Gym Management System
User & Development Documentation

Braeden Mercer, Stephen Badcock, Mimya Hafiz
Group 7 SD12 Java Final Sprint

Introduction

The Gym Management System is a console-based Java application primarily used to oversee, manage and control the day to day operations and revenues of a fitness center. It recognizes and permits access to the necessary functions for Administrators, Trainers & Members, respectively. This is achieved through role-based logins after a user registers for the first time using a username, password, email, phone number, address and their role.

The application runs a menu-based interface allowing users to navigate to complete the desired actions with ease. When starting the application for the first time the user will be prompted to register a new account or use saved info to log into an existing account. After logging in the user will see a role-specific menu only containing the functions that that user's role is allowed to access. When logged in as an admin you will be able to see all users and their contact info, permanently delete users from the system, and view all memberships and total revenue. When logged in as a trainer, you will be able to create, update and delete workout classes, view a list of all workout classes assigned to that trainer, and purchase a gym membership. Finally if you are logged in as a member you will be able to browse workout classes, view total expenses and purchase a gym membership.

The system is connected to a PostgreSQL database to persistently store and retrieve data related to users, memberships and workout classes. When a user performs an action the application interacts with the database to update the relevant information.

Classes

User Class

Purpose

Represents a generic user in the system. It is a base class for user specific roles.

Attributes

- id: Numeric value used to uniquely identify each person
- userName: created name for a user during registration
- password: a user created password for log in purposes (hashed with bcrypt)
- phoneNumber: Contact number of the user
- address: user email address
- role: user role (admin, member, trainer)

Methods

Getters and setters for attributes

Interactions

Inherited by Admin, Trainer, and Member. Used by UserDAO for database operations and UserServcie for business logic.

Admin Class

Purpose

Represents an administrator user with the most privileges. Inherits attributes and methods from user class

Interactions

Interacts with UserDAO, MembershipDAO and WorkoutClassDAO through UserService, MembershipService and WorkoutClassService to perform admin-specific operations.

Trainer Class

Purpose

Represents a trainer user who can manage workout classes. Inherits Attributes and Methods from user class.

Interactions

Interacts with WorkoutClassDAO through WorkoutClassService to manage classes and with MembershipDAO through MembershipService to purchase memberships.

Member Class

Purpose

Represents a member of the fitness center who can browse classes and purchase memberships.

Interactions

Interacts with WorkoutClassDAO through WorkoutClassService to view classes and with MembershipDAO through MembershipService to purchase memberships and view personal expenses..

Membership Class

Purpose

Represents a membership for the fitness center

Attributes

- Id: numeric value used to uniquely identify each person
- membershipType: The membership level the user decided to purchase
- membershipDescription: Brief explanation of what the level of membership includes
- membershipCost: cost of the membership
- availableCredits: number of credits available for member to use
- startDate: start date for member's current membership
- endDate: end date for member's current membership
- memberID: foreign key referencing user

Methods

Getters and setters for attributes

Interaction

Interacts with MembershipDAO for database operations and MembershipService for business logic related to memberships. Linked to user through memberId

MemberClass Class

Purpose

Represents a workout class offered at the gym.

Attributes

- Id: numeric value used to uniquely identify each class
- MemberClassType: name of the class offered
- MemberClassDescription: brief description of the class
- trainerId: foreign key referencing user with the role 'Trainer'

Methods

Getters and setters for the attributes

Interactions

Interacts with WorkoutClassDAO for database operations and WorkoutClassService for business logic related to workout classes. Linked to the user through trainerID.

Data Access Objects (DAOs):

UserDAO

Purpose

Handles all database interactions related to the user entity. Similar DAOs created for admin, member and trainer.

Methods

- addUser
- updateUser
- getAllUsers
- getUserByEmail
- getUserById
- deleteUser
- buildUserFromResultSet

MembershipDAO

Purpose

Handles all database interactions related to the Membership entity

Methods

- addMembership
- getMembershipById
- getMembershipsByMemberId
- getAllMemberships
- updateMembership

- deleteMembership
- getTotalRevenue

MemberClassDAO

Purpose

Handles all database interactions related to the WorkoutClass entity

Methods

- getMembersByClassId
- getClassesByMemberId
- isMemberEnrolledInClass
- addMemberToClass

Services

UserService

Purpose

Contains the business logic for user-related operations, such as registration, login, retrieving user information, and deleting users. Similar services created for admin, trainer and member.

Methods

- registerUser
- loginUser
- listAllUsers
- updateUser
- deleteUser
- isValidRole

MembershipService

Purpose

Contains the business logic for membership-related operations, such as purchasing memberships, viewing membership details, and tracking revenue

Methods

- addMembership
- getMemberId
- getMembershipType
- getMembershipById
- getAllMemberships
- updateMembership
- deleteMembership
- getTotalRevenue
- getMembershipsByMemberId

WorkoutClassService

Purpose

Contains the business logic for member class-related operations, such as adding, updating, deleting, and viewing classes.

Methods

- createWorkoutClass
- listAllWorkoutClasses
- getWorkoutClassDetails
- getClassRoster

- unassignTrainerFromClass
- updateWorkoutClass
- deleteWorkoutClass

Starting & Using the System

Prerequisites

Java, Maven & PostgreSQL must be installed on the machine. A PostgreSQL database created for the system.

Setting up Database Connection

Locate the configuration file for database connection details. It can be found under target/classes/db.properties. Update the file with your personal PostgresSQL database credentials (database name, username, password)

Build the project using Maven

Run “mvn clean install” in the terminal. This will download any dependencies, compile the java code, and package the application.

Run The Application

Using the System

The application will start in the console and display a main menu. You wil be given the option to login an existing user or register a new user. If you choose to login, you will be prompted to enter an existing username and password. If you choose to register, you will be prompted to enter your username, password, email, phone number, address and desired role. Once logged in, depending on your role you will see role-specific menus. Under the admin menu you will see the options to view all users, view all memberships and total revenue. Under the trainer menu you will see the option to create, update, and delete workout classes, view assigned classes, and purchase a membership. Finally, under the member menu you will see the options to browse workout classes, view membership expenses, and purchase a new membership. Follow on=screen prompts to naviogate the menus and perform actions. Enter the correct numbers to select options. Their will be a logout option in the menus to leave the system.

Development Documentation

Introduction

This section provides information for developers who will be working on or maintaining the Gym Management System. It covers technical aspects of the program including, but not limited to, code documentation, project structure and build process.

Javadoc Documentation

Throughout the system javadocs can be found, used for all key methods and documents. This is important to the system because it helps users and developers understand the code and having easy to understand code increases maintainability and collaboration within the development of the system

Project Directory

A well organized project directory is a must for a good Java system for many reasons but the biggest being organization and ease to understand for new developers on the system.

Key Directories

- Pom.xml - The heart of maven. It defines project metadata, dependencies, build configuration, and plugins
- src/main/java/com/group7/gym - Contains the man Java source code for the application
- App.java - Entry point of the application
- dao/ - Class that handles data access operation for each entity. (CRUD operations)
- models/ - All necessary classes for the system

- service/ Entities used to call the dao requests.
- README.md - A file providing high-level overview of project and setup instructions

Build Process

1. Pom.xml Configuration: This file defines the build process. The file is used by maven to download dependencies, compile the code, run tests and package the application.
2. Maven Lifecycle: Some common commands used in the maven lifecycle are clean, compile, test, package, and install.
3. Plugin Management: Maven plugins are used to extend functionality. For example, maven-compiler-plugin is used for compiling Java Code.

Dependencies

Dependencies used in this system include:

- PostgreSQL JDBC Driver - used to establish a connection to PostgreSQL server, execute SQL queries and retrieve data from database
- BCrypt - provides BCrypt algorithm for securely hashing passwords.
- JUnit for Testing - Widely used unit testing framework for Java. Allows writing and running automated tests for single units of code.

These dependencies are essential for the system to be able to Interact with the PostgreSQL, securely handle passwords, and write and run automated tests to ensure code quality.

Setting up Database

1. Install PostgreSQL - follow PostgreSQL documentation for installation instructions
2. Connect to PostgreSQL - use a client tool such as pgAdmin to connect the database to the system.
3. Create the database - Use the following command "CREATE DATABASE gym_management_system;" in pgAdmin to create database
4. Create Tables - create required tables for the database to be functional, such as users, memberships, and workout classes.

5. Configure connection in application - Ensure DAOs are set up to properly connect to the database

Cloning and Running the Project

1. Obtain repository url from github page
2. Clone repository by using “git clone <repository_url> command
3. Navigate to correct directory just created using cd command
4. Ensure all necessary software is downloaded (JDK, MAven, PostgreSQL)
5. Configure database connections in DAO files if not already completed.
6. Run command “mvn clean install” to start project