Health & Fitness Microservices Platform

Подробный План Разработки

1. Обзор Архитектуры

1.1 Название Проекта

Health & Fitness Microservices Platform — комплексная система для отслеживания здоровья и физической подготовки с микросервисной архитектурой, ролевым доступом и возможностью интеграции AI для персональных рекомендаций.

1.2 Основные Характеристики

- **Архитектура:** Микросервисы на Spring Boot
- **Коммуникация:** REST (синхронная) + Kafka (асинхронная)
- **БД:** PostgreSQL (одна БД на сервис)
- **Аутентификация:** JWT токены
- **Авторизация:** Role-Based Access Control (RBAC)
- **Контейнеризация:** Docker + Docker Compose
- **Инструменты разработки:** IntelliJ IDEA, Maven

2. Основные Сервисы

2.1 Обязательные Сервисы

1. Auth Service (Сервис Аутентификации и Авторизации)

Порт: 8001
Описание: Управление пользователями, аутентификация, генерация JWT токенов, управление ролями.

Ключевые функции:

- Регистрация пользователей
- Вход (логин/пароль)
- Генерация и валидация JWT токенов
- Управление ролями (ADMIN, USER, DOCTOR, TRAINER)
- Обновление профиля

Сущности:

```
User (id, email, password, firstName, lastName, dateOfBirth, roles, createdAt)
Role (id, name, permissions)
UserRole (userId, roleId)
```

2. Cardio Service (Сервис Кардионагрузок)

Порт: 8002
Описание: Отслеживание кардиотренировок, пробежек, велопрогулок и других аэробных нагрузок.

Ключевые функции:

- Создание/редактирование кардиотренировки
- Записать результаты: дистанция, время, пульс, калории
- Просмотр истории тренировок
- Получить статистику (средний пульс, общее расстояние, сожженные калории)
- Фильтрация по типу активности, дате, длительности

Сущности:

```
CardioWorkout (id, userId, workoutType, distance, duration,
               averageHeartRate, maxHeartRate, caloriesBurned,
               startTime, endTime, location, notes, createdAt)
WorkoutType (id, name, description)
```

3. Strength Service (Сервис Силовых Тренировок)

Порт: 8003
Описание: Отслеживание силовых упражнений, подходов, повторений, используемых весов.

Ключевые функции:

- Создать программу тренировок
- Логировать упражнения: вес, подходы, повторения, интенсивность
- Просмотр истории упражнений
- Отслеживание прогресса по упражнениям
- Получить рекомендуемые веса на основе предыдущих тренировок

Сущности:

```
StrengthWorkout (id, userId, workoutDate, duration, notes)
Exercise (id, workoutId, exerciseName, sets, reps, weight, bodyPart)
```

```
BodyPart (id, name)
```

4. Medical Service (Сервис Медицинских Анализов)

Порт: 8004
Описание: Хранение и управление результатами медицинских анализов и показателей здоровья.

Ключевые функции:

- Добавить результаты анализов (кровь, мочу, биохимию и т.д.)
- Сохранить показатели жизнедеятельности: давление, пульс, вес, температура
- Просмотр истории анализов
- Сравнение результатов с нормами
- Генерация отчетов по здоровью
- Доступ врачей к данным пациентов (через роли)

Сущности:

```
MedicalTest (id, userId, testType, testDate, results)
HealthMetric (id, userId, metricType, value, unit, measurementDate)
MedicalRecord (id, userId, doctorId, diagnosis, prescription, recordDate)
TestType (id, name, normalRange, unit)
MetricType (id, name, normalMin, normalMax, unit)
```

5. Nutrition Service (Сервис Приёмов Пищи)

Порт: 8005
Описание: Логирование приёмов пищи, подсчет калорий и макронутриентов.

Ключевые функции:

- Добавить прием пищи (завтрак, обед, ужин, снеки)
- Поиск блюд по названию
- Добавить собственные рецепты
- Подсчет калорий, белков, жиров, углеводов
- Просмотр дневной статистики питания
- Установить целевые значения калорий
- Отслеживание соответствия нормам

Сущности:

```
Meal (id, userId, mealType, mealDate, totalCalories,
      totalProtein, totalFat, totalCarbs, notes)
```

```
MealItem (id, mealId, foodId, quantity, unit)
Food (id, name, caloriesPer100g, proteinPer100g,
      fatPer100g, carbsPer100g, source)
MealType (id, name)
```

2.2 Рекомендуемые Дополнительные Сервисы

6. AI Recommendation Service (Сервис AI Рекомендаций)

Порт: 8006
Описание: Анализ данных пользователя и предоставление персональных рекомендаций.

Ключевые функции:

- Анализ тренировок и здоровья
- Генерация рекомендаций по физической активности
- Рекомендации по питанию
- Анализ прогресса
- Интеграция с внешними AI моделями (Gemini API, OpenAI)
- Использование RAG для улучшенных рекомендаций

Взаимодействие с другими сервисами:

- Получает данные из Cardio, Strength, Medical, Nutrition сервисов
- Публикует события в Kafka с рекомендациями
- Отправляет уведомления пользователям

7. Notification Service (Сервис Уведомлений)

Порт: 8007
Описание: Отправка уведомлений пользователям (email, push, in-app).

Ключевые функции:

- Отправка email уведомлений
- Push уведомления мобильным приложениям
- In-app уведомления
- Управление параметрами уведомлений
- История уведомлений

8. Analytics Service (Сервис Аналитики)

Порт: 8008
Описание: Агрегирование и анализ данных для формирования статистики и дашбордов.

Ключевые функции:

- Формирование дашбордов пользователя
- Расчет KPI (пройденное расстояние, сожженные калории, пройденные дни)
- Сравнение периодов времени
- Экспорт отчетов
- Общая статистика (рейтинги, достижения)

9. Social Service (Социальный Сервис)

Порт: 8009
Описание: Социальные функции: друзья, челленджи, соревнования.

Ключевые функции:

- Добавление друзей
- Создание и участие в челленджах
- Участие в соревнованиях
- Лидерборды
- Совместные тренировки

10. Appointment Service (Сервис Записи к Врачам)

Порт: 8010
Описание: Запись и управление приемами у врачей, тренеров.

Ключевые функции:

- Поиск специалистов
- Запись на консультацию
- Управление расписанием
- Уведомления о приемах
- История консультаций

3. API Gateway и Service Discovery

3.1 API Gateway (Spring Cloud Gateway)

Порт: 8000
Описание: Единая точка входа для всех запросов.

Функции:

- Маршрутизация запросов к нужным сервисам
- Валидация JWT токенов (предварительная проверка)
- Rate limiting
- Логирование запросов
- CORS обработка

Конфиг маршрутизации:

```
spring:
  cloud:
    gateway:
      routes:
        - id: auth-service
          uri: http://auth-service:8001
          predicates:
            - Path=/api/auth/**
        - id: cardio-service
          uri: http://cardio-service:8002
          predicates:
            - Path=/api/cardio/**
        # ... и т.д.
```

3.2 Service Registry (Eureka Server)

Порт: 8761
Описание: Регистрация и обнаружение сервисов.

Функции:

- Регистрация всех микросервисов
- Health check сервисов
- Самообнаружение сервисов
- Load balancing

4. Коммуникация между Сервисами

4.1 REST Communication (Синхронная)

Используется для:

- Получение данных пользователя из Auth Service
- Синхронные запросы между сервисами
- Запросы, требующие немедленного ответа

Пример: Cardio Service запрашивает данные из Auth Service

```
GET http://auth-service:8001/api/auth/users/{userId}
Header: Authorization: Bearer {JWT_TOKEN}
```

4.2 Kafka Communication (Асинхронная)

Используется для:

- Событийные уведомления между сервисами
- Асинхронная обработка
- Интеграция с AI Service
- Уведомления

Основные Kafka Topics:

Topic Name	Producer	Consumers	Событие
`cardio-workout-created`	Cardio Service	AI, Analytics, Notification	Новая кардиотренировка
`strength-workout-created`	Strength Service	AI, Analytics, Notification	Новая силовая тренировка
`meal-logged`	Nutrition Service	AI, Analytics, Notification	Новый прием пищи
`medical-test-updated`	Medical Service	AI, Notification, Appointment	Новый медицинский анализ
`recommendation-generated`	AI Service	Notification, Analytics	Новые рекомендации
`user-created`	Auth Service	Notification, Analytics	Новый пользователь
`appointment-scheduled`	Appointment Service	Notification	Новый прием
`health-alert`	Medical Service	Notification	Предупреждение о здоровье

Пример Producer (Cardio Service):

```
@Service
@RequiredArgsConstructor
public class CardioEventPublisher {
    private final StreamBridge streamBridge;

    public void publishWorkoutCreated(CardioWorkout workout) {
        CardioWorkoutEvent event = new CardioWorkoutEvent(
            workout.getId(),
            workout.getUserId(),
            workout.getDistance(),
            workout.getCaloriesBurned()
        );
        streamBridge.send("cardio-workout-created", event);
    }
}
```

Пример Consumer (AI Service):

```
@Service
public class CardioEventConsumer {

    @Bean
    public Consumer&lt;CardioWorkoutEvent&gt; processCardioWorkout() {
        return event -&gt; {
            // Получить полные данные пользователя из Auth Service
            // Проанализировать тренировку
            // Сгенерировать рекомендации
            // Опубликовать в topic recommendation-generated
        };
    }
}
```

5. Аутентификация и Авторизация

5.1 JWT Токены

Структура JWT:

```
{
  "sub": "user-id-12345",
  "email": "user@example.com",
  "roles": ["USER", "TRAINER"],
  "iat": 1234567890,
  "exp": 1234571490
}
```

5.2 Роли и Разрешения

Роль	Описание	Разрешения
ADMIN	Администратор системы	Управление пользователями, всеми данными
USER	Обычный пользователь	Просмотр и редактирование своих данных
DOCTOR	Врач	Просмотр данных пациентов, добавление медицинских записей
TRAINER	Тренер	Просмотр данных клиентов, рекомендации
AI_SYSTEM	AI система	Доступ к данным для анализа и рекомендаций

5.3 Защита Эндпойнтов

На уровне Gateway:

```
@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {
        http
            .authorizeExchange()
            .pathMatchers("/api/auth/login", "/api/auth/register").permitAll()
            .pathMatchers("/api/**").authenticated()
            .and()
            .oauth2ResourceServer()
            .jwt();
        return http.build();
    }
}
```

На уровне Микросервисов:

```
@RestController
@RequestMapping("/api/cardio")
public class CardioController {

    @PostMapping("/workouts")
    @PreAuthorize("hasRole('USER') or hasRole('TRAINER')")
    public ResponseEntity&lt;CardioWorkoutDTO&gt; createWorkout(@RequestBody CardioWorkou
        // Логика
    }

    @GetMapping("/workouts/{userId}")
    @PreAuthorize("hasRole('ADMIN') or @securityService.isOwnerOrDoctor(#userId)")
    public ResponseEntity&lt;List&lt;CardioWorkoutDTO&gt;&gt; getUserWorkouts(@PathVariab
        // Логика
    }
}
```

6. Структура Базы Данных

6.1 PostgreSQL Схема

Каждый сервис имеет свою БД:

```
auth-service-db
├── users
├── roles
├── user_roles
└── permissions

cardio-service-db
├── cardio_workouts
├── workout_types
└── cardio_stats

strength-service-db
├── strength_workouts
├── exercises
└── body_parts

medical-service-db
├── medical_tests
├── health_metrics
├── medical_records
├── test_types
└── metric_types

nutrition-service-db
├── meals
├── meal_items
├── food
└── meal_types

ai-service-db (кеш, история)
├── recommendations
├── analysis_history
└── user_preferences

analytics-service-db
├── user_stats
├── daily_aggregates
└── achievements
```

7. Docker Compose для Локальной Разработки

```
version: '3.8'

services:
  # PostgreSQL Databases
  auth-db:
```

```
    image: postgres:15
    environment:
      POSTGRES_DB: auth_service
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"

  cardio-db:
    image: postgres:15
    environment:
      POSTGRES_DB: cardio_service
      POSTGRES_PASSWORD: postgres
    ports:
      - "5433:5432"

  # Kafka
  kafka:
    image: confluentinc/cp-kafka:7.5.0
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    ports:
      - "9092:9092"
    depends_on:
      - zookeeper

  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
    ports:
      - "2181:2181"

  # Eureka Server
  eureka-server:
    build:
      context: ./eureka-server
    ports:
      - "8761:8761"

  # API Gateway
  api-gateway:
    build:
      context: ./api-gateway
    ports:
      - "8000:8000"
    environment:
      EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server:8761/eureka/
    depends_on:
      - eureka-server
      - kafka
```

```
  # Auth Service
  auth-service:
    build:
      context: ./auth-service
    ports:
      - "8001:8001"
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://auth-db:5432/auth_service
      SPRING_DATASOURCE_PASSWORD: postgres
      EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server:8761/eureka/
      KAFKA_BOOTSTRAP_SERVERS: kafka:29092
    depends_on:
      - auth-db
      - eureka-server
      - kafka

  # Cardio Service
  cardio-service:
    build:
      context: ./cardio-service
    ports:
      - "8002:8002"
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://cardio-db:5432/cardio_service
      SPRING_DATASOURCE_PASSWORD: postgres
      EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server:8761/eureka/
      KAFKA_BOOTSTRAP_SERVERS: kafka:29092
      AUTH_SERVICE_URL: http://auth-service:8001
    depends_on:
      - cardio-db
      - eureka-server
      - kafka
      - auth-service

  # ... другие сервисы по аналогии
```

8. Project Structure (Структура Проекта)

```
health-fitness-platform/
├── pom.xml (parent)
├── eureka-server/
│   ├── pom.xml
│   ├── src/main/java/com/healthfitness/eureka/
│   │   └── EurekaServerApplication.java
│   └── src/main/resources/application.yml
├── api-gateway/
│   ├── pom.xml
│   ├── src/main/java/com/healthfitness/gateway/
│   │   ├── ApiGatewayApplication.java
│   │   └── config/SecurityConfig.java
│   └── src/main/resources/application.yml
├── auth-service/
│   ├── pom.xml
│   ├── src/main/java/com/healthfitness/auth/
```

```
│   │   ├── AuthServiceApplication.java
│   │   ├── controller/AuthController.java
│   │   ├── service/
│   │   │   ├── AuthService.java
│   │   │   ├── UserService.java
│   │   │   └── JwtTokenProvider.java
│   │   ├── entity/
│   │   │   ├── User.java
│   │   │   ├── Role.java
│   │   │   └── UserRole.java
│   │   ├── dto/
│   │   │   ├── LoginRequest.java
│   │   │   ├── RegisterRequest.java
│   │   │   ├── JwtResponse.java
│   │   │   └── UserDTO.java
│   │   ├── repository/
│   │   │   ├── UserRepository.java
│   │   │   └── RoleRepository.java
│   │   └── security/
│   │       ├── CustomUserDetails.java
│   │       └── JwtAuthenticationFilter.java
│   ├── src/main/resources/
│   │   ├── application.yml
│   │   └── db/migration/ (Flyway миграции)
│   └── src/test/
├── cardio-service/
│   ├── pom.xml
│   ├── src/main/java/com/healthfitness/cardio/
│   │   ├── CardioServiceApplication.java
│   │   ├── controller/CardioController.java
│   │   ├── service/
│   │   │   ├── CardioService.java
│   │   │   └── CardioEventPublisher.java
│   │   ├── entity/CardioWorkout.java
│   │   ├── dto/CardioWorkoutDTO.java
│   │   ├── repository/CardioWorkoutRepository.java
│   │   ├── event/CardioWorkoutEvent.java
│   │   └── client/AuthServiceClient.java
│   └── src/main/resources/application.yml
├── strength-service/
│   ├── pom.xml
│   └── src/main/java/com/healthfitness/strength/
│       ├── StrengthServiceApplication.java
│       ├── controller/StrengthController.java
│       ├── service/StrengthService.java
│       ├── entity/
│       │   ├── StrengthWorkout.java
│       │   └── Exercise.java
│       └── dto/ExerciseDTO.java
├── medical-service/
│   └── ...
├── nutrition-service/
│   └── ...
├── ai-service/
│   ├── pom.xml
│   └── src/main/java/com/healthfitness/ai/
```

```
│       ├── AiServiceApplication.java
│       ├── consumer/
│       │   ├── CardioEventConsumer.java
│       │   ├── MealEventConsumer.java
│       │   └── HealthMetricEventConsumer.java
│       ├── service/
│       │   ├── RecommendationService.java
│       │   ├── GeminiAiService.java
│       │   └── DataAggregationService.java
│       ├── entity/Recommendation.java
│       └── dto/RecommendationDTO.java
├── notification-service/
│   └── ...
├── docker-compose.yml
└── README.md
```

9. Основные Зависимости (pom.xml)

```
&lt;parent&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
    &lt;artifactId&gt;spring-boot-starter-parent&lt;/artifactId&gt;
    &lt;version&gt;3.1.5&lt;/version&gt;
&lt;/parent&gt;

&lt;properties&gt;
    &lt;java.version&gt;17&lt;/java.version&gt;
    &lt;spring-cloud.version&gt;2022.0.4&lt;/spring-cloud.version&gt;
&lt;/properties&gt;

&lt;dependencies&gt;

    &lt;dependency&gt;
        &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
        &lt;artifactId&gt;spring-boot-starter-web&lt;/artifactId&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
        &lt;artifactId&gt;spring-boot-starter-data-jpa&lt;/artifactId&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.postgresql&lt;/groupId&gt;
        &lt;artifactId&gt;postgresql&lt;/artifactId&gt;
        &lt;version&gt;42.6.0&lt;/version&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.projectlombok&lt;/groupId&gt;
        &lt;artifactId&gt;lombok&lt;/artifactId&gt;
        &lt;optional&gt;true&lt;/optional&gt;
```

```
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
    &lt;artifactId&gt;spring-boot-starter-security&lt;/artifactId&gt;
&lt;/dependency&gt;
&lt;dependency&gt;
    &lt;groupId&gt;io.jsonwebtoken&lt;/groupId&gt;
    &lt;artifactId&gt;jjwt-api&lt;/artifactId&gt;
    &lt;version&gt;0.12.3&lt;/version&gt;
&lt;/dependency&gt;
&lt;dependency&gt;
    &lt;groupId&gt;io.jsonwebtoken&lt;/groupId&gt;
    &lt;artifactId&gt;jjwt-impl&lt;/artifactId&gt;
    &lt;version&gt;0.12.3&lt;/version&gt;
&lt;/dependency&gt;
&lt;dependency&gt;
    &lt;groupId&gt;io.jsonwebtoken&lt;/groupId&gt;
    &lt;artifactId&gt;jjwt-jackson&lt;/artifactId&gt;
    &lt;version&gt;0.12.3&lt;/version&gt;
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.cloud&lt;/groupId&gt;
    &lt;artifactId&gt;spring-cloud-starter-netflix-eureka-server&lt;/artifactId&gt;
&lt;/dependency&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.cloud&lt;/groupId&gt;
    &lt;artifactId&gt;spring-cloud-starter-netflix-eureka-client&lt;/artifactId&gt;
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.cloud&lt;/groupId&gt;
    &lt;artifactId&gt;spring-cloud-starter-gateway&lt;/artifactId&gt;
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.kafka&lt;/groupId&gt;
    &lt;artifactId&gt;spring-kafka&lt;/artifactId&gt;
&lt;/dependency&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.cloud&lt;/groupId&gt;
    &lt;artifactId&gt;spring-cloud-stream-binder-kafka&lt;/artifactId&gt;
&lt;/dependency&gt;


&lt;dependency&gt;
    &lt;groupId&gt;io.micrometer&lt;/groupId&gt;
    &lt;artifactId&gt;micrometer-registry-prometheus&lt;/artifactId&gt;
&lt;/dependency&gt;
```

```
    &lt;dependency&gt;
        &lt;groupId&gt;org.springframework.cloud&lt;/groupId&gt;
        &lt;artifactId&gt;spring-cloud-starter-sleuth&lt;/artifactId&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
        &lt;artifactId&gt;spring-boot-starter-validation&lt;/artifactId&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
        &lt;artifactId&gt;spring-boot-starter-webflux&lt;/artifactId&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.modelmapper&lt;/groupId&gt;
        &lt;artifactId&gt;modelmapper&lt;/artifactId&gt;
        &lt;version&gt;3.1.1&lt;/version&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.flywaydb&lt;/groupId&gt;
        &lt;artifactId&gt;flyway-core&lt;/artifactId&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;com.google.ai.client.generativeai&lt;/groupId&gt;
        &lt;artifactId&gt;google-generativeai&lt;/artifactId&gt;
        &lt;version&gt;0.3.0&lt;/version&gt;
    &lt;/dependency&gt;


    &lt;dependency&gt;
        &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
        &lt;artifactId&gt;spring-boot-starter-test&lt;/artifactId&gt;
        &lt;scope&gt;test&lt;/scope&gt;
    &lt;/dependency&gt;
&lt;/dependencies&gt;

&lt;dependencyManagement&gt;
    &lt;dependencies&gt;
        &lt;dependency&gt;
            &lt;groupId&gt;org.springframework.cloud&lt;/groupId&gt;
            &lt;artifactId&gt;spring-cloud-dependencies&lt;/artifactId&gt;
            &lt;version&gt;${spring-cloud.version}&lt;/version&gt;
            &lt;type&gt;pom&lt;/type&gt;
            &lt;scope&gt;import&lt;/scope&gt;
        &lt;/dependency&gt;
    &lt;/dependencies&gt;
&lt;/dependencyManagement&gt;
```

10. Пример Реализации: Auth Service

10.1 Entity (User.java)

```
@Entity
@Table(name = "users")
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String password;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "date_of_birth")
    private LocalDate dateOfBirth;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set&lt;Role&gt; roles = new HashSet&lt;&gt;();

    @Column(name = "created_at")
    private LocalDateTime createdAt;

    @Column(name = "updated_at")
    private LocalDateTime updatedAt;

    @PrePersist
    protected void onCreate() {
        createdAt = LocalDateTime.now();
        updatedAt = LocalDateTime.now();
    }

    @PreUpdate
    protected void onUpdate() {
        updatedAt = LocalDateTime.now();
```

```
    }
}
```

10.2 DTO (LoginRequest, JwtResponse)

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class LoginRequest {
    @NotBlank(message = "Email is required")
    @Email
    private String email;

    @NotBlank(message = "Password is required")
    private String password;
}

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class JwtResponse {
    private String token;
    private Long userId;
    private String email;
    private List&lt;String&gt; roles;
    private LocalDateTime expiresAt;
}

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UserDTO {
    private Long id;
    private String email;
    private String firstName;
    private String lastName;
    private List&lt;String&gt; roles;
}
```

10.3 Service (AuthService.java)

```
@Service
@RequiredArgsConstructor
@Slf4j
public class AuthService {
    private final UserRepository userRepository;
    private final RoleRepository roleRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtTokenProvider jwtTokenProvider;
```

```
    private final StreamBridge streamBridge;

    public JwtResponse login(LoginRequest loginRequest) {
        User user = userRepository.findByEmail(loginRequest.getEmail())
            .orElseThrow(() -&gt; new UsernameNotFoundException("User not found"));

        if (!passwordEncoder.matches(loginRequest.getPassword(), user.getPassword())) {
            throw new BadCredentialsException("Invalid password");
        }

        String token = jwtTokenProvider.generateToken(user);
        List&lt;String&gt; roles = user.getRoles().stream()
            .map(Role::getName)
            .collect(Collectors.toList());

        return JwtResponse.builder()
            .token(token)
            .userId(user.getId())
            .email(user.getEmail())
            .roles(roles)
            .expiresAt(jwtTokenProvider.getExpirationDate(token))
            .build();
    }

    public UserDTO register(RegisterRequest registerRequest) {
        if (userRepository.existsByEmail(registerRequest.getEmail())) {
            throw new IllegalArgumentException("Email already exists");
        }

        User user = User.builder()
            .email(registerRequest.getEmail())
            .password(passwordEncoder.encode(registerRequest.getPassword()))
            .firstName(registerRequest.getFirstName())
            .lastName(registerRequest.getLastName())
            .dateOfBirth(registerRequest.getDateOfBirth())
            .build();

        // Добавить роль USER по умолчанию
        Role userRole = roleRepository.findByName("USER")
            .orElseThrow(() -&gt; new IllegalArgumentException("Default role not found"));
        user.getRoles().add(userRole);

        User savedUser = userRepository.save(user);

        // Опубликовать событие в Kafka
        streamBridge.send("user-created", new UserCreatedEvent(
            savedUser.getId(),
            savedUser.getEmail(),
            savedUser.getFirstName(),
            savedUser.getLastName()
        ));

        return convertToDTO(savedUser);
    }

    private UserDTO convertToDTO(User user) {
```

```
        List&lt;String&gt; roles = user.getRoles().stream()
            .map(Role::getName)
            .collect(Collectors.toList());

        return UserDTO.builder()
            .id(user.getId())
            .email(user.getEmail())
            .firstName(user.getFirstName())
            .lastName(user.getLastName())
            .roles(roles)
            .build();
    }
}
```

10.4 Controller (AuthController.java)

```
@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
@Slf4j
@CrossOrigin(origins = "*")
public class AuthController {
    private final AuthService authService;

    @PostMapping("/login")
    public ResponseEntity&lt;JwtResponse&gt; login(@Valid @RequestBody LoginRequest login
        JwtResponse response = authService.login(loginRequest);
        return ResponseEntity.ok(response);
    }

    @PostMapping("/register")
    public ResponseEntity&lt;UserDTO&gt; register(@Valid @RequestBody RegisterRequest reg
        UserDTO user = authService.register(registerRequest);
        return ResponseEntity.status(HttpStatus.CREATED).body(user);
    }

    @GetMapping("/users/{userId}")
    @PreAuthorize("hasRole('ADMIN') or #userId == authentication.principal.id")
    public ResponseEntity&lt;UserDTO&gt; getUserById(@PathVariable Long userId) {
        UserDTO user = authService.getUserById(userId);
        return ResponseEntity.ok(user);
    }
}
```

11. Интеграция AI для Рекомендаций

11.1 Использование Gemini API

```
@Service
@RequiredArgsConstructor
@Slf4j
public class GeminiAiService {

    @Value("${gemini.api.key}")
    private String apiKey;

    private final GenerativeModel model;
    private final RestTemplate restTemplate;

    @PostConstruct
    public void init() {
        // Инициализация Gemini API
    }

    public RecommendationDTO generateFitnessRecommendation(UserHealthProfile profile) {
        String prompt = buildPrompt(profile);

        try {
            // Вызов Gemini API
            GenerateContentResponse response = model.generateContent(prompt);
            String recommendation = response.getText();

            return RecommendationDTO.builder()
                .userId(profile.getUserId())
                .type(RecommendationType.FITNESS)
                .content(recommendation)
                .generatedAt(LocalDateTime.now())
                .build();

        } catch (Exception e) {
            log.error("Error calling Gemini API", e);
            throw new RuntimeException("Failed to generate recommendation", e);
        }
    }

    private String buildPrompt(UserHealthProfile profile) {
        return String.format("""
            Based on the following health profile, provide personalized fitness recommend

            Recent workouts: %s
            Average weekly exercise: %d hours
            Health metrics: %s
            Nutrition logs: %s

            Provide 3-5 specific, actionable recommendations to improve fitness and healt
            """,
            profile.getRecentWorkouts(),
            profile.getWeeklyExerciseHours(),
            profile.getHealthMetrics(),
            profile.getNutritionLogs()
        );
```

```
    }
}
```

11.2 Event Consumer для AI

```
@Service
@RequiredArgsConstructor
@Slf4j
public class HealthEventConsumer {

    private final GeminiAiService geminiAiService;
    private final RecommendationService recommendationService;
    private final StreamBridge streamBridge;
    private final DataAggregationService dataAggregationService;

    @Bean
    public Consumer&lt;CardioWorkoutEvent&gt; processCardioWorkout() {
        return event -&gt; {
            try {
                log.info("Processing cardio workout event: {}", event.getWorkoutId());

                UserHealthProfile profile = dataAggregationService.getUserProfile(event.g
                RecommendationDTO recommendation = geminiAiService.generateFitnessRecomme

                recommendationService.save(recommendation);

                // Опубликовать новое событие
                streamBridge.send("recommendation-generated", recommendation);

            } catch (Exception e) {
                log.error("Error processing cardio workout event", e);
            }
        };
    }

    @Bean
    public Consumer&lt;MealEvent&gt; processMealLogged() {
        return event -&gt; {
            try {
                log.info("Processing meal event: {}", event.getMealId());

                UserHealthProfile profile = dataAggregationService.getUserProfile(event.g
                RecommendationDTO nutritionRec = geminiAiService.generateNutritionRecomme

                recommendationService.save(nutritionRec);
                streamBridge.send("recommendation-generated", nutritionRec);

            } catch (Exception e) {
                log.error("Error processing meal event", e);
            }
        };
    }

    @Bean
    public Consumer&lt;HealthMetricEvent&gt; processHealthMetric() {
```

```
        return event -&gt; {
            try {
                log.info("Processing health metric event: {}", event.getMetricType());

                // Проверить, если показатель вне нормы
                if (!event.isWithinNormalRange()) {
                    streamBridge.send("health-alert", new HealthAlertEvent(
                        event.getUserId(),
                        event.getMetricType(),
                        "Ваш показатель " + event.getMetricType() + " вне нормы",
                        LocalDateTime.now()
                    ));
                }

            } catch (Exception e) {
                log.error("Error processing health metric event", e);
            }
        };
    }
}
```

12. Monitoring и Logging

12.1 Prometheus Metrics

```
management:
  endpoints:
    web:
      exposure:
        include: health,metrics,prometheus
  metrics:
    distribution:
      percentiles-histogram:
        http:
          server:
            requests: true
```

12.2 Centralized Logging (ELK Stack)

```
logging:
  level:
    root: INFO
    com.healthfitness: DEBUG
  pattern:
    console: "%d{yyyy-MM-dd HH:mm:ss} - %logger{36} - %msg%n"
    file: "%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n"
  file:
    name: logs/app.log
    max-size: 10MB
    max-history: 10
```

13. Фазы Разработки

Фаза 1: Инфраструктура (Неделя 1-2)

- [ ] Setup Eureka Server
- [ ] Setup API Gateway
- [ ] Setup Docker Compose
- [ ] Setup Kafka

Фаза 2: Core Services (Неделя 3-4)

- [ ] Auth Service
- [ ] Cardio Service
- [ ] Strength Service

Фаза 3: Data Services (Неделя 5-6)

- [ ] Medical Service
- [ ] Nutrition Service
- [ ] Интеграция REST между сервисами

Фаза 4: Event-Driven (Неделя 7-8)

- [ ] Kafka Topic Configuration
- [ ] Event Publishers & Consumers
- [ ] Notification Service
- [ ] Analytics Service

Фаза 5: AI Integration (Неделя 9-10)

- [ ] Gemini API Integration
- [ ] AI Recommendation Service
- [ ] Data Aggregation Service
- [ ] Тестирование AI рекомендаций

Фаза 6: Дополнительно (Неделя 11+)

- [ ] Social Service
- [ ] Appointment Service
- [ ] Mobile App Backend
- [ ] Performance Optimization

14. Тестирование

14.1 Unit Tests

```
@SpringBootTest
class AuthServiceTests {

    @MockBean
    private UserRepository userRepository;

    @Autowired
    private AuthService authService;

    @Test
    void testLoginSuccess() {
        // Arrange
        User user = User.builder().email("test@test.com").build();
        LoginRequest request = new LoginRequest("test@test.com", "password");

        // Act &amp;amp; Assert
        // Написать тест
    }
}
```

14.2 Integration Tests

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class AuthControllerIntegrationTests {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void testRegister() {
        // Test registration flow
    }
}
```

15. Развертывание (Deployment)

15.1 Kubernetes (Production)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-service
```

```
spec:
  replicas: 3
  selector:
    matchLabels:
      app: auth-service
  template:
    metadata:
      labels:
        app: auth-service
    spec:
      containers:
      - name: auth-service
        image: health-fitness/auth-service:v1.0
        ports:
        - containerPort: 8001
        env:
        - name: SPRING_DATASOURCE_URL
          valueFrom:
            secretKeyRef:
              name: db-secrets
              key: auth-db-url
```

15.2 CI/CD (GitHub Actions)

```
name: Build and Deploy

on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up JDK 17
      uses: actions/setup-java@v2
      with:
        java-version: '17'

    - name: Build with Maven
      run: mvn clean package

    - name: Build Docker image
      run: docker build -t health-fitness/auth-service .

    - name: Push to Docker Hub
      run: docker push health-fitness/auth-service
```

16. Best Practices

✅ **Следуй Single Responsibility Principle**

- Каждый сервис отвечает за одну область

✅ **Используй DTO для коммуникации**

- Не передавай Entity напрямую через REST/Kafka

✅ **Валидация на всех уровнях**

- Controller, Service, Database constraints

✅ **Обработка ошибок**

- Custom Exception classes
- Global Exception Handler

✅ **Logging**

- Логируй все важные операции
- Используй proper log levels

✅ **Security First**

- Валидируй JWT на Gateway и в сервисах
- Используй HTTPS в production

✅ **Database Migrations**

- Flyway для контроля версий БД

✅ **Testing**

- Unit tests для service layer
- Integration tests для контроллеров
- Contract tests для Kafka events

17. Расширение для AI Recommendations

Потенциальные Интеграции:

1. **Gemini AI** - Основные рекомендации
2. **RAG (Retrieval-Augmented Generation)** - Персональные рекомендации на основе истории
3. **Machine Learning модели** - Прогнозирование прогресса
4. **Natural Language Processing** - Анализ отзывов пользователей

Data Pipeline для ML:

```
User Data → Aggregation Service → Feature Engineering →
ML Model → Recommendations → Notification Service
```

Заключение

Эта архитектура обеспечивает:

- ✅ Масштабируемость
- ✅ Высокую доступность
- ✅ Декоупленность сервисов
- ✅ Гибкость для расширений
- ✅ Готовность к AI интеграции
- ✅ Production-ready security
- ✅ Асинхронная обработка через Kafka
- ✅ Простота локальной разработки через Docker Compose

Начни с базовых сервисов (Auth, Cardio, Strength), потом добавляй остальные постепенно. Каждый сервис можно разрабатывать и деплоить независимо!