Gym membership app

Summary

This project is a gym management system that handles the daily operations of a fitness center. The main purpose is to manage members, their subscriptions, personal training sessions, group classes, and attendance tracking**.**

It's implemented as a REST API using Spring Boot framework with MySQL for data storage.

Business requirements

The system should fulfill the following 10 requirements:

1. Member registration: Members can register with their name, email, phone number and date of birth. The email must be unique to avoid duplicates.
2. Membership plans: The gym offers different plans (Basic, Premium, VIP). Each plan has its own price and duration. Some plans include personal training, others don't.
3. Personal training: Members with active subscriptions can book sessions with a trainer. Cancellations must be made at least 24 hours in advance.
4. Group classes: The gym offers group classes such as yoga, spinning, and crossfit. Members can enroll but each class has a capacity limit.
5. Attendance tracking: Members check in when they arrive and check out when they leave. The system only allows check-in for members with active subscriptions.
6. Trainer management: Trainers have profiles with their specialization (weightlifting, cardio, etc.) and hourly rate. They can view their scheduled sessions.
7. Subscription types: Each membership plan has different features and pricing. The system handles all plan variations.

8. Class capacity: Group classes have a maximum number of spots. Members cannot enroll in full classes.
9. Cancellation rules: Bookings (training sessions or class enrollments) can only be cancelled at least 24 hours before the scheduled time.
10. Membership validation: The system validates membership status before allowing access to any service (check-in, bookings, enrollments).

Features

The app focuses on 5 main features:

Feature 1 - Member management

This is the core functionality of the system. It allows adding new members, updating their information, searching for them, and removing them when needed. Deletion uses soft delete, meaning members are marked as inactive rather than removed from the database.

Endpoints:

- Create a new member
- Get a member by their ID
- Get all members (with pagination)
- Search for members by name or email
- Update member information
- Delete (deactivate) a member

Feature 2 - Subscriptions and plans

This feature handles membership plans and subscriptions. Different plans can be created with various prices and features, and members can subscribe to them.

Functionalities:

- Create, read, update and delete membership plans

- Subscribe a member to a plan (automatically calculates end date based on plan duration)
- View active and expiring subscriptions
- Cancel or renew subscriptions

Feature 3 - Trainers and training sessions

This handles personal training. Trainers have profiles and members can book sessions with them.

The system:

- Manages trainer profiles (add, edit, view, delete)
- Allows members to book training sessions
- Checks for scheduling conflicts to prevent double-booking
- Displays trainer schedules
- Allows canceling or marking sessions as complete

Feature 4 - Group classes

Handles group fitness classes like yoga or spinning. The gym can create classes and members can enroll.

Functionalities:

- Create and manage group classes
- Enroll members in classes
- Validate capacity before enrollment
- Cancel enrollments (with 24h rule)
- View class schedules and enrollment lists

Feature 5 - Attendance

Tracks when members enter and leave the gym. Useful for monitoring gym usage.

Functionalities:

- Check in members (validates membership first)
- Check out members and calculate visit duration
- View attendance history per member
- Generate reports for specific date ranges

Technical details

Database entities

The application uses 9 entities:

1. User - Base class that Member and Trainer extend from. Contains common fields such as name, email, and phone.
2. Member - Extends User, adds date of birth. Has relationships to subscriptions, attendance records, training sessions and class enrollments.
3. Trainer - Extends User, adds bio, specialization and hourly rate. Connected to training sessions.
4. MembershipPlan - Defines available plans (name, price, duration, included features).
5. Subscription - Links a member to a plan. Contains start date, end date and status.
6. TrainingSession - A booking between a member and trainer. Contains date/time, duration and status.
7. GymClass - A group class definition. Contains name, instructor, capacity, schedule and type.
8. ClassEnrollment - Join table linking members to classes (many-to-many relationship).
9. Attendance - Records check-in and check-out times for members..

Relationships between entities

The following relationships are implemented:

- User to Member/Trainer: inheritance (JOINED strategy)
- Member to Subscription: one-to-one (each member has one active subscription)
- MembershipPlan to Subscription: one-to-many (one plan can have many subscriptions)
- Member to Attendance: one-to-many (one member, many attendance records)
- Trainer to TrainingSession: one-to-many

- Member to GymClass: many-to-many (through ClassEnrollment)

Validation

I used Jakarta validation annotations on the DTOs:

- @NotBlank for required text fields
- @Size for length limits
- @Email for email format
- @Pattern for phone numbers
- @Past for date of birth (has to be in the past obviously)
- @DecimalMin and @Digits for prices and rates

API Documentation

The API is documented with Swagger/OpenAPI. When you run the app you can go to:

- Swagger UI: http://localhost:8080/swagger-ui.html
- OpenAPI JSON: http://localhost:8080/api-docs

Every endpoint has descriptions and shows what responses you can expect.

How to run it

What we need

- Java 25
- Maven 3.9 or newer
- MySQL 8.0 or newer

Steps

1. Create a MySQL database: **CREATE DATABASE gymapp;**

2. Update the database credentials in src/main/resources/application.properties if yours are different from root/root
3. Run the app: **mvn spring-boot:run**
4. Open http://localhost:8080/swagger-ui.html to see the API docs

Testing

Tests use H2 in-memory database (MySQL not required):

mvn test

A Postman collection is available in the postman folder for manual testing.

Project structure

src/main/java/com/gymapp/backend/

```
├── config/        - Configuration (Swagger/OpenAPI)
├── controllers/   - REST controllers (handle HTTP requests)
├── dtos/                  - Data transfer objects (requests and responses)
├── entities/      - JPA entities (database tables)
├── enums/         - Enums for status types
├── exceptions/    - Custom exceptions and error handling
├── mappers/       - MapStruct mappers (convert between entities and DTOs)
├── repositories/  - JPA repositories (database access)
└── services/      - Business logic
```

What I learned

Building this project helped me understand how Spring Boot applications are structured. I learned about JPA relationships and how to configure them properly, input validation using annotations, and API documentation with Swagger.

The most challenging part was configuring the entity relationships correctly, especially the cascade operations and the many-to-many relationship for class enrollments.