Computer Networks and IOT Lab Project Report

Chatterbox: A Chat Messenger

Jaypee Institute of Information Technology

Submitted by

Ananya Kapoor	**20103104**	**B4**
Dhairya Sachdeva	**20103098**	**B4**

Abstract

The purpose of this project is to develop a network chat messenger that will provide a platform for people to communicate and exchange information in real-time. The chat messenger will enable users to communicate with one another by sending text messages. The project will be developed using Python programming language with a focus on creating a user-friendly interface that will allow users to easily navigate through the application.

The network chat messenger will be developed using socket programming to ensure that users can communicate seamlessly in real-time. The project will leverage the client-server model to enable users to connect to a central server from where they can exchange messages with other users. The server will be responsible for maintaining user details, routing messages to the appropriate recipient, and ensuring the security of user data.

The chat messenger will be developed with features that will ensure the security of user data. These features will include end-to-end encryption, user authentication, and data privacy. The project will also include a user-friendly registration and login system to ensure that users' data is protected and only accessible by authorized persons.

Overall, the development of this network chat messenger will provide users with a reliable and secure platform for communicating with one another in real-time. The project will be developed with the aim of meeting the needs of users who require an easy-to-use chat messenger that is secure and reliable.

Scope of the Project

a) Functional Requirements

Here are some potential functional requirements for a network chat messenger project:

1. Chatting functionality: The chat messenger should allow users to send and receive messages in real-time, including text messages, voice notes, and video messages.
2. Group chats: The chat messenger should allow users to create or join group chats with multiple users.

3. User profile: The chat messenger should allow users to customize their profiles, including adding profile pictures and updating personal information.
4. Privacy and security: The chat messenger should include features that ensure the privacy and security of user data, including end-to-end encryption, user authentication, and data privacy.

b) Non-functional requirements

Here are some potential non-functional requirements for a network chat messenger project:

1. Performance: The chat messenger should be able to handle a high volume of messages and users without experiencing delays or crashes.
2. Usability: The chat messenger should be intuitive and easy to use, with a user-friendly interface that requires minimal training.
3. Reliability: The chat messenger should be reliable, with minimal downtime or outages.
4. Scalability: The chat messenger should be designed to scale as the number of users and messages increases over time.
5. Security: The chat messenger should be designed with security in mind, with features such as end-to-end encryption, two-factor authentication, and data privacy.

c) Description of the modules of the project

Here are some potential modules for a network chat messenger project:

1. User management module: This module would be responsible for user registration, login, and profile management. It would also handle user authentication and authorization.
2. Chatting module: This module would handle the core functionality of the chat messenger, including sending and receiving messages, group chats, and file sharing.
3. Security module: This module would handle encryption and decryption of messages, as well as user authentication and data privacy.

4. Maintenance module: This module would handle maintenance and updates to the chat messenger, ensuring that it remains functional and up-to-date over time.

Tools and Technologies Used

Here are some potential tools and technologies that could be used in a network chat messenger project:

1. Programming languages: The chat messenger could be developed using a variety of programming languages, including Python, Java, or JavaScript.
2. Socket programming: Socket programming could be used to enable real-time communication between users on the chat messenger.
3. WebSockets: WebSockets could be used to enable two-way communication between the client and the server.
4. Encryption libraries: Encryption libraries such as OpenSSL or PyCrypto could be used to provide end-to-end encryption for messages.

Design of the Project

The project has the following key components:

Server: The server component of the project is responsible for receiving and broadcasting messages to all the clients connected to it. It creates a socket object and binds it to a particular IP address and port number. It listens for incoming connections from clients and maintains a list of active clients. When a client connects to the server, the server creates a separate thread to handle the client's request. The server also provides support for sending and receiving encrypted messages, audio messages, files, and pictures.

Client: The client component of the project is responsible for sending and receiving messages to and from the server. The client creates a socket object and connects to the server's IP address and port number. Once connected, the client sends its username to the server, which adds the client to the list of active clients. The client then has the option to send private messages to other clients

or broadcast messages to all clients. The client can also send and receive encrypted messages, audio messages, files, and pictures.

GUI: The graphical user interface (GUI) of the project is developed using Tkinter, a standard Python GUI toolkit. The GUI provides an intuitive interface for the user to interact with the application. It allows the user to enter their username, connect to the server, send messages, and select files and pictures to send. The GUI also displays the messages received from other clients, including private messages and broadcast messages.

Encryption: The project supports encryption of messages using a symmetric encryption algorithm such as AES or DES. When a client sends a message, the message is first encrypted using a pre-shared key. The encrypted message is then sent to the server, which decrypts the message using the same key and broadcasts it to all clients. The encryption key is never transmitted over the network, ensuring secure communication between clients.

Audio Messages, Files, and Pictures: The project supports sending and receiving audio messages, files, and pictures. When a client wants to send an audio message, they select the audio file and send it to the server. The server then broadcasts the audio file to all clients. The same process is followed for sending and receiving files and pictures.

In summary, our web chat room application project is a robust and secure client-server architecture that supports private and broadcast messaging, encryption of messages, and sending and receiving of audio messages, files, and pictures. The project's graphical user interface provides a user-friendly experience, making it easy for users to interact with the application. Overall, this is an impressive project that demonstrates your proficiency in socket programming and Tkinter.

Implementation

This is a simple chat application where clients can connect to a server and exchange messages in a group or privately. The server is implemented using Python's socket library and the client is implemented using the Tkinter library for the GUI and the socket library for the network communication.

The server creates a socket object and listens for incoming connections from clients. When a client connects, the server prompts the client for a username and

adds the client to the list of active clients. It also sends a list of all active users to all clients. The server then spawns a new thread to listen for incoming messages from the client.

The client creates a socket object and attempts to connect to the server. If the connection is successful, the client prompts the user for a username and sends it to the server. It then spawns a new thread to listen for incoming messages from the server. The GUI consists of a text box to display messages, a text box to type messages, and buttons to send messages and switch between group and private messages.

Overall, the application seems to be working fine and can be used as a starting point for a more feature-rich chat application.

we import the necessary modules **socket**, **ast**, and **threading**. We define **HOST** and **PORT** as the IP address and port number, respectively, that the server will be running on. **LISTENER_LIMIT** refers to the maximum number of clients that can connect to the server at any given time. **active_clients** is an empty list that will contain information about all clients that are currently connected to the server. **users** is an empty dictionary that will store the username of each client that connects to the server along with its associated IP address.

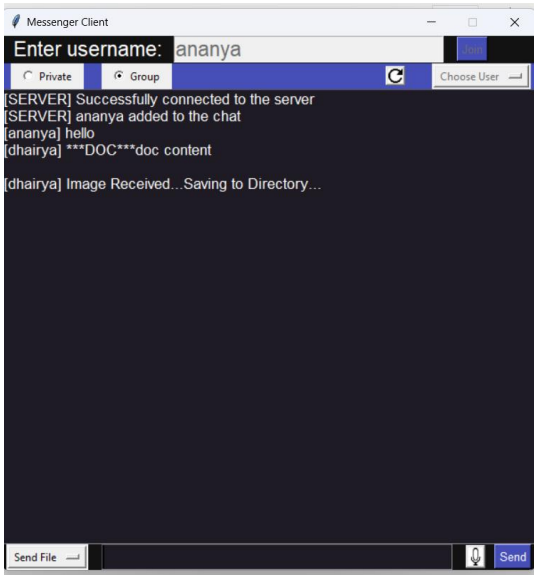
function listens for messages from a particular client. The **client** parameter refers to the socket object that is created when a client connects to the server, and the **username** parameter is the username associated with that client.

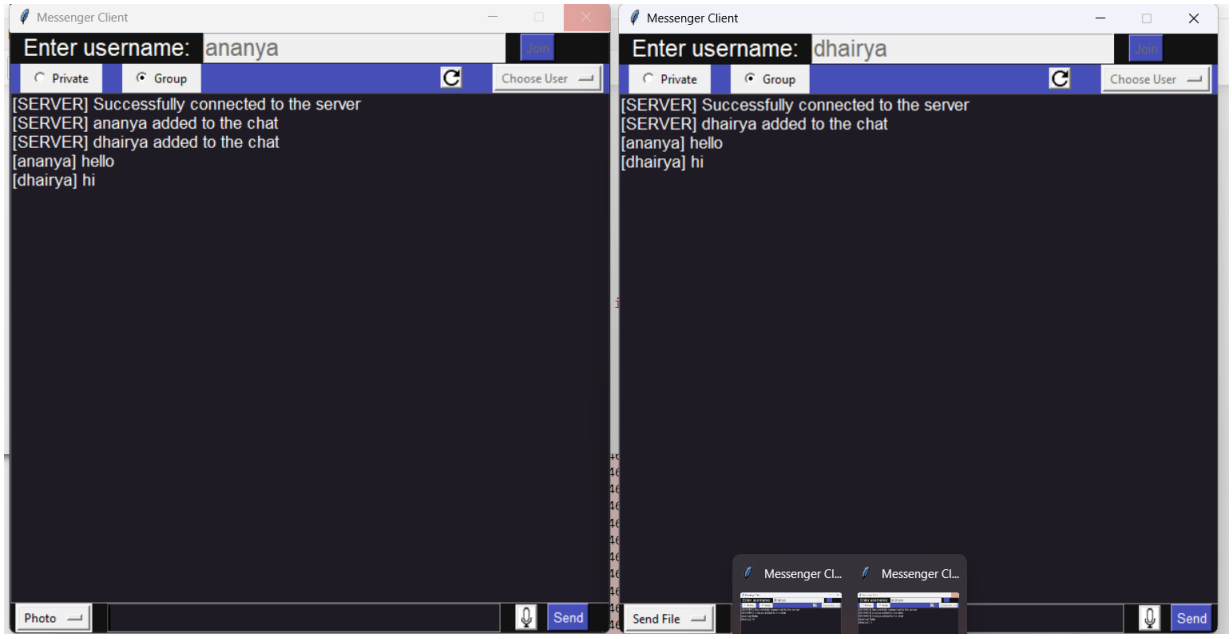
The **while** loop continuously listens for incoming messages from the client. If the message contains the string **"***PRIVATE***"**, it is considered a private message and the server extracts the username of the target client from the message. If the target client is in the **users** dictionary, then the message is forwarded to that client only. Otherwise, an error message is printed indicating that the target client is not in the active clients list.

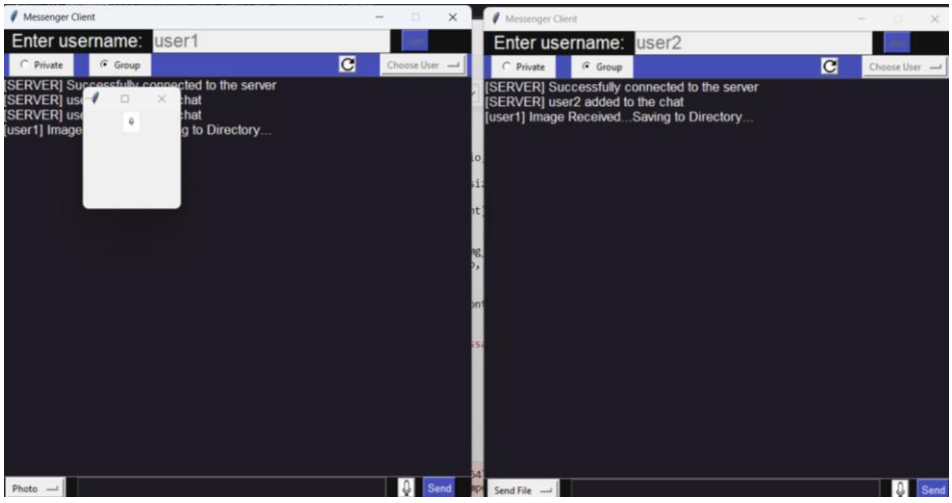
If the message does not contain **"***PRIVATE***"**, it is considered a group message and is sent to all connected clients by calling the **send_messages_to_all()** function. If the message is empty, a message is printed indicating that the message from the client is empty.

two functions are helper functions for sending messages to clients. The **send_message_to_client()** function takes a client socket object and a message as input and sends the message to the specified client.

Testing Details







References

[1] https://realpython.com/python-sockets/

[2]https://www.packtpub.com/product/python-network-programming-third-edition/9781789958092

[3] https://docs.python.org/3/library/socket.html

[4] https://www.udemy.com/course/socket-programming-in-python-beginner-to-advanced/

[5] https://realpython.com/python-sockets/