HEALTHCARE PLATFORM : HEALTHY BUFFS

Team Members: Payoj Jain, Rahul Chowdhury, Ganesh Chandra Satish

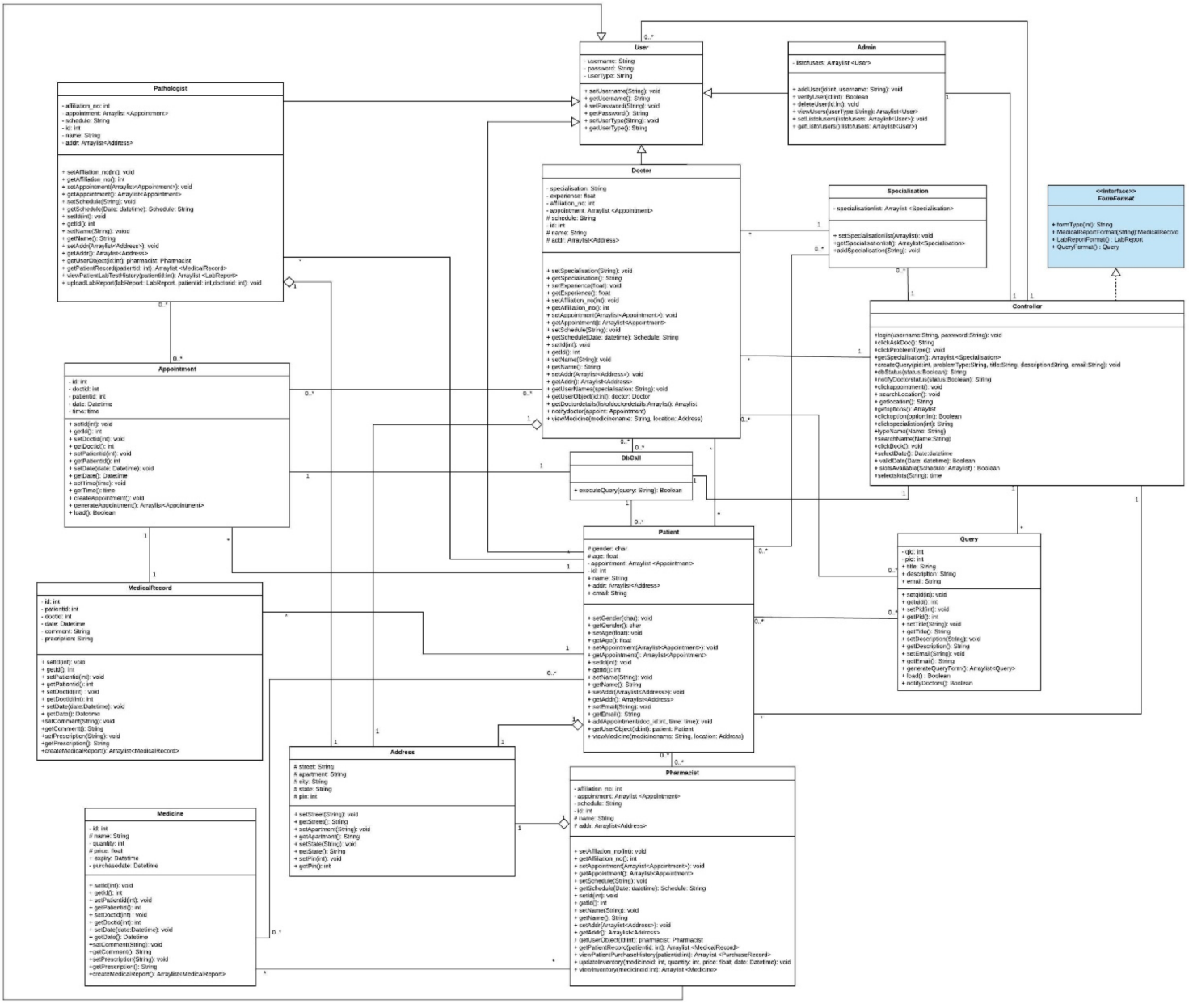
Team Number: 33

Title of the Project: HealthyBuffs

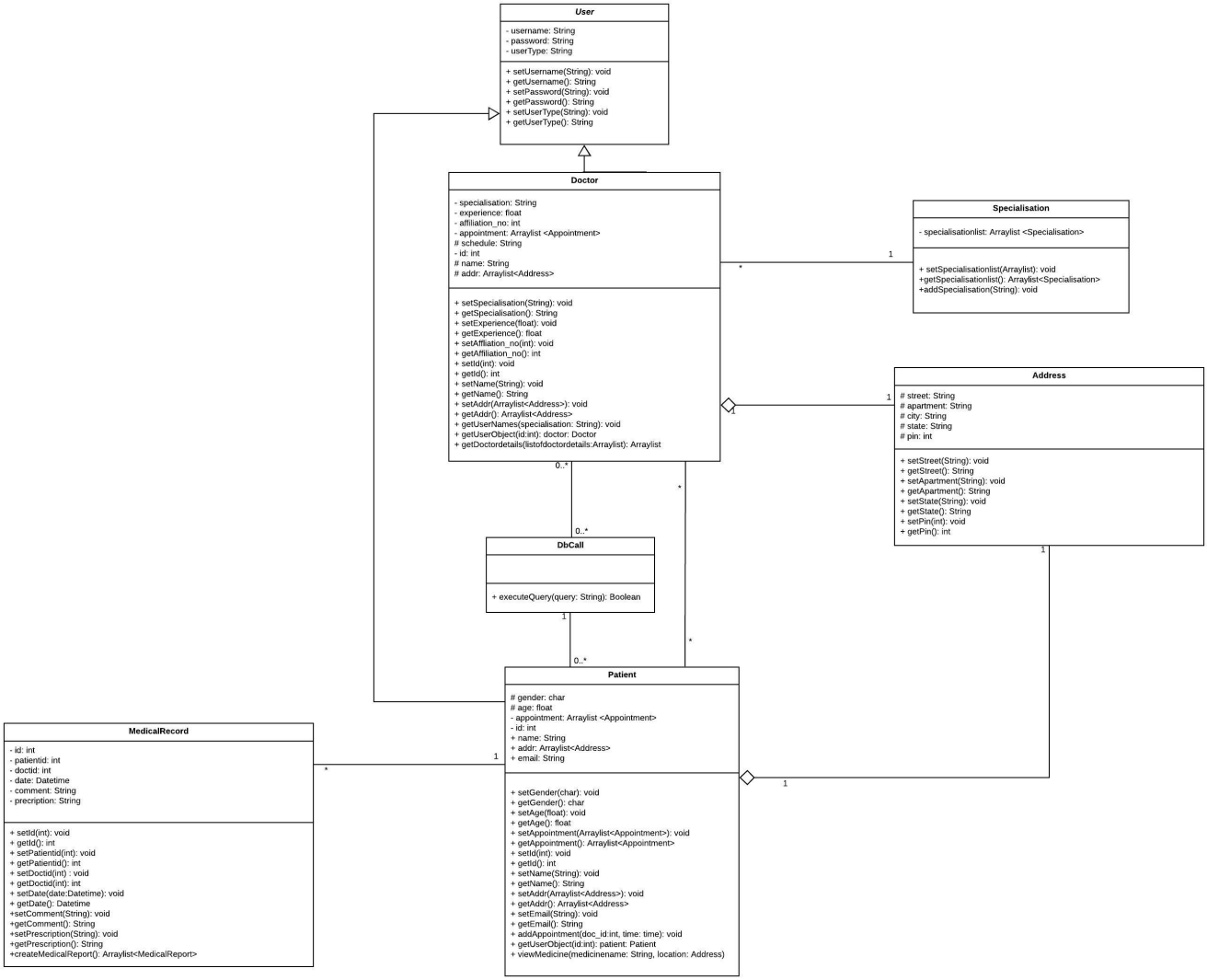
Vision: We are building a platform for CU members to search for doctors, pharmacies, pathologies etc. easily. This product helps people to book appointment instantly and look for medicine availability in nearby pharmacies. People can also consult doctors online anonymously by asking over the platform and get experts' answers.

Project Summary: HealthyBuffs is a healthcare management platform exclusively for students, faculties and other members of CU Boulder where people can locate doctors, pharmacists, pathologists, consultants and book appointments. This platform can also be used for looking up medicines availability in pharmacies. The portal also manages medical records/history of each patient so that it is easier for doctors to make their diagnosis and pharmacists can suggest medicines to patients.

Previous Class Diagram:



Completed Class Diagram:



Summary:

- Corrected the class diagram to include suggested changes.
- We started with setting up Spring MVC on our systems and come up with a strategy of how to implement our idea. Since we are developing a healthcare platform we thought to implement those classes and methods first which are most important for this platform.
- We have implemented following classes in Java
  - Doctors, Patients, Address, Specialisation and Medical Record.
  - Methods in these classes involve functionalities such as viewing and updating medical record for a particular patient. Also, viewing doctor details based on specialisation.
- For this we have setup a MySQL database. We have implemented getting connection with the database using JDBC driver. Functions in the database involves inserting, deleting, updating and fetching records for Doctor class.
- Created Completed class diagram for the implemented classed

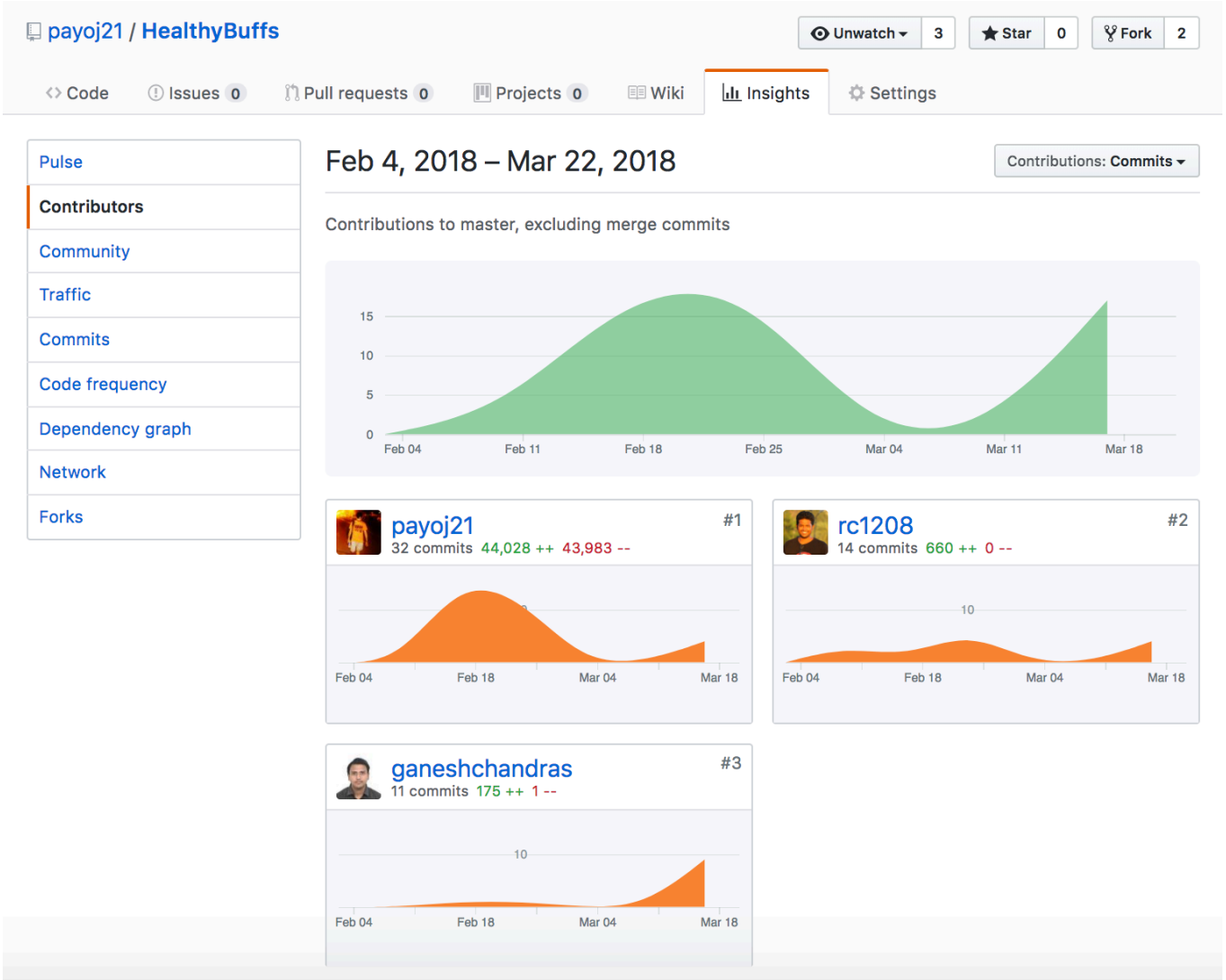
Breakdown:

Rahul Chowdhury: Corrected the class diagram as per the feedback, Created Current Class diagram, implemented the JDBC connection. Implemented classes Doctor and DBCall.

Ganesh Chandra Satish: Documentation, Setup Database, Implemented classes User, Specialisation and Address.

Payoj Jain: Setup Spring MVC, Integrated database connection with the code. Wrote queries to make db calls for functions. Implemented classes User, Patient, MedicalRecord.

Github Graph:



Estimate Remaining Effort:

The remaining work involves the implementation of the Spring Model View Controller. We are working on the service layer and Data Access Object (DAO) layer. We have to implement the DAO layer, where all the database connections and the communication with the database has to be established and all the querying to the database will be done on this layer. Then moving to service layer, where we will be calling different DAO layers. And the controller where all the api's will be created and will call the service layer to fetch the data from the database. We will inject dependency based on the requirement of different functionality in POM.xml. And we will call all xml files that are dependent on controller. We will add those resources in web.xml. For the database

connections we have to create the properties file in the resource folder of the project. And we will also keep track on no cycle creation in pom.xml. For the unit test cases we will work on api client where we will check the stability and consistency of the API's.

Estimated Hours Remaining:

Implementation of classes and design pattern: 20 hours

Refactoring: 5 hours

UI: 15 hours

Next Iteration:

We are planning to implement the rest of the classes with as much functionality as possible. Also, we will be testing out current code on some sample users and try to fix bugs.

We have already setup the important and necessary steps required in the implementation.

A. Implementation of the Following classes:

1. Pharmacists
2. Pathologist
3. Medicine
4. Query
5. Appointment

B. Inclusion and Implementation of the design patterns. We currently believe that using Facade and Singleton design patterns would be appropriate for our project.

C. Unit testing