OOADI

Mini project

Project Report

Group 552:
Rasmus Hjøllund Davidsen
Simon Loi Baks
Mads Dahl Andersen
Camilla Høy Kusk

Aalborg University
Computer Engineering

Contents

1 Introduction **2**

2 Analysis **3**
2.1 Use Case Description . . . 3
2.2 Use Case Diagram . . . 3
2.3 FURPS+ . . . 5
2.4 Domain Model . . . 7

3 Design **8**
3.1 System Design Description . . . 8
3.2 Class Diagram . . . 8
3.2.1 User and GUI . . . 9
3.2.2 Server, Connection, and Logs . . . 9
3.3 System Sequence Diagram . . . 10

4 Implementation **12**
4.1 System Implementation Description . . . 12
4.1.1 Sending and Receiving on Sockets . . . 12
4.1.2 Login . . . 12
4.1.3 User Registration . . . 13
4.1.4 Main window . . . 13
4.1.5 One-on-One Chat . . . 19
4.1.6 Group chat . . . 20
4.1.7 Persistent Logs . . . 21

Introduction 1

In this OOADI report the student group will be analyzing, designing and implementing the chosen mini-project ***encrypted chat***. The primary design of the encrypted chat will be including the following features:

- A separate server that only handles key and message exchange
- Encrypted group chats
- Persistent chat logs that are loaded if you close and reopen the program
- Assume the server can be trusted with key exchange

It should be noted that the server *should not* be able to decode the messages between clients.

Analysis 2

In this chapter a use case description will be given based on the chosen mini-project. It will include a use case description, aswell as a use case diagram to show how users interact with the system, and a description of these. Next the FURPS model will be used to analyze any requirements of the system such as functionality, usability and etc. At the end of the analysis a domain model will be made as a source of inspiration for designing the software objects.

2.1 Use Case Description

In the following section a scenario based on the system in mind will be presented as an example.

1. The user opens the computer and starts up the program.
2. The user is presented with a login screen before entering the chat menu.
3. After successful login the user can choose to either create a group chat or chat one on one.
4. To begin chatting the system prompts the user for a key for validation.
5. After a successful validation the user enters a chat GUI.
6. When the user is done with chatting, the user can simply click the return button to return to the chat menu.
7. Before closing down the program, the user is prompted to logout.

2.2 Use Case Diagram

The following use case diagram is used to define the system's general functionalities, and show the interaction between actors and the system. The use case diagram can be seen on figure 2.1 below.

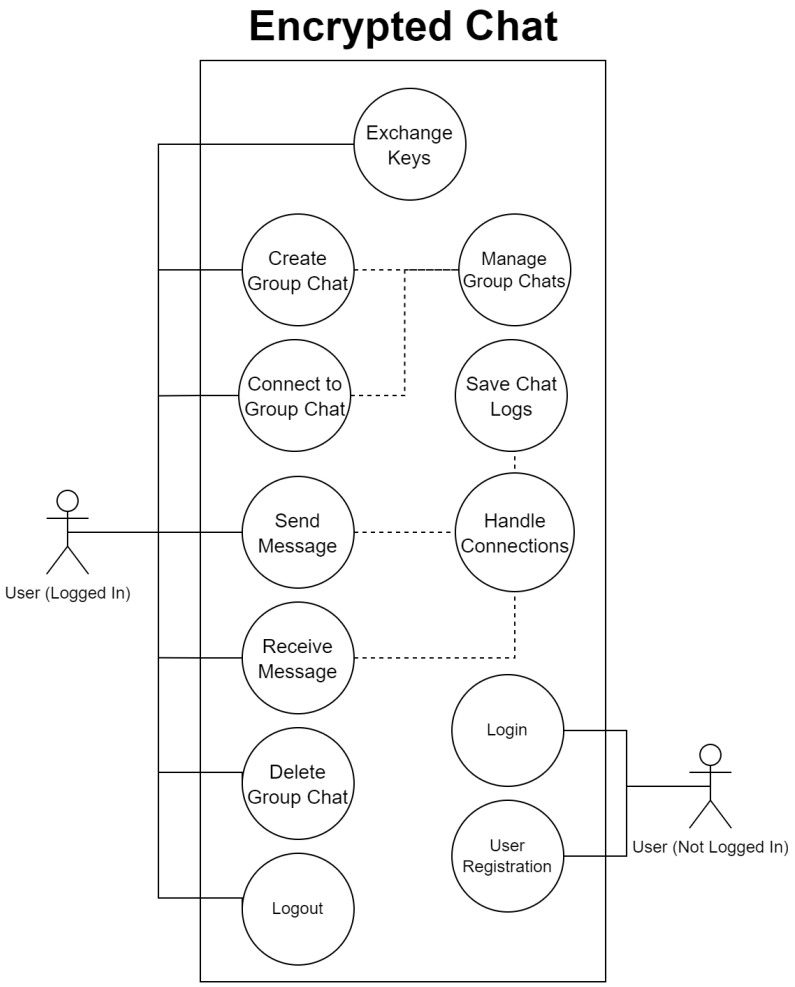


Figure 2.1: The figure is a use-case diagram which shows the functionalities and the interaction between actors and the system.

Use Cases and Actors

The following actors are the ones interacting with the system.

- User (Logged in)
- User (Not Logged in)

There are two types of actors interacting with the system, the logged in user and the not logged in user. If the user is not logged in, they have the option to either login or if it is a new user, register as a new user. If the user login they are able to create, delete, connect and chat.

The list below lists all the use cases in the system, such as create, delete, manage group chat and etc.

- Exchange Keys
- Create Group Chat
- Manage Group Chats
- Connect to Group Chat

- Save Chat Logs
- Send Message
- Handle Connections
- Receive Message
- Delete Group Chat
- Login
- Logout
- User Registration

The use cases manage group chat, handle connections, exchange keys, save chat logs and user registration are all expected to be handled by the server. It will manage the group chats and handle the connections, in the sense that it keeps the connection between users alive, so they do not have to exchange keys every time they want to chat. As stated in the mini-project description the server is also expected to handle the exchange of keys between users. The server also handle the registration of user and chat logs between users. The chat logs are expected to be encrypted so the server can not see the content.

The other use cases are expected to be on the client-side. The logged in users should be able to exchange keys, create group or individual chats, delete chats, connect to chats as well as sending and receiving messages.

2.3 FURPS+

In this section FURPS+ will be applied to the encrypted chat. FURPS+ are in principle requirements for system.

Functionality:

The functionality of the system refers to what the system should do.

- The user shall be able to login into an account or logout.
- The user shall be able register an account and create a chat.
- The user shall be able to delete a existing group chat.
- The user shall be able to send and receive messages in the chat.
- The user shall be able to view messages in the chat.
- The user shall be able to exchange keys with other users.
- The system shall be able to forward the exchanged keys.
- The system shall be able to manage the group chats.
- The system shall be able to handle incoming connections.

- The system shall be able to handle login and logout attempts.
- The system shall be able to handle user registration.
- The system shall be able to save encrypted chat logs.

Usability:

The usability of the system, should describe what kind of User interface is needed.

- A screen is needed for a user to see.
- A keyboard is needed for the user to interact with the chat.
- A computer mouse is needed to interact with the GUI.

Reliability:

Reliability refers to the tolerance of the system to failures. In another sense this could be considered how durable the system is.

- The system is expected to deny access to users who have attempted to login more than three times.
- If the user tries to connect to chat and enter the wrong exchange key three times, they should be denied access / Forcefully logged out.

Performance:

The performance requirements considers aspects such as response times, accuracy, availability, resource usage etc.

- The response time should not exceed more than 2 seconds.
- A given chat shall be available to all users invited to it.
- Users part of a chat shall be able to write in a chat.
- All chats shall be end to end encrypted.

Sustainability:

- An admin shall be able modify groups.
- The creator of a group shall be able to modify the created chat.

Extra (+):

-

2.4 Domain Model

The domain model represents the whole chat system split into objects that are interconnected to each other. The objects are split into a client-server side to differentiate between which objects are part of the server and the client. The client represents the side which the user interacts with the GUI.
When the user starts the program, a login screen will appear, where the user is required to log on to be able to access the group chats. When a group chat is initiated the "client connection" where the "Key exchange" is responsible

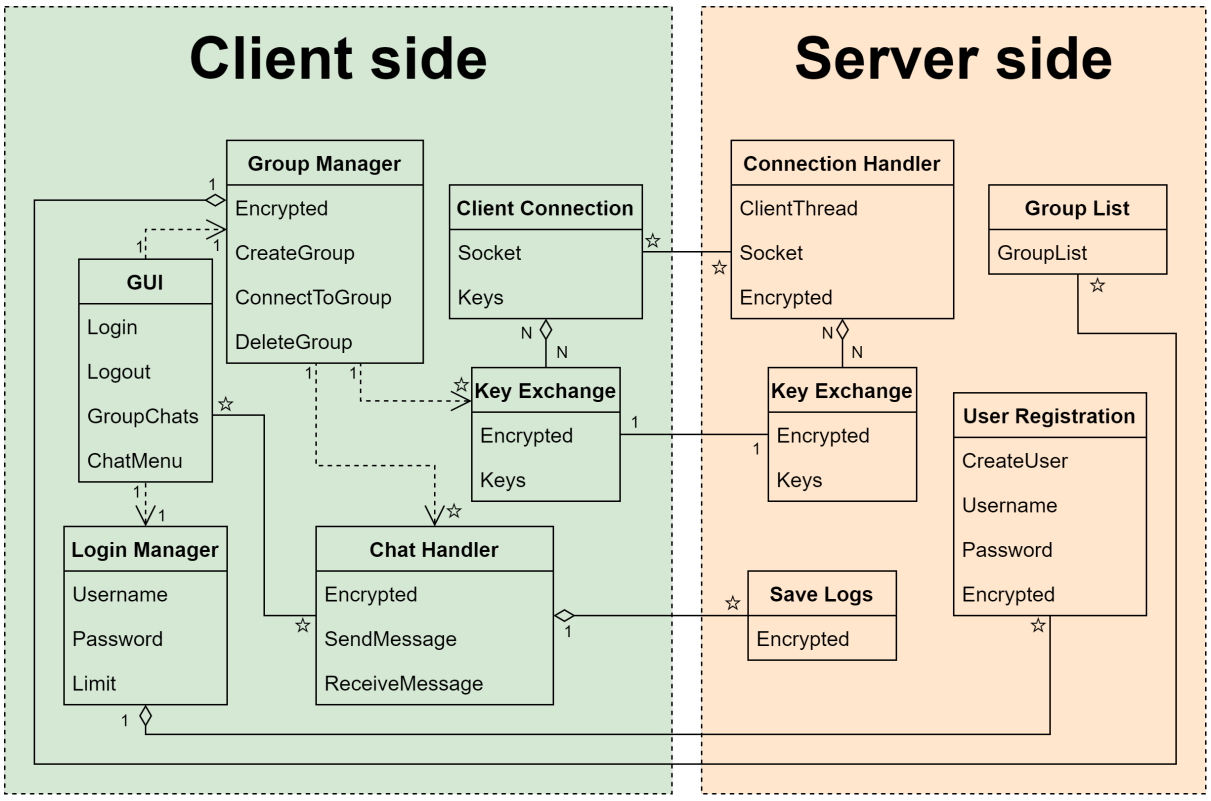


Figure 2.2: Domain model showing stuff

Design 3

In this chapter a general description of the system in mind will be given, as well as a class and sequence diagram to help model the objects of the system, and how the objects interact with each other. To start off a description of the system's design will be given.

3.1 System Design Description

As it can be seen from the domain model in chapter 2 subsection 2.4, the intended design is a system which is divided into a client and a server side. When the user open the program a login prompt will be given, where the user have to pass the registered login credentials to be able to chat. If the user does not have a registered account, there will be an option to register. The server is intended to handle the login and registration of users.

With a successful login the user will be presented with a chat menu, where it will be possible to check current group chats, logout, create, join and delete group chats. The user will also be able to change any general settings such as font size, nickname/username, password change, etc. When the user tries to connect to, join or create a group chat, there will be a user key exchange prompted to ensure the validity of the user. When the key exchange has been approved the user will be able to send and receive messages. The server is expected to be in charge of handling key and message exchanges, but should not be able to decode messages but rather forward messages to another client. Any messages exchanges between users will be encrypted and stored in a database on the server, which will allow users to have persistent chat logs.

3.2 Class Diagram

As a description of system has been made earlier 3.1. A class diagram is made better under each objects attributes and functionalities of each given object in the system. The class diagram is like a blueprint for a system, as it also includes the relations between each object.

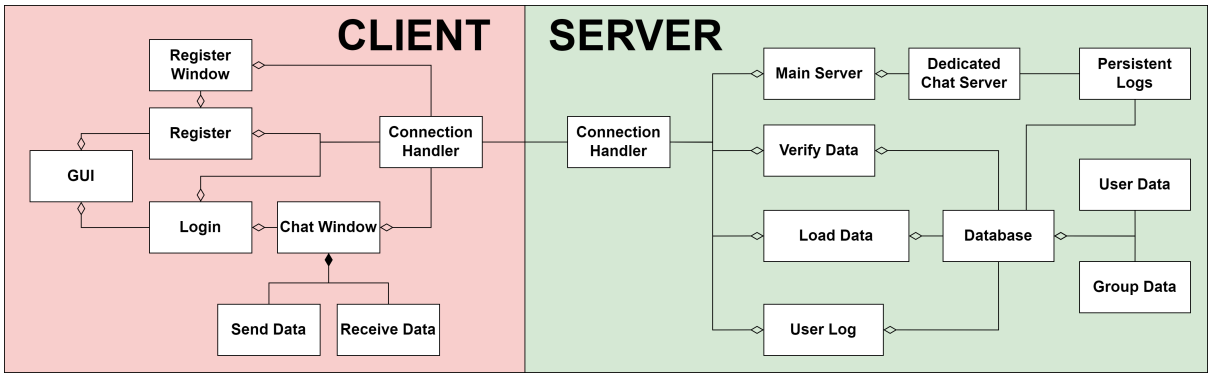


Figure 3.1: An overview of the different classes in the *Client* and *Server*.

A more in depth class diagram is made to, where attributes and methods are included as well as the relationship between each object as the diagram before. The next subsections will focus on a specific aspect of the chat, which will be used for the implementation.

3.2.1 User and GUI

It is common for a chat programs to give each chat user a username and a password, which they can use to login. Each user has these two attributes to login to the chat. The user will always interact with the GUI to use and communicate in the chat program.

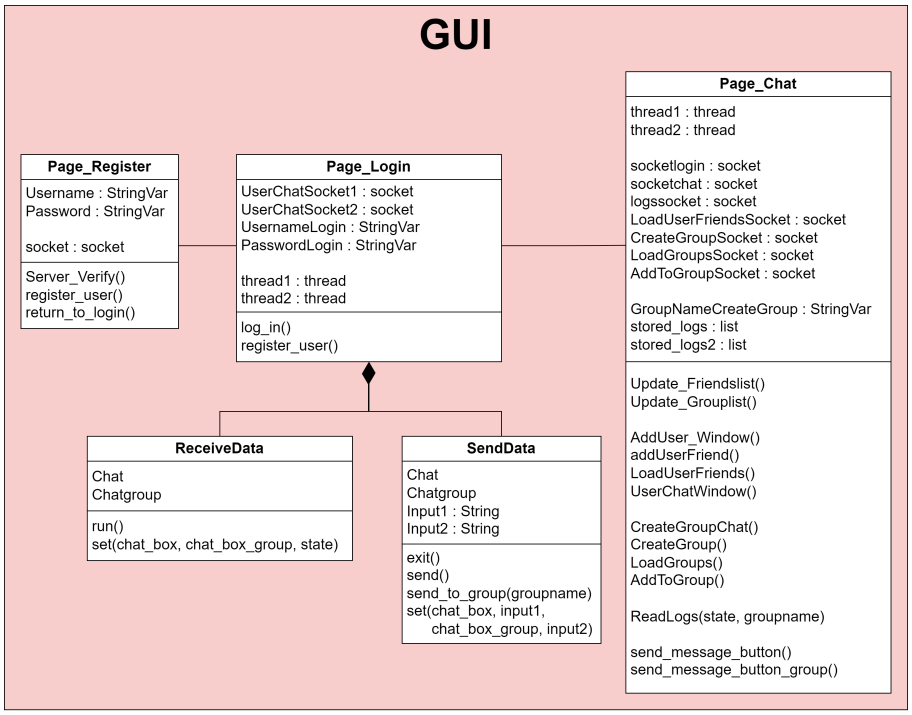


Figure 3.2: Class diagram which show the different classes and their relationship in the GUI.

Chat

The chat will contain a list of the group chats and individual chats with the users friends. The class chat will contain methods related friends, group chats, etc. In the chat, it is only possible to have one chat active at the time that the user is able to see and write to.

3.2.2 Server, Connection, and Logs

The server is responsible for connecting users and keeping a log of every conversation in the chat program. Every message is sent and received is logged at the server, which means that each time the user logs on again the server will have kept the log of every conversation that user has had.

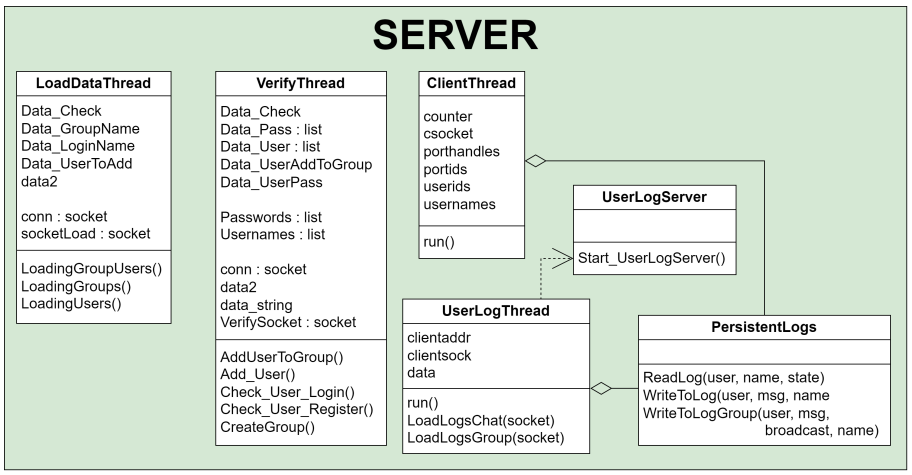


Figure 3.3: Class diagram which show the different classes and their relationship in the server.

3.3 System Sequence Diagram

In this section, a system sequence diagram (SSD) will be made based on the system design description 3.1. The SSD can visualise how classes should interact and respond to some of the essential functionalities in the system. This section will go through a common scenario in a chat program.

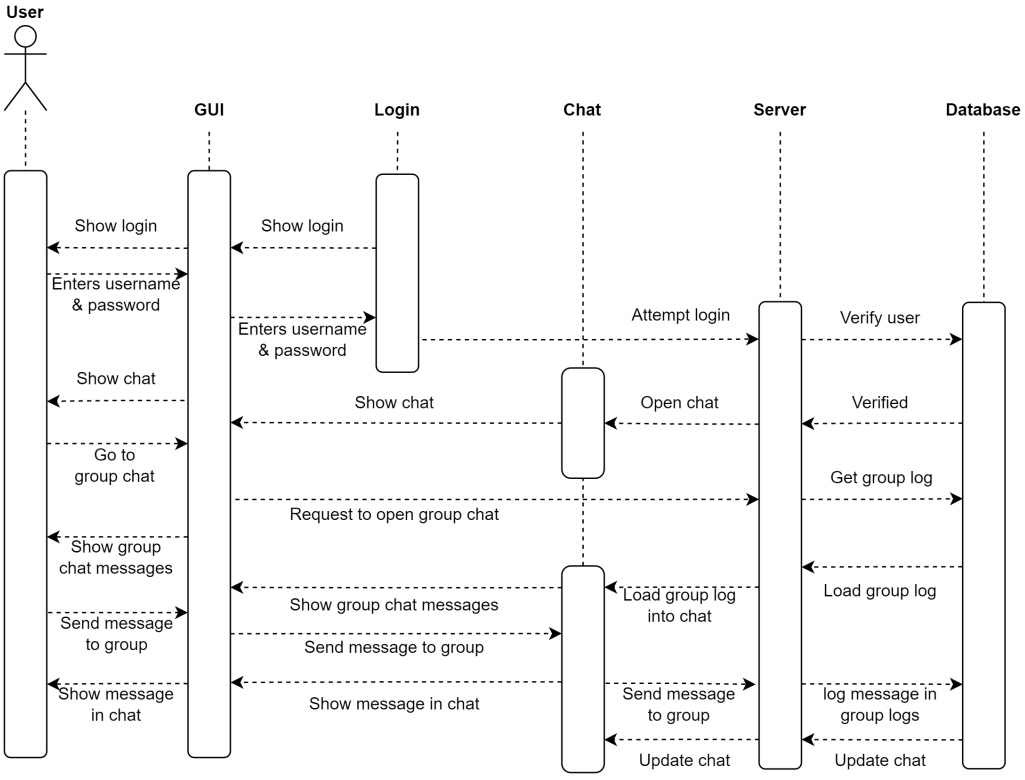


Figure 3.4: A System Sequence Diagram that illustrates a successful login and a message sent to a group chat.

In the figure 3.4, the first half of the diagram will show an user logging in. When the user starts the program they will be presented with the login screen. The user will always interact with the system through the GUI. The user will have two options, which are to login with a username and a password, or register if they are new. In this case, the user is already

registered, and has now attempted to log in, where a request will be sent to the server to verify if the username and password combination is the same as what is the user has registered themselves as. In this case the users username and password is verified, and the server will allow the user access to their chat profile. If the username and password did not match the login screen would be presented once again.
In the last part of the diagram the user wants to enter a group chat that has already been created and used to message the other members of the group before. The request to open the chat is sent to the server that has the logs of their previous messages. The group log will be loaded from the group logs contained in the database back to group chat. The user then proceeds to send at message in the group chat, which will go through the server, and be logged in the database. The update, or rather the new message, will now be seen by all members in chat.

Implementation 4

4.1 System Implementation Description

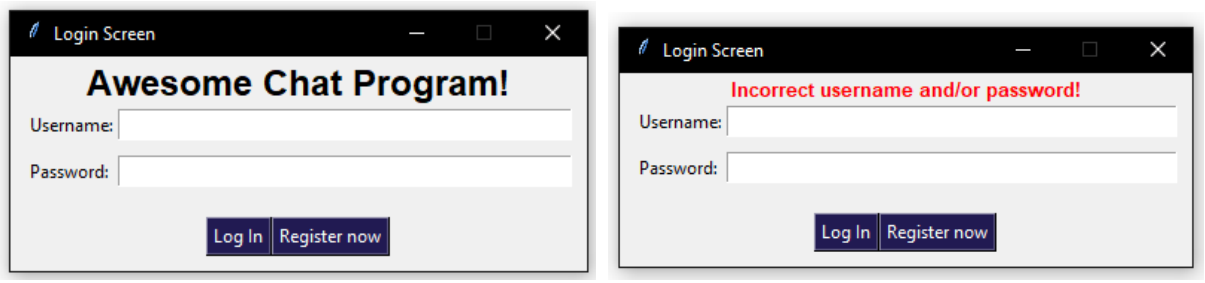
In this chapter the implementation of the system will be explained. For the actual implementation of the system, some of the requirements and use cases were not met because of time constraint. There are also some parts in the design phase that were not implemented or rather re-designed during the implementation part. Though most of the requirements and use cases has been implemented and will be further explained in the following sections.

4.1.1 Sending and Receiving on Sockets

The Python module "Pickle" is a module used for serializing and de-serializing a Python object structure. It is used for serializing and de-serializing Python objects so that they can be sent over a network connection using sockets. Pickle enables the objects to be reconstructed from their serialized form when received.
The core part of the system is the exchange of data between multiple clients. To make the system able to communicate with other clients over the internet, "network sockets" were used to deal with the communication. The sockets are bound both to the server and client in order for them to communicate. The basic concept is binding the host's (the server's) IP address to the socket and a port number. The client then connects to the server's IP address and port to instantiate the communication. To be able to receive and read the data between client and server, the Python "Pickle" module is used because it can convert an object to a byte stream that can easily be sent over a socket. The library can also do the opposite of the receiving end of converting the bytestream back into a Python object.

4.1.2 Login

When the user open the client program a GUI with a login screen appear, figure 4.1a. On the screen there is an entry for "Username" and "Password", where it is possible to login if you have a registered username and password. If the entered username or password do not exist, the screen prints that the username and/or password is incorrect. If the "Log In" button is pressed, the username and password is saved as strings, pickled and sent via a socket to the server, where the data is unpickled and processed to see if it is a valid login. The server then responds whether or not the login is successful. If the login is successful, a connection to the part of the server responsible for the chat is created and then the server creates a separate thread for handling each connected user. The user is also able to register by clicking the "Register now" button.



(a) Login Screen **(b)** Wrong username or password

Figure 4.1: The opening screen of the program.

4.1.3 User Registration

On the "User Registration" page, a username and password can be entered to create a new user as seen on figure 4.2. When the "Register now" button is pressed the username and password are sent to the server. The server compares the username to all the users and passwords stored in the database, and return a message if the user registration was successful or if a user with that name already exists. Figure 4.3a shows the GUI when the registration is successful and figure 4.3b shows what happens if a user already exists with that username. After a successful registration the user is able to return to the login page and login by clicking the "Return" button.

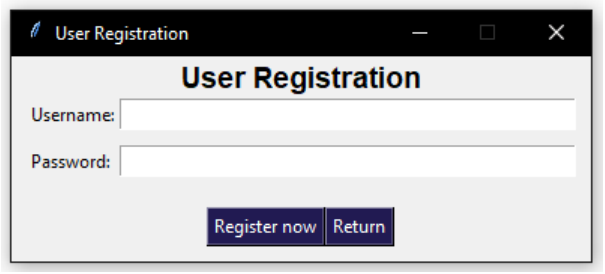
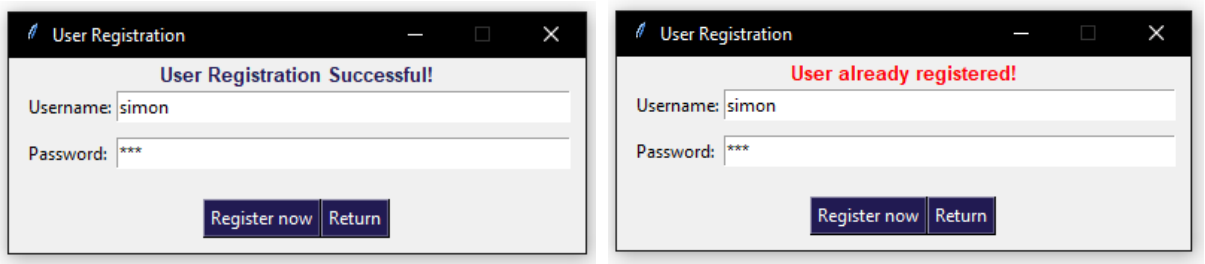


Figure 4.2: User Registration window.



(a) User successfully registered. **(b)** User already registered.

Figure 4.3: User Registration screen.

4.1.4 Main window

Once the user has successfully logged in they will be presented with the chat GUI as shown in 4.4. The chat window consists of multiple frames, and in the upper left corner the main chat menu options are presented. In the chat menu the user is able to press the buttons "Add User", "Create Group", "Add to Group", "Refresh Friends" and "Refresh Groups". Each button

opens a new frame a presents the user with the option stated on the button text. Below the chat menu options the friend list and group list of the user is presented, each with its own window and scroll bar attached to it. On the right side of the main window the chat window is shown, which both functions as the chat window, and the window presenting the option to add a user, create a group and add a user to a group.

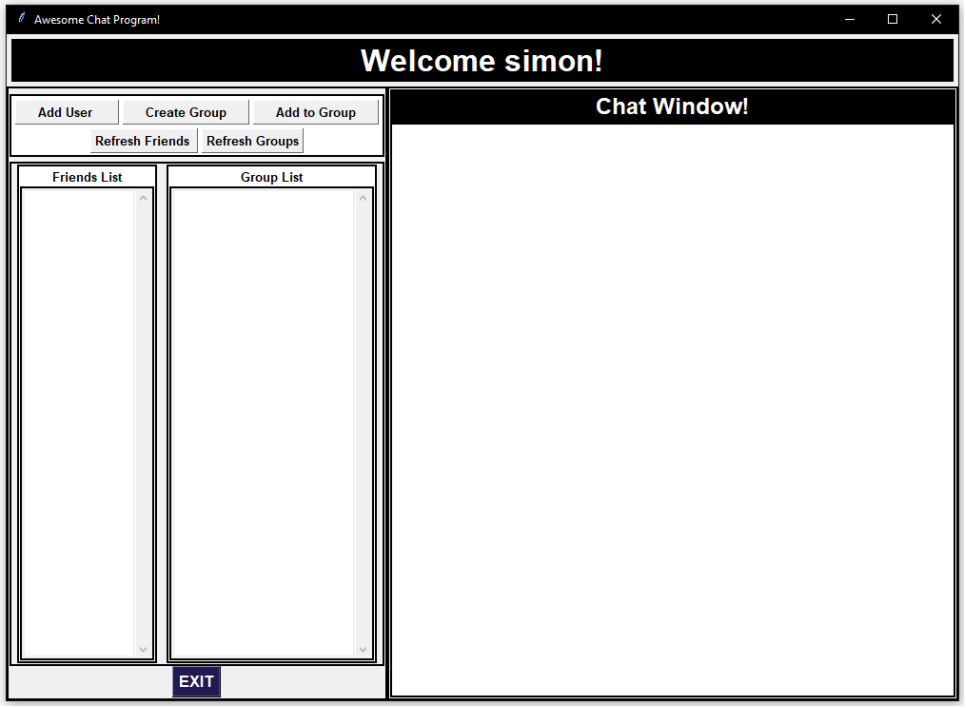


Figure 4.4: The main chat window displaying the logged in user.

Add User Window

To add a user to the "Friends List" the "Add User" button can be clicked. When this is done a window where the username of a user can be entered, figure 4.5.

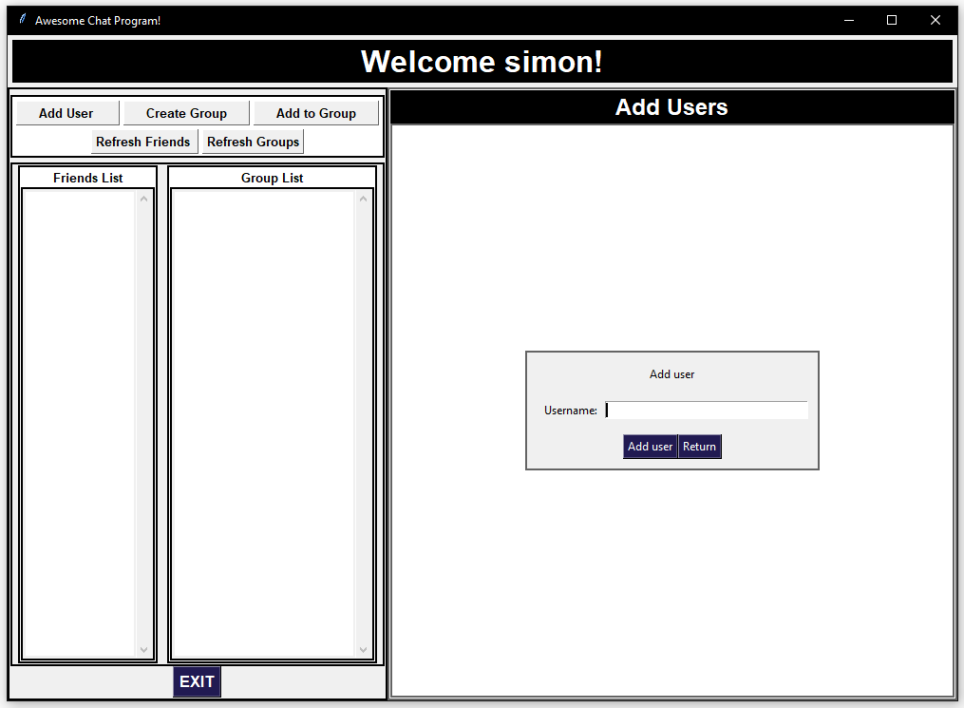
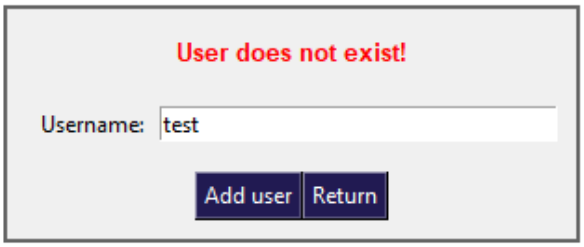
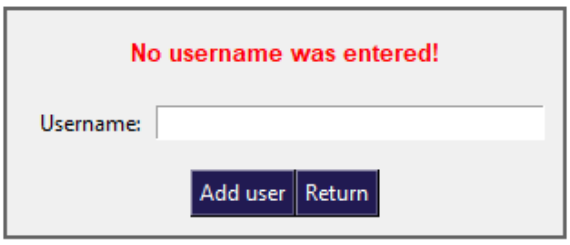


Figure 4.5: Placeholder.

If nothing is entered in the username input field, a red string containing "No username was entered!" is displayed as can be seen on figure 4.6b. If a username is entered the client will send the username to the server and the server will respond with either the username if the add was successful, "ALREADY IN FRIENDLIST!" if the user is already in the clients friend list, "TRIED TO ADD YOURSELF!" if the user is trying to add themselves or "USER DOES NOT EXIST!" if the username entered does not match a registered user, which as an example can be seen on figure 4.6a. If the username entered matches a registered user a green message is displayed and the user appears in the friend list. This can be seen on figure 4.7.



(a) Placeholder.



(b) Placeholder.

Figure 4.6: Add User Window.

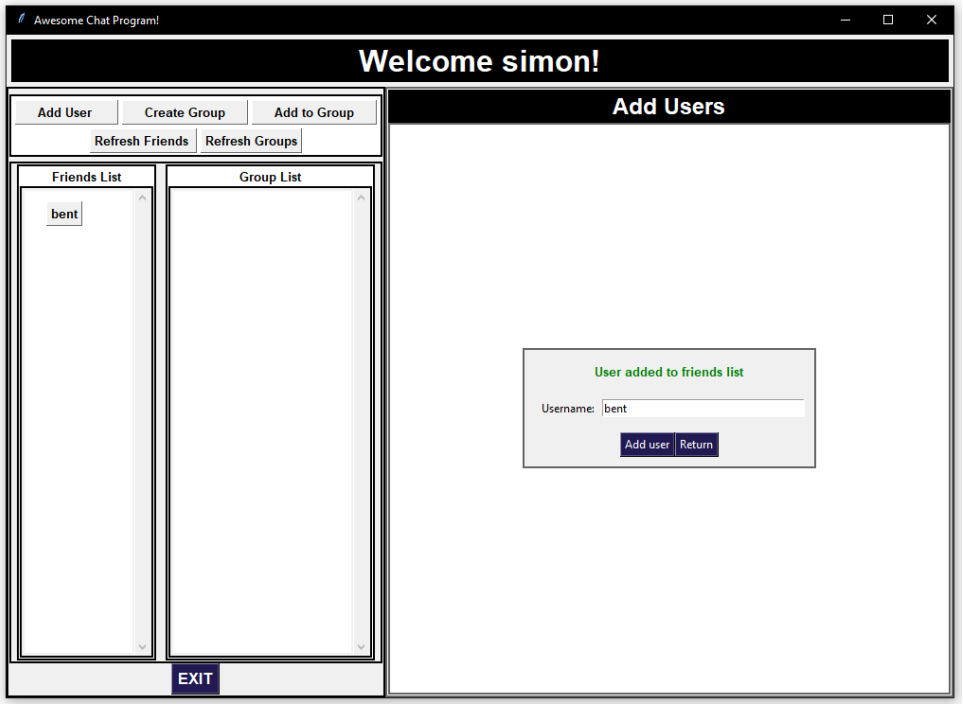


Figure 4.7: User successfully added.

Once a user has been added to the friend list, the the button can be clicked and a chat window for that friend is opened as seen on figure 4.8.

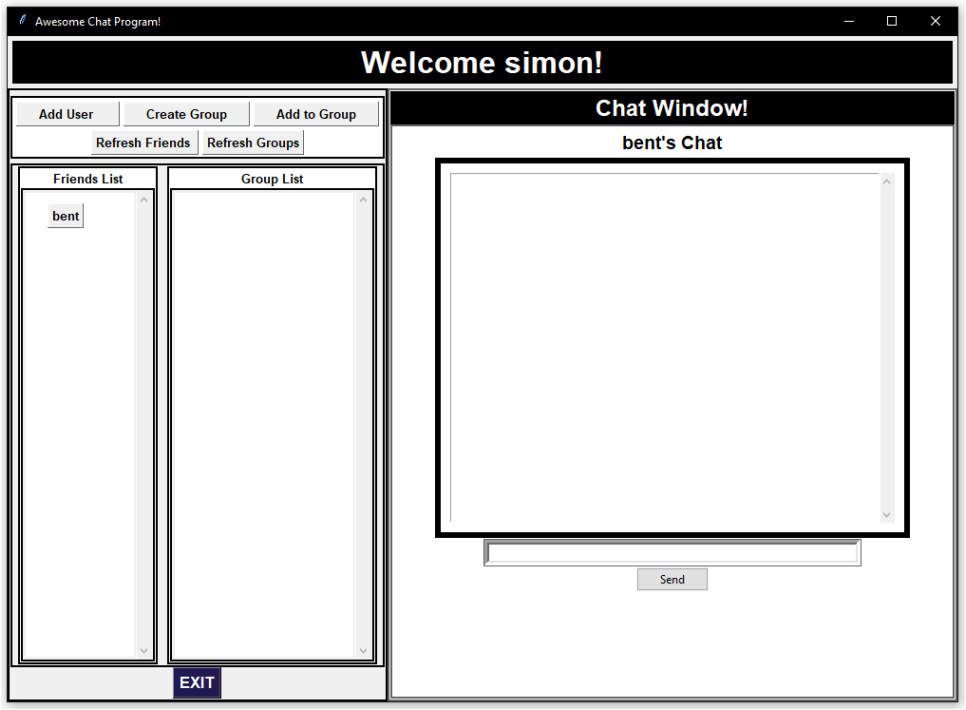


Figure 4.8: Placeholder.

Create Group Window

Figure 4.9 shows the GUI when the "Create Group" button has been pressed. Just like in the case for Adding a User to the Friends List, there is a popup window here where a group

name can be entered.

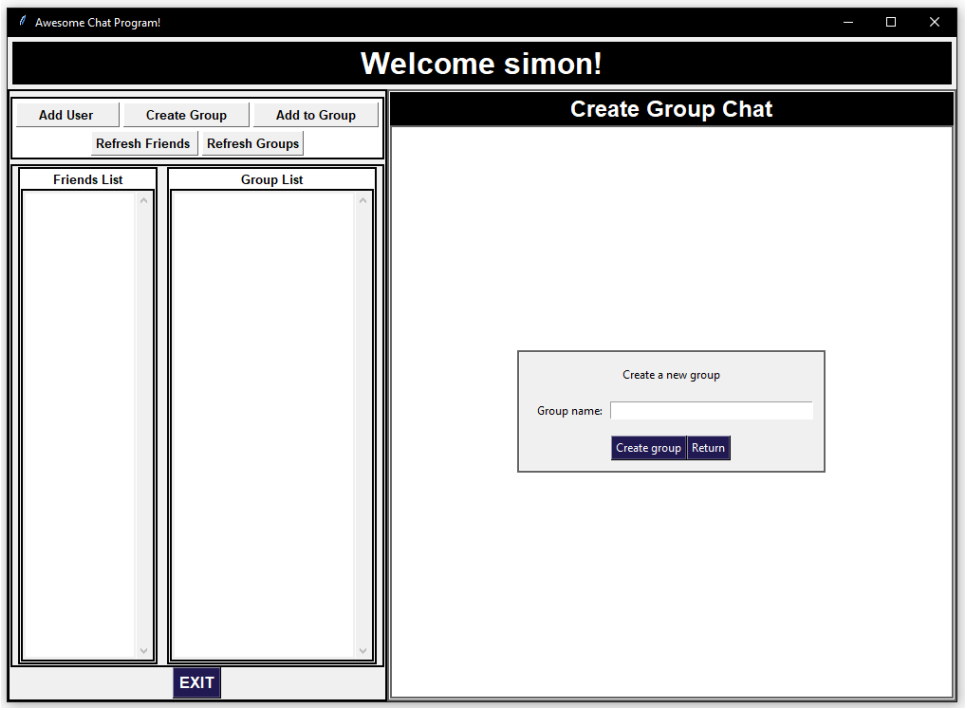
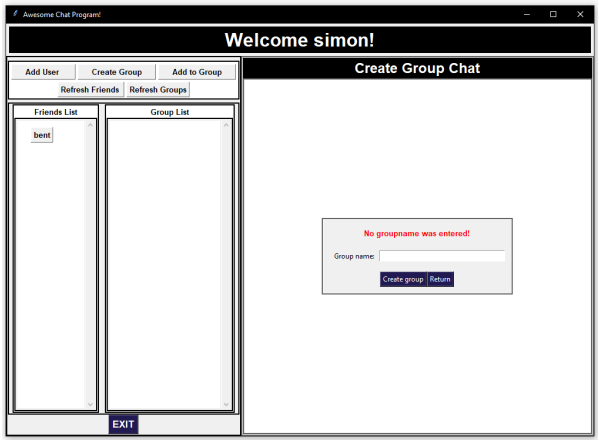
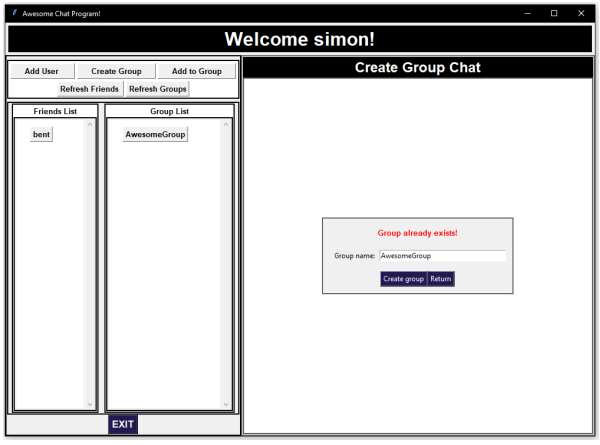


Figure 4.9: Create group window.

When the "Create group" button is clicked, a method called "CreateGroup" is called. This method first checks if a group name has been entered and if not display a message on the GUI as seen on figure 4.10a. If a group name was entered it creates a new connection to the server and requests the creation of the group. If a group with that name already exists, the server responds with "GROUP ALREADY EXISTS", a message is shown on the GUI which can be seen on figure 4.10b.



(a) Placeholder.



(b) Placeholder.

Figure 4.10: Create Group Window fails.

If the server responds with "GROUP CREATED" it means the the creation was successful and the group will appear in the "Group List". A green message is also shown on the GUI. These two things can be seen on figure 4.11.

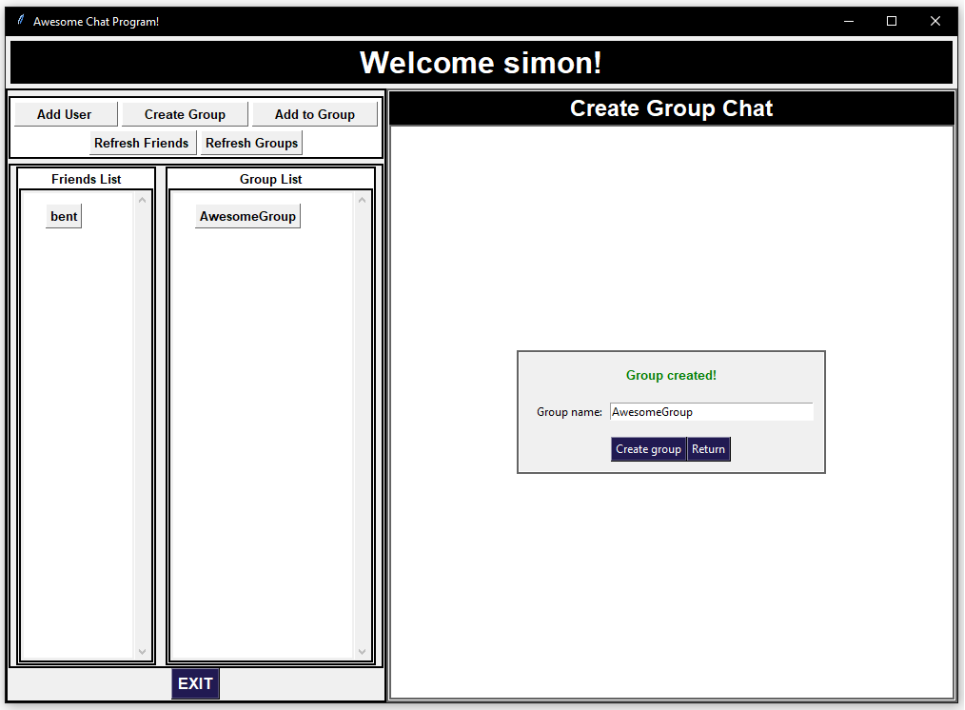


Figure 4.11: Creation of group successful.

Add to Group Window

When the client wants to add a user to a group chat they can click on the "Add to Group" button. When this button is pressed a window pops up, figure 4.12, where the client's groups and friends can be chosen in drop down menus.

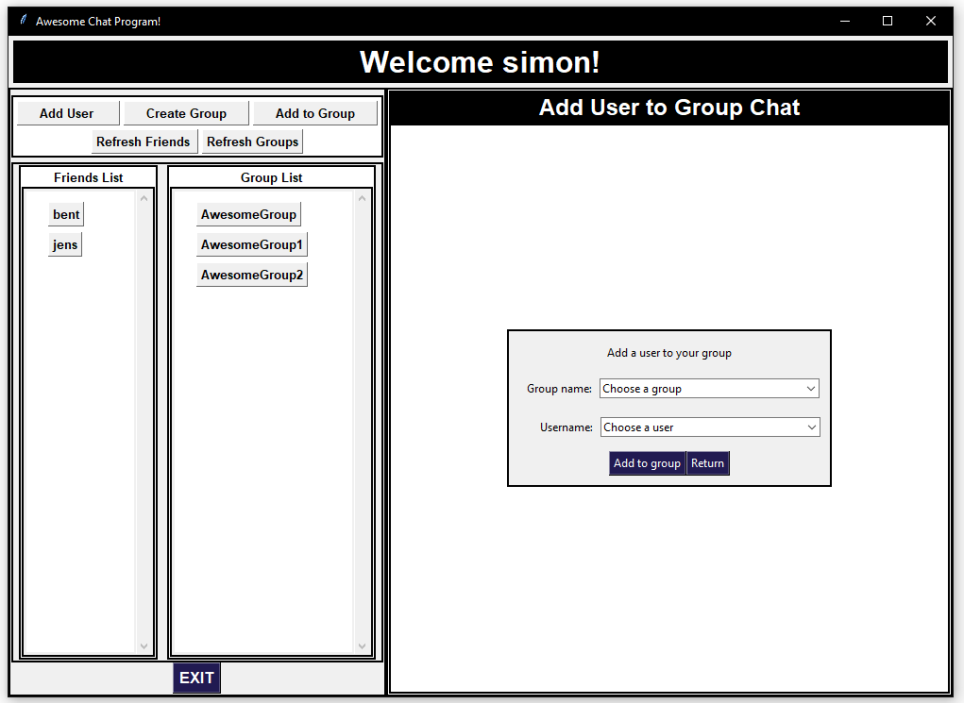
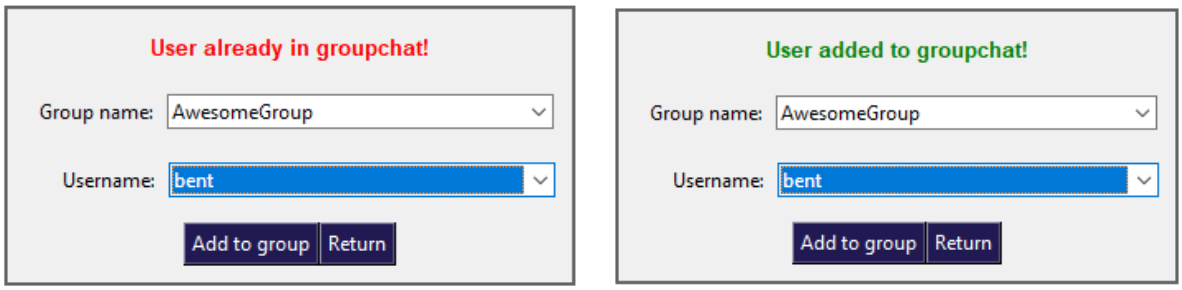


Figure 4.12: Add to Group GUI window.

Once the "Add to group" button is pressed the program checks if a user and group was

chosen and creates a connection to the server. Then the server processes the user and group and responds with either that the user was added to the group or that the user is already in that group, both responses can be seen on the GUI in figures 4.13a and 4.13b.



(a) Placeholder. **(b)** Placeholder.

Figure 4.13: Create Group Window Fails.

4.1.5 One-on-One Chat

When a user is trying to chat with another user one on one they can send messages by typing the message in the input field and pressing the "Send" button or the Enter key on their keyboard. When a message is sent to a single user it is specific in the message sent to the server that it's a message to a single user and also what user it message is to. With this information the receiving thread on the server side can go through the list of users connected to the server and send the message to the correct socket. Even if the target user isn't connected to the server the message will be stored in the appropriate log file which will be described in section 4.1.7. Screenshots of an ongoing chat between users "simon" and "bent" can be seen on figures 4.14 and 4.15.

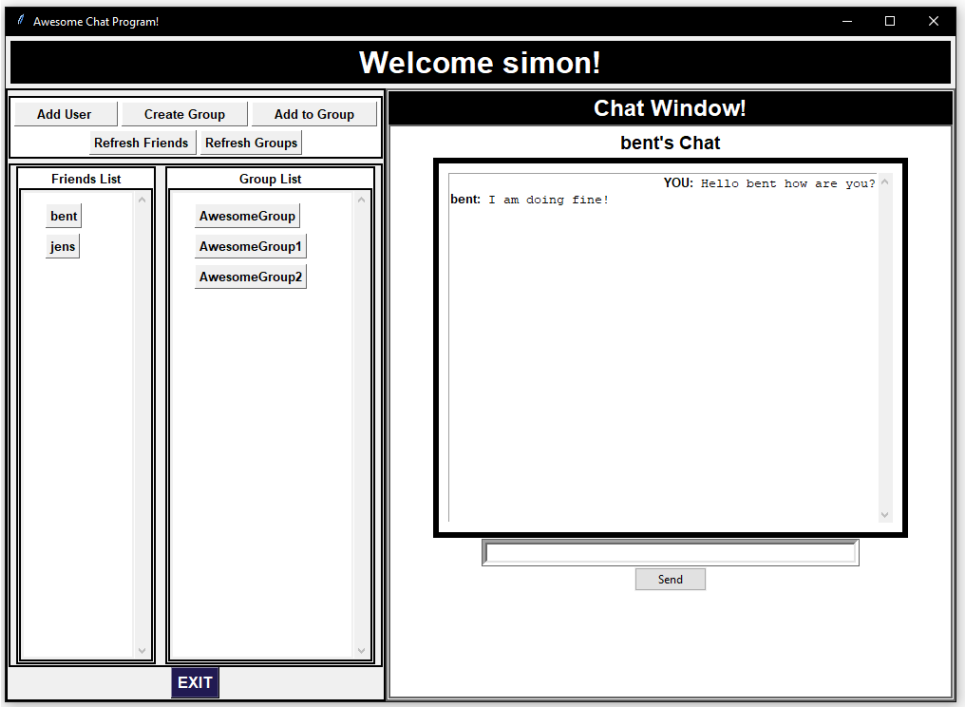


Figure 4.14: Chat conversation from Bent's point of view

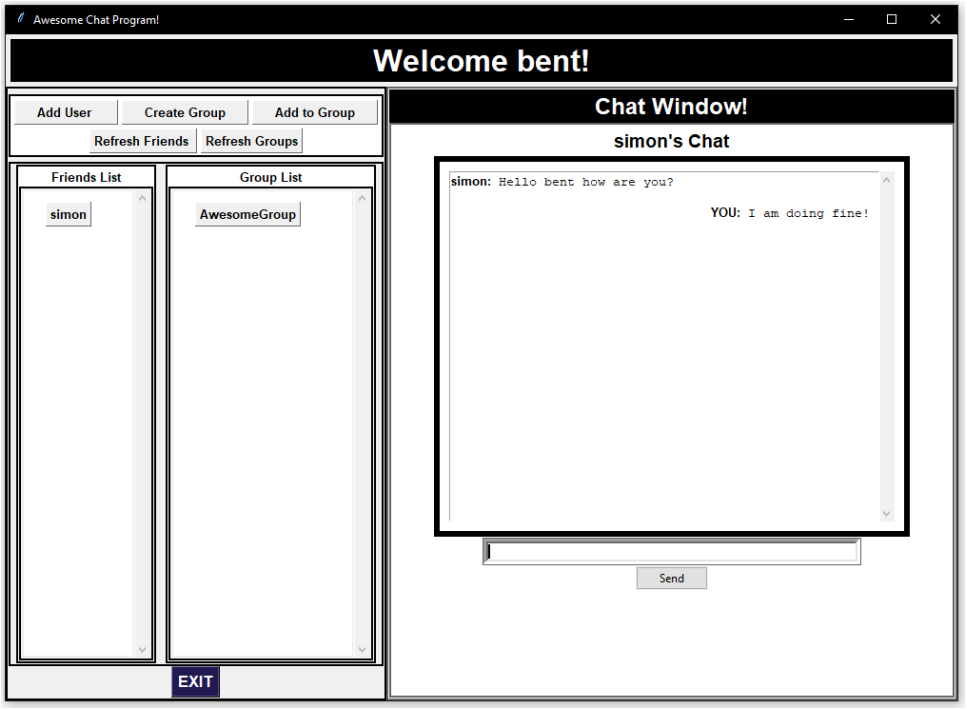


Figure 4.15: Chat conversation from Simon's point of view

4.1.6 Group chat

In the Group chat it possible to write with other users. First the user need to create a group chat and add the users in the friend list. The user is able to pick which ever group chat the user wants to chat with as long as the user is a part of the group chat. In figure 4.16, the user "simon" is part of a group chat with "bent" and "jens". In the chat window a message from each member can be seen. When a user sends a message to the chat each users part of the group chat will receive the message and be individually logged for each user. When the user reopens the program and opens the same group chat window, the messages sent back and forth will be shown.

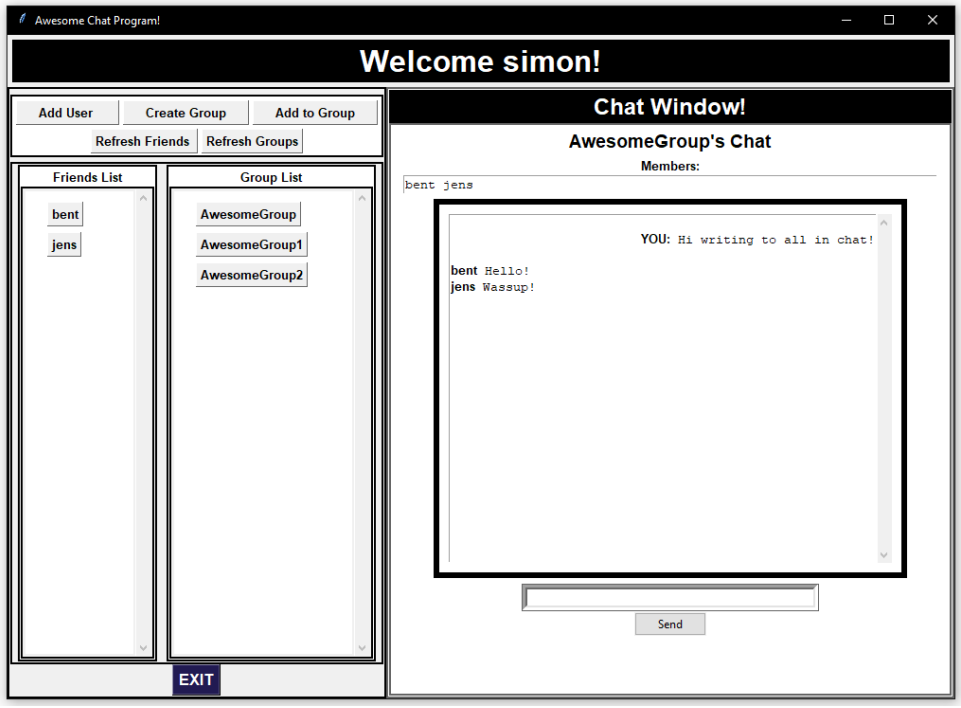


Figure 4.16: Shows the chat-window for the user simon.

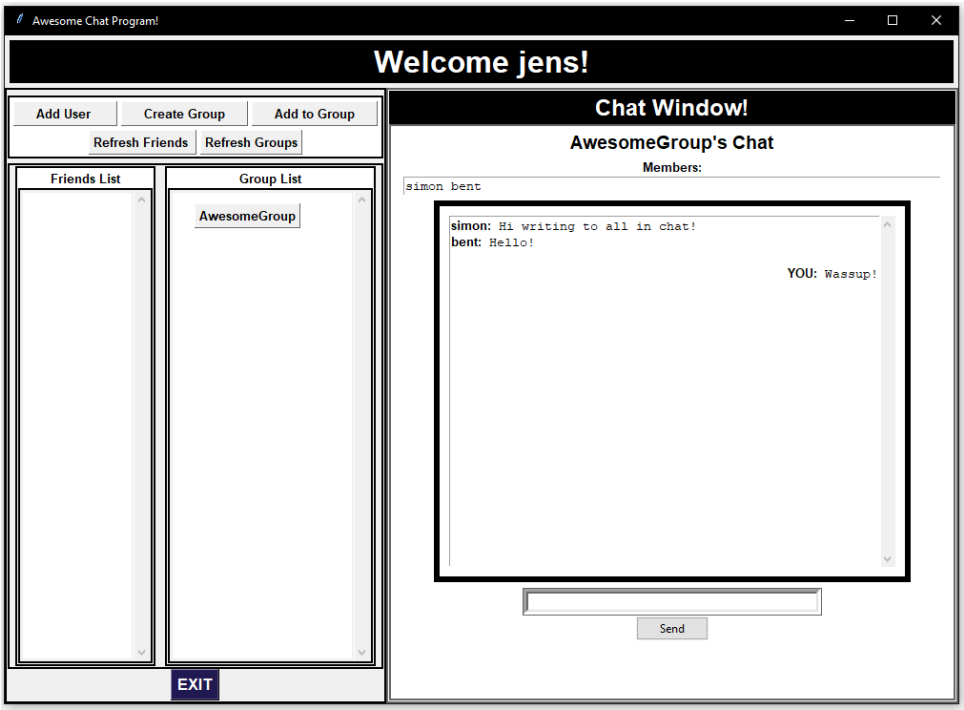


Figure 4.17: Shows the chat-window for the user jens.

4.1.7 Persistent Logs

Writing to Logs

On the server a class called "PersistentLogs" is created, which has methods to write to logs and read logs. These methods are used to save to a specific log file whenever a message is received on server.

Reading of Logs

On the client side, once a user click on a different chat, the logs have to be loaded for the user to be able to see any messages that were sent to them while they didn't have the chat active, as well so that they can see the all previous messages. This is done by calling the "ReadLogs" method of the "PageChat" class. In this method a socket is created to contact the server to retrieve the logs. One of the inputs to the method is whether the chat is with a single user or if its a group chat. Once the user socket receives the logs for a specific chat, the logs are entered into the currently active chat window to be seen by the user. While this transfer of the logs is happening a loading GIF is being played, which is done by a class that loads a GIF frame by frame and continuously updates the frame being displayed..

Saving of Logs

The logs are saved in ".txt" files on the server and there is a log file for each chat a user is part of. The structure of the log file directory can be seen on figure 4.18. Each of the log files just contains the lines of text that has been transmitted in the conversation as seen on figure 4.19.

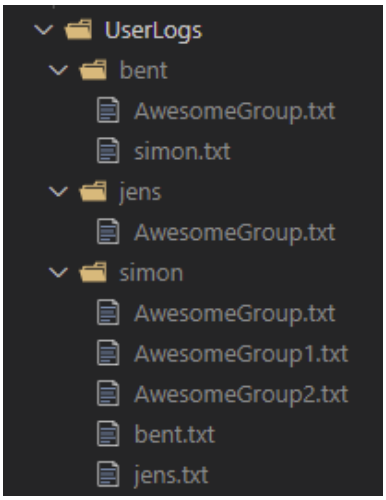


Figure 4.18: UserLogs directory.

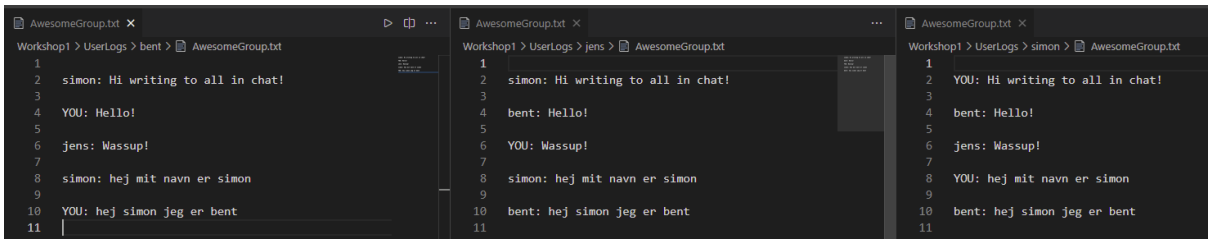


Figure 4.19: Inside log files.