CPS 888 Final Project Report

**Group ID:** Group 01

**Group Members / Roles**

Jason Yeung (500597019) /Project Manager and Network programmer,

Kazi Farzana Saad (500497835) /Tester and GUI programmer,

Manuel Guanipa Larice (500718372) /System Integrator and Network programmer,

Joshua Kwong Chip (500633419) /System Integrator, Tester and GUI programmer

## Introduction

The purpose of this project was to design and implement an online instant messenger that can support multiple clients through the usage of sockets and a main server. The project was developed with Java and contains an interactive graphical user interface (GUI), which will allow clients to view and send messages amongst each other clients through a console. Clients communicate with other clients through the Server, which then relays the messages from the sender client to the receiving client.

## Requirements

### General Purpose of Software

As stated previously, the purpose of this project was to create an online instant messenger system that could support at least two clients with a server that connected them. Fortunately, we were able to design a main server that could handle communications between multiple clients within a chat room, where they could talk with one another; in other words, a group chat. Java was used to create this project within the NetBeans IDE environment. As was said in the proposal, socket programming was used to create and manage the communication between the server and the clients. When a client wants to send a message to the other clients through the group chat, its first has to go through the server, and then the server would send it to everyone else.

### Functional Requirements

In this project’s proposal, the initial intention was to implement a login system, enable file transfer between clients, and allow emojis to be used in instant messaging. However, in the final product, the following features were implemented:

(1) Login System,

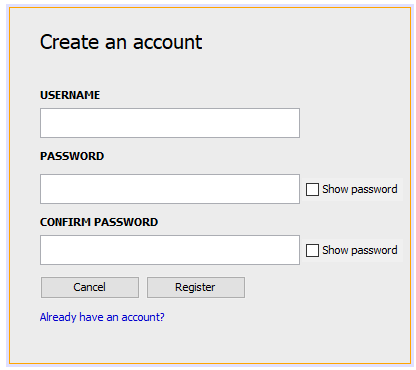
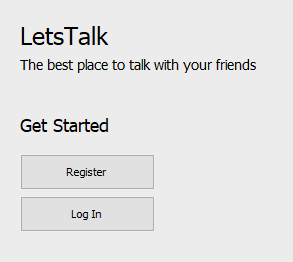
(2) List of Online Users,

(3) Private Messaging,

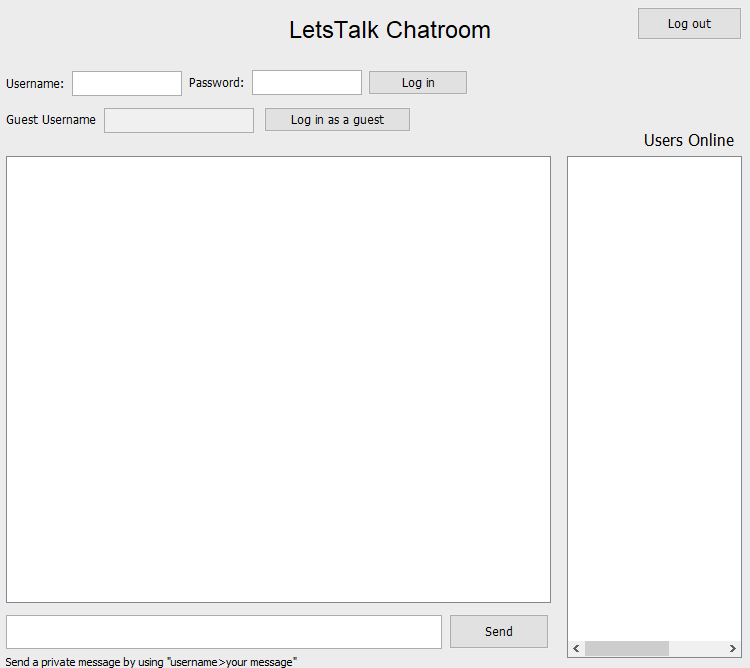
(4) Database Implementation.

After research was collected regarding file transfer and emoji in Swing Java, the group decided the work will be difficult to deviate appropriately, and it was more practical to spend extra time on improving functional and nonfunctional requirements than create decorative features. As a result, file transferring and emoji implementation were disregarded in favor for a database, list of online users, and private messaging.

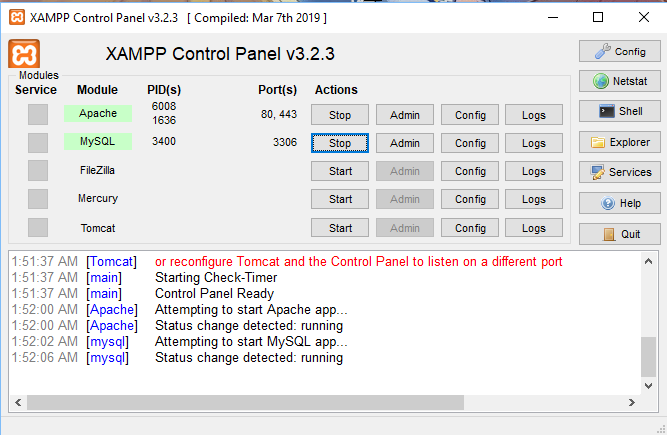
When the instant messenger application was opened, clients are greeted with the chat’s home page, which offers them two choices: register for a new account or log in with a previously owned account (as seen in Figure 1). The login system enforces users to create an account and choose their own personalized username to differentiate themselves from other users through the registration window. This can be viewed in Figure 2. Another additional hidden feature not mentioned beforehand, is the ability for users to log in anonymously if they would like to join the chat as a guest as can be seen in Figure 3. They will appear as “LetsTalk(XXX)”, with the Xs being a randomly generated number. A list of online users will appear on the right side of the client window, and will display all guests and users that are currently connected to the server.

Figure 1. Home Page Figure 2. Registration Page

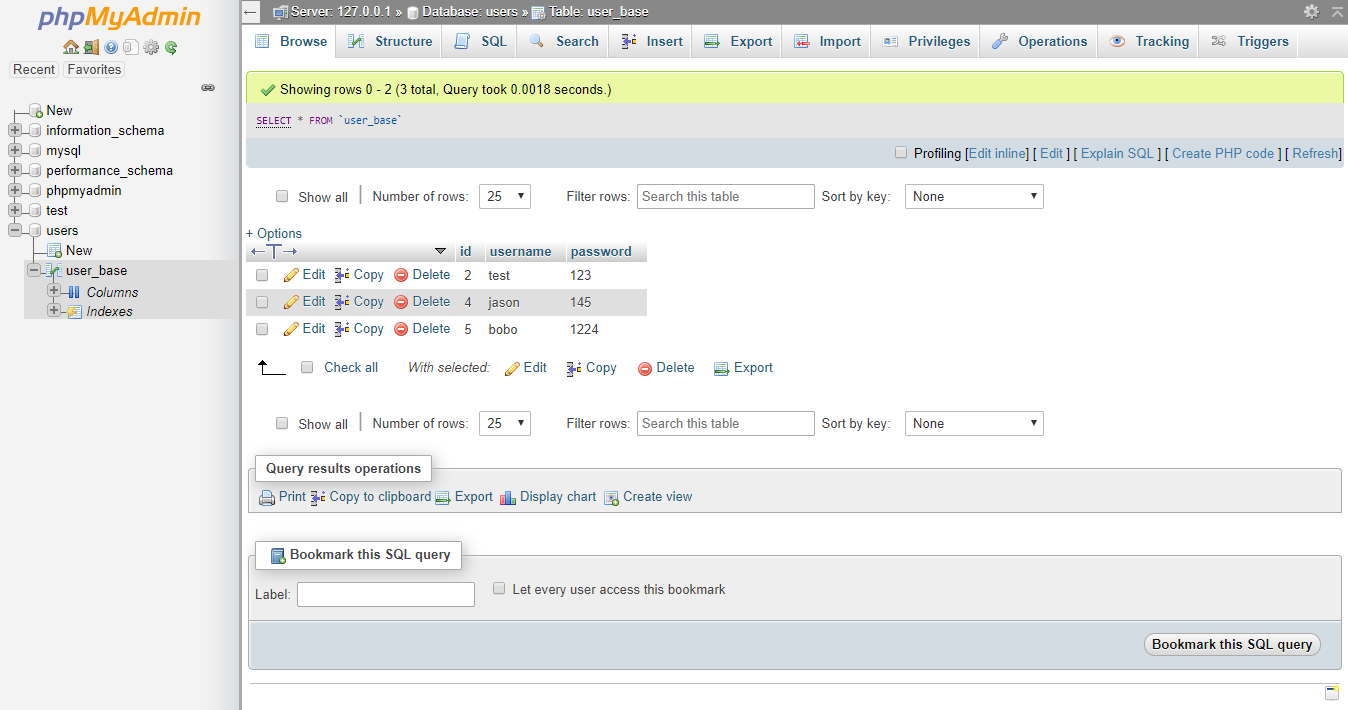
Users can talk with other users or guests by typing their messages within the message field, which is located beside the send button. The private messaging feature can be performed when the username of the person you wish to message is followed by “>” symbol and your desired message. For example, if you would like to privately message Bob the following message, “Hello World”, the following will be typed in the bottommost window, “Bob>Hello World”. A reminder of how to private message is written on the bottom on the client, as shown in Figure 3.

Figure 3. Group Chat Room

The database implementation was implored to save all the usernames and its associated passwords. Whenever a new user would register for an account, their username and password would be saved within the mysql database as long as the username is not already taken. When logging in, the database would be accessed to check if the user exists and if the credentials are 100% correct. To access the mysql database, XAMPP would have to be installed, so that both the apache ports and mysql ports could be turned on. This can be seen from Figure 4. It should be noted that these port numbers and PIDs will vary across different users and different devices.

 Figure 4. XAMPP control panel with the ports on.

When all the necessary ports are online, your localhost phpmyadmin software should be accessible on the web. This is where the mysql database is located as seen in Figure 5. It can be seen that a database called users was created with one table in it that contains 3 columns called: id, username, and password. Id would autoincrement for every new user account created and the username and password would be recorded in the table.

Figure 5. phpMyAdmin software.

The server is one of the biggest parts of the instant messenger. Without a server, there is no communication, and with no communication, there are no conversations going on between multiple users. When the server is opened up, it would automatically start up on it’s own with us not needing to do anything like press a button as you can see in Figure 6. It has the option to view all the online users or clear the servers field of text if there’s too much on it already. The server will display information that’s being sent between it and the clients. For example, it will receive and send all the messages being said through the group chat and it will also show when people log in or log out. To close the server, all you have to do is close the actual window. When the server window is closed, clients will no longer be able to communicate with each other. They’ll know this due to something like “Not connected to server” popping up into the client chat.

Within the client window, it can be noted that there is also a users online area, which shows who is currently logged into the chat. If you’re not currently logged in, nothing will appear, but once you log in, all the users in the chat will be on there. When you or someone else logs out, they will immediately be removed off the list.

Some last things to mention are the multiple ways one can perform an action. For example, instead of having to press send for every message you write to the users in the group chat, you can also press ‘Enter’ on your keyboard. It will provide the same function. It’s basically the same thing with logging in. This can also be done for the logout button and closing the window. If a user were to just close the window instead of logging out beforehand, they will be automatically logged out.

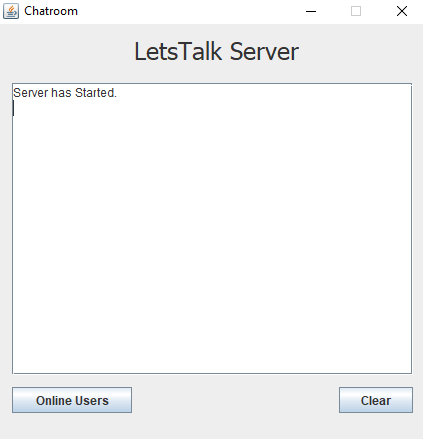


Figure 6. Server Startup screen displaying startup message

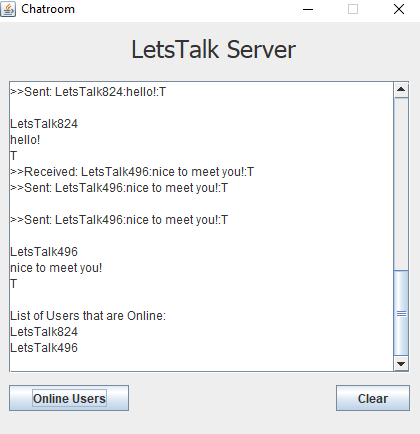


Figure 7. Server displaying online users and messages sent between people displayed above.

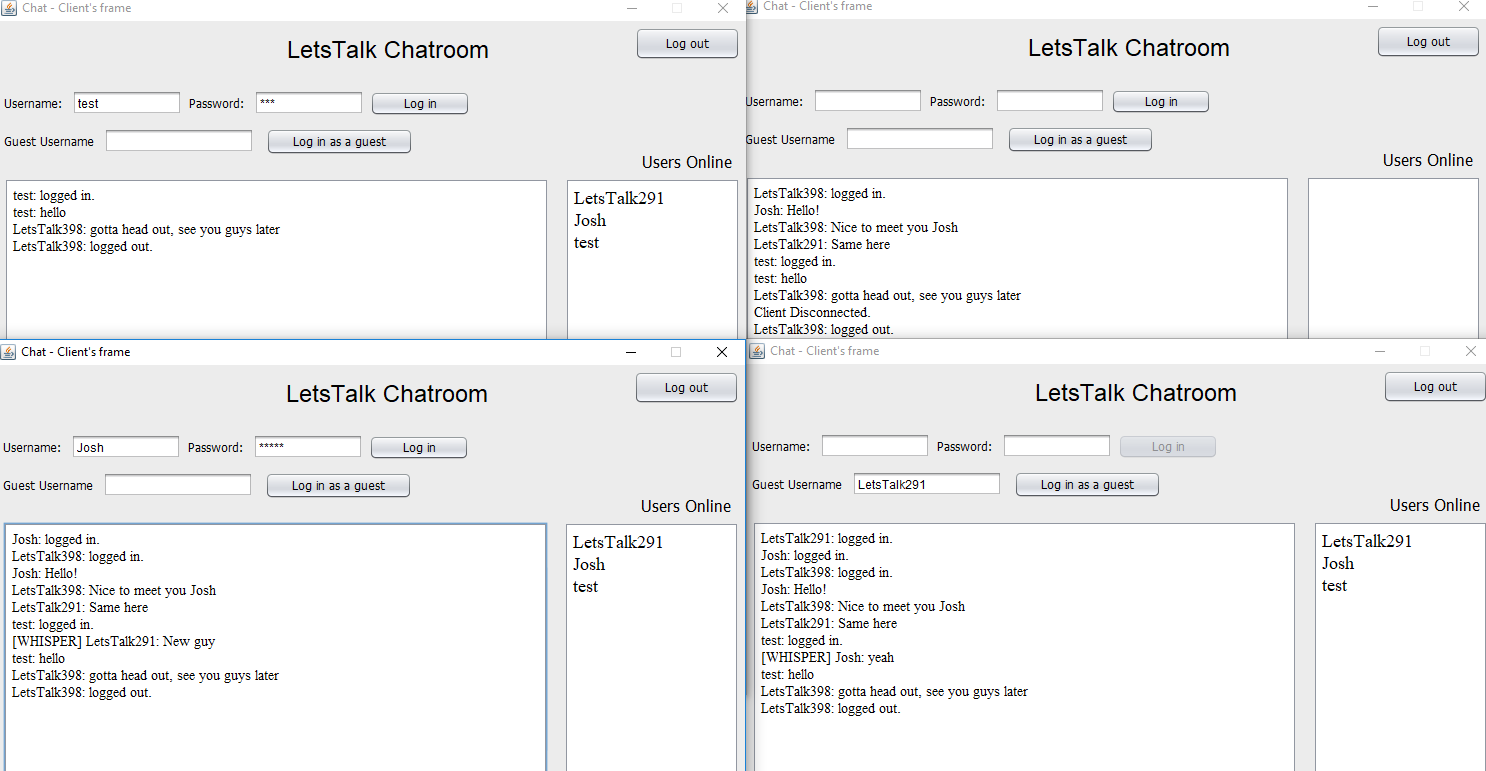
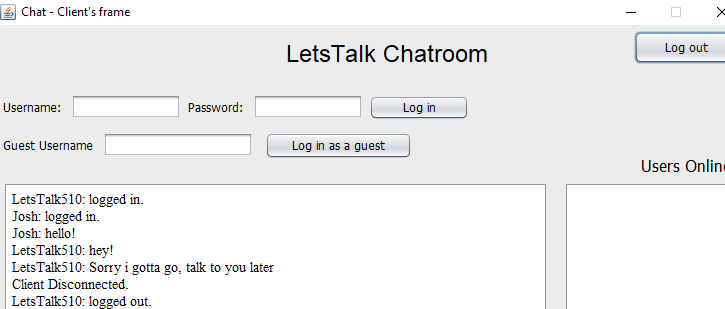
Figure 8. Multiple clients that showcases the multiple features like guest login, list of online users, private messaging, and displaying associated usernames on top field and its associated password in asterisks.

Figure 9. User logging out of client, which will disconnect user from the application and the window can be closed safely. When closing server, the entire window just clears.

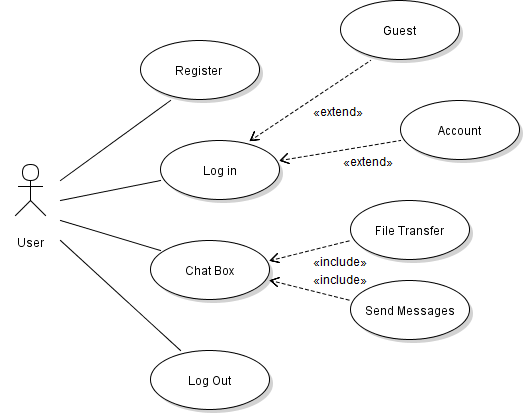


Figure 10. Before version of the use case diagram.

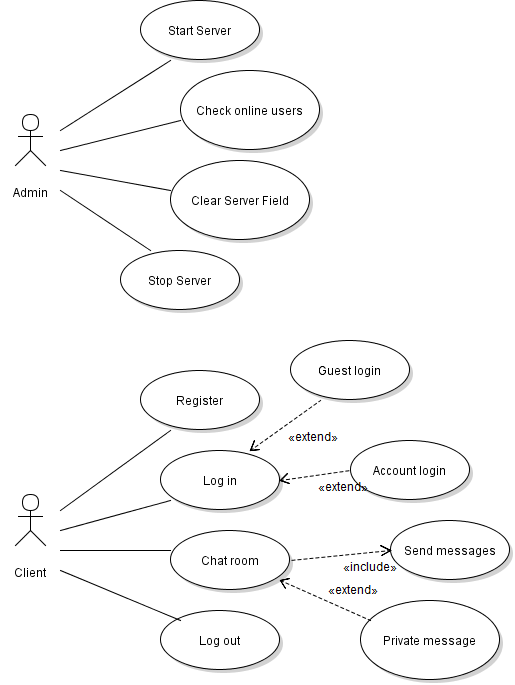


Figure 11. After version of the use case diagram.

Right above are the two use case diagrams for this project, one before the project was started and one after the project was complete. Both use case diagrams are extremely similar, but the before use case diagram includes file transfer, which we didn’t implement in the end, but instead was replaced with private message as can be seen in Figure 11. There is also a separate use case diagram in the after version due to there being an admin that has to turn on and maintain the server part of the instant messenger.

### Non-Functional Requirements

One of the biggest features that the instant messenger contains is the security and privacy it provides to its users by encrypting every message. The messages that clients and server share are encrypted with the Data Encryption Standard (DES), which is a [symmetric-key algorithm](https://en.wikipedia.org/wiki/Symmetric-key_algorithm) for the [encryption](https://en.wikipedia.org/wiki/Encryption) of data. DES encryption was selected because it takes less computing power compared to public key algorithms like RSA and since the messenger deals with large chunks of text it would have been impractical to use an encryption algorithm that will slow down the system with high computational tasks. A symmetric key algorithm uses the same key to encrypt and decrypt the data, this means that all the clients and server contain the same exact key to be able to successfully encrypt and decrypt the messages being shared between them. The key distribution system is done by the server which acts as a Key Distribution Center where the key is generated and then shared in a secure connection to every client. Another way of distributing the key is to manually provide the key to every party interacting in the instant messenger.

A major asset about the project’s features is their fluidity with one another. All of the designed features either work with one or more other features. The login system allows the user to login with a previous owned account, or register a new one. This will require access into another feature, the database, to withdraw a pre-existing or create a new username and password to and from the phpMyAdmin local database (through the XAMPP application). Once a correct login is entered, the user will enter the chatroom with all features enabled, including private messaging and the online list of users. Therefore, it is evident that the features play a role in supporting the other features of this product, creating a great flow throughout this application.

Another nonfunctional requirement of this instant messenger is the ergonomic usability of this messenger. Some quality of life changes were added to this project, to ease the usage of this application for first time users, or users unfamiliar to online messaging. For example, upon registering a new account, the user will be prompted with a new window with a few empty fields. The password fields has a “show password” option as shown in Figure 2, which lets the user see the desired password they wish to use for their new account. Some users may or may not appreciate this option, but this added feature allows users who feel uncomfortable with typing their passwords blindly, enjoy the messenger more and feel at ease when registering a new account.

Accessibility of the messenger and database is the third nonfunctional requirement added to this application. All softwares and database, such as XAMPP, Netbeans, and MySQL, are open-source programs so any users who wish to use this messenger has the ability to access it.

A couple constraints were encountered and resolved during the collaboration of this messenger. Firstly, some features (such as the database implementation) required more time to implement than planned. This forced the group to reconsider the important features that should be implemented. As a result, the group decided the database and login system are important features to maintain, whereas file transferring and emoji implementation were scrapped for more practical features (such as private messaging, and a list of online users). Another major constraint was the usage of Github as a web-based hosting service. During the group’s tenure on this project, the inexperience of our team members caused some irreversible changes to our Github, such as committing duplicate folders, or pushing files into the master branch instead of working branches (overwriting previously working code). To circumvent this issue, the consensus was to use Google Drive as the main platform for any work done. The folders within this drive were dated, archived, and documented with README files (as shown in Figure 12) to keep track of our progress, and the work was later committed into Github as a group during the final phases of the project (as shown in Figure 13).

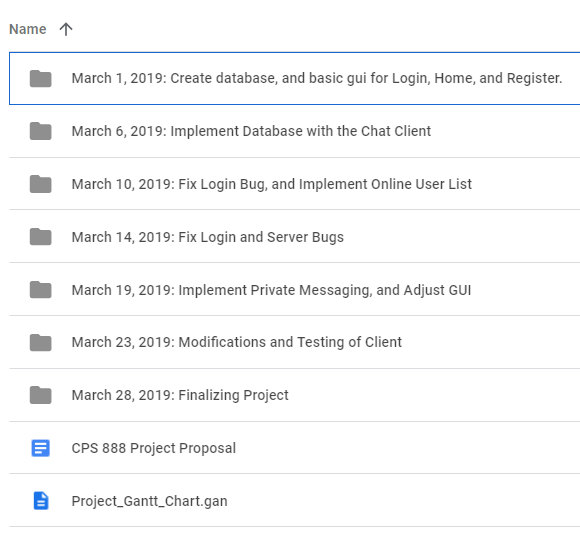
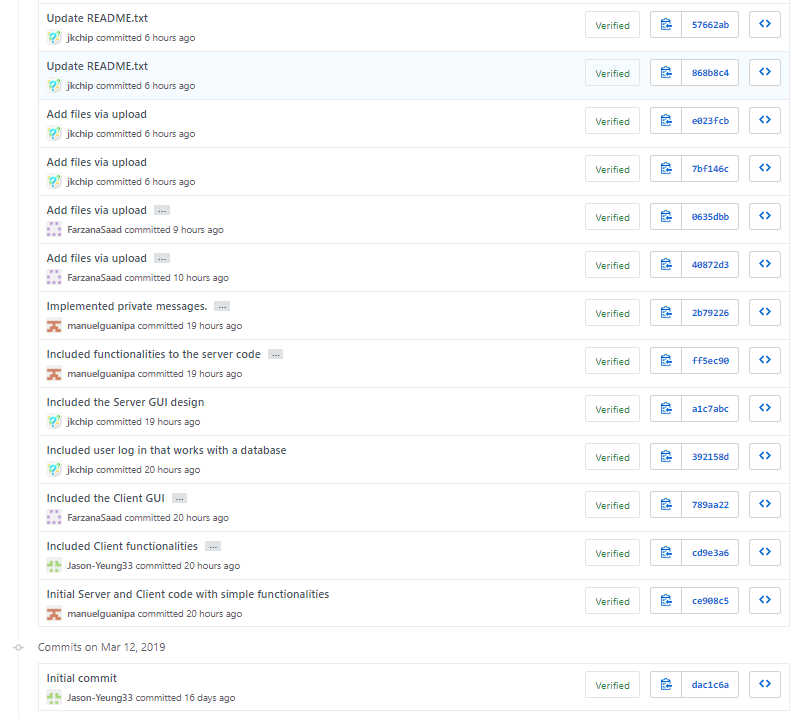
 

Figure 12. Google Drive Commits Figure 13. Github Commits

## Software Design

The application was developed using Java and the NetBeans IDE. The application consists of 5 major classes , Server, client, Register, HomeScreen and My\_connection, that communicate with each other to provide the desired functionalities. The class diagram, showed below, provide a depiction of the attributes and methods used by each class along with the relationship amongst them.

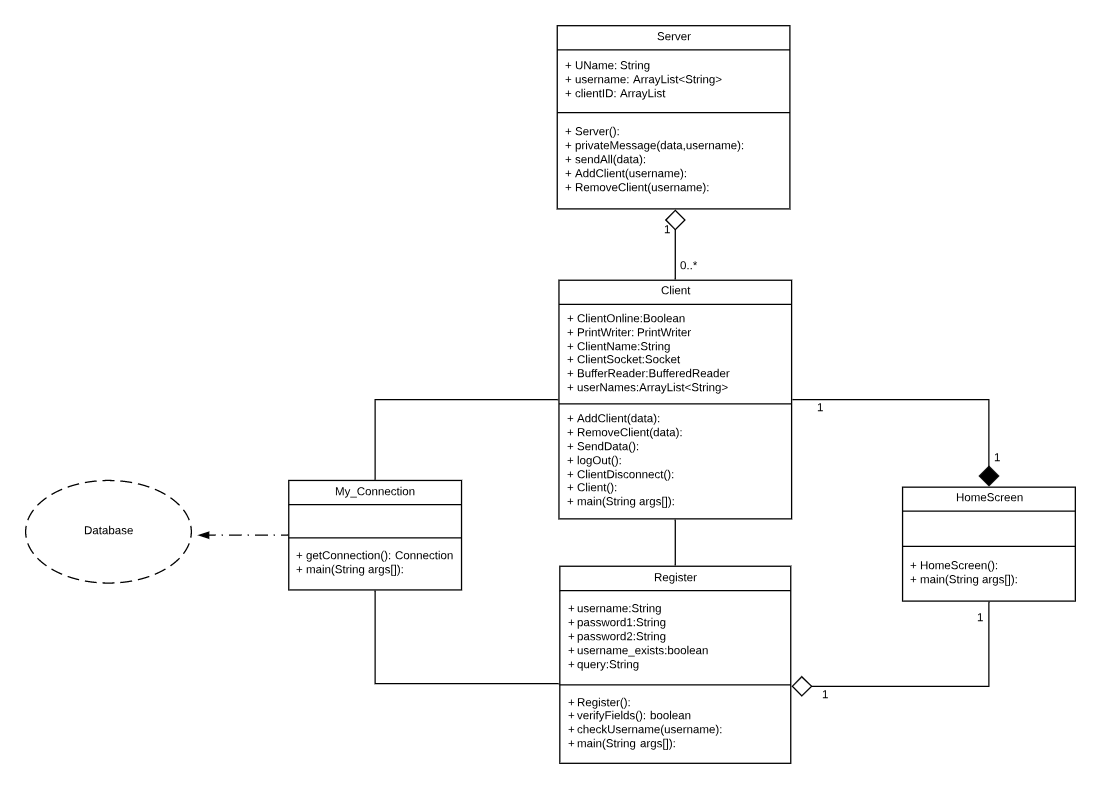


Figure 14. UML Class Diagram.

As described previously, the application starts by starting the server. This is done using the Server() method in the Server class. This method calls the initcomponents() method that initializes the GUI component of the server. The server calls the client class when a new client logs in using the AddClient() method and RemoveClient() when a client logs out. The private message() method in the server class is used when a client wants to send a personal message to another while the sendAll() method sends a message to all the clients currently in the chat. The server class then connects with the client class in an aggregation with a one to many relationship signifying that one server can support multiple clients whereas one client can have only one server.

The client class has several methods that are used when a client wishes to login and register, to send messages, log out and to disconnect. The Addclient() method in the client class is used to add the client to the list of online users in the chat, whereas the RemoveClient() method removes the client from the list. The SendData method is used to update the online list of clients. The ClientDisconnect() method is used to disconnect the client from the chat after they have logged out. This client class communicates with the HomeScreen class when the initiated. The HomeScreen class gives the user the option to login using an existing username and password or they are asked to register. If the client chooses to register, they are then directed to the register class. The Register class asks the user to input a desired username and password. The username is then checked using the checkUsername() method and both fields verified using the verifyFields() method. Once, the username and password fields are filled by the user the register class calls My\_Connection class. This class deals with updating the database with the new data inputted by the user. Though the database is connected to only the My\_Connection class, the client creates multiple clients by recalling the My\_Connection class, and therefore allow each instance of the client to connect to the database.

## Software Architecture

There are two main architectural patterns that we used and implemented into our instant messenger. Those would include the client-server pattern and the layered pattern. For the client-server pattern, it is basically a distributed system model that has a set of stand-alone servers that provide different functions across a network to a set of clients who use these specific services. In our case, we had two separate kind of servers, which included a database server and a chat server. If a client wished to utilize the log in function of our application, then they would need to call on the database server in order to register their information and retrieve it to log in. On the other hand, if a user or guest just wanted to chat on our messenger, they would need to connect and use the chat server to be able to send messages to one another. For the layered architecture pattern, it can be seen that our instant messenger works in layers with the top of the diagram being the presentation level to the bottom being the data level. For our project, the top layer would be the user interface, follow by the registration, user login, and guest login functions where users can communicate with the software, to the types of messaging and encryption for the chat, and then finally the user database and chat server.

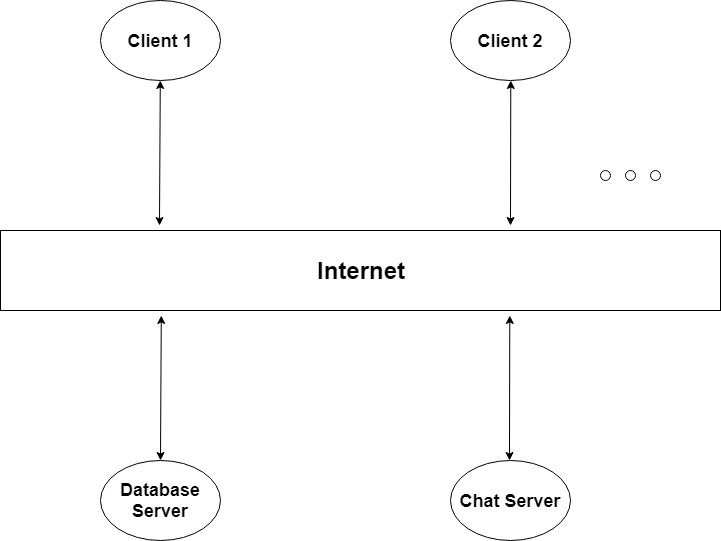
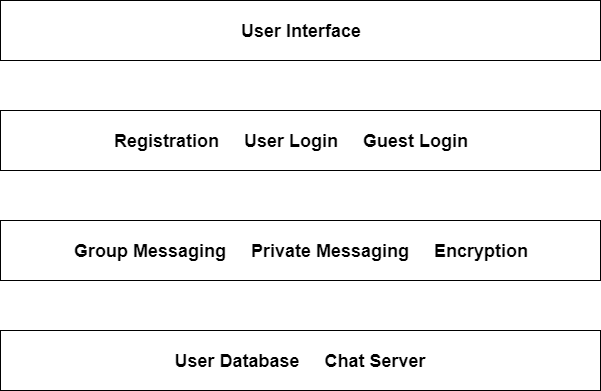


Figure 15. The client-server architecture pattern diagram.

 Figure 16. The layered architecture pattern diagram.

## Software Testing

### System Testing

System testing was implemented and done for this instant messenger. Basically, tests were done to make sure that the completed software met the requirements that we decided on. To do this, we mainly implemented try-catch statements. If the features didn’t function correctly, the program would catch that error and print out some kind of message. For example, the server and messaging system. If the message of the users didn’t go through to the other people in the chatroom, it would instantly catch that problem and print out a message saying ‘message was not sent’. If a catch was found, we would have to go through the code again to fix it to make it to meet our final requirements, but if the try went through and it performed the task we wanted it to, it would be a success. Another example could be when someone tries to connect or log in to the client chat. If the log in goes through properly, great, but if it doesn’t, it will catch the error and print out something like ‘cannot connect! Try again.’ This kind of system testing was done for all of the features implemented into this software.

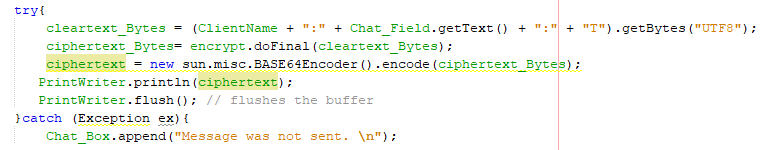


Figure 17. Message not sent catch statement.

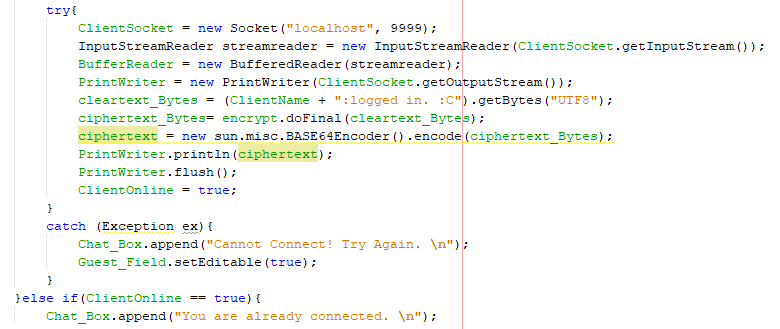


Figure 18. Cannot connect catch statement.

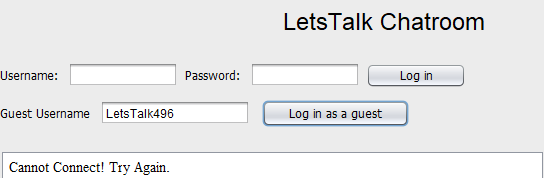


Figure 19. The cannot connect catch statement going through on the chat.

### Unit Testing

Unit testing was implemented at several developmental stages of the instant messenger. It was infeasible to use JUnit testing with a GUI, but it was possible to test the different methods using the Netbeans Console. For example, it was possible to unit test the private messaging by returning the expected values at the end of the method. Specifically, before the chatroom displays the user’s name and the associated private message to the client, the console will show where the two strings are stored in the temporary data array. As you can see in Figure 20, the “test” user is privately messaging user “LetsTalk716” a message, and the message was received by the latter successfully. In Figure 21, the console displays the Strings stored in the two elements in the temporary array: the username and the message. By displaying the array contents onto the console (implemented on line 98 and 99 on client.java), the method that implements that private messaging is deduced to work properly since the expected message and username entered into the client, is appearing as intended in the console with the appropriate client and message.

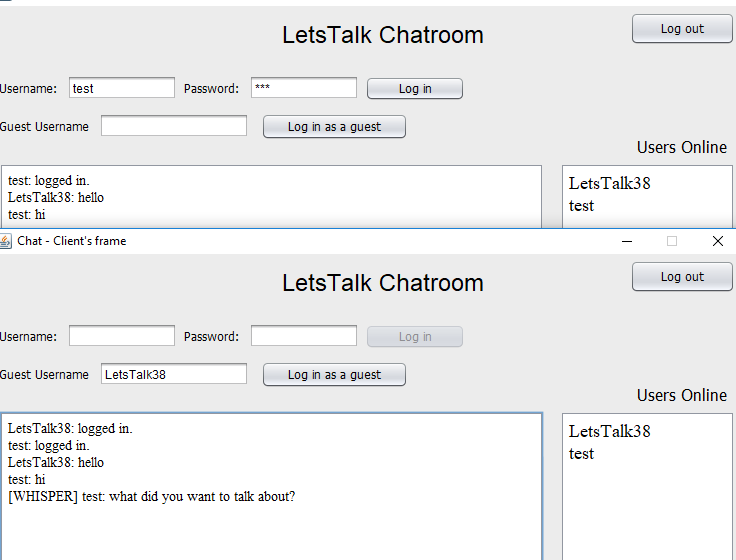


Figure 20. Private messaging between two clients

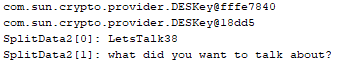


Figure 21. Netbeans Console returning the contents of the array elements

## Github Link

The following github repo was used during the course of this project in order to commit and share code.

<https://github.com/Jason-Yeung33/CPS-888-Software-Engineering>

The team also used personal messages through facebook/Whatsapp/In-person in order to collaborate and communicate the needs of the project. The team shared google drive also was used to exchange and collaborate.

## Contributions

Jason: - Project Management

- Socket Programming

- Testing

Manuel: - Socket Programming

- GUI Integration

- Testing

Farzana: - GUI Design

- Testing

- UML diagrams

Joshua: - GUI Design

- Testing

- Database Implementation

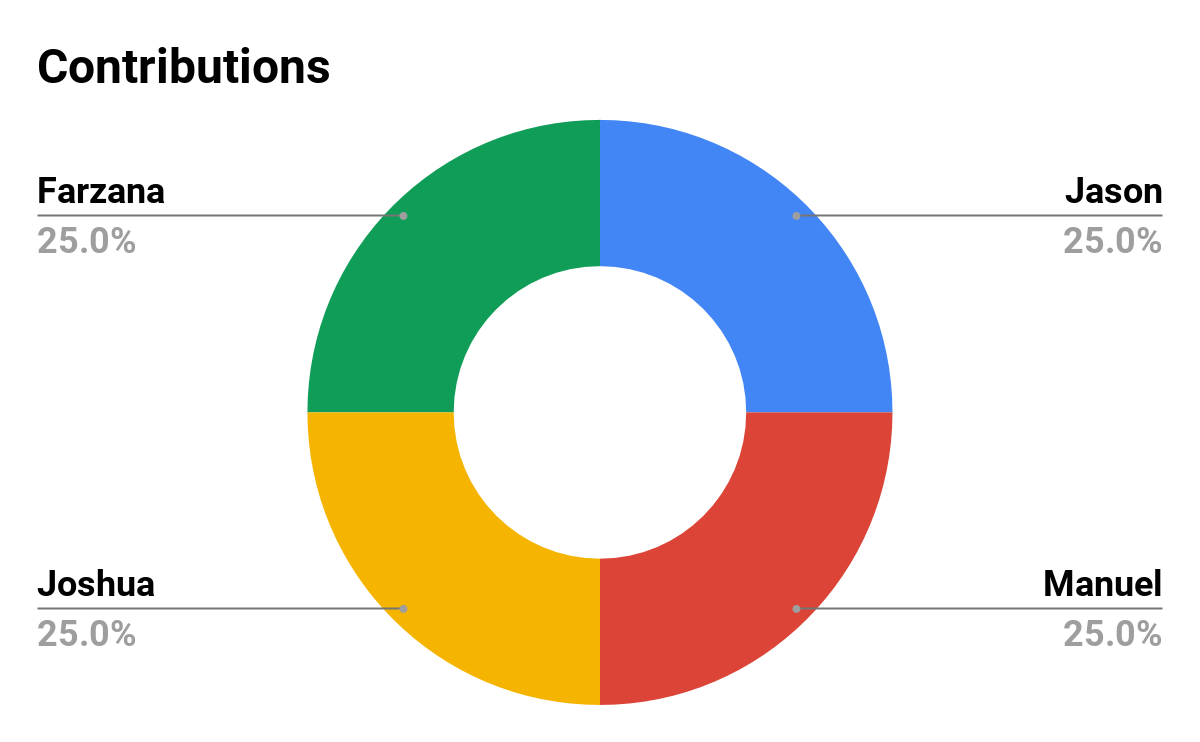
****

Figure 22. Approximate Contribution Percentages.

## References

<https://www.inloox.com/company/blog/articles/a-guide-to-dependencies-constraints-and-assumptions-part-2-managing-constraints/> (Project Constraint Model)

<https://www.digitaldesignjournal.com/best-free-bootstrap-chat-templates/>

<https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.14/gtps7/s7symm.html>

<https://courses.ryerson.ca/d2l/le/content/254308/viewContent/2326382/View>