CSCI 5448 Project Part 4

Health Tracker

Team: Prithvi Nath Manikonda
Priyanka Pashte
Shreya Joshi

1. What features were implemented?

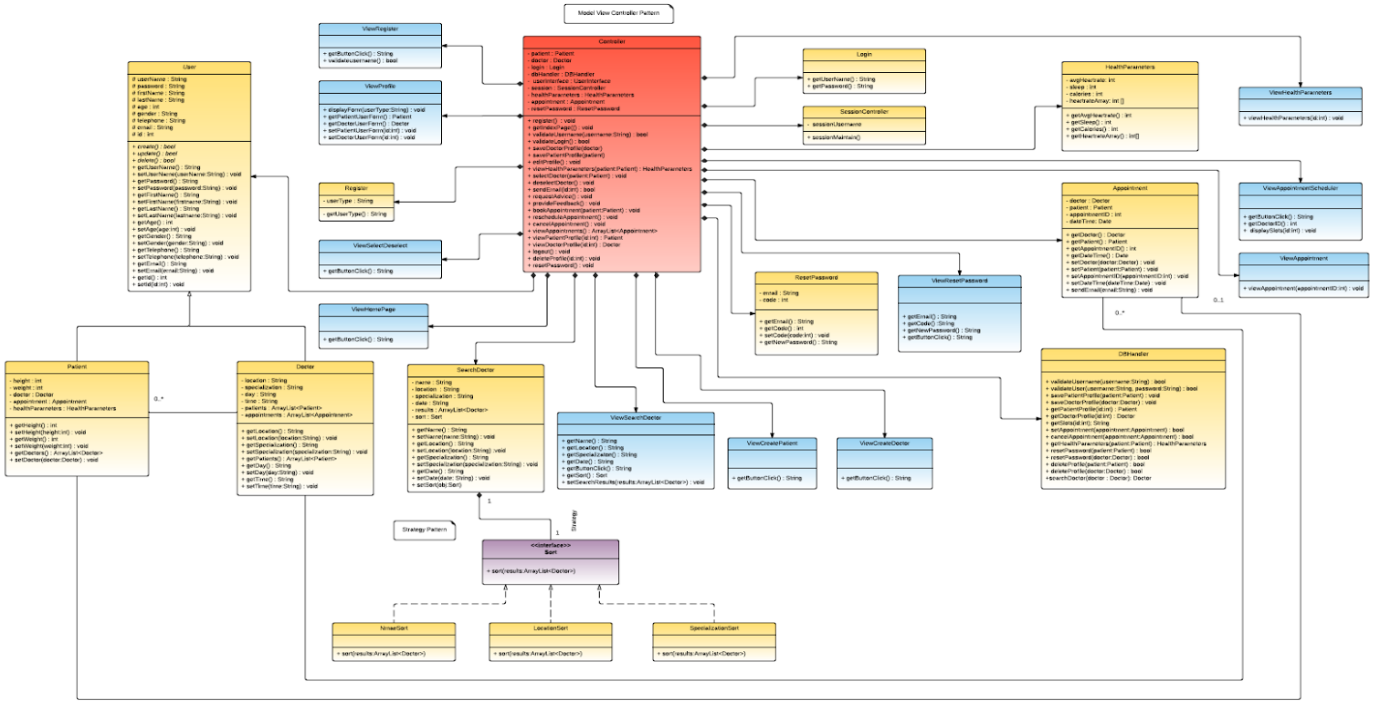
- FR-01: The system verifies that the username is unique and password meets a set of guidelines at the time of registration.
- FR-02: The system is able to differentiate between a Patient and Doctor Login.
- UR-01: Patient and Doctor can register and create their respective profile.
- UR-02: Patient and Doctor can login and view their respective homepage.
- UR-03: Patient and Doctor can edit their profile.
- UR-06(a): Patient can select an advising Doctor from the registered ones to monitor their profile.
- UR-06(b): Patient can deselect the advising Doctor.
- UR-08(a): Patient can Book an appointment with a specific Doctor.
- UR-08(c): Patient can cancel a booked appointment with a specific Doctor.
- UR-09: Doctor can view the profile of his Advising patients.
- UR-10: Doctor can view the health parameters of his advising patients.
- UR-14: Patient and Doctor can logout.
- BR-01: While executing a search, the system must be able to display 10 search results per page.
- NFR-01: Every webpage that is accessed must be loaded within 5 seconds.

2. Which features were not implemented from Part 2?

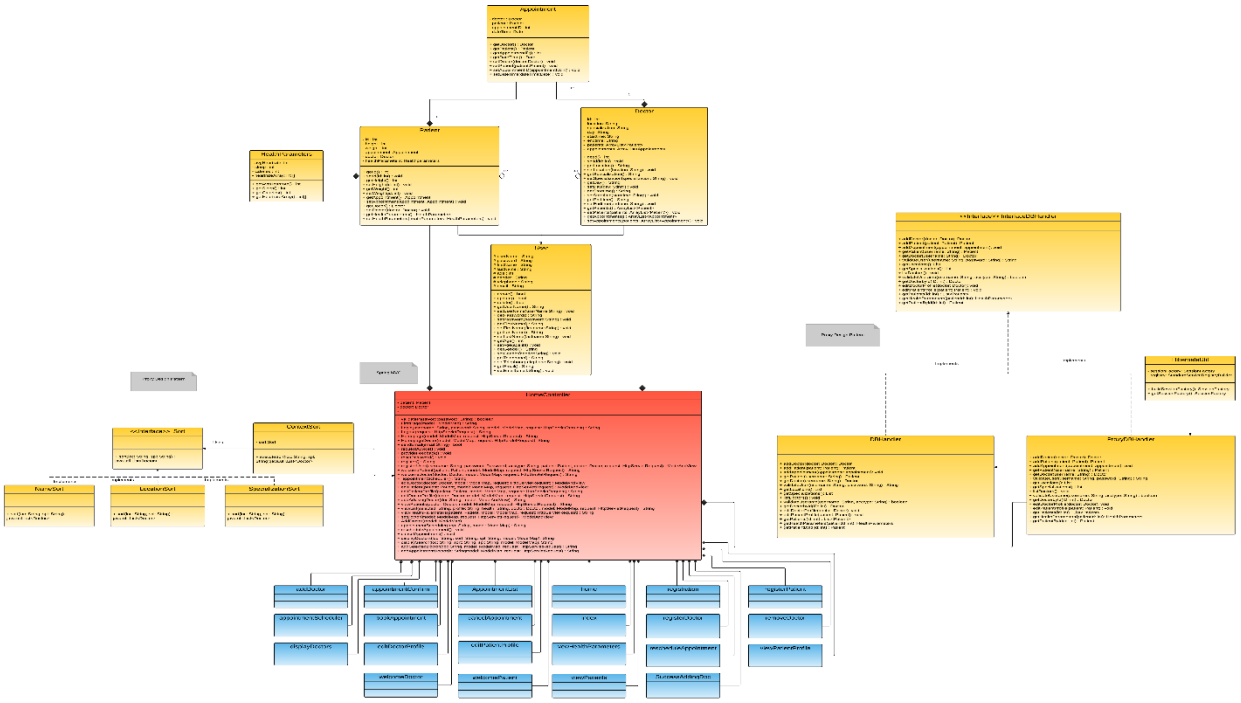
- FR-03: If inactivity on the webpage persists for over an hour, then the Patient/Doctor is automatically logged out.
- UR-04: Patient can view his health parameters for the past 3 days.
- UR-05: Patient can view various Doctor's profiles.
- FR-04: Logging involving authentication via two factor authentication through E-mail.
- UR-07: Patient can request for health advice from his advising Doctor.
- UR-08(b): Patient can reschedule a booked appointment with a specific Doctor.
- UR-11: Doctor can send feedback/emergency notification to his patients.
- UR-12: Doctor can cancel his appointment with a Patient.
- UR-13: Patients and Doctors can delete their respective profiles.
- UR-15: Patient and Doctor are allowed to reset the password.

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Original Class Diagram: For Closer look of the class diagram please refer: [Class Diagram Part 2]



New Class Diagram: For Closer look of the class diagram please refer: [Class Diagram Part 4]



During the implementation we stuck to most of the parts proposed in the class diagram during Part 2. We had decided to use MVC pattern for our design and developing the class diagram accordingly provided a good base while working on the implementation and lead to a faster development process.

However, the primary functions remained the same but the internal design of classes changed a bit. Some models, such as model of search doctor, were felt unnecessary and has been removed from the new class diagram. Also, during the implementation of Proxy Design Pattern we realized the need of an Interface to facilitate creation of one object on demand and hence that change has also been incorporated into the new class diagram.

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

The analysis of the project beforehand and creating class diagrams before even starting the implementation helped a lot with the implementation. We did a good job by identifying various models, views and controller. However, when we started with the implementation we realized that we could have used other design patterns and successfully incorporated few of those.

The following are the design patterns that have been used:

1. **Proxy Design Pattern :**

   Virtual Proxy Design Pattern has been used to avoid overhead on the DBHandler class. A DBHandler Interface was created. The ProxyDBHandler and DBHandler classes implements DBHandler interface. This allowed us to ensure that only one DBHandler object is created and used. Hence, this would help avoid object duplication and save memory. Also, this Design pattern provides us the flexibility to change the underlying database without changing the code in the client side.

2. **Strategy Design Pattern :**

   Strategy Design Pattern has been used that allows us to dynamically select the sorting algorithm during the run time. A sort interface was created which was implemented by various classes namely NameSort, LocationSort and SpecializationSort. And a ContextSort class was created which has the responsibility to execute the sorting algorithm. The Controller creates the object for ContextSort based on the request of the client. For example, if sort by name is selected by the user NameSort object would be created to execute the strategy.

3. **Model View Controller :**

   MVC Architectural Design pattern was applied for separation of concerns. The separation of model, view and controller allowed us to work independently on various models without worrying about the views. Also, it allowed us to reuse the same model across multiple views. Using ORM along with models allowed hibernate to create table for each model in the relational database.

The following are the design patterns that could have been used:

1. **Iterator Design Pattern :**

   Iterator Design Pattern could have been utilized to iterate through list of the patient doctor is advising. Every Doctor has a specific set of advising patients and iterator could have been used to go through the list of patients and display the patient's information to the doctor.

2. **Observer Design Pattern:**

   The Advantages of Observer Design Pattern could have been leveraged to notify the doctor when a particular patient has added the doctor as advising doctor or has requested the doctor for health advice. Similarly, the patient could be notified when the doctor has provided feedback.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

This project helped us to understand the software development cycle through practical implementation. We also observed the benefits of object-oriented approach such as understanding of a system, reusability of models and programming, modifiability and reduced costs associated with change, and reduced development risk.

The initial documentation preparation wherein we listed our requirements, use cases and created activity, sequence and class diagrams helped us to build a concrete foundation and determine and understanding the exact flow of the project.

We initially started the design by using just Java and JDBC, but upon realizing the extensive requirements of mappings for each and every class, we decided to explore spring and Hibernate frameworks that could be incorporated.

We found that Hibernate was much better than plain JDBC. JDBC required writing JDBC statement, setting the parameters, executing query and processing the result by hand is lot of work. Hibernate on the hand automatically generates and executes the SQL statements. This saved a lot of development time and all tedious efforts.

Finally, after we had implemented many of the requirements we could identify places where there was smelly code and worked towards refactoring it using the various method that were taught.