Network Chat Application

CSC3002F
RBLCAM001, VMRGEO003, SNGRIN001

Classes

Two classes were used to implement this network chatting approach in Python. A Server script that will be run by the Server machine creates a Socket to the Client machines, allowing the Server to send and receive messages and facilitate the chat features. This class uses threading to allow the Server to be constantly sending and receiving messages to and from the Client machines. These messages include protocols that act as instructions to the concerned device to begin a certain behaviour. The server IP address is required to connect to it, meaning that this application can be run on any machine where the IP address is known. There is also a password feature that ensures only certain client machines may connect to and send messages to this server. This application makes use of the UDP protocol to send these messages amongst the Server and Client machines.

Libraries

pyodbc, os: Implements database connectivity using a relative path to the database given by os.
Base64: Used to ensure base-64 encryption of messages sent to and from client machines.
Binascii: Used to convert between binary and ASCII when encrypting.
Socket: Allows for easy creation of Server- and Client-side sockets that enable communication between the two machines.communication
Threading: Implements threading to allow the Client to send and receive messages simultaneously.
Time: Used for delaying receive messages for proper formatting

Features

Chat

If the client inputs ‘CHAT’ into the main menu, they will enter the chat to a single person functionality of the program. The client is then prompted to enter the username of the person that they want to chat to. The client can type a message and press return to send it to the desired user. The following protocol will be sent to the server: ‘CHAT#username#message’. When the protocol is sent, a confirmation is sent back by the server to the client whether their message was successfully received by the server and then successfully delivered to the desired client. If the desired user does not exist then the server will tell the client that the person they are sending to does not exist and that their message was not successfully sent. If the client types the word ‘LEAVE’ then they will exit the chat feature and will return back to the main menu. (See Figure 1a in the Appendix).

Group

If the client inputs the word ‘GROUP’ in the main menu, they will enter the group chatting feature of the program. They will first be prompted to choose whether they want to join an existing group or create a new group by choosing the option ‘E’ or ‘N’.

- If the client chooses the existing group option then they will be opted to type the name of the group to join. The following protocol will be sent to the server ‘JOIN_GROUP#group_name’. The server will return a flag to determine if the group exists or not and let the client know (Figure 3e).
- If the option is chosen to create a new group then the client will be prompted to input the name of the new group, followed by the usernames of the people that they want to add. The following protocol will then be sent to the server ‘NEW_GROUP#group_name#members1#member2#...’ where a group will be stored. The user is then prompted with a list of users that were not found and confirmation of a successfully created group (Figure 3d).

The client is now in the group chat function and any message that they send will be sent to the server as the following protocol: ‘GROUP_CHAT # group_name # message’ where the server will send the message to all active users in that group. When the client types LEAVE then they will exit the group chat function and be sent back to the main menu. (See Figure 1b in the Appendix).

Broadcast

If the client inputs the word ‘BROADCAST’ in the main menu, the server will be sent the protocol ‘CHAT#All#Message’. The server will then forward the message onto all active users on the server by looping through the list current_ips. The sending client will receive a ‘SENT’ and ‘RECEIVED’ with the receiving client’s ip address attached to inform the sending client who their message was sent to (Figure 3c).

Active

If the client inputs ‘ACTIVE’ in the main menu, the server will be sent the command ‘ACTIVE’ and then a string version of the list of active users (current_ips) will be sent to the client and displayed on their side. From this list, the client will be able to see who is available to chat (Figure 3f).

Exit

If the client inputs ‘EXIT’ in the main menu, the server will be sent the following protocol: ‘EXIT#clientName’. The server will then get the name and client address of the client that sent the request and remove them from the list of active users (current_ips). This change will be shown on the server side when the list of users is displayed. After the client has sent the protocol to the server, it will call os._exit(0) which will exit the program.

Encryption

To encrypt messages we used an existing library called Base64. Before a message is sent from the client to the server, it is encrypted and this encrypted version is included in the protocol. The message is sent to the server and then onto the specific client and only then is the message decrypted and displayed on the client side. This is to ensure that while the message is being sent between clients, it cannot be read on the server side or by a client it is not meant for and all private information is protected.

Hashing

Computes the hash of the message to be sent using md5 in the python library hashlib and attaches it to the end of the message. This hash value will be checked on the receiving client side to ensure that there has been no message loss during transfer. If the hash function computed on the receiving client side does not match the one computed on the sending client side, the message will not be shown to the receiving client and instead the receiving client will be told that the message was corrupted and the sending client will be made aware of the corrupted message. This was implemented due to the lack of guaranteed lossless transmission under the UDP protocol.

Message Verification

When a message is sent by a client and the server receives it, a message will be sent back to that original client to notify them that the server has received the message ("SERVER: SENT"). The server then forwards that message onto the intended client and when it is received, the recipient then sends a message back to the server and finally then to the original client to notify them that their message was received by the intended client ("SERVER: 'clientName' RECEIVED").

User and server passwords

In order to ensure secure, controlled access to this communication server, there is a password to 'unlock' the server and make it ready to use. This password has been set to 'password' for simplicity's sake. Additionally, each user has a password that is saved in the database that they will use upon every login attempt. New users who have not accessed the server before will be prompted to choose a password to use in the future.(See Figure 1c in Appendix A)

Persistent storage

A connection was set up to a MS Access database (called 'db.accdb') that is used to store each user's password, name, IP address and unique User ID. This allows for a secure log-in of each user to the server and allows for new users to create passwords that will be saved persistently on the server side. (See Figure 2 of Appendix A for the database design and relationships)

Creativity

To add creativity to our code, we included encryption and authentication and used a database to store user IDs and passwords.
We encrypted each message that is sent to the clients online. If a message is being sent from one client to another client, the message will pass through the server and to the other client in an encrypted form so that no unauthorised users can see what the message contains. Any message from the server that is sent to a client is also encrypted and only decrypted when the intended client receives the message.
When a client tries to access the server, they are prompted to enter their username and login information which is then stored in a database. This is to keep a record of past users and add new users to the database.

Protocol Messages

Each message sent by the client is sent in a specific format so that the server understands exactly how to process the commands and inputs the user supplies it with. Here is a list of protocols implemented in our code:

- CHAT#Name#Message`hash
- GROUP_CHAT#GroupName#Message`hash
- CHAT#All#Message`hash
- ACTIVE
- EXIT
- LEAVE
- JOIN_GROUP#GroupName
- CHECK_FOR_GROUP#GroupName

Each protocol contains a header but only some protocols have a body that is sent. In the first example above, the header is 'CHAT#Name', where 'CHAT' is the command and 'Name' is the recipient's information, while the body is the message and hash combined ('#Message`hash'). The protocol for the group chats and broadcasting features follow the same format. 'ACTIVE', 'EXIT' and 'LEAVE' only have a header, and more specifically, a command.

Error Handling

Various methods have been used to deal with erroneous input and user behaviours. These would include non-standard input such as asking to chat with a user that is not online or does not exist in the database and message corruption warnings.

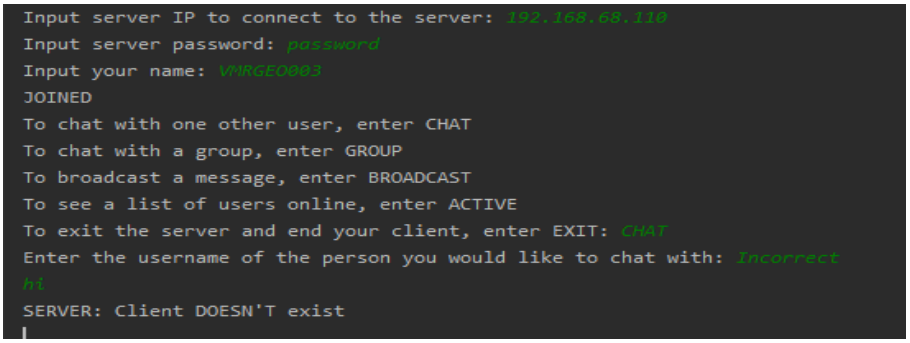


Figure 4a: Screenshot showing the outcome of trying to chat with a user that is currently not on the server.

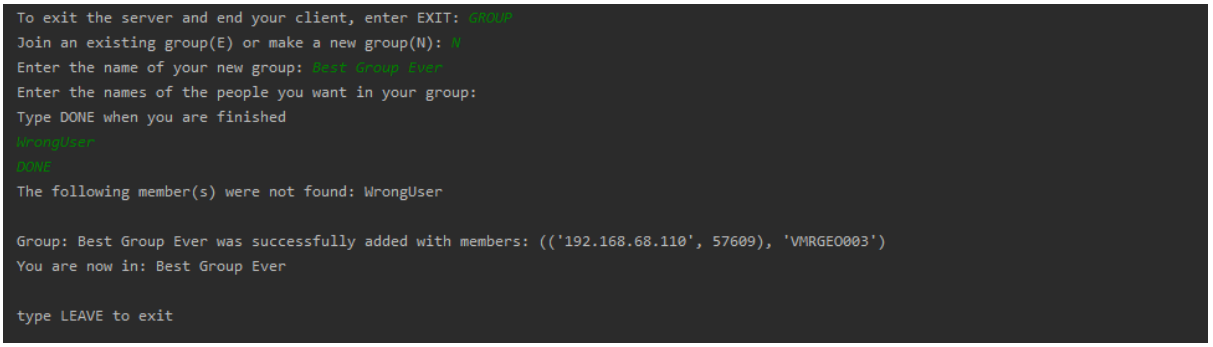


Figure 4b: Screenshot showing the outcome of trying to add a user that is not online to a group chat.

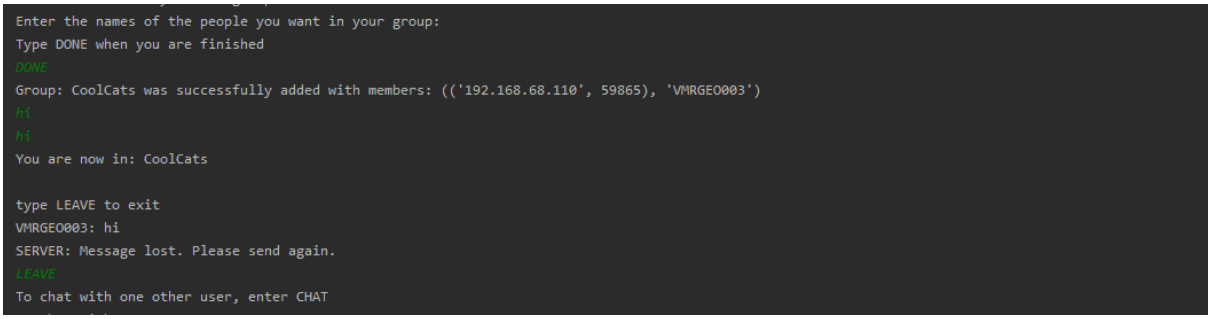


Figure 4c: Screenshot showing the outcome of sending a message before the server is ready to receive said message, resulting in a lost message.

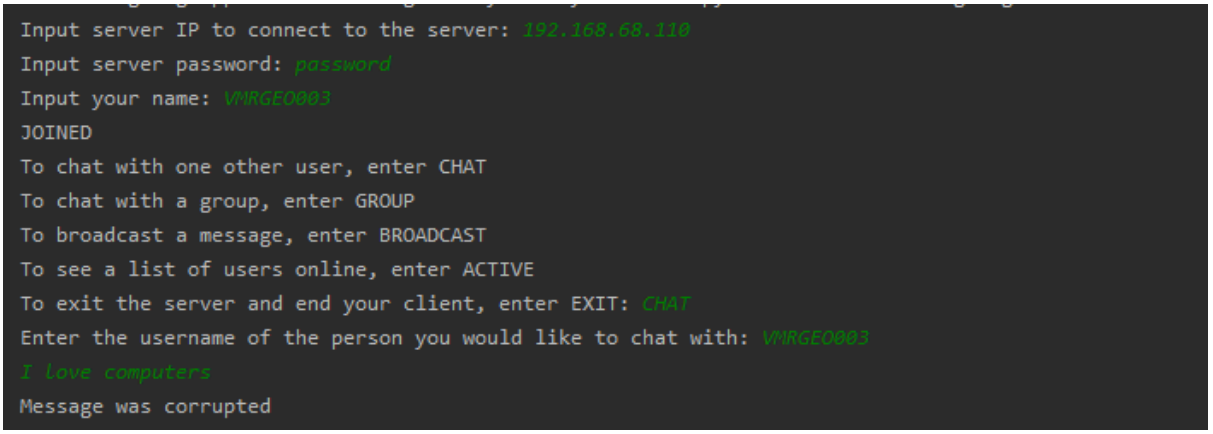


Figure 4e: Screenshot simulating the outcome ("Message was corrupted") of lost messages or packets being sent.

Group Members and Contributions

- Rinya Singh SNGRIN001
- Cameron Rebelo RBLCAM001
- Georgie van der Merwe VMRGEO003

All group members met for a few hours almost everyday for the past two weeks since receiving the assignment. In the evenings after our meetings took place, each group member worked on a specific part of code. For example, Rinya was in charge of implementing a database, Cameron was responsible for the group chat feature, while Georgie was responsible for the one-on-one chat feature and encryption. The report was also a collaborative effort.

Appendix

Figure 1a:

Sequence Diagram depicting the Chat state

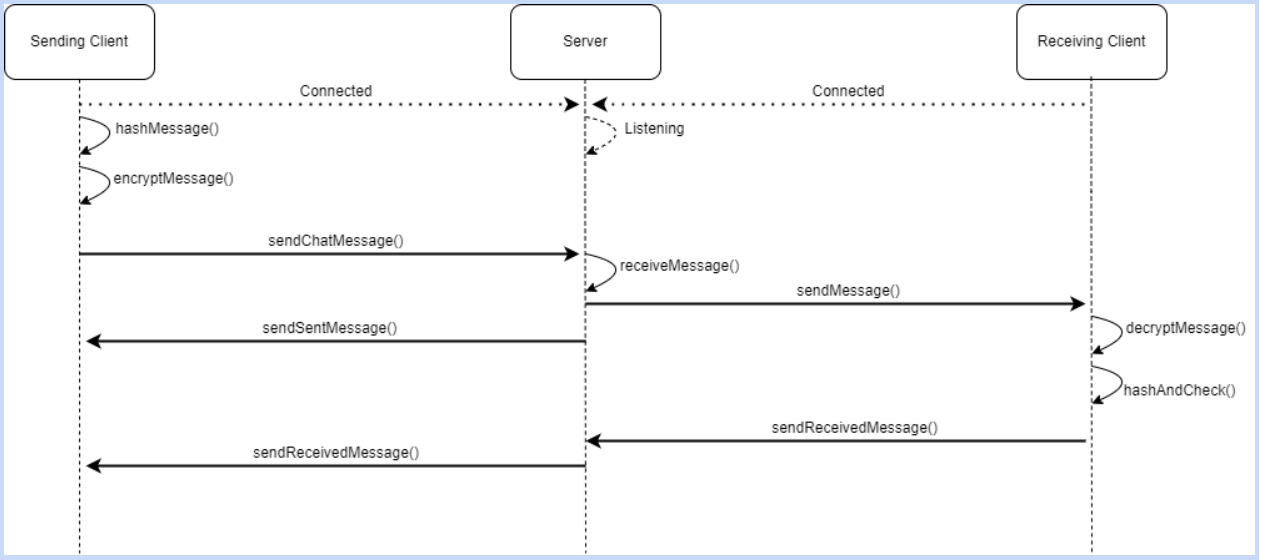


Figure 1b:

Sequence Diagram depicting the group chat state

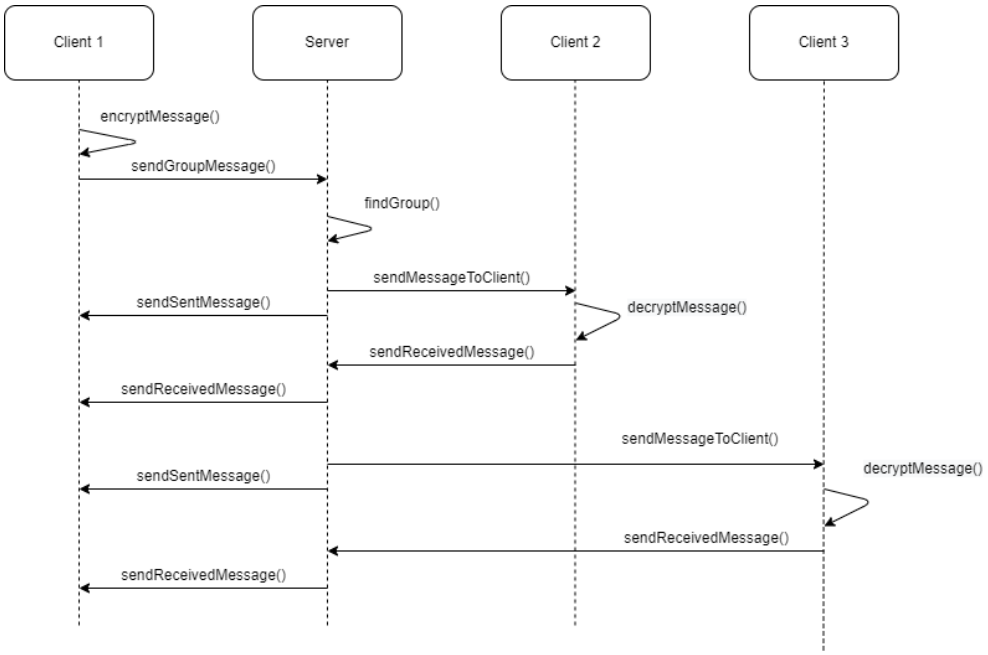


Figure 1c:

Sequence Diagram depicting Server/Database connection state

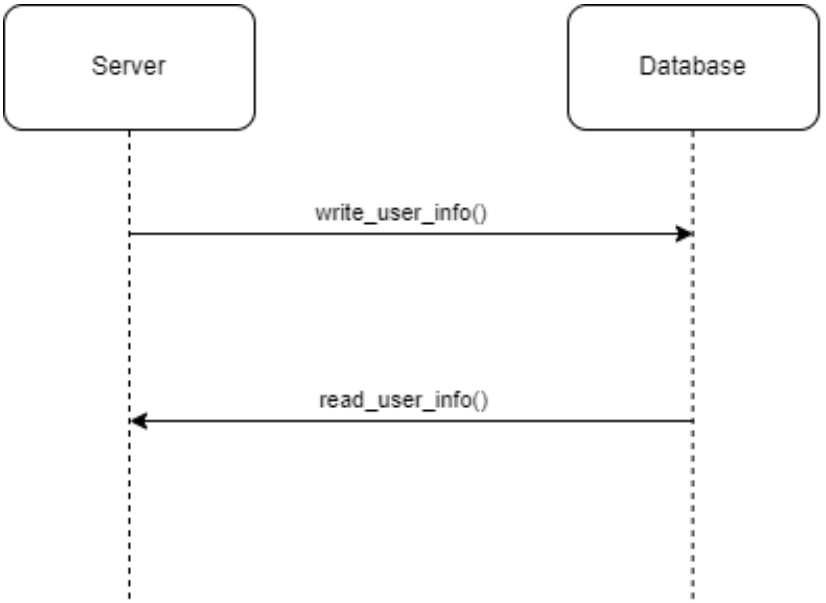


Figure 1d:

Sequence Diagram depicting the EXIT state

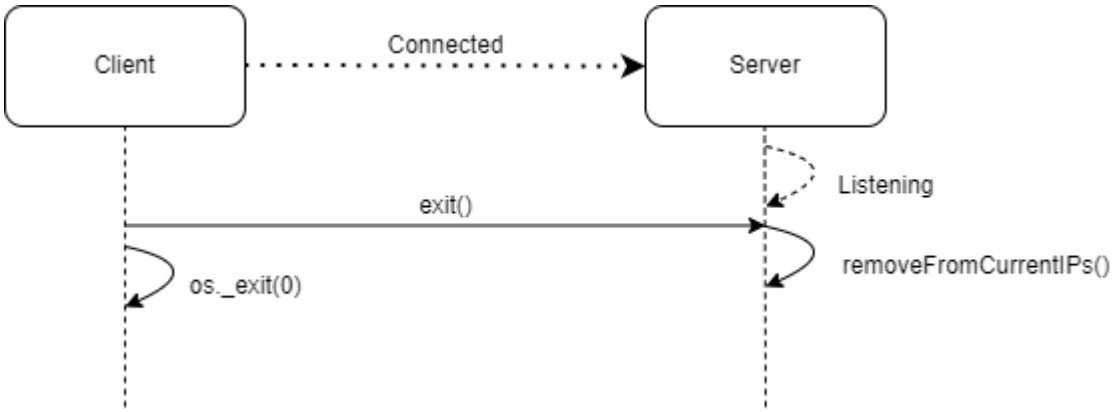


Figure 1e:

Sequence Diagram depicting the ACTIVE action

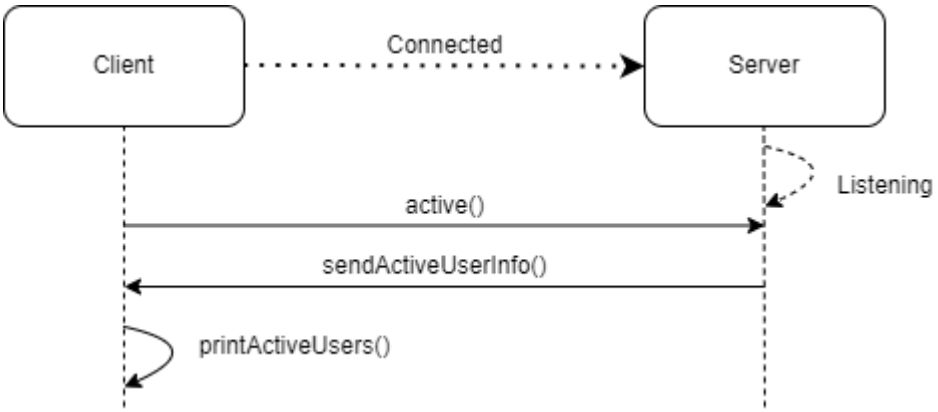
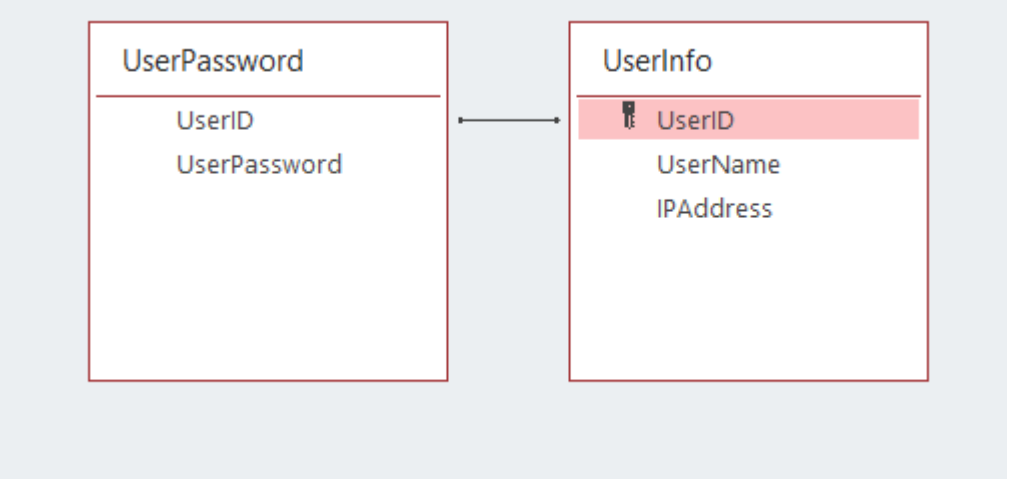


Figure 2:

Relational database



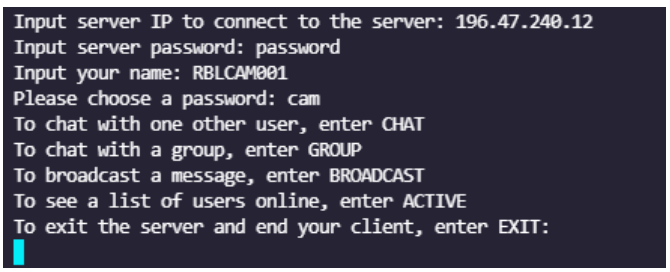


Figure 3a: Main Menu of program

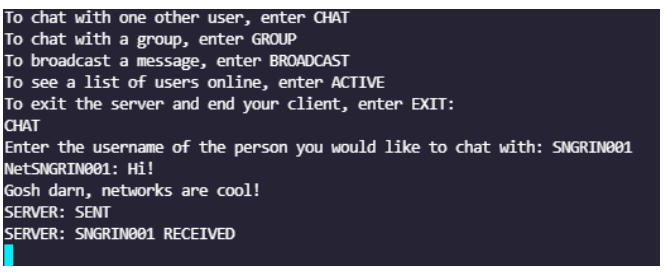


Figure 3b: Chat feature with one person

```
Input server IP to connect to the server: 196.47.240.12
Input server password: password
Input your name: RBLCAM001
Please enter your password: cam
LOGGEDIN
To chat with one other user, enter CHAT
To chat with a group, enter GROUP
To broadcast a message, enter BROADCAST
To see a list of users online, enter ACTIVE
To exit the server and end your client, enter EXIT:
BROADCAST
Hey guys!
SERVER: SENT
SERVER: VMRGEO003 RECEIVED
SERVER: SENT
SERVER: SNGRIN001 RECEIVED
VMRGEO003: hi
```

Figure 3c: Broadcast message sent to all active user

```
To chat with one other IP, enter CHAT
To chat with a group, enter GROUP
To broadcast a message, enter BROADCAST
To see a list of users online, enter ACTIVE
To exit the server and end your client, enter EXIT: GROUP
Join an existing group(E) or make a new group(N): N
Enter the name of your new group: CS: Group
Enter the names of the people you want in your group:
Type DONE when you are finished
RBLCAM001
VMRGEO003
DONE
Group: CS: Group was successfully added with members: (('196.47.240.12', 56575), 'SNGR
IN001')(('196.47.192.72', 56444), 'RBLCAM001')(('196.42.77.126', 64726), 'VMRGEO003')
You are now in: CS: Group

type LEAVE to exit
```

Figure 3d: Creating a new group

```
Group: CS: Group was successfully added with members: (('196.47.240.12', 56575), 'SNGR
IN001')(('196.47.192.72', 56444), 'RBLCAM001')(('196.42.77.126', 64726), 'VMRGEO003')
You are now in: CS: Group

type LEAVE to exit
RBLCAM001: I <3 Databases
text filez rule!
SNGRIN001: text filez rule!
SERVER: SENT
SERVER: RBLCAM001 RECEIVED
SERVER: SENT
SERVER: VMRGEO003 RECEIVED
LEAVE
To chat with one other IP, enter CHAT
To chat with a group, enter GROUP
To broadcast a message, enter BROADCAST
To see a list of users online, enter ACTIVE
To exit the server and end your client, enter EXIT: EXIT
PS C:\Users\rinya\Projects\NetChat\FINAL>
```

Figure 3e: Joining an existing group, sending and receiving messages then leaving the group and then the server

```
Input server IP to connect to the server: 196.47.240.12
Enter the server password: password
Input your name: VMRGEO003
Please choose a password: password123
To chat with one other user, enter CHAT
To chat with a group, enter GROUP
To broadcast a message, enter BROADCAST
To see a list of users online, enter ACTIVE
To exit the server and end your client, enter EXIT: ACTIVE
[(('196.42.77.126', 63411), 'VMRGEO003'), (('196.47.192.72', 59313), 'RBLCAM001')]
To chat with one other user, enter CHAT
To chat with a group, enter GROUP
To broadcast a message, enter BROADCAST
To see a list of users online, enter ACTIVE
To exit the server and end your client, enter EXIT: ACTIVE
[(('196.42.77.126', 63411), 'VMRGEO003'), (('196.47.192.72', 59313), 'RBLCAM001'), (('196.47.240.12', 64464), 'SNGRIN001')]
To chat with one other user, enter CHAT
To chat with a group, enter GROUP
To broadcast a message, enter BROADCAST
To see a list of users online, enter ACTIVE
To exit the server and end your client, enter EXIT:
```

Figure 3f: Checking active users, then a user leaves and checking again to see if they have left