Gym App Documentation

17th April 2025
Final Sprint

Khoa Pham

User Documentation

Introduction

The **Gym Management System** is a Java-based console application designed to help gyms run their operations smoothly. This system allows **Admins**, **Trainers**, and **Members** to interact with the application based on their roles using a role-based login system. Built with **Java**, managed using **Maven**, and connected to a **PostgreSQL** database, this project offers secure, persistent, and user-specific features for everyday gym management.

Users start by registering a new account with basic details like username, password, email, phone number, address, and role. Passwords are hashed using **BCrypt** before being stored, which helps keep user credentials safe.

After logging in, each user is shown a **menu specific to their role**:

- **Admins** manage users, view memberships, and track revenue.
- **Trainers** handle class creation and management.
- **Members** can browse and sign up for classes, as well as purchase memberships.

This documentation is divided into three sections:

- **User Guide** – for people using the system.
- **Developer Guide** – for those building or maintaining the code.
- **Personal Report** – where each team member shares their work and challenges.

What the System Does

The Gym Management System lets users:

- Create accounts and login securely
- View personalized menu options based on their role

- Manage gym memberships and workout classes
- Interact with PostgreSQL-backed data that's saved even when the system is shut down

Each user type has limited permissions:

- Admins can do high-level operations like seeing all users and revenue
- Trainers can create, edit, or delete workout classes
- Members can see available classes, sign up, and buy memberships

System Roles & Class Responsibilities

User (Base Class)

Purpose: Acts as the foundation for all users (Admin, Trainer, Member).
Attributes: ID, username, hashed password, email, phone, address, role.
Managed by: UserDAO, UserService
Inheritance: Admin, Trainer, and Member extend this class.

Admin

Role: Full control over the system.
Abilities:

- View all users with their contact info
- Delete users
- View all memberships and track total revenue

Trainer

Role: Manage gym workout classes.
Abilities:

- Create, edit, and remove workout classes
- See their assigned classes

- Purchase memberships for themselves

Member
Role: Regular gym user.
Abilities:

- View available workout classes
- Track their membership costs
- Buy a membership

Membership
Purpose: Represents a gym membership purchased by a user.
Attributes: ID, type, description, cost, available credits, start/end date, member ID.
Managed by: MembershipDAO & MembershipService
WorkoutClass
Purpose: Represents a fitness class at the gym.
Attributes: ID, class name/type, description, trainer ID.
Managed by: WorkoutClassDAO & WorkoutClassService

Data Access Layer (DAOs)

UserDAO
Handles database operations related to users:

- Create, read, update, and delete users
- Search users by ID or email
- Retrieve all users
- Converts database result sets into User objects

MembershipDAO
Responsible for:

- Managing memberships in the database
- Viewing revenue

- Fetching membership details for specific users

WorkoutClassDAO

Handles all of the workout classes data:

- Assigning trainers to classes
- Listing classes for users
- Adding or removing members from classes

Service Layer (Business Logic)

UserService

Takes care of user operations:

- Registration
- Login
- Role validation
- User management

MembershipService

Handles memberships:

- Buying or updating memberships
- Viewing personal memberships
- Getting total gym revenue

WorkoutClassService

Manages class-related logic:

- Adding or updating classes
- Listing class info
- Removing trainers
- Viewing class rosters

Starting & Using the System

Prerequisites

- Java JDK installed (version 21+)

- PostgreSQL installed and running
- Maven installed (for building the project)
- PostgreSQL database named gym_management_system

Setting Up the Database

- Open pgAdmin or psql CLI
- Run:

  CREATE DATABASE gym_management_system;
- Create the necessary tables: users, memberships, workout_classes
- Fill out the db.properties file with your credentials (username, password, DB name)

Build the Project

From the terminal or your IDE:

    mvn clean install

This will download needed dependencies, compile the code, and package the app.

Running the Application

- After compiling, run App.java
- You will create the user and apply the user name and password in order to log in.
- Based on your role, you will get the information:
  - Admins: manage users, see revenue
  - Trainers: manage classes, purchase membership
  - Members: view classes, expenses, and purchase memberships
- Change the user name and password according to your role that you want to perform in order to see other information

Conclusion

This Gym Management System simplifies operations for gym administrators. From handling new member registrations to keeping track of memberships and workout routines, the system is designed to be user-friendly, efficient, and expandable for future features.

Development Documentation

Introduction

This development documentation explains the technical design and structure of the Gym Management System, including how to run, compile, and extend the project. It also covers the OOP class structure and entity relationships.

Javadoc

All major methods and classes include Javadoc comments, explaining:

- What the method does
- What parameters it uses
- What it returns

Javadoc helps other developers understand and modify the code easily.

Directory Structure

```
_Final_Sprint_Java
    _.idea
        +.gitignore
        +misc.xml
    _src
        _main
```

```
        _java\org\keyin (contains all java files)
            _databases
                +DatabaseConnection.java
            _memberships
                +Membership.java
                +MembershipDAO.java
                +MembershipService
            _user
                _childclasses
                    +Admin.java
                    +Member.java
                    +Trainer.java
                +User.java
                +UserDao.java
                +UserService.java
            _workoutclasses
                +WorkoutClass.java
                +WorkoutClassDAO.java
                +WorkoutClassService.java
            +GymApp.java
            +GymAppUtils.java
        _resources
            +scripts.sql
    _test\java
        +UserTest.java
        +MembershipTest.java
        +WorkoutClassTest.java
_docs (contains documentation files)
    +GymApp.pdf
_video (contains video)
    +Video
_.gitignore
_pom.xml
```

Build & Run Process

1/Edit db.properties with:
db.url=jdbc:postgresql://localhost:5432/(your data base name)
db.user=your_username
db.password=your_password
2/Install dependencies and compile:
mvn clean install
3/Run the main app

- In terminal: java -cp target/classes com.gym.App
- Or directly from IDE (e.g., right-click GymApp.java > Run)

Required Tools / Dependencies

- Java Development Kit (JDK 11+)
- Git (for version control and GitHub submission)
- A Java IDE or text editor (e.g., VS Code, IntelliJ)

Cloning & Running from GitHub

1/Clone the repo:
git clone https://github.com/KhoaPham-2002/Final_Sprint_Java/blob/main/src/main/java/org/key in/database/DatabaseConnection.java
2/Navigate into the folder:
Cd Final_Sprint_Java
3/Run the system
Mvn clean install
Make sure your PostgreSQL server is running when you launch the app.

Conclusion

This Gym Management System is structured using clean OOP principles, with well-separated responsibilities across classes. The code is readable, extendable, and easy to maintain. The documentation and class design support easy contribution and debugging by any developer familiar with Java.

Individual Report

Introduction

The Gym Management System is a Java-based application designed to streamline the management of a gym's members, trainers, workout plans, and memberships. The system was built using Object-Oriented Programming (OOP) principles, ensuring that the code is modular, maintainable, and easy to extend. The system allows gym owners and managers to track members, assign trainers, create personalized workout plans, manage memberships, and generate reports. The primary goal of the system is to provide a solution to efficiently manage gym operations, particularly with respect to managing memberships and workout plans.

Problem Statement

Running a gym involves handling a variety of tasks, including tracking memberships, assigning workout plans, and ensuring trainers are appropriately assigned to members. Traditionally, these tasks are handled manually, which can be inefficient and prone to error. The Gym

Management System addresses these issues by automating the management of gym members, trainers, workout plans, and memberships.

- The system helps gym managers easily manage members and their workout plans.
- It ensures that trainers are assigned to the correct members, and membership expiry is properly tracked.
- By automating these processes, the system reduces human error and improves overall efficiency.

System Design and Architecture

Contributions

As a solo developer for the Gym Management System project, I was responsible for developing, testing, and deploying the entire system. My contributions included:

1. User Management and Authentication:
   - I implemented user registration and login functionality for Admins, Trainers, and Members. This involved creating the User, UserDAO, and UserService classes, ensuring secure password storage using BCrypt hashing.
   - I designed role-based access control (RBAC) where each user type has specific access privileges, such as Admins managing users, Trainers creating workout classes, and Members viewing classes and purchasing memberships.
2. Membership and Workout Class Management:
   - I developed the membership management functionality, allowing Members and Trainers to

purchase memberships. Admins were able to track membership revenue.
    - I created CRUD operations for the Membership and WorkoutClass entities, ensuring that Trainers could add, update, or delete workout classes, and Members could view available classes.
3. Database Integration:
    - Integrated PostgreSQL to store user, membership, and class data. I designed the database schema, ensuring proper relationships between users, memberships, and workout classes.
    - I created SQL scripts to populate the database and implemented the necessary CRUD operations in Java for each entity.
4. Console-Based Interface:
    - I developed a user-friendly console interface using the Scanner class for interacting with the system, allowing users to register, log in, and perform tasks according to their roles.
5. Testing and Debugging:
    - I wrote unit tests for key functionality such as user authentication, membership purchases, and class management to ensure the correctness of the system.
    - I debugged and resolved issues related to database connectivity and user authentication, ensuring a smooth user experience.

Challenges Faced During Development

While working on the Gym Management System, I faced a couple of key challenges:

1. Updating Membership and Workout Class Logic:
   - Initially, I had difficulties in correctly updating the membership information and associating members with workout classes. Some issues arose when updating membership statuses or adding new classes, which caused inconsistent data in the system.
   - I refactored the data flow to ensure smooth updates between users, memberships, and workout classes, ensuring the system maintained data integrity.
2. Maven and Java SE 21 Compatibility:
   - During the setup, I faced issues with the Maven build configuration. The project initially used an older version of Java, which caused compatibility problems with certain dependencies and features.
   - I had to update the Maven configuration to use Java SE 21, ensuring compatibility with the latest libraries and tools. This process involved troubleshooting dependency issues and testing the system after each change to ensure everything worked correctly.
3. Database Integration and SQL Queries:
   - Setting up PostgreSQL and ensuring the Java application could correctly communicate with the database presented a challenge. There were some difficulties in getting the database connections to work seamlessly with the Java code, especially when dealing with CRUD operations.
   - I worked through these issues by carefully reviewing the connection settings and modifying the database queries to ensure they were optimized and functional.