Project Part - 6 : Final Report

- **Team:** Mridula Natrajan
  Hari Shreenivash Madras Vanamamalai
  Rajarshi Basak

- **Title:** ChatApp

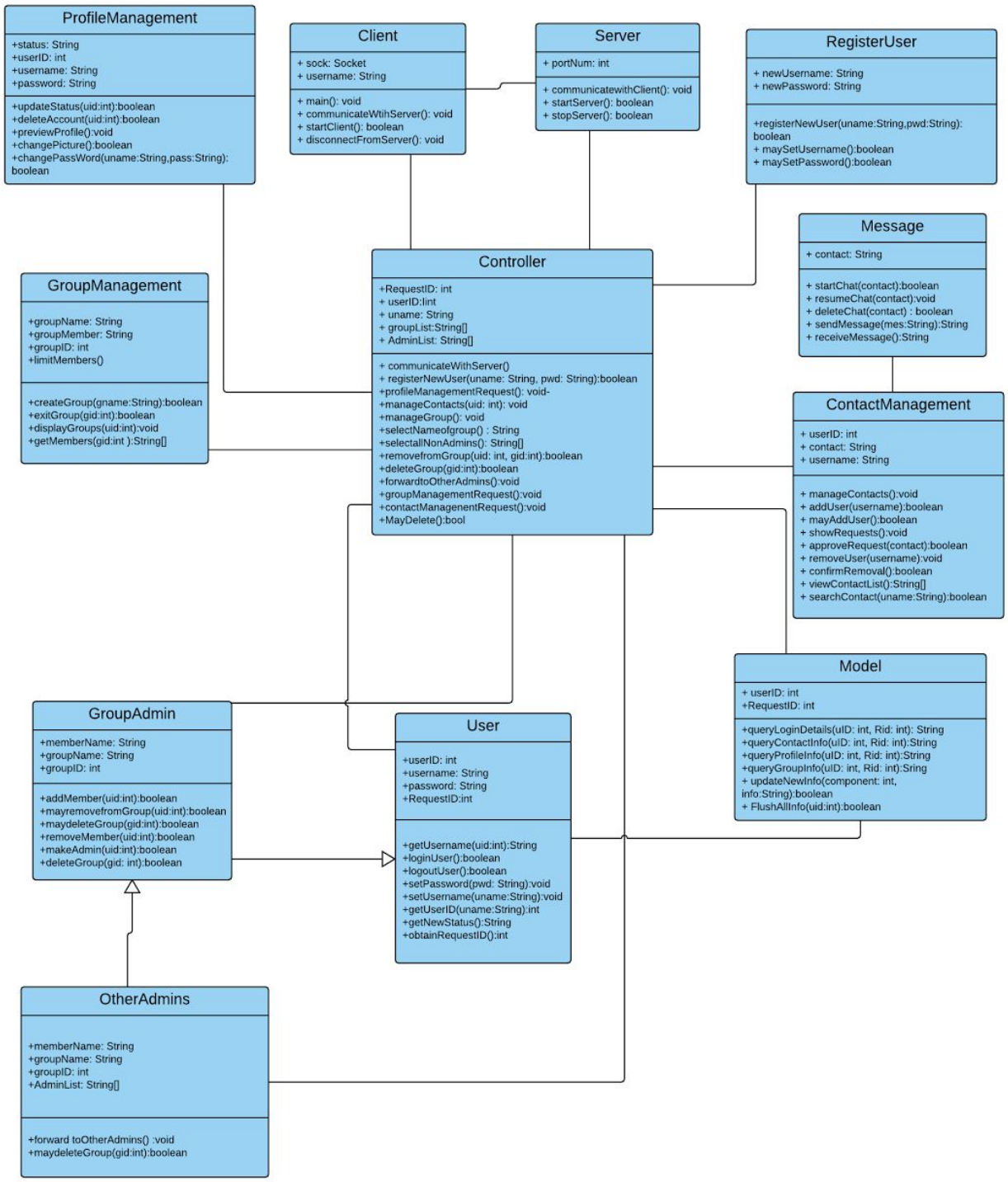
Features that were implemented	
ID	**Title**
US – 01	Register for the service
US – 02	Login to the system
US – 03	Change my display picture
US – 04	Update my status
US – 05	View my profile
US – 06	Delete my account
US – 07	Add a new contact to my contact list
US – 08	Remove an existing contact from my contact list
US – 10	View my contact list
US – 11	Select a contact from my contact list

US – 13	Start a chat with a selected contact
US – 16	Create a new group
US – 18	Add users to a group
US – 19	Remove an user from a group
US - 21	Delete an existing group

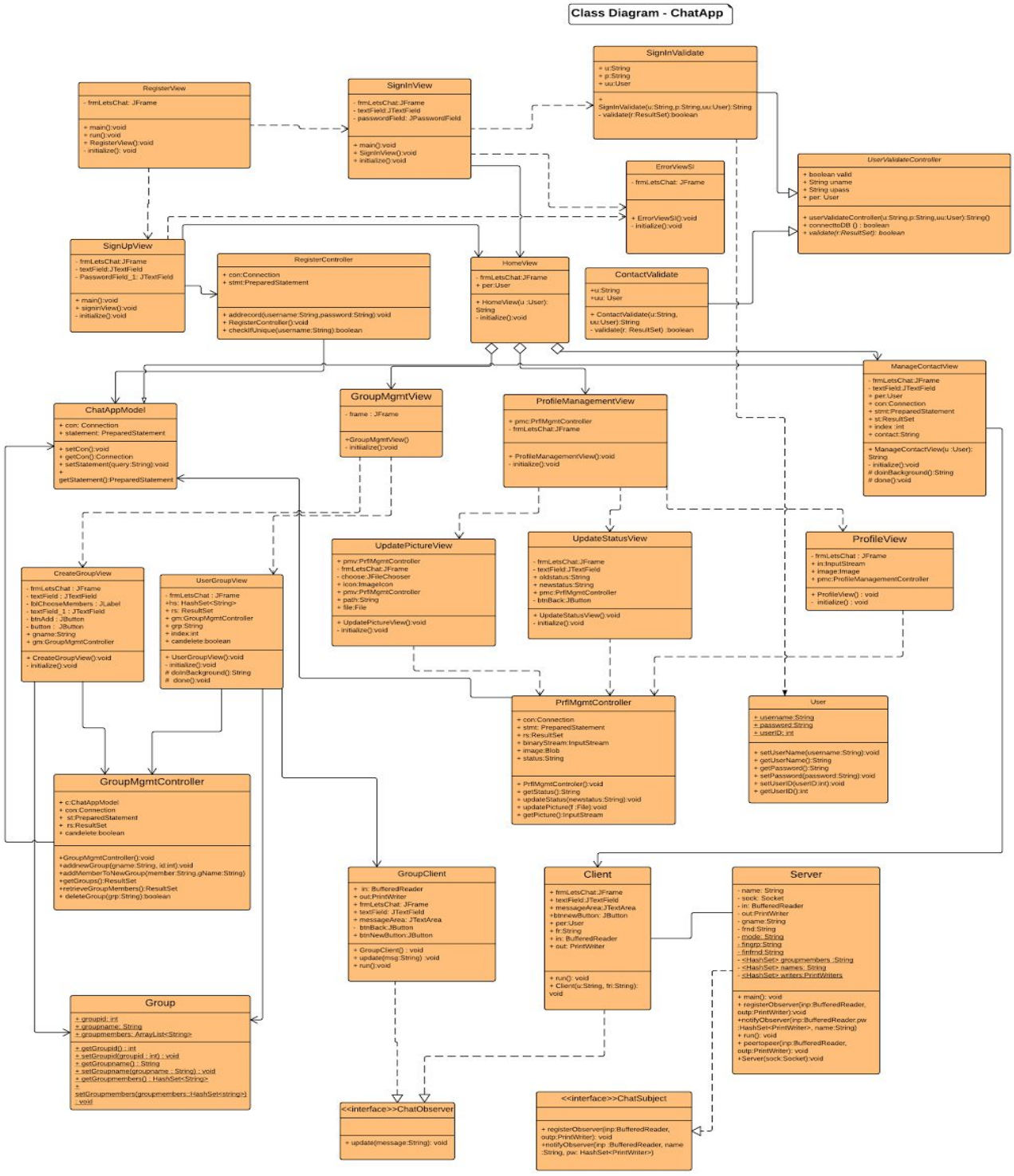
US - 22	Change my password
US - 23	Log out

Features that were not implemented (from Part - 2)	
ID	**Title**
US - 09	Accept a request to add a contact
US – 12	Search a contact from my contact list
US – 14	Resume a cha t with a selected contact when I have already started a chat
US – 15	Delete a chat with a selected contact
US – 20	Make a group member an administrator

Old Class Diagram



Final Class Diagram



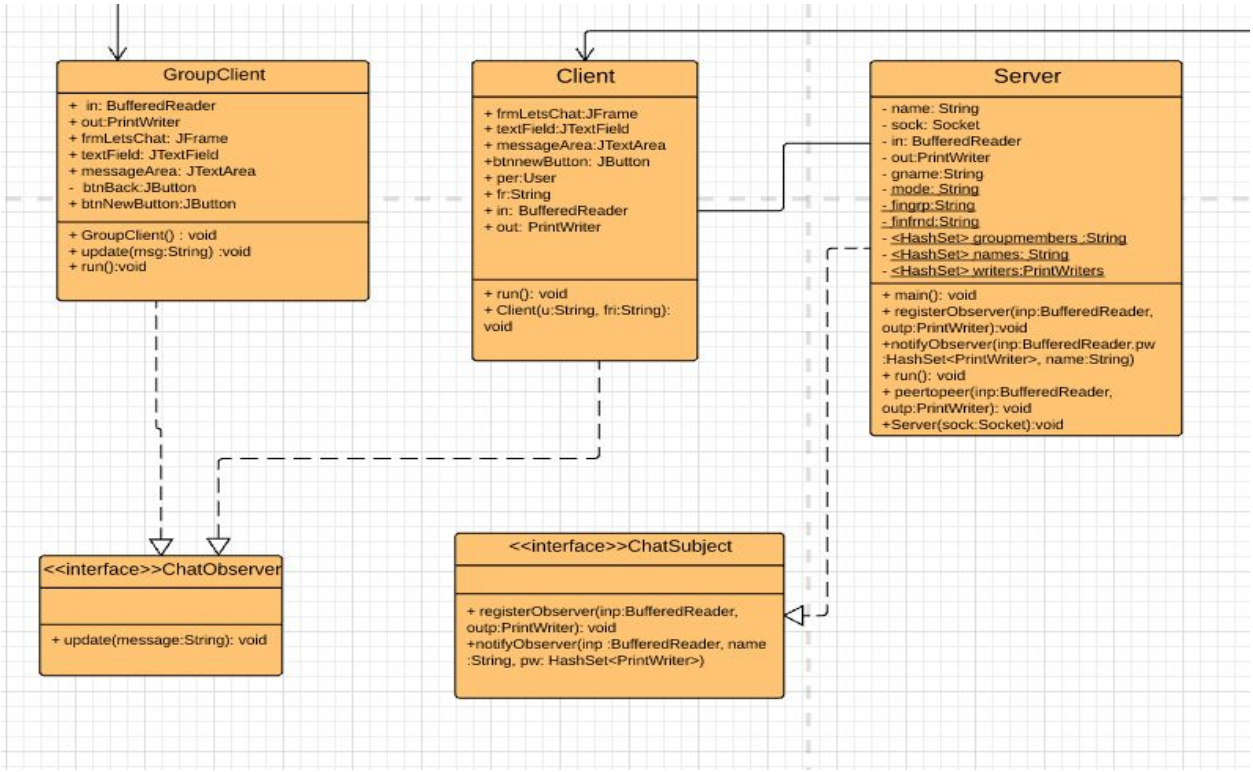
The link is given below for final class diagram:
https://www.lucidchart.com/documents/edit/2d13421c-9c4d-4c91-84be-1df95f98a9f1

Changes that were made:

- Progressing through the code, we realised that our design of having one central controller was not sufficient and caused a lot of confusions therefore we decided to implement a separate controller for each of our Main functionalities- Group Management, Profile Management and Contact Management.
- We also made use of abstract classes and inheritance when implementing validation of a user credential- as this functionality was being used in multiple places, it seemed best to have an abstract class with with an abstract method to validate the contact, that allowed the classes inheriting the abstract class to define their custom validations.
- We also realised that a few classes were not needed that were originally present in the class diagram- such as OtherAdmins.

Design Patterns Used:

Observer Design Pattern



The Observer design pattern seemed fit for our application. The server is implementing the subject class and the observers in this application are clients- peer to peer and group. These clients register to the server by calling the registerObserver method. Once they have been registered, the server stores their output streams and notifies only those output streams which are part of the list of observers. The notifyObserver method helps achieves this. When the server notifies the observer of this change the observer (clients) in this case read in the the changed value from the input stream and display it on the text area.

References

The code for the server and client programs are were referenced from http://cs.lmu.edu/~ray/notes/javanetexamples/

In the server code we have modified the run() method and implemented notifyObserver() and peertopeer() methods. In case of the client we modied the run method and implement the upload method.

What we learnt from the process:

Having gone through the process of creating a software (desktop application in our case) from scratch, which involved

(1) Coming up with the idea of the project
(2) Deciding the functionalities of our project
(3) Designing the project using UML
(4) Coding the application in Java (Eclipse)
(5) Refactoring our design to accommodate the new code and class structure, and finally
(6) Getting it all together (FInal report, Group and Individual Videos, and the Demonstration)

we learnt that the process of software development is a tedious one, but in the end, when it all comes together, it provides the team a sense of achievement having been able to create a real-world application that serves the society.