</scope>  
   </dependency>  
   *<!-- Bouncy Castle -->*<dependency>  
    <groupId>org.bouncycastle</groupId>  
    <artifactId>bcprov-jdk18on</artifactId>  
    <version>1.78.1</version>  
   </dependency>  
     
   *<!-- Validation -->*<dependency>  
    <groupId>jakarta.validation</groupId>  
    <artifactId>jakarta.validation-api</artifactId>  
    <version>3.0.2</version>  
   </dependency>  
   <dependency>  
    <groupId>org.hibernate.validator</groupId>  
    <artifactId>hibernate-validator</artifactId>  
    <version>8.0.1.Final</version>  
   </dependency>  
     
   *<!-- JWT -->*<dependency>  
    <groupId>io.jsonwebtoken</groupId>  
    <artifactId>jjwt-api</artifactId>  
    <version>0.11.5</version>  
   </dependency>  
   <dependency>  
    <groupId>io.jsonwebtoken</groupId>  
    <artifactId>jjwt-impl</artifactId>  
    <version>0.11.5</version>  
    <scope>runtime</scope>  
   </dependency>  
   <dependency>  
    <groupId>io.jsonwebtoken</groupId>  
    <artifactId>jjwt-jackson</artifactId>  
    <version>0.11.5</version>  
    <scope>runtime</scope>  
   </dependency>  
     
   *<!-- Тестирование -->*<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-test</artifactId>  
    <scope>test</scope>  
   </dependency>  
   <dependency>  
    <groupId>org.springframework.security</groupId>  
    <artifactId>spring-security-test</artifactId>  
    <scope>test</scope>  
   </dependency>  
   <dependency>  
    <groupId>joda-time</groupId>  
    <artifactId>joda-time</artifactId>  
    <version>2.12.7</version>  
    <scope>compile</scope>  
   </dependency>