Fitness Database Management System

COMP3005

By: Rishabh Jain, Wren Liu, & Maggie Zheng

December 1st, 2025

Overview

The Fitness Database Management System is a PostgreSQL-based program and database management system, written in Java, that allows Members, Trainers, and Administrators to interact with an application that enables users to register as members, register for classes and personal training sessions, and set fitness goals. They will be able to monitor their own measurable health, which will be logged to keep track of their vital records: height, weight, body fat percentage, and heart rate.

Aside from Members, Trainers will also have access to the database to log and set their availability for personal training sessions that Members can register for, provided those sessions do not coincide with any of their existing classes or times when they are unavailable. They can view the metrics of the members who participate in their classes and personal training sessions to see if progress is being made.

Finally, the administrators will have full access to the entire system. They will be responsible for billing and invoicing, maintenance tickets for equipment, and managing classes. They overlook the whole system, and while the system itself should not have errors occur, they will be there to correct them should anything be amiss manually.

The report will outline and summarize the relationships between tables and data, and provide a simple overview of how the project is set up and the functions we implemented in our application.

ER Schemal, Diagram, & Entity Sets

The following are all the entity sets within this database.

Users
members(**member_id**, name, date_of_birth, gender, email, phone, join_date)
trainers(**trainer_id**, name, email, phone, specialization, hire_date)

Room & Equipment
rooms(**room_id**, location)
equipment(**equipment_id**, *room_id*, name, is_operational)

- room_id → rooms(room_id)

Goals & Health Metrics
goal_types(**type_id**, name, unit)
fitness_goals(**goal_id**, *member_id*, *type_id*, target_value, target_date, start_date, is_completed)

- member_id → members(member_id)
- type_id → goal_types(type_id)

health_metrics(**metric_id**. *member_id*, timestamp, heart_Rate, body_fat, weight, height)

- member_id → members(member_id)

Trainer Availability & PT Sessions

trainer_availability(**availability_id**, *trainer_id*, start_timestamp, end_timestamp, recurrences)

- trainer_id → trainers(trainer_id)

pt_sessions(**session_id**, *trainer_id*, *member_id*, *room_id*, start_timestamp, end_timestamp)

- trainer_id → trainers(trainer_id)
- member_id → members(member_id)
- room_id → rooms(room_id)

Classes & Registration

classes(**class_id**, *trainer_id*, *room_id*, name, capacity, start_timestamp, end_timestamp)

- trainer_id → trainers(trainer_id)
- room_id → rooms(room_id)

class_registration(***class_id***, ***member_id***, register_date)

- PK = (**class_id, member_id**)
- class_id → classes(class_id)
- member_id → members(member_id)

Maintenance Tickets

maintenance_tickets(**ticket_id**, *equipment_id*, report_date, description, being_repaired, is_repaired, resolved_date)

- equipment_id → equipment(equipment_id)

Invoice, Items, & Payments

invoices(**invoice_id**, *member_id*, issue_timestampe, total, is_paid)

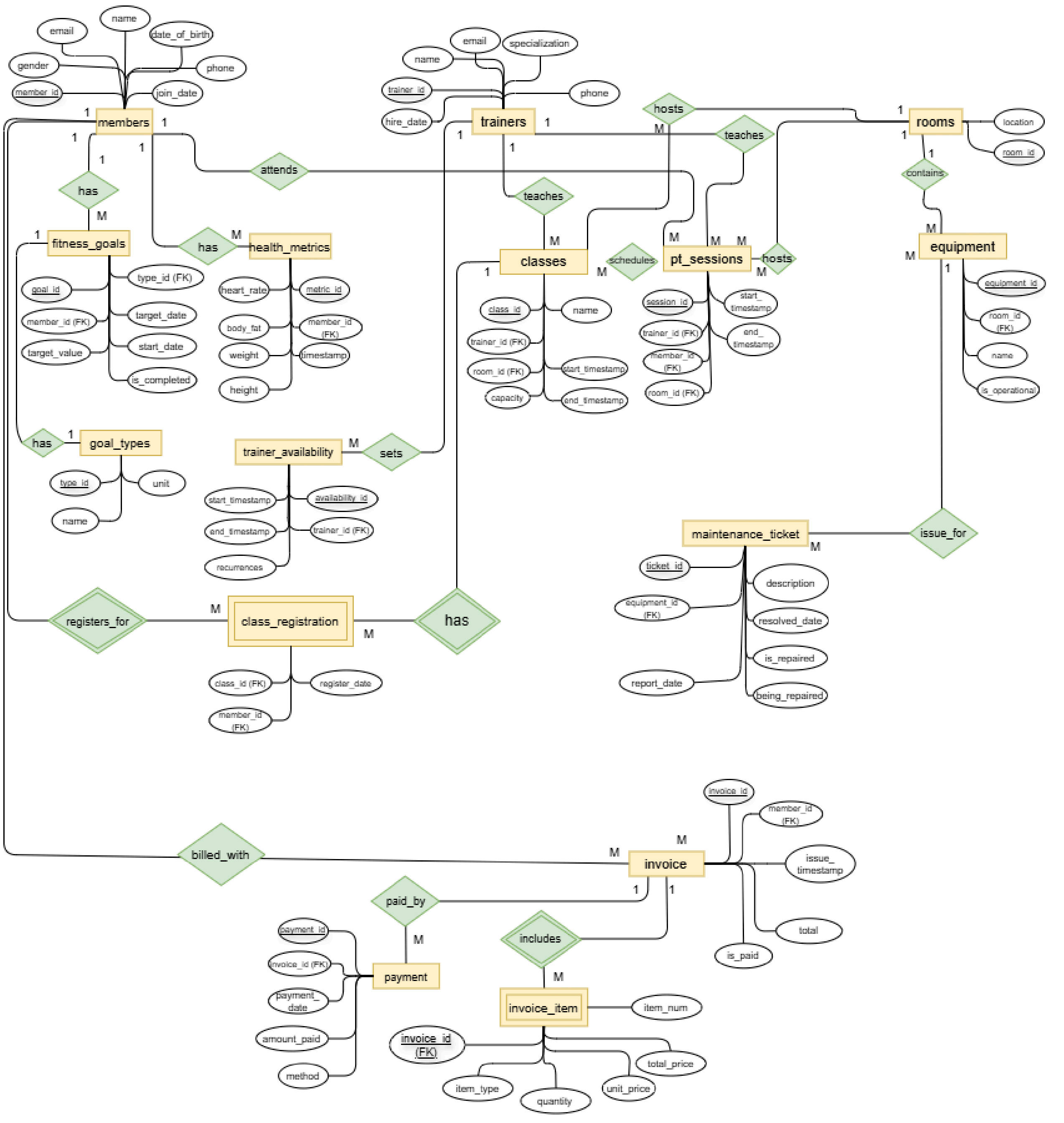
- member_id → members(member_id)

invoice_items(**invoice_id**, item_num, item_type, quantity, unit_price, total_price)

- PK = (**invoice_id**, **item_num**)
- invoice_id → invoices(invoice_id)

payments(**payment_id**, *invoice_id*, amount_paid, method, payment_date)

- invoice_id → invoices(invoice_id)



(Figure 1: ER Diagram)

Link to the draw.io file should the image not be clear enough:

All relationships in the diagram were initially broken down into categories within the above-stated sets before being further connected in the database. Members have the most essential features: the ability to view their health metrics, set fitness goals, and register for classes and personal training sessions. Trainers would then primarily be organized by trainer_availability and pt_sessions, since they are scheduled around the trainer and their availability. Rooms, equipment, and maintenance tickets are to be managed via the CLI for our project, enabling admins to log in and oversee these functions. They are there to correct any errors that the system may miss. This logic was used throughout to slowly build up for the database and connect the tables with each other where necessary. All relations have been reduced to 1:M relationships and normalized to 3NF.

There are two weak entities in this diagram: invoice_items and class_registration. Invoice_items have a composite key consisting of invoice_id and item_no. This is because item number, when logically implemented on an invoice, cannot always be unique; thus, it should be determined relative to each invoice, dictated by invoice_id. This way, multiple invoices can have item numbers in a chronological, sensible order, hence why invoice_items does not have a key of its own. Class_registration is a weak entity created to normalize the N:M relationship between members and classes. Although similar to personal training sessions, classes have a capacity and can accommodate multiple members (as opposed to one member per personal training session), which requires normalization to the 1:M level. This ensures our system remains 3NF. As such, it uses a composite key that references both member_id and class_id, and stores the date each member registered for each class.

Informal breakdown of all the relationships:

members 1 — M fitness_goals

goal_types 1 — M fitness_goals

members1 — M health_metrics

trainers 1 — M trainer_availability

trainers 1 — M pt_sessions

members 1 — M pt_sessions

rooms 1 — M pt_sessions

trainers 1 — M classes

rooms 1 — M classes

rooms 1 — M equipment

equipment 1 — M maintenance_tickets

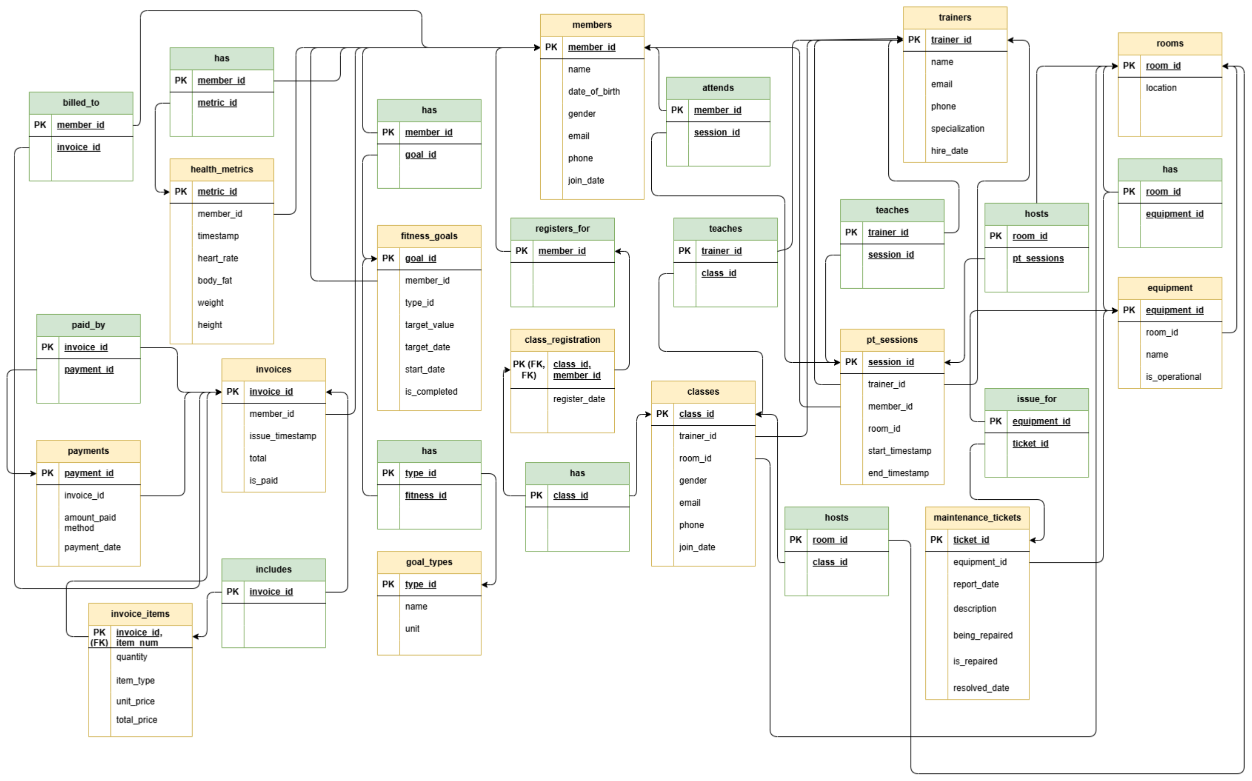
members 1 — M invoices

invoices 1 — M invoice_items *(weak)*

invoices 1 — M payments

members M — N classes via weak entity class_registration
→ members 1 — M class_registrations
→ classes 1 — M class_registrations
Normalization of members to classes relation into members to class_registrations and classes to class_registrations.

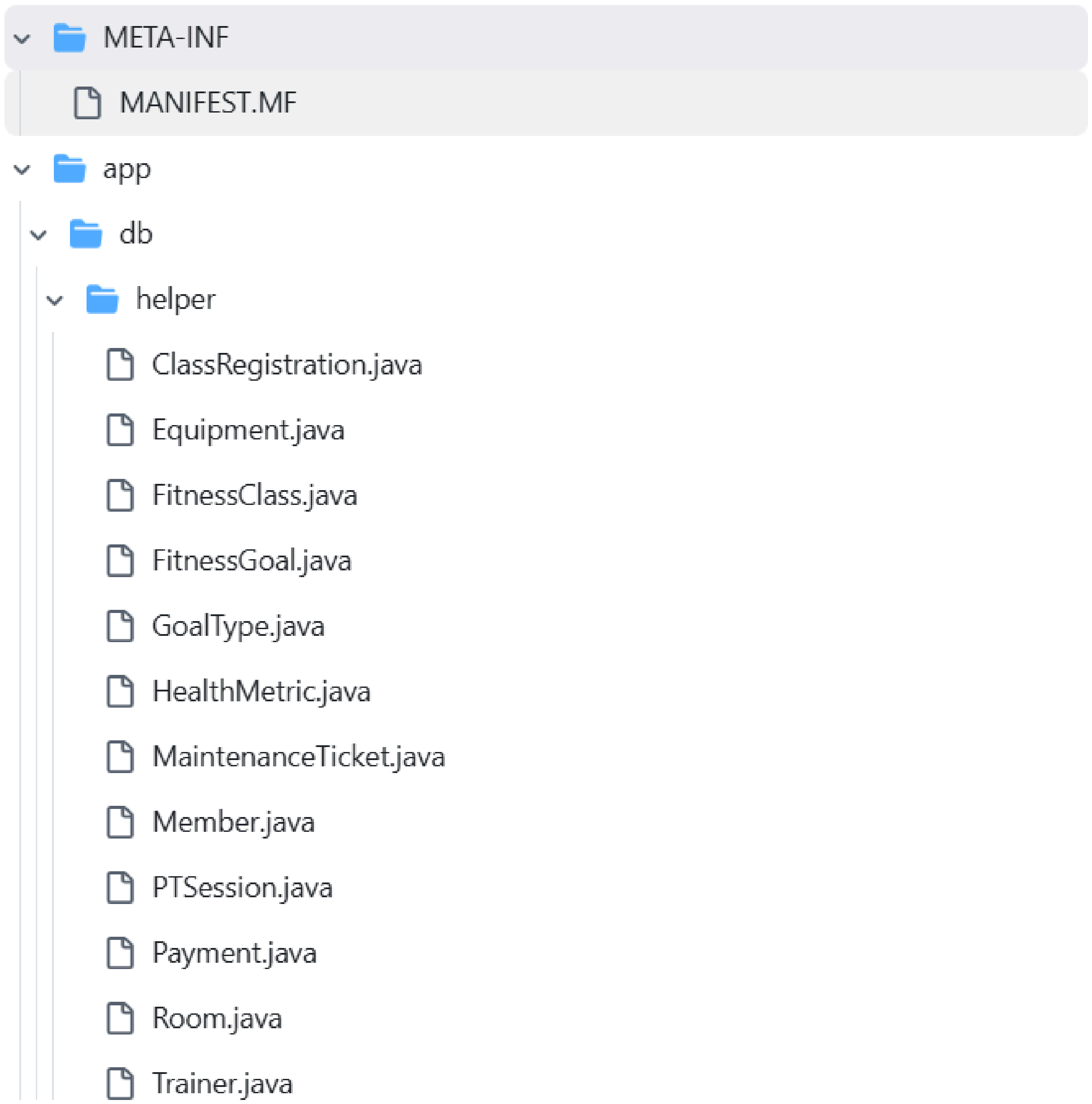
ER Schema



(Figure 2: ER Schema diagram)
Link to the draw.io file should the image not be clear enough:

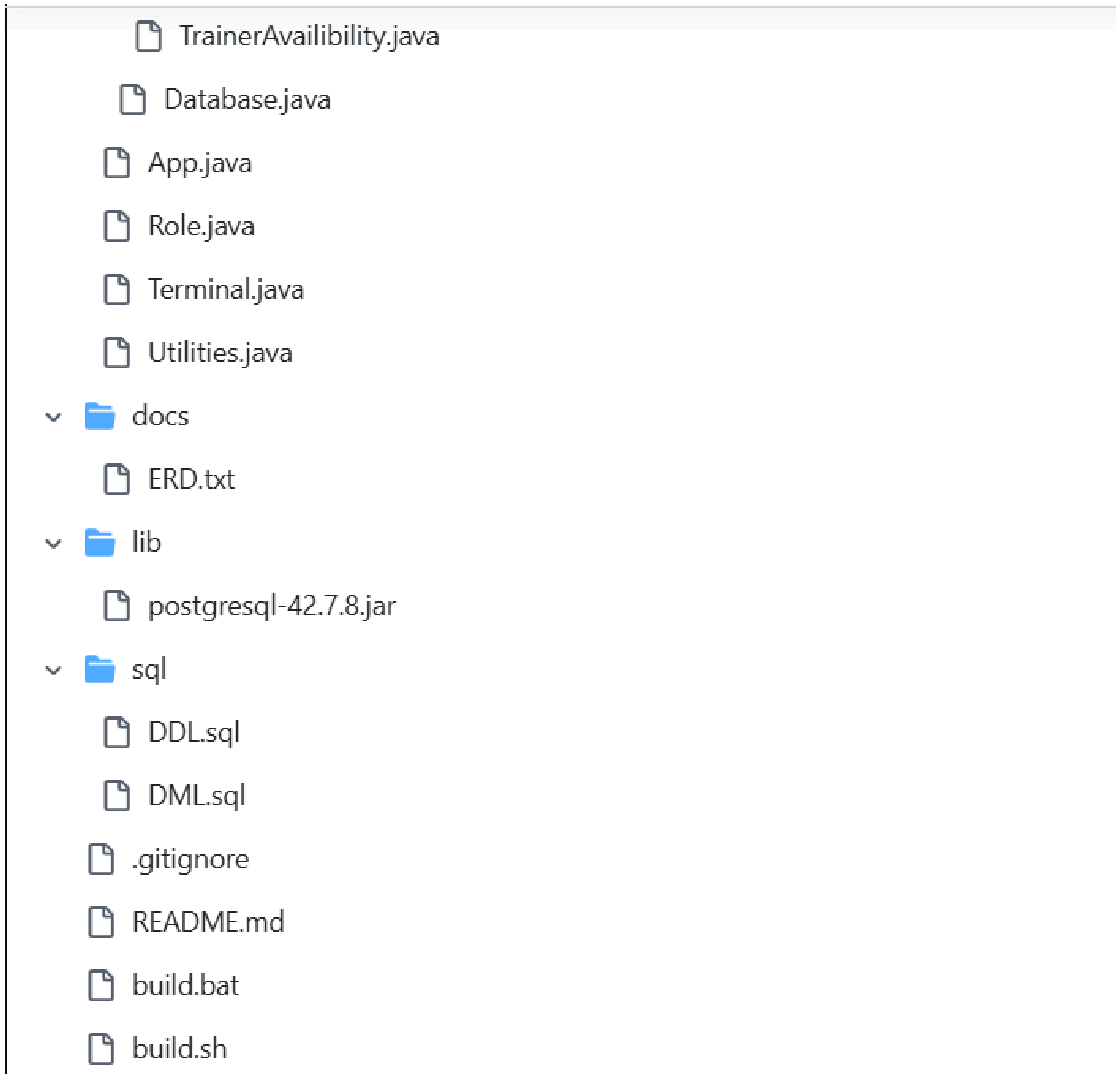
Project Structure & Setup

File Structure:



```
META-INF
    MANIFEST.MF
app
    db
        helper
            ClassRegistration.java
            Equipment.java
            FitnessClass.java
            FitnessGoal.java
            GoalType.java
            HealthMetric.java
            MaintenanceTicket.java
            Member.java
            PTSession.java
            Payment.java
            Room.java
            Trainer.java
```

(Figure 3a: Project structure)


TrainerAvailibility.java
Database.java
App.java
Role.java
Terminal.java
Utilities.java
docs
ERD.txt
lib
postgresql-42.7.8.jar
sql
DDL.sql
DML.sql
.gitignore
README.md
build.bat
build.sh

(Figure 3b: Project structure cont'd)

The main folders in the root directory are META-INF, app, docs, lib, sql, .gitignore, README.md, build.bat, and build.sh.

META-INF:

The META-INF directory contains MANIFEST.MF determines the Class-Path and Main-Class that connect our code to the class that will be packaged into a jar file, using the postgresql-42.7.8.jar file in the lib directory.

app:

The main functionality of the code is in the app directory, with the db directory containing all functions for manipulating the database. Inside the db directory, there is another helper folder that further breaks down the application's functions into helper functions for fetching and retrieving requested information. Each Java class file represents an entity (with the same name; the only difference is that the file names follow Java naming conventions), each with helper functions that can retrieve information from its respective table in the database.

lib:

Our lib directory contains the postgresql-42.7.8.jar file, which is what we will primarily use to connect to our PostgreSQL database.

sql:

The sql directory contains ddl.sql and dml.sql, which set up the initial database tables and initial values. For our project, we assume the database is pre-populated.

docs:

The docs directory includes all the PDFs and files associated with the project. This includes the report, the ER Diagram, and the ER Schema.

Our CLI (in the terminal) will run the program and display visuals that present three views, with the proper implementations and limitations for the three different user types.

The build.bat is the build file for all Windows systems, while the build.sh will be for Linux systems. They have all the detailed commands to create an App.jar file that compiles the code, allowing us to run the management system by connecting to the database.