Communication Project

Group 4

Anh Truong

Nicholas Bui

Shih Tung Yapp

Revision History

| Date | Revision | Description | Author |
| --- | --- | --- | --- |
| 7/17/2022 | 1.0 | Initial Version | Anh Truong, Nicholas Bui, Shih Tung Yapp |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1.1 Introduction

Communication program that allows staff to chat with each other privately or in a group. Admin has the ability to look at logs of all users. Chat is limited to text only, no images.

1.1 Goals and Objectives

This document describes the implementation of all methods used for a chat system using a client and server.

1.2 Statement of Scope

Decisions in this document are made based on usability, simplicity, and maintainability.

1.3 Software Context

Communication will be serviced through the **clientHandler** objects. The various commands from the **client** will be processed by the client handler which will interact with the **server** to provide all chat services.

1.4 Major Constraints

Issue 1: How would the user connect to the server?

Option 1.1: TCP/IP connection through ClientConnection objects requesting access to the server. Server will grant access to the client request.

Option 1.2: Connecting through local host based on the requirements.

Decision: Based on the requirement document, the Communication program would be required to be connected through TCP/IP.

Issue 2: Where to store logs?

Option 2.1: Store all messages on the (users) client and send to ChatLog objects. ChatLog objects will store into a separate .txt file and will be read by the server upon a client request.

Option 2.2: Store all logs in a database.

Decision: Option 2.1 is selected because database implementation would take longer time as developers have no database experience.

Issue 3: How to update user logs?

Option 3.1: Update logs of both sender and receiver and all members if it is group chat.

Option 3.2: Centralized all chat logs into one text file. Update chat log once only.

Decision: Option 3.2 is the best option because there is no need of keeping up with multiple .txt files.

Issue 4: How to load messages?

Option 4.1: Look for a text file that matches username.

Option 4.2: Chat class object with Chat ID and use loadLog method to pull up all messages.

Decision: Option 4.2 is selected because it provides a method loadLog to easily pull up messages and there is no need to search through .txt files.

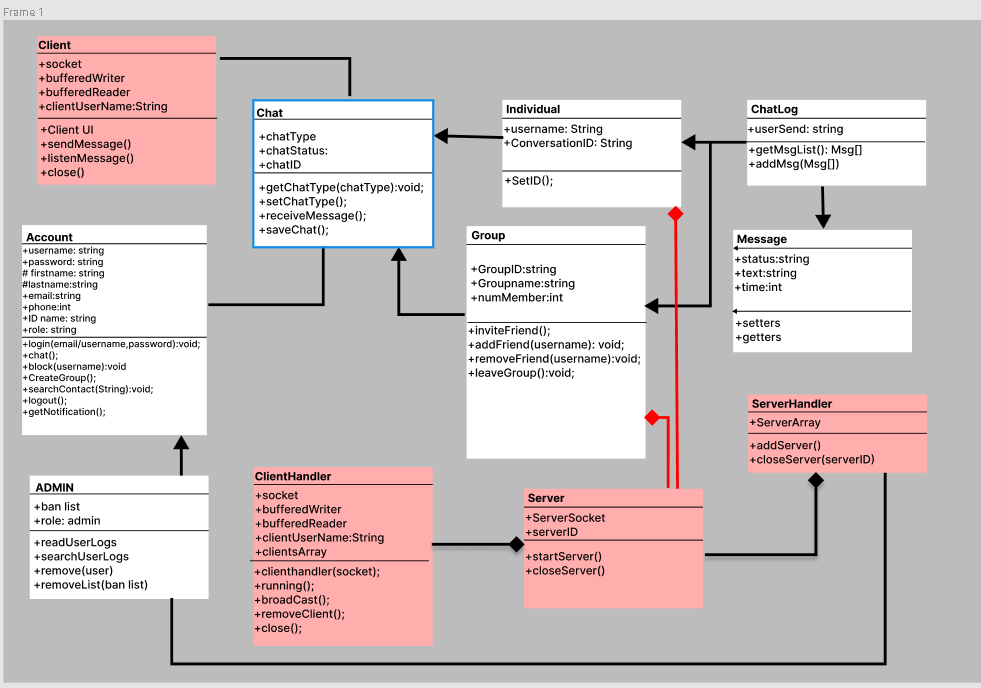
Issue 5: How will the server handle client requests?

Option 5.1 : Client handler will send server requests and

In client class, we can use buffer Writer, and Object Output Stream, to pass an object into the socket. Then, in server class, we can use buffer Reader, and Object Input Stream to listen from the client. Then, use while loops/boolean to broadcast.

Option 5.2: Multiple Clients connected to a specific server at the same time.

Decision: Option 5.1 is the best option because ClientHandler class will handle all Clients and no need for multiple Clients trying to connect to the server.

2.0 Data Design

2.1 Client Side

Descriptions on methods

* **Account Class: *#login*** will be used first when the users enter the UI, users will have to input the name, password. Then, ***#chat*** will call the ***#Client UI*** to update the interface.

***#block(username)*** will delete the chat from the others users if it is on the Client UI.

***#CreateGroup*** will create an Group server with information user input.

***#searchContact*** will find the others Clients username on Client UI

***#logout*** will turn off UI

Whenever ClientHandler ***#broadcast something***, ***#getNotification*** will store that information and pop on Client UI screen.

* ***#listenMessage*** will listen for Message objects.
* **Group:*#addFriend*** will give the user a textbook to enter another user’s ID and add the ID to the user’s friend list.
* **Group:*#removeFriend*** will delete the entered user ID from the friend list.
* **Group:*#inviteFriend*** will send a request to another User to join a group message.
* Sending Message Implementation

**Client Class:** ***#sendMessage*** will send the message content to chatLog, chatLog objects will execute ***#addMsg*** add message will use buffer Writer, and Object Output Stream, to pass an object into the socket. Then, in server class, we can use buffer Reader, and Object Input Stream to listen from client. Then, use while loops/boolean to broadcast through client handler.

***#listenMessage*** will automatically listen to ClientHandler using buffer Reader, and Object Input Stream.

***#ClientUI*** will update the UI interface, so users can see the message from the UI instead the terminal output.

* ***#close*** will terminate the ***#ClientUI*** application.
* **Server Handler:** pick the server from the server Array.
* **Client Handler Class:** ***#receiveMessage*** will automatically listen to Server using buffer Reader, and Object Input Stream. ***#broadcast*** will use for each loops to send the objects from the servers, to send to Client.
* **Admin:** will have access to ban list. Ban list functions will use inherited ClientHandler functions (**#removeClient**, **#addFriend**) to manipulate a list of users and change the account roles to banned.
* **ChatLog: *#addMsg***will create new instance of a message class.
* **ChatLog:** ***#getMsgList*** will retrieve the Msg array from the message .txt files for the users requesting their chatLogs from the client.

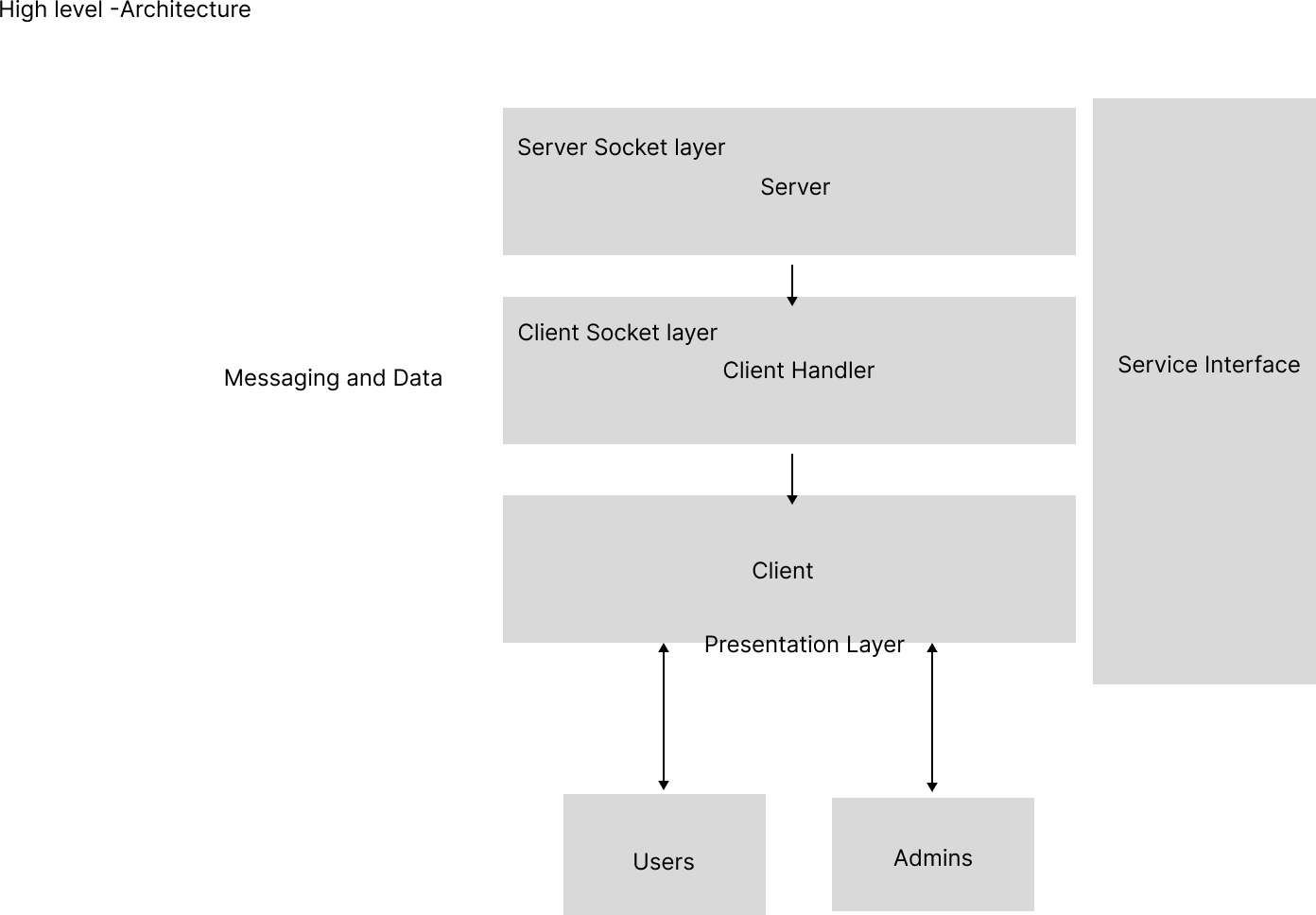
2.2 Server Side

* **ServerHandler:** ***#addSever*** add the server object to //ServerArray

***#closeServer(server ID)*** call ***#closeServer*** from specific server class to close server.

* **Server: #closeServer** would close the specific server that is open. No server can be closed if no Server is open.
* **Server:** **#startServer** will create the first instance of server.

3.0 Architectural and Component-level Design



3.1 Program Structure

The Chat system runs as a client-server application. The application will provide a GUI for communication interfaces which will send and receive data from the Client handler. The Client Handler will communicate with the server to provide real time communication between users on the client.

3.2 Description of Client

3.2.1 Client processing narrative

Every user would interact with their client to send messages to users or groups. User’s text will be stored in a Message object and be sent with sendMessage method in Client class. Client will send the object to the Server to further forward the message.

3.2.2 Client interface description

Client will read the text entered into the text box by the user. This text will be stored in a Message object for the Client’s sendMessage method to send over to another user or group. When a message is sent from another end, listenMessage will receive the message. This new message will show up on the receiver’s end.

3.2.3 Client processing details

Message object that is received and sent needs to be evaluated to who its receiver and sender is. After the sender and receiver is determined, the Message object is sent to the Server. The Server would send the Message object to the intended receiver.

3.3 Description of Server

Server Handler would create a Server object for every user who successfully logs in. Server will be closed when the user chooses to log out or a user disconnects. Server object is connected to Client object which receives and sends messages. Server will take in a Message object sent by Client and determine which user to forward it to.

3.3 Software Interface Description

3.3.1 External Interfaces (Public Class)

Account details will be stored into a text file and will be public for the client to manipulate user details and the client handler will process changes.

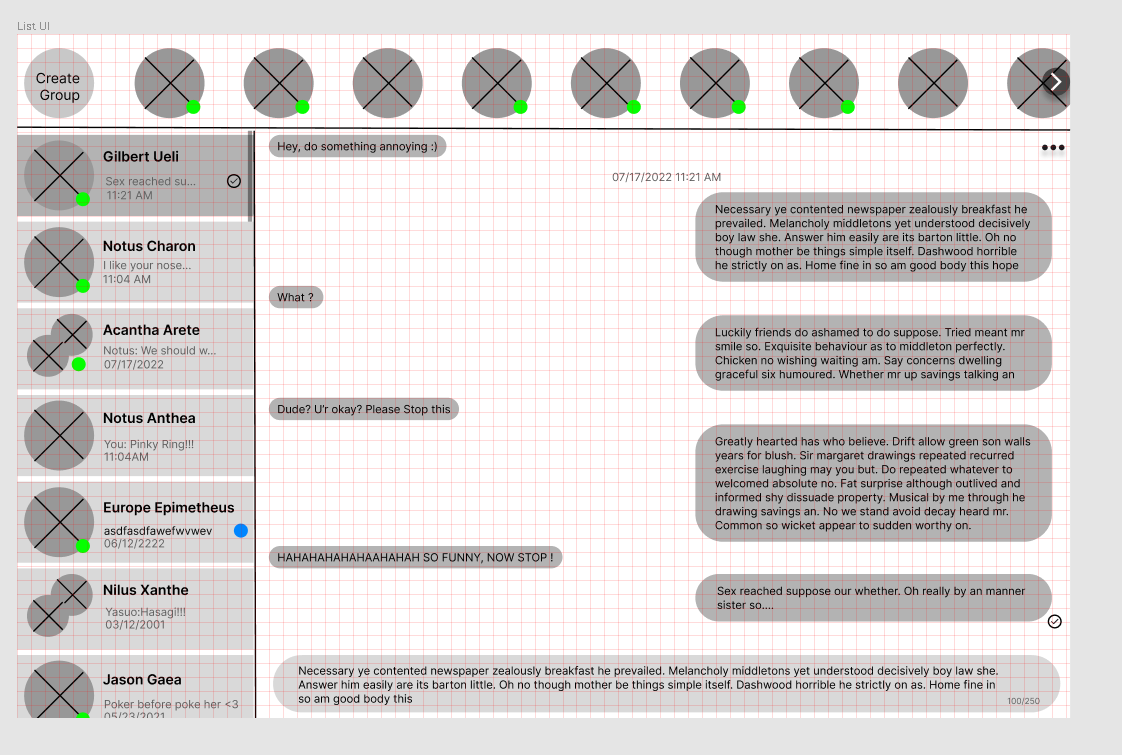
3.3.2 Internal Interfaces (Private Class)

Client, Client Handler, Server, Server Handler, will all be private interfaces processing all the communication processes for the user.

3.3.3 Human interfaces

* GUI: Buttons will be used to execute interface commands.
* Registration/User Profile: Text boxes will be used to store the user information fields.
* Messaging: A text box is available for the user to enter their message.
* A green dot beside a username would indicate that the user is online.
* A blue do t on a chat is a notification that there are unread messages.
* Bubble on the left is a message sent from the other user while bubble on the right is a message sent by the user.

4.0 User Interface Design



5.0 Restrictions, Limitations, and constraints

* As per the requirement, the communication program will be limited to text messages only, and images and videos are not allowed.
* There will be no encryption and privacy is limited because admins will be able to look through logs of all users.
* There is a group size limit of 20 because an overpopulated group would decrease readability and the server may not be able to keep up due to limited bandwidth available.
* Data is stored in .txt files because implementation of database storing is not required.
* Text size is limited to 256 characters as per the requirements.
* The program will be written in Java. This allows the program to be cross platform.
* The strength of user password will not be tested as implementation of password testing would be very time consuming.

6.0 Testing Issues

6.1 Testing the maximum number of users connected to the server at one time. There is not enough test subject to test the limit of the server.

6.2 Incomplete Coverage - Full manual regression testing will be challenging (Different OS, multiple platforms).

6.3 Facing biases.

6.4 Lack of experience for creating test plans.

6.5 Insufficient Requirements Gathering - Inadequate or insufficient requirements analysis has consequences on the quality of the message software.

6.6 Conflicts and schedule delay impact quality, availability and efficiency of test environments

6.7 Testing on client and server interaction could be difficult because there is limited computer to do the test.