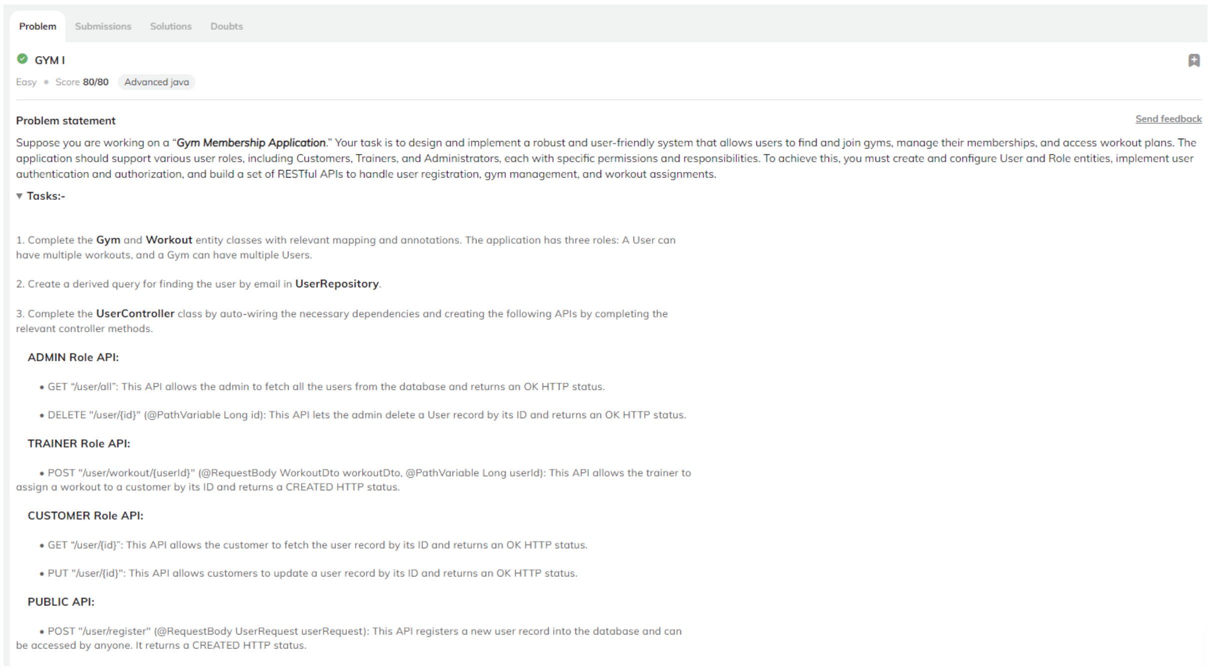
GYM APPLICATION I

Problem Statement:



Problem   Submissions   Solutions   Doubts

GYM I

Easy • Score 80/80   Advanced java

Problem statement

Send feedback

Suppose you are working on a "**Gym Membership Application**." Your task is to design and implement a robust and user-friendly system that allows users to find and join gyms, manage their memberships, and access workout plans. The application should support various user roles, including Customers, Trainers, and Administrators, each with specific permissions and responsibilities. To achieve this, you must create and configure User and Role entities, implement user authentication and authorization, and build a set of RESTful APIs to handle user registration, gym management, and workout assignments.

▾ Tasks:-

1. Complete the **Gym** and **Workout** entity classes with relevant mapping and annotations. The application has three roles: A User can have multiple workouts, and a Gym can have multiple Users.

2. Create a derived query for finding the user by email in **UserRepository**.

3. Complete the **UserController** class by auto-wiring the necessary dependencies and creating the following APIs by completing the relevant controller methods.

ADMIN Role API:

- GET "/user/all": This API allows the admin to fetch all the users from the database and returns an OK HTTP status.
- DELETE "/user/{id}" (@PathVariable Long id): This API lets the admin delete a User record by its ID and returns an OK HTTP status.

TRAINER Role API:

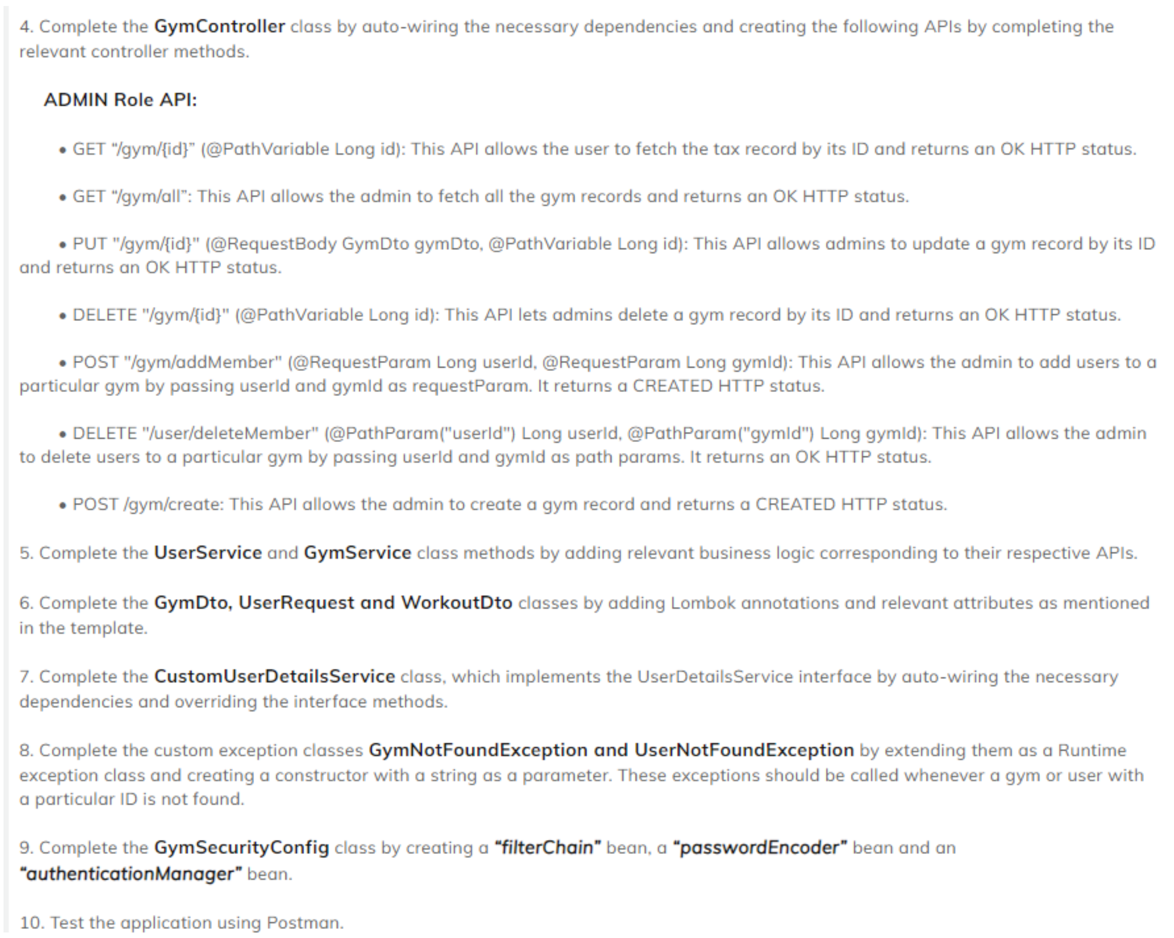
- POST "/user/workout/{userId}" (@RequestBody WorkoutDto workoutDto, @PathVariable Long userId): This API allows the trainer to assign a workout to a customer by its ID and returns a CREATED HTTP status.

CUSTOMER Role API:

- GET "/user/{id}": This API allows the customer to fetch the user record by its ID and returns an OK HTTP status.
- PUT "/user/{id}": This API allows customers to update a user record by its ID and returns an OK HTTP status.

PUBLIC API:

- POST "/user/register" (@RequestBody UserRequest userRequest): This API registers a new user record into the database and can be accessed by anyone. It returns a CREATED HTTP status.



4. Complete the **GymController** class by auto-wiring the necessary dependencies and creating the following APIs by completing the relevant controller methods.

ADMIN Role API:

- GET "/gym/{id}" (@PathVariable Long id): This API allows the user to fetch the tax record by its ID and returns an OK HTTP status.
- GET "/gym/all": This API allows the admin to fetch all the gym records and returns an OK HTTP status.
- PUT "/gym/{id}" (@RequestBody GymDto gymDto, @PathVariable Long id): This API allows admins to update a gym record by its ID and returns an OK HTTP status.
- DELETE "/gym/{id}" (@PathVariable Long id): This API lets admins delete a gym record by its ID and returns an OK HTTP status.
- POST "/gym/addMember" (@RequestParam Long userId, @RequestParam Long gymId): This API allows the admin to add users to a particular gym by passing userId and gymId as requestParam. It returns a CREATED HTTP status.
- DELETE "/user/deleteMember" (@PathParam("userId") Long userId, @PathParam("gymId") Long gymId): This API allows the admin to delete users to a particular gym by passing userId and gymId as path params. It returns an OK HTTP status.
- POST /gym/create: This API allows the admin to create a gym record and returns a CREATED HTTP status.

5. Complete the **UserService** and **GymService** class methods by adding relevant business logic corresponding to their respective APIs.

6. Complete the **GymDto, UserRequest and WorkoutDto** classes by adding Lombok annotations and relevant attributes as mentioned in the template.

7. Complete the **CustomUserDetailsService** class, which implements the UserDetailsService interface by auto-wiring the necessary dependencies and overriding the interface methods.

8. Complete the custom exception classes **GymNotFoundException and UserNotFoundException** by extending them as a Runtime exception class and creating a constructor with a string as a parameter. These exceptions should be called whenever a gym or user with a particular ID is not found.

9. Complete the **GymSecurityConfig** class by creating a ***"filterChain"*** bean, a ***"passwordEncoder"*** bean and an ***"authenticationManager"*** bean.

10. Test the application using Postman.