Project 5 Report

Team 65 - Jennifer Cahillane, Rishabh Chaudhary, Kenzie Duby, Leo Li, Jiayu Zhou

Part One: Project Description

Overall Description

For this project, our team implemented Option 3, which was developing a social network messaging application. This application is a user-friendly program that allows for creating easy connections with new users. There are three different graphical user interfaces associated with this project. The first is the login and user profile page.

When the user logs in, they have the opportunity to either sign in to an old account or create a new one. Our program notifies the user if the account they are attempting to create or sign into is valid, as determined by the rule that each user must have a unique username or identifier. Furthermore, if a user has been banned by the moderator, they will also receive this message. Multiple users are allowed to use this program at any given time, as allowed by our multi-threaded server. Once logged in, they will have the option to see their chats or edit their profile. They may change the background and text colors of the application and edit the profile. The color change application asks for red green and blue color values, which range from 0 to 255. This is a great aspect for users who prefer dark modes or have trouble seeing certain colors. For the option to edit the profile, the user can change their password if they would like to. They enter the new desired password twice, and after making sure they match, the password is changed.

Once the user accesses the chats, they are shown a screen that displays a scrollable list of their chats, ordered from the ones with the most recent activity to the oldest. The user is able to click on any of the chat names to access that list. On the top of the screen, they are shown four buttons: “Back,” “New Chat,” “New Group Chat,” and “Import Chat.” The “Back” button allows the user to return to their profile. The “New Chat” button allows for the creation of a new chat with one other user, whereas the “New Group Chat” button allows for the creation of a new group chat with more than one user. “Import Chat” allows for users to import data from other conversations from an external file.

After the user has selected a chat to enter, they are shown the history of the chat along with several buttons on the right and a chat bar at the bottom. The chat history shows who said the message and what was said. The chat bar allows the user to enter new messages, which appear instantly, as new data is continuously being read in. The buttons along the side allow the user to edit past messages they sent, delete past messages they sent, and delete the chat in whole. There is also a button that allows the user to return to the previous page where all of the chats are listed.

If the user wishes to sign out of the application, they simply have to click the red circle on the top left of the graphical user interface. There is absolutely no reason to worry about losing data since an external file is continuously written to as new chats are sent, ensuring that if the server shuts down, no data will be lost. As alluded to before, there is also a moderator or administrator login option that allows for complete access to all of the chats on the server and has an additional feature that allows the moderator to ban any users who seem to be violating the rules of the platform. These banned users cannot access the platform.

Project Structure

During our first meeting, a general structure was created for a baseline of what needed to be accomplished. A document was created to list the general steps that would need to happen for the project to be successful, including outlining how the user registration process would go and what the input and output files would look like. This meeting also involved brainstorming. Looking at other sources of inspiration in similar messaging applications, we decided to model it like iMessage or GroupMe—two very common applications. As mentioned before, this project was split into three parts and assigned to different group members to ensure that we were working efficiently towards a solution. A group chat was created to ensure that teammates could quickly respond if one member was encountering roadblocks in their code.

For the user registration process, we went through possible solutions. We all agreed that having a re-enter password box would be smart, as it confirms that the user knows they entered the correct password. We did not want to make anything in this program hard to understand, so the layout is simply a text box and a label next to it. There is also a username text box. This part was fairly straightforward, and there was not much variance in possible solutions. We also decided to have a button for returning users and one for new users, which is used for almost every application. The layout is very similar to what you would see on login pages for social media applications, such as Instagram and Snapchat. During the startup process of the program, we also decided to implement a multi-threaded server, to make sure that many users could access the program at the same time.

For the user profile, we decided to implement a color-changing option, which was created by logging the RGB values of the user in the user data file and reading that data at different points in the program. By putting the color data in the user file, it was an efficient way of storing data, as other parts of the program also use that file. There were many discussions about how the data should be stored. It was decided that having all of the user information (username, password, and color data) in one file and having all of the chat information in another was the best way to do it, as it did not create an unruly number of files. That said, if a user decides to export their chat, it will create a singular file for that chat. This choice was made to ensure that if thousands of users were using the platform, there wouldn’t be thousands of files as well.

For the chat list portion of the program, many ActionListeners were implemented to make sure that all of the user’s actions were being accounted for. This was taken as inspiration from past assignments that we completed in this course. For the listing of the chat names, the research concluded that using a JList and a DefaultListModel was the best way to store information. By using a JList, it was easy to transfer information from an ArrayList containing the names to a JList by copying in all of the values. Furthermore, a list is easy to change and rearrange, making it ideal for a program that will alter the order depending on how recent the messages are. For this stage, many different classes were defined to clearly separate the function of it; there is a FindChat class that will return information about certain chats and a ChatInfo class that will return information based on a line from a file that you send it. Both of these simplify the process of this program, allow for easy debugging, and allow for easy identification of the source of the problem.

For the chat portion of the program, we once again decided on a simple format to make it extremely user-friendly. A unique feature of this code is that it uses a timer to read information from the files in a certain amount of time. This ensures that new chat information will be displayed on the user’s screen without completely crashing the program. There is only a small pause in the display of the information. Many lists were also used in this portion of the project to display the user’s options of which messages they can edit and delete if they so choose.

Part Two: Reflections

Weihang Li / Leo

My contribution to this project can be summarized in three separate perspectives.

The first segment is designing the overall planning and structure of the entire program. I have worked on designing the underlying data structure of data storage and logic flows in three separate files each containing user information, chat list information, and conversation information for each conversation. This will allow us to break the overall program into three sections: login, chat list, and conversation. The three sections will require minimum data transfer to run independently.

The second segment is writing the ChatList section. I have created the overall structure of the portion, including the server and client connection, multi-thread capability, GUI design, and the function for starting a new chat and starting a new group conversation. These two functions will read in each individual user name, check if they exist in the user list then it will check if it exists in the chat list. If both checks are passed, a new chat will be created in the file with the chat index number, both users in the chat, chat name, and time stamp. My program will also read through the chat list file and find the ones that include my username and return the ArrayList for further use.

The third segment is the integration of the three sections. This includes modifying each section of code to make them capable of working together. Adding a connection section to enable them to transit from one to another section, along with delivering the important information needed for the other section. This also involved testing the cooperation on the connection of the three sections.

If I can start over again there are several things I will change. First of all, for the overall data structure. Since this is the first time I worked on a big project like this, I lack experience with setting up a data standard. The data structure I planned wasn't the same as the final data structure. This has caused many issues throughout the project, and thus we have to redo a lot of parts in order for the program to function properly. We should also make a database for all the methods. When we put all the parts together, we realized that we have reused many methods like reading the file and acquiring all the usernames, etc. If there is a database for all these methods, we wouldn't need to redo much of the work. The last thing I think I would redo is to save the file before making any changes. This has caused me lots of issues. When I wanted to make modifications to the code, it sometimes suddenly stopped working and running into many errors. When I want to go back to the older version, I often realized that the last version is the one saved a long time ago and will require me to redo lots of the work to get it back up running again.

Rishabh Chaudhary

My contribution to the project includes building both the client-side and multi-threaded server for user registration, authentication, account settings, and GUI customization feature. Additionally, I contributed to the integration of the codebase and supported my team with debugging.

User registration and authentication is the “starting point” for the overall platform as it initializes a user and stores their information into a file that persists when the server or client shuts down. Users have the capability of creating accounts and logging in. Users can also delete their account and change their password which is updated in the data file shared across the platform. The main portion of my contribution includes the server and client connection, multi-threading capability, GUI design, and storing/updating data in a file shared across the platform. As an optional feature, I built the GUI customization feature which enables users to customize the colors of the messenger background and text. The color settings are saved to the user’s account into the user data file. These settings are saved for when the user logs back in. The chat messenger GUI will set background and text color based on the user’s settings. When a user creates a new account, the color settings will set to default until they customize it.

Apart from building my part of the project, I worked on supporting my team with the integration of the codebase. I worked on integrating my part of the code into the overall platform. Additionally, I supported my team members with building multithreading capability for chat messenger. Initially, the chat messenger server was single threaded which only allowed a single client. I worked on transforming the single-threaded server to a multi-threaded server that accepts multiple clients. I also supported my team with implementing swing utility workers which prevented GUI frames from freezing.

If I was given the opportunity to redo the project, there are a few things I would do differently and the same. The one thing I would do the same is project execution. At the beginning of the project, we discussed ways that we should proceed on building the project. My initial suggestion was that we create a Javadoc highlighting all the classes and methods before writing code. A team member suggested we split the project into individual parts and jump straight into the coding. Given the complexity of the project and the timeline we were given, jumping straight into coding was a wise choice. It would have been hard to create a Javadoc as we only figured out which methods and features we needed as we built out the project. This is something I would do again. Two things I would do differently are version control and server/client-side. Although we used GitHub regularly, our system of saving and contributing was not systematic. We updated the entire codebase every time we made commits. A more efficient method would be where the entire team works on a shared version and we commit and merge our individual code into a single project. When we started building the individual portion of the project, each of our parts had separate servers. This created some challenges while merging as we had unique servers for each part of the project. Although this created some challenges while merging, this method was effective as we were able to test our parts without conflict from unfinished or buggy code from other parts of the project. Creating separate servers from the beginning is something I would do again.

Jiayu Zhou/ Jenny

My contribution to the team project is the entire chat panel, including the client GUI and server. It is one of the most important parts of the project because you need to make sure users can send messages to each other and to the group. The message should appear at the correct users’ chat panel. I built the data structure in a way that all the message history for each chat is saved in a log file that can be accessed by the server. By doing it this way, the history won’t be lost if the user disconnects or exits the chat, and it could be reloaded once the user is back. Users will be able to see the messages they receive even if they are offline. For the client GUI side, once a user selects a chat from the chat list, the chat GUI will show up and automatically request history from the server. The client will keep scanning for updates from the server to ensure the user receives a real-time message.

The second contribution to the project is the detailed features of chat. The first feature is a show and hide time feature. Users can customize their panel by choosing if the time for each message is showing or not. The next feature is to edit or delete messages. Each user could select only their past message to edit and delete. After editing, the message will show an “edited.” After deleting, the message will show as being withdrawn. Again, all these will be shown in real-time. Lastly, there is a “member” button that can show all participants in this chat.

The third contribution is to have an export and import CSV file feature and delete chat feature. This is a part I worked on both the chat part and chat list part. Within the chat, a user can export the chat history as a CSV file with all detailed information. The CSV file will be saved in the client's directory. Within the chat list panel, a user can import the CSV file to the chat. It will go through the server for quick validation and show it on the chat list. A user can also delete their chat from the chat panel. By doing this, this chat will be deleted, and the user won’t see it on the chat list, but this chat will still be visible to other participants.

The fourth contribution is on two extra features. The first is to add a superuser – the administrator. The administrator will log in through a special client and this client is limited to the administrator. The administrator can ban users, and the user won’t be able to log in again. The administrator can also view all chats by all users as a monitoring role. The second is help with the color setting. I took the color setting values from the setting panel and applied them into the chat panel. Each user can set and see different colors in the background and text.

The fifth contribution is to combine the authorization, chat list, and chat part together, with some help from other group members. I rewrote the main methods from all parts to make them have the same format and logic, and combined the GUI into one client file, and combined all server parts together. I worked on connecting the panels and buttons so the correct panel shows up when an action happens. I re-built and combined all the loops from each server to make it integrated and work with clients without errors. The last thing is to make sure everything is working correctly, such as adding validations to the color inputs and debugging the issues we found during the testing.

If I would redo the project from the beginning, I would like to change the data structure of receiving updates. The current way of doing it is when a user does something, for example sending a message, the message will be sent to the server and stored into a log file. The server will not take more action. It doesn’t look for who should receive this message. Instead, each user client requests updates from the server each second. The client checks if the messages match the log file on the server side. If not, the client will update the messages showing on the JTextArea. By doing this, the client can get updated if there is a new, edited, or deleted message in real-time. But in real life, this could be inefficient. It can occupy a lot of memory and use a lot of wireless data. What I would like to do but is much harder is that when the server receives something, it goes ahead and searches for the user who should receive this message or update. The server sends information to the client with the details of the new message or updates only. The client will only need to keep listening to a single message instead of a lot of history. This method could also make it easy to publish a “there is a new message” in the chat list. If I could do this way, I can reduce the wireless usage and memory usage significantly and it can be closer to what we have in real life.

Kenzie Duby

For my contributions, I mainly worked on the GUI of the Chat List, along with some other features such as arranging chats in order of time, listing the chats, and creating a delete chat function.

The first contribution I made was developing a baseline for the Chat List GUI, which includes a few buttons and a list of names. I did a lot of experimentation with figuring out how to best list the names of the different chats. Initially, I aimed to use buttons, but fitting them to the correct size proved to be a bit of a struggle, as the buttons for the group names changed size when a new group was added to the list. Thus, I decided to implement a JList/DefaultListModel to organize the names. This was the ideal option as it displays the chats with a decent-sized text and allows for many different groups to be added. A JScrollPane was also implemented to ensure that a user with many chats would still be able to access the older ones. This was an extremely important part of the program, as the user must be able to access and look at their chats in an easy, sophisticated way.

The next contribution I made, which aligns with my previous one, is the organization of the chat names based on which chats were more recently accessed. The user file that we created has a timestamp of the last time a message was sent in the chat. My sorting method reads in that information from the ChatInfo class, allowing me to identify both the group name and the time stamp. Using the time stamp, I implemented a bubble sorting method to order the names and times from most recent to oldest. Once the sort is done, the new JList of the names is turned into a DefaultListModel, as mentioned above, and is sent to the main client function to append it to the content pane. This was also an important contribution, as it is a characteristic that most messaging applications have.

Furthermore, I also worked on the “Delete Chat” button and making sure the list of names shown on the GUI updates as chats are added or deleted. I additionally started working on a new message indicator for the code, but because the project was finishing up soon, we decided not to make any changes to ensure that our code would still work after fully integrating it. I was working on a function that would add an asterisk next to the chats that had new messages.

If I could have done anything differently, I would have worked on the chatting side of the program. I was really interested in that part and wish I could have been more involved in that portion of the project, as I think it would have been a good challenge. I was not extremely comfortable with the idea of threads before this project, so I think working on the chat portion of the project would have helped me develop my understanding of and confidence with them. It also would have been nice to work on the file input and output, as I felt really comfortable with that during this semester and would love to work on that some more. Additionally, if I had the opportunity to do it again, I would have kept the optional features in mind earlier on, as I could have created the new message function while I was working on the sorting list for the chats. Overall, I am really proud of what I accomplished for this project given my minimal background in Java, but I wish I could have expanded my horizons on what I worked on a bit more.

Jennifer Cahillane

For my contributions to this project, I worked to help with the individual messaging part of the code. I also tried to help with the integration part of the code.

Towards the beginning of the project, I made sure to talk through the overall project and expectations with the group. We had several zoom calls where we discussed what we wanted the overall platform to look like once it had been completed. Once we had the general layout, I helped with the individual message part of the code. I had begun working, albeit semi-slowly due to a massive other project in a different class, on setting up the initial GUI pop-up, with the text boxes for the messaging and the send button and whatnot. I also had begun working on the communication between the server and client for the storage and displaying of messages for any member in the chat. Unfortunately, due to a misunderstanding on my part, I did not realize that a different group member had been working on the same thing and had made more progress than I. After checking with that member to see that we essentially had the same things, the code was combined and the other version became the one we further built off of.

I also tried to help with the integration part of the code, specifically turning what we had into something that would function with multi-threading. My efforts in this matter were not completely fruitless, but unfortunately, I could not manage to eliminate all of the bugs. This was mainly due to the fact that I am not entirely comfortable with threading, which made it very difficult for me to try and do this task. Another group member stepped in to help, which meant that my code for this, since it was not as efficient, was not used in the final code. While my work in this regard did not ultimately show up in the final version, I did feel that what I learned in these efforts though did help make me a stronger coder for the future.

Overall, I wish I had taken more initiative to try to be assigned the parts of the project I actually understood and felt strong with at the beginning. For example, I felt really strong with the GUI appearance parts of the code, and the file IO aspects. However, other group members had jumped on these parts early, and I did not ask to be put on those parts instead. I know my strengths and my weaknesses, and I often got jobs that aligned strongly with my weaknesses, which meant that while I was trying my hardest on the sidelines to learn more and get it done, I worked more slowly. Therefore, other members were often able to figure out the issues before I could make a strong contribution or a final solution. All in all, I just wish I had been more confident and had done more research to feel more confident before the project had actually begun.