**Mthokozeleni Sithole STHMTH002 (Individual Work) Report**

**Description of functionality and features**

**App Architecture**

The app consist of 4 classes split into 2 python script files. There is a client class which launches and controls the login GUI and initiates the sockets responsible for sending and receiving messages. The client class has 4 function including the constructor, the methods are genPort, login and startRun . genPort is used to generate a random port number to be used by the client. The login method is called when the “LOGIN” or “GROUP” buttons are pressed, it collects the username and the recipient’s name from the entry widgets and sends it to the server. And finally startRun creates an instance of sockets are used to communicate with the server, creates the login GUI and invokes message handler threads which are used during the chat session to send and receive messages and listen for GUI events.

The messageHandler class inherits from the threading class and overrides the run method. The run method constantly listens to receive messages the from the server sockets and display them on the text field whilst listening for events on the GUI. There are getters and setters method used to communicate with other methods and the parent Client class, other significant methods are the display method used to display messages on the text widget once they are received from the server. The getHistory  method is used to retrieve the users chat history including messages sent while user is offline. The chat history is retrieved by reading a text file created by the server to keep chat logs. Lastly there is the sendRply method  used to fetch text from entry widget and send to the server.

On the server side, apart from a couple of getters there is the startRun , sendMsg  and groupMsg  on the Server class . startRun Initiates sockets and bind then listens for login details then updates local variables. After that the method attempts to read from chat logs file if it already exist if there are any recent messages to be sent to the client that has just logged in. client handler threads are then spawned which handle receiving messages from clients and sending them to appropriate recipients. Client handler threads are created according to the type of recipient the user select, then command messages are to notify the users of any changes. Below are the sendMsg and groupMsg used to send to messages to private chat and group chats respectfully. The message is first concatenated meta data required then sent.

The clientHandler class  also inherits from the threading class. Apart from getters from setters the class has the writeChatLogs method  which appends chat logs or create a new file if it does not exist then it write chat logs. The run method continuously receives and sends messages to the stipulated recipient in the meta data and write or update chat logs. All the features of the App, login window/GUI, chatting window/GUI ,private and group chat and retrieving chat history are implemented through the above classes and methods. All libraries used are built in python libraries including Tkinter which is used to create the graphical user interfaces-GUI.

**Class analysis diagram**

Diagram

Description automatically generated**Features**

Once the scripts are running the client receive a login screen where they can login at different times if one user logs in and sends messages to an offline user they are stored in the history and will be retrieved by the recipient when they login. To login clients/user enter details, click login then close the login window. To login into a group clients enter their user name then click on the group button then close the login window. Clients in a private chat are separated from group chat activity and cannot send or receive group messages. The get history function ensure reliability by writing chat logs that can be accessed if messages are not sent. While chat once the user press send the entry is cleared to improve the user experience.

**Private Chat login**Graphical user interface, treemap chart

Description automatically generated

**Chat windows**

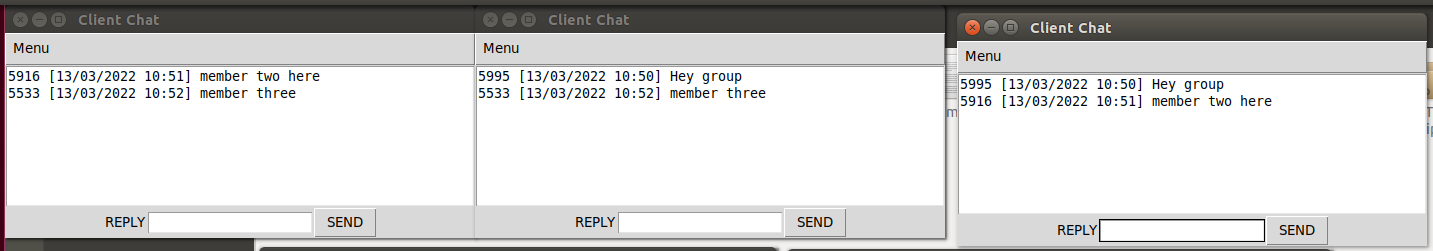
Graphical user interface, text, application

Description automatically generated

**Group Chat**

Text

Description automatically generated



Messages are sent to every member of the group chat except the sender. A member can log off and login later then retrieve the chat history to access the missed messages.

Get History and Server monitoring activity

Text

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedText

Description automatically generated

**Specification (Protocol design & specification (sequence diagrams & message formats/structure))**

**Message Protocol**

Login

|  |  |
| --- | --- |
| Source Port | Destination Port |
| length | Checksum |
| Body: login message= Username +\n+ | Recipient +\n+ Sender port number |

Chat

|  |  |
| --- | --- |
| Source Port | Destination Port |
| length | checksum |
| Body: Message +\n+ sender Port number +\n+ | Port number +\n+ Recipient username |

The header of the UDP protocol includes the source and destination port, length and checksum which are required by the sockets and the body of the carries the message encoded into bytes. The format of the message helps the server implement our manual protocol, the server utilizes the username, recipient name and sender port number to create a connection from the login packet/data. Once in the chat state the app uses the format of the message to identify the appropriate recipient and a port number to maintain connection to respond and for the identify whom the message comes from.

**Sequence Diagram(If not visible PNG folder is attached in submission)**

Graphical user interface, application

Description automatically generated

The sequence diagram explains the flow of user input from the GUI into the server and to the recipient. And the change of states from login state to the chat state throughout the process.

Conclusion

The are many challenges that I faced while trying to implement the chat app, the major challenges include communicating and updating variables in the main thread as they are manipulated by the client or message handler threads. The Tkinter GUI library can only be accessed or manipulated by the main thread on which is created and the mainloop method is called, this lead to the creation of a separate login and chat GUI. Simulation errors was also a challenge as I do not how that can be implemented. Whilst logging in control messages sent by server are not received until the login window is closed. Beyond that the chat app is quite robust and can handle different situations such linked logins where one user is in a private chat then a third user logs in to chat with one the active users.

**References**

docs.python.org, [www.guru99.com](http://www.guru99.com), stackoverflow.com , [www.simplilearn.com](http://www.simplilearn.com), [www.w3schools.com](http://www.w3schools.com), [www.tutorialspoint.com](http://www.tutorialspoint.com), superuser.com, [www.adamsmith.haus](http://www.adamsmith.haus), realpython.com, [www.geeksforgeeks.org](http://www.geeksforgeeks.org), python.tutorialink.com, [www.freecodecamp.org](http://www.freecodecamp.org), clouds.eos.ubc.ca, pythontic.com, [www.reddit.com](http://www.reddit.com), github.com , help.socketlabs.com, careerkarma.com, cppsecrets.com, [www.techwithtim.net](http://www.techwithtim.net), medium.com, developer-shubham-rasal.medium.com, python.plainenglish.io, [www.linkedin.com](http://www.linkedin.com), pythonprogramming.net, python-docs.readthedocs.io, [www.programiz.com](http://www.programiz.com), www.techwalla.com, [www.delftstack.com](http://www.delftstack.com), [www.studytonight.com](http://www.studytonight.com), [www.programcreek.com](http://www.programcreek.com), codefather.tech, [www.learndatasci.com](http://www.learndatasci.com), [www.onooks.com](http://www.onooks.com), python-list.python.narkive.com, [www.codespeedy.com](http://www.codespeedy.com), www.youtube.com,