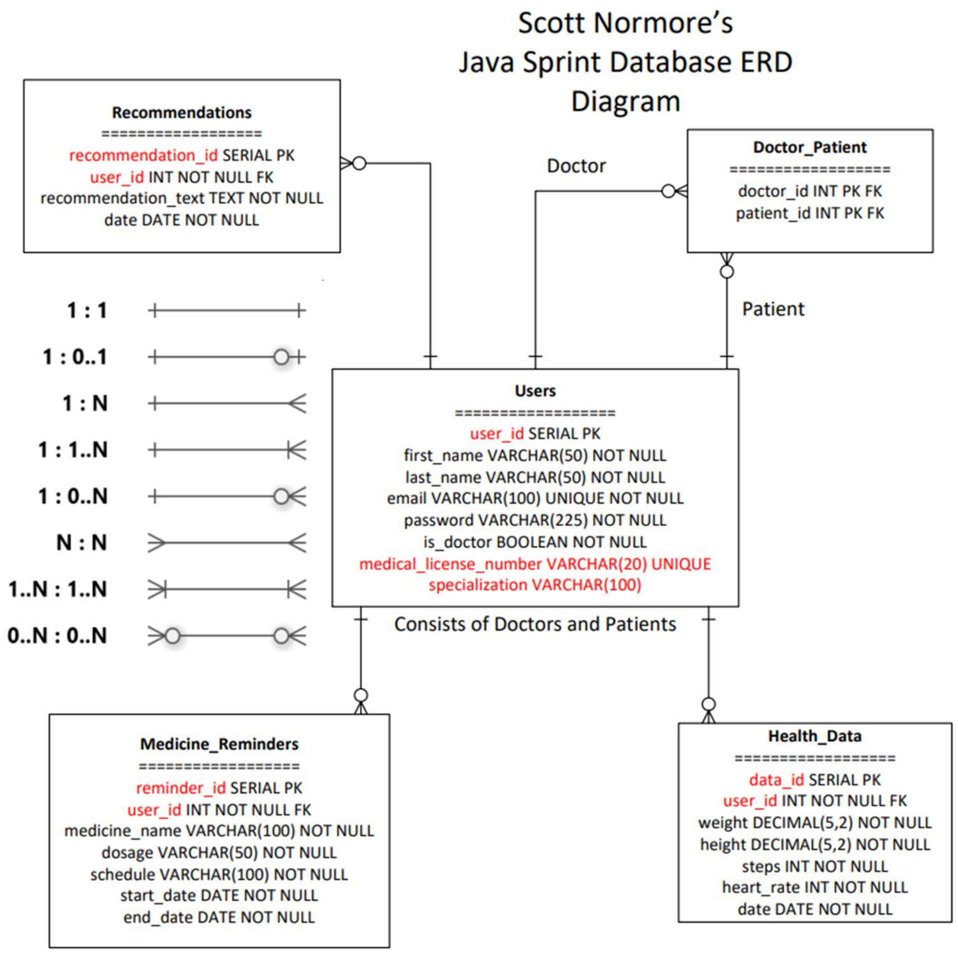
SCOTT NORMORE’S FINAL SPRINT: HEALTH MANAGEMENT SYSTEM

Scott Normore

Introduction

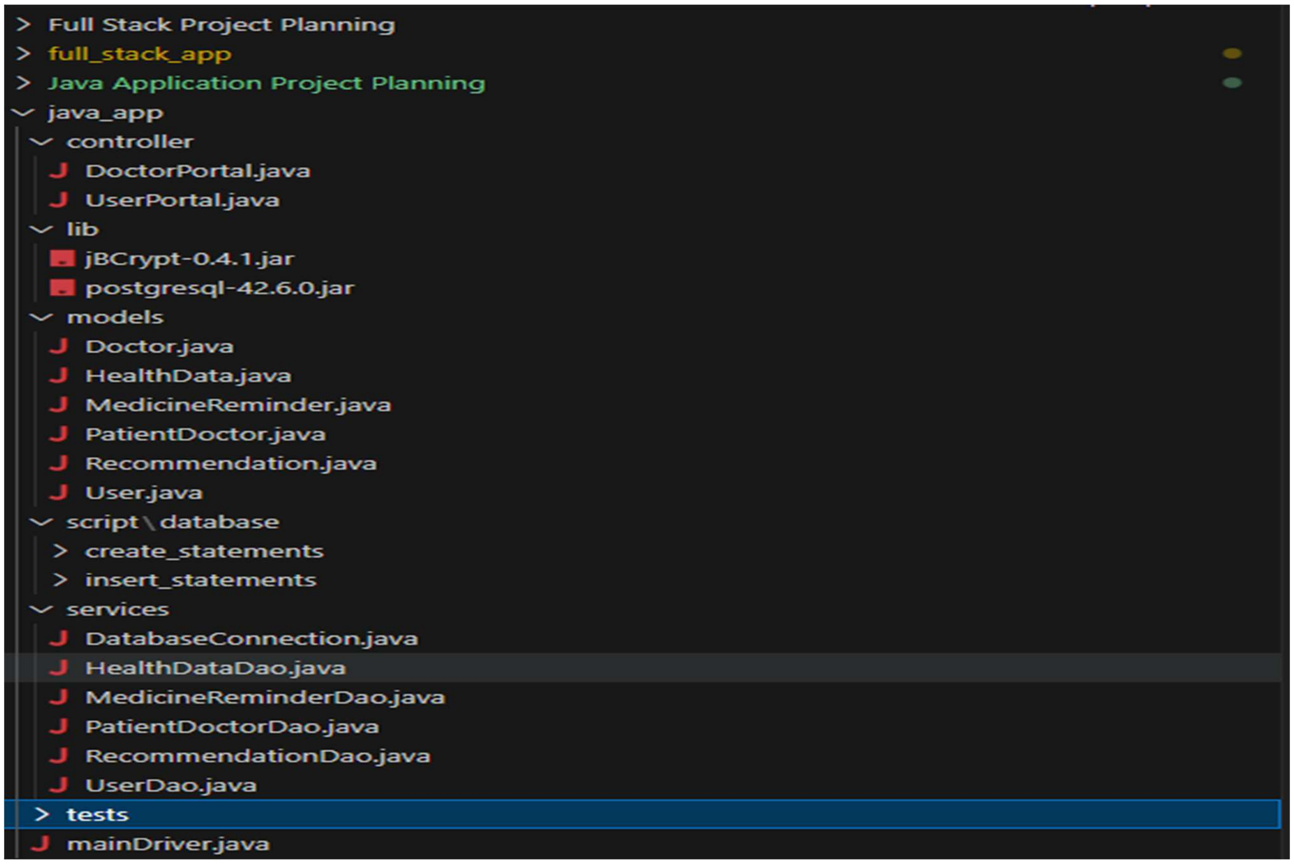
This is Scott Normore's Submission for the java final sprint. In this document, I will showcase the directory structure of the document, the classes in the application, and how to run my application through visual studio code.

Database Structure



This is my database ERD. The Structure of the database used the given script with a couple of changes. Those changes are highlighted in red. I have changed the id field in users from id to user_id, and done similar in recommendations, medicine reminders, and health data. In addition, I have added license number and specialization to users to handle doctors in the database. Whilst this breaks 3NF, I found it quite handy to do this as it allows for quick checks to see if a user is a doctor or not without having to query another table.

Directory Structure



```
> Full Stack Project Planning
> full_stack_app
> Java Application Project Planning
∨ java_app
  ∨ controller
    J DoctorPortal.java
    J UserPortal.java
  ∨ lib
    jBCrypt-0.4.1.jar
    postgresql-42.6.0.jar
  ∨ models
    J Doctor.java
    J HealthData.java
    J MedicineReminder.java
    J PatientDoctor.java
    J Recommendation.java
    J User.java
  ∨ script \ database
    > create_statements
    > insert_statements
  ∨ services
    J DatabaseConnection.java
    J HealthDataDao.java
    J MedicineReminderDao.java
    J PatientDoctorDao.java
    J RecommendationDao.java
    J UserDao.java
  > tests
  J mainDriver.java
```

Above is the visual representation of the directory structure. Since I'm using the one git repository for all my work, you should ignore the full_stack_app and Full Stack Project Planning folders. These two folders contain information on the other final sprint.

The most important folder is the java_app folder. In that folder contains the application code. At the highest level in that folder, we have 5 sub directories (models, controller, lib, script, services) and the mainDriver file. The main driver file is used to run the health monitoring system application.

The other directories also include some important items. The script folder contains the scripts required to create the database tables and to populate the database tables with records.

The lib folder contains libraries used by the application; the postgresql is used to establish connection to the SQL database, and the jbcrypt is used to encrypt the passwords stored on the database, as well as verify/authenticate entered passwords to the database.

The other three folders contain classes used by the database. In particular, the service classes are used to establish connection to the database and return the information from the database into java objects/models. The models folder contains those mentioned models. Finally, the controller app contains the hybrid controller/view classes for the database. More on these classes in the next section.

Classes

Models

User – A data model that represents a user. A user has an id, a first and last names, an email that they use to login, a password they use to login, and a boolean to represent if they are a doctor or not (used by the system).

Doctor – A data model that represents a doctor. It is a subclass of user with only two additional fields; medicalLicenceNumber, and specialization. It is retrieved from the same table in the database; only the model and function used to retrieve it assumes it will return a doctor instead of a user.

MedicineReminder- A data model that represents a medicine reminder in the database. Used by the user to create new medicine reminders. Doctors can also issue reminders to users. Contains an id, an associated user id, a medicine name, a dosage amount, a schedule, and the start and end dates of the prescription.

PatientDoctor – A data model that represents relationship between a patient and a doctor. Doctors can assign patients under them and view patients under them. Patients can view doctors assigned to them.

Recommendation – A data model that represents a recommendation object. This is by the system when a user generates recommendation based on the inputted health data. Doctors can also create recommendations for users. The recommendation only has 3 attributes; an id, a user id, and the recommendation

HealthData – A model that represents health data inputted from the user. It contains an id, a user id, weight, height, steps, heartrate, and a date object. The user id represents the owner of the health date, the weight is in kg, the height in meters, steps in the steps taken that day, and a date object of when the health data has been taken.

Services

DatabaseConnection – used to establish a connection to the database. Used by the other Database Access Objects (DAOs)

UserDAO – Contains the Basic Crud operations for anything to do with the User Table. Also contains a function for verifying logins. Unlike the other DAO objects, it has two model classes due to Users being both Users and Doctors!

HealthDataDAO – Contains the Basic Crud operations for anything to do with the User Table.

PatientDoctorDAO – Contains the Basic Crud operations for anything to do with the User Table.

RecommendationDAO – Contains the Basic Crud operations for anything to do with the User Table.

MedicineReminderDAO – Contains the Basic Crud operations for anything to do with the User Table.

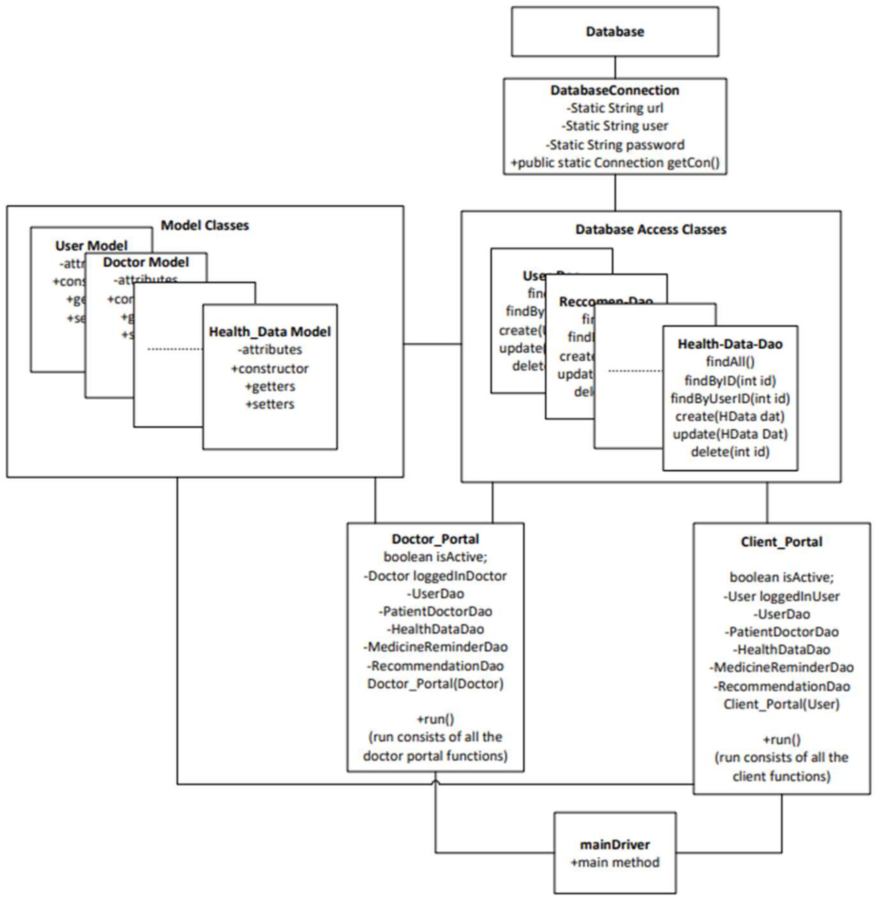
Controllers

DoctorPortal – A controller/view object for the doctor portal. While it only has the run() function, the run function is capable of viewing personal data of the logged in doctor, changing said data, viewing assigned patients, assigning new patients, viewing health data of a patient, creating a new recommendation of a patient, and creating a new medicine reminder for a patient.

UserPortal - A controller/view object for user portal. While it only has the run() function, the run function is capable of viewing personal data of the logged in user, changing said data, creating, viewing, and deleting reminders, creating, viewing health data, generate recommendations, clearing recommendations, and viewing assigned doctors. Recommendations are generated off the latest health data, and use the information to calculate BMI to recommend the next course of action.

MainDriver – A controller/view object for the health data system. It is a controller object because it also handles user creation and account login. Not the most proud of the route taken this project; I'll elaborate more in the next section.

Class Diagram



This is the architecture of the application. It used a database to store the information. The application used the DatabaseConnection.java class to connect to the database. From there, we use Database Access Objects to get the data from the database, and output that data into models used by the application. The portal classes use these database objects to give the application functionality. Finally, the main driver runs the portal classes.

The problem with this design is that I did not segregate the controllers from the views. Because of this, the portals are very cluttered classes that handle both logic and the displaying of data. In retrospect, I should have split the doctorportal class into a doctorController and a doctorView. The same should have been done with user. Finally, I should have a guest controller and view class for the main driver instead of having the main driver also handle that aspect.

With all of the above being said, the application does work, with only some minor bugs. Those bugs being associated with the lack of authorization checking, so a logged in user could in theory delete the reminders of other users.

Setup/Run

To run, open the **java_app folder** in visual studio code. Then use the run button to run the application.

Java Documentation

Included in all non model classes are documentation on the functions and attributes.