# Project Report

On

**Chatting Application**

Submitted in fulfillment of the requirement for the award of the degree of

## Master of Computer Applications

(Batch 2021 - 2023)



**By**

**Simran Soni, Jyoti Chauhan**

**1321728,1321712**

## Under the Supervision of

Dr. Bharti Sharma

Assistant Professor

## M.M. Institute of Computer Technology & Business Management Maharishi Markandeshwar (Deemed to be University), Mullana (Ambala)

**(NAAC accredited Grade ‘A++’ University)**

**April 2023**

# DECLARATION

I hereby declare that the work which is being presented in this project report entitled **“Chatting Application”,** in partial fulfillment of the requirement for the award of the degree of **MASTER OF COMPUTER APPLICATIONS** submitted at M.M. Institute of Computer Technology & Business Management, **Maharishi Markandeshwar (Deemed to be University), Mullana, Ambala** is an authentic work done by me during a period from 20 Feburary 2023 to 10 May 2023

Under the Guidance of Dr. Bharti Sharma (Associate Professor) **( Supervisor appointed by the Institute).**

The work presented in this project report has not been submitted by me for the award of any other degree of this or any other Institute/University.

**Signature**

**Simran Soni , Jyoti Chauhan**

**Roll no. – 1321728, 1321712**

**This is to certify that the above statement made by the candidate is correct to best of my knowledge and belief.**

**Date : Place :**

**Signature**

**Dr.** Bharti Sharma

# ACKNOWLEDGEMENT

I would like to express my gratitude and appreciation to all those who gave me the possibility to complete this report. It would not have been possible without the kind support and help of many individuals and organizations. I am highly thankful to Dr. Bharti Sharma for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project. I would like to express my gratitude towards Dr.Sumit Mittal Principal,for their kind cooperation and encouragement which helped me in completion of this project. I would also like to acknowledge with much appreciation the crucial role in my colleagues whose help, stimulating suggestions and encouragement, helped me in developing the project.

Simran Soni (1321728)

Jyoti Chauhan(1321712)

# Contents:

* Chapter 1: Abstract
* Chapter 2: Introduction
* Chapter 3: Main Objective
* Chapter 4: The proposed Solution
* Chapter 5:Requirement
* Chapter 6: Flow Diagram
* Chapter 7: Algorithm
* Chapter 8: Implentation Details
* Chapter 9: Result/Output
* Chapter 10: Operational Concept and Scenarios
* Chapter 11: Limitations & Fututre Scope
* Chapter 12: Conclusion
* References

# INDEX:

|  |  |  |
| --- | --- | --- |
| S.No. | Topic | Page No. |
| 1 | Abstract | 6 |
| 2 | Introduction | 7-8 |
| 3 | Main Objective | 9-11 |
| 4 | The proposed solution | 12-15 |
| 5 | Requirement | 16-17 |
| 6 | Flow diagram | 18 |
| 7 | Algorithm | 19 |
| 8 | Implementation Details | 20-30 |
| 9 | Result/ Output | 31 |
| 10 | Operational Concept and Scenarios | 32-33 |
| 11 | Limitation and Future Scope | 34-35 |
| 12 | Conclusion | 36 |
| 13 | References | 37 |

# CHAPTER-1:**Abstract**

Teleconferencing or chatting refers to any kind of communication that offers a real-time transmission of messages from sender to the receiver. Chatting is a method of using technology to bring people and ideas together despite the geographical barriers. The technology to provide the chatting facility has been available for years, but the acceptance is quite recent. Analysis of chatting provides an overview of the technologies used, available features, functions, system of the architecture of the application, the structure of database of an Instant Messaging application: IChat(IC). The objective of IC application is to facilitate text messaging, group chatting option, data transfer without size restriction which is commonly seen in most of the messaging applications.

# CHAPTER-2: INTRODUCTION

Chatting Application is a desktop based application.

This client server chat application is based on python and used socket package. its simple and easy and require only Basic Python knowledge. The objective of chatting application is to facilitate text messaging, group chatting option, data transfer without size restriction which is commonly seen in most of the messaging applications.

This application/program is a good example of using Python, Socket Programming to create a chat application. A beginner of Python language, who is familiar with this packages can able, be beneficiate.

Chatting is a method of using technology to bring people and ideas “together” despite of the geographical barriers. The technology has been available for years but the acceptance it was quit recent. Our project is an example of a multiple client chat server.

It is made up of 2 applications the client application, which runs on the user’s Pc and server application,which runs on any Pc on the network. To start chatting client should get connected to server. We will focus on TCP and UDP socket connections which are a fundamental part of socket programming.

Keywords: sockets, client-server, Python programming, network programming-socket functions, Multicasting etc.

The Chat Application is very common today offered either via a web application or mobile application. Learning to write a Chat Application is good for understanding many network communication concepts and can be useful to build other network applications. Chat Application provides communication between two parties i.e. sender and receiver. The sender is someone who initiates and send a message to other known as receiver; receiver at other end receives the message. The role of sender and receiver is not fixed and keep exchanging during communication, so in simple words, at a point, someone who sends the message is a sender and who receive the message is called receiver. In networking terms, sender and receiver are denoted as source and destination respectively.

Communication can be of many types depending upon the method of communication and the number of parties involved. Some of the scenarios are :

1. **Simplex or one-way communication:** Only one party is able to send the message and other parties can only receive.
2. **Duplex or two-way communication:** Both parties can send and receive messages.

**Duplex communication** is a common way of communication and can be **one-to-one** (simple chat) or **many-to-many** (chat room)

Generally, in real-world communication is done directly using voice in an ideal situation (distance between communicating parties, identification of parties)  where **sender** speak out and the intended **receiver** respond after listening. So, what is the **medium of communication**here? Indeed, it is the air which helps our voice to travel to the **receiver** and successful communication depends upon air (high wind and long-distance can cause trouble). In online or digital communication the role of air is played by network channel (coaxial cable, fiber optics, etc.) and communication is controlled by a **server**. A **server** is a program which regulates the communication between **sender** and **receiver**.

So, to create a **Python** **Chat Application**, one has to write a **server** program and **client** program/s (**sender** and **receiver**). Suppose, two parties Alice and Bob want to chat with each other and ask you to develop a chat application then being a developer you have to write a **server** **program** and a **client** **program** (different instance of the same program will be used by both Alice and Bob or even more users).

we will demonstrate the aforementioned scenario and will develop a **Python** **Chat Application**for Alice and Bob. **Python** has many modules which can help us to create **network-r**elated application, the **socket** is one of such popular default **Python** modules for **low-level network programming**. We will first list and explain the steps for **server** and **client** programs and then implement the same using **Python**.

**CHAPTER-3: MAIN OBJECTIVE**

The aim of this project is to express how we can implement a simple chat application between a server and a client? The application is a desktop based application and is implemented using Tkinter. The project is developed in Python language executed on a single stand-alone Python across a network using loop back address concept. A chat application makes it easy to communicate with people anywhere in the world by sending and receiving messages in real time. With a web or mobile [chat app](https://www.pubnub.com/use-case/in-app-chat/), users are able to receive the same engaging and lively interactions through custom messaging features, just as they would in person. This also keeps users conversing on your platform instead of looking elsewhere for a messaging solution.

Application consists of two programs:

* **Server**
* **Client**

**Server:**

The server module of the application waits for the client to connect to it. Then if connection is granted a client interacts communicates and connects to the server, it can mutually communicate with the server. The duty of the server is to let clients exchange the messages.

**Server program** has all the logic to control and regulate the **Chat**, so most of the chat logic is implemented with a **server** program. So first step of communication is to identify the users, how server do this? In network communication, users are identified by a **socket** which is nothing but a combination of **IP address** and **port address**. So, for human understanding, Alice and Bob will be chatting but for a **network**, it is two **sockets** process which is sending and receiving bytes.  Steps involved in this process is as follows:

1. Create socket
2. Communicate the socket address
3. Keep waiting for an incoming connection request/s
4. Connect to client
5. Receive the message
6. Decode the destination user and select the socket
7. Send a message to the intended client
8. Keep repeating step 5 & 6 as per users wish
9. Exit i.e. end the communication by terminating the connection

**Client:**

The client module is the one that utilizer sends requests to the server. Utilizer utilizes the client as the means to connect to the server. Once he establishes the connection, he can communicate to the connected server.

**Client** script is run by the user, so the same **client** code will be run by a different user but each will have a separate **socket** so they will have their unique **communication channel**. Client script uses to be thin because it has very less work i.e. it only connect with the **server** and send and receive messages. The steps involved in **client script** are:

1. Create a unique client socket per instance/user
2. Connect to the server with given socket address (IP and port)
3. Send and receive messages
4. Repeat step 3 as per configuration
5. Close the connection

**CHAPTER-4: THE PROPOSED SOLUTION**

This project is to express how we can implement a simple chat application between a server and a client? The application is a desktop based application and is implemented using Tkinter. The project is developed in Python language executed on a single stand-alone Python across a network using loop back address concept.

Chatting is a method of using technology to bring people and ideas “together” despite of the geographical barriers. The technology has been available for years but the acceptance it was quit recent. Our project is an example of a multiple client chat server.

It is made up of 2 applications the client application, which runs on the user’s Pc and server application,which runs on any Pc on the network. To start chatting client should get connected to server. We will focus on TCP and UDP socket connections which are a fundamental part of socket programming.

This client server chat application is based on python and used socket package. its simple and easy and require only Basic Python knowledge. The objective of chatting application is to facilitate text messaging, group chatting option, data transfer without size restriction which is commonly seen in most of the messaging applications.

Socket Programming

Sockets can be thought of as endpoints in a communication channel that is bi-directional and establishes communication between a server and one or more clients. Here, we set up a socket on each end and allow a client to interact with other clients via the server. The socket on the server side associates itself with some hardware port on the server-side. Any client that has a socket associated with the same port can communicate with the server socket. 

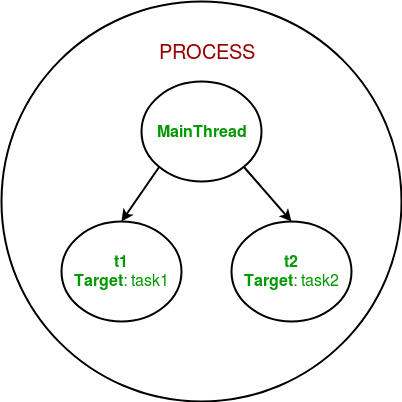
Multi- Threading

Running several threads is similar to running several different programs concurrently, but with the following benefits −

* Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
* Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

* It can be pre-empted (interrupted)
* It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

**Usage:**

This server can be set up on a local area network by choosing any on the computer to be a server node, and using that computer’s private IP address as the server IP address.   
For example, if a local area network has a set of private IP addresses assigned ranging from 192.168.1.2 to 192.168.1.100, then any computer from these 99 nodes can act as a server, and the remaining nodes may connect to the server node by using the server’s private IP address. Care must be taken to choose a port that is currently not in usage. For example, port 22 is the default for ssh, and port 80 is the default for HTTP protocols. So these two ports preferably, shouldn’t be used or reconfigured to make them free for usage.   
However, if the server is meant to be accessible beyond a local network, the public IP address would be required for usage. This would require port forwarding in cases where a node from a local network (node that isn’t the router) wishes to host the server. In this case, we would require any requests that come to the public IP addresses to be re-routed towards our private IP address in our local network, and would hence require port forwarding.   
For more reading on port forwarding: [link](https://en.wikipedia.org/wiki/Port_forwarding)  
To run the script, simply download it from the GitHub link specified at the bottom of the post, and save it at a convenient location on your computer.

/\* Both the server and client script can then be run

from the Command prompt (in Windows) or from bash

Terminal (Linux users) by simply typing

"python chat\_server.py " or "python client.py ".

For example, \*/

python chat\_server.py 192.168.55.13 8081

python client.py 192.168.55.13 8081

Below is the Server side script that must be run at all times to keep the chatroom running.

**CHAPTER-5: REQUIREMENTS**

* **External Interface Requirements**

**User Interface**

The user interface required to be developed for the system should be user friendly and attractive.

There are two sets of Python APIs for graphics programming:

Tkinter (Abstract Windowing Toolkit).

* Tkinter Module was introduced in Python3. Most of the GUI components have become obsolete and should be replaced by newer GUI components.
* Tkinter, a much more comprehensive set of graphics libraries that enhances the GUI, was introduced as part of Python3 Foundation Classes after the release of Python2.

**Software Interfaces**

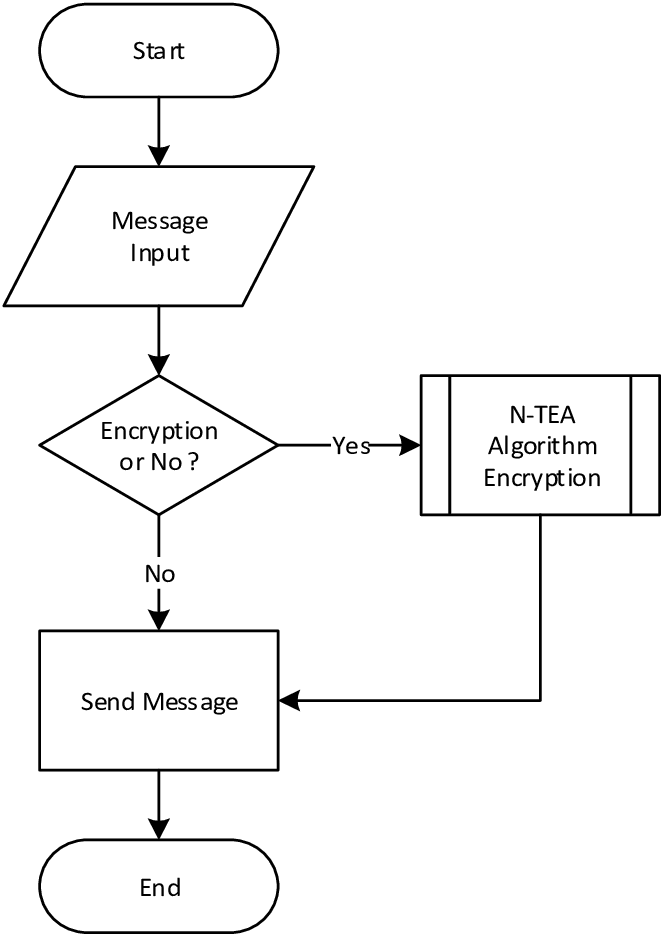
**Programming Language**: Python and Socket Programming

**Compiler/Interpreter:** CPython

**Text Editor**: Visual Studio

**Other Tools:** Tkinter

**CHAPTER-6: FLOW DIAGRAM**



**CHAPTER-7: ALGORITHM**

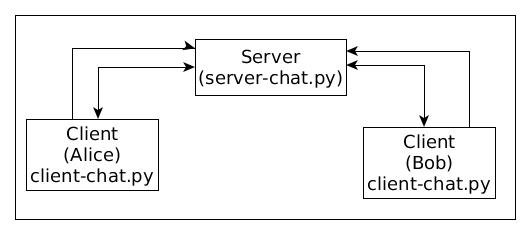
Here is an algorithm chat application using socket programming:

* Here is an algorithm of a server-side chat application [**1**](https://www.studentcpu.com/2009/12/how-implementation-of-chat-between.html):

1. Start the program.
2. Declare the variables and structure.
3. Create a socket using socket functions.
4. Bind the socket to a specified port.
5. Listen for incoming connections.
6. Accept incoming connections.
7. Send and receive messages between the client and server.
8. Close the connection.

* Here is an algorithm of a client-side chat application [**1**](https://www.studentcpu.com/2009/12/how-implementation-of-chat-between.html):

1. Start the program.
2. Declare the variables and structure.
3. Create a socket using socket functions.
4. Connect to the server.
5. If the connection is successful, then send a message to the server.
6. Receive messages from the server.
7. Respond to messages from the server.
8. Close the connection.

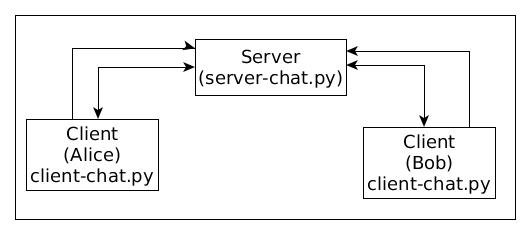


**CHAPTER-8: IMPLEMENTATION DETAILS**

This project can be mainly divided into two modules:

1. Server

2. Client



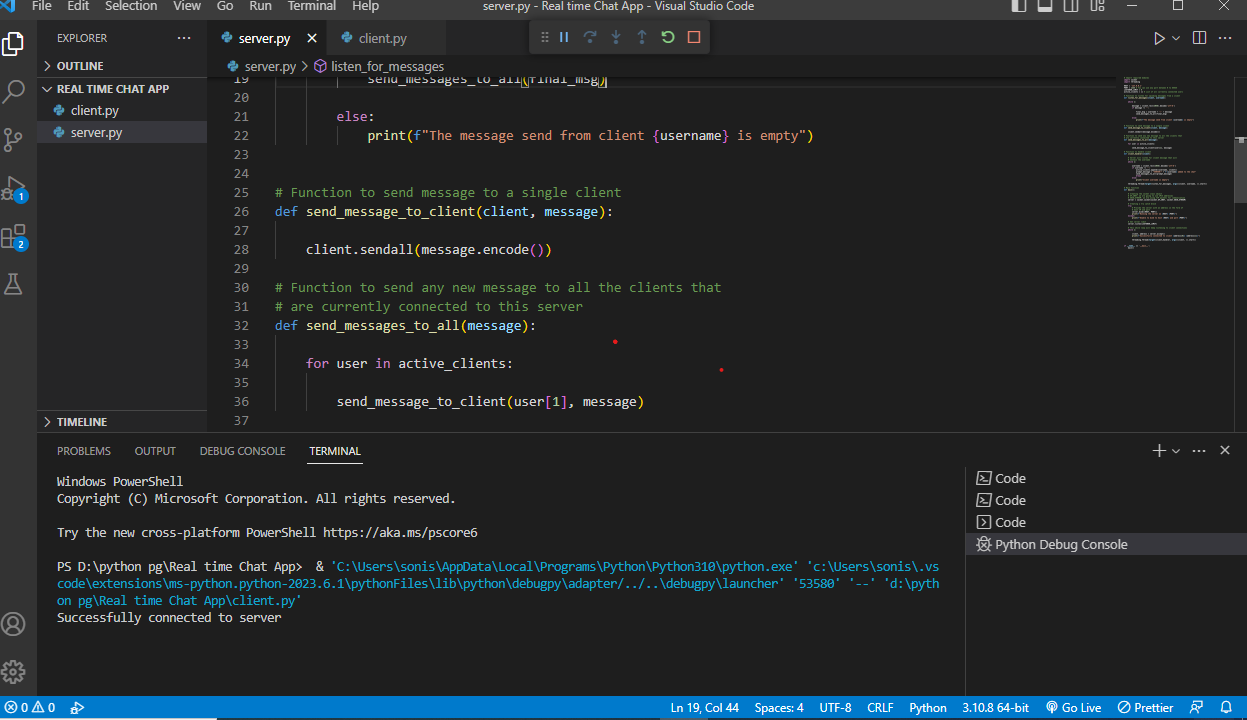
This project is mainly depended on client/server model. The client requests the server and server responses by granting the clients request. The proposed system should provide both of the above features along with the followed ones:

* Server

A server is a computer program that provides services to other computer programs (and their users) in the same or other computers. The computer that a server program runs in is also frequently referred to as a server. That machine may be a dedicated server or used for other purposes as well. Example Server, Database, Dedicated, Fileserver, Proxy Server, Web Server. The server is always waiting for client’s requests. The client come and go down but the server remains the same.

A server application normally listens to a specific port waiting for connection requests from a client. When a connection request arrives, the client and the server establish a dedicated connection over which they can communicate. During the connection process, the client is assigned a local port number, and binds a socket to it. The client talks to the server by writing to the socket and gets information from the server by reading from it. Similarly, the server gets a new local port number (it needs a new port number so that it can continue to listen for connection requests on the original port). The server also binds a socket to its local port and communicates with the client by reading from and writing to it. The client and the server must agree on a protocol that is, they must agree on the language of the information transferred back and forth through the socket. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

THE SERVER SCREEN



**The server and client are two separate programs:**

Server.py :

# Import required modules

import socket

import threading

HOST = '127.0.0.1'

PORT = 1234 # You can use any port between 0 to 65535

LISTENER\_LIMIT = 5

active\_clients = [] # List of all currently connected users

# Function to listen for upcoming messages from a client

def listen\_for\_messages(client, username):

    while 1:

        message = client.recv(2048).decode('utf-8')

        if message != '':

            final\_msg = username + '~' + message

            send\_messages\_to\_all(final\_msg)

        else:

            print(f"The message send from client {username} is empty")

# Function to send message to a single client

def send\_message\_to\_client(client, message):

    client.sendall(message.encode())

# Function to send any new message to all the clients that

# are currently connected to this server

def send\_messages\_to\_all(message):

    for user in active\_clients:

        send\_message\_to\_client(user[1], message)

# Function to handle client

def client\_handler(client):

    # Server will listen for client message that will

    # Contain the username

    while 1:

        username = client.recv(2048).decode('utf-8')

        if username != '':

            active\_clients.append((username, client))

            prompt\_message = "SERVER~" + f"{username} added to the chat"

            send\_messages\_to\_all(prompt\_message)

            break

        else:

            print("Client username is empty")

    threading.Thread(target=listen\_for\_messages, args=(client, username, )).start()

# Main function

def main():

    # Creating the socket class object

    # AF\_INET: we are going to use IPv4 addresses

    # SOCK\_STREAM: we are using TCP packets for communication

    server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    # Creating a try catch block

    try:

        # Provide the server with an address in the form of

        # host IP and port

        server.bind((HOST, PORT))

        print(f"Running the server on {HOST} {PORT}")

    except:

        print(f"Unable to bind to host {HOST} and port {PORT}")

    # Set server limit

    server.listen(LISTENER\_LIMIT)

    # This while loop will keep listening to client connections

    while 1:

        client, address = server.accept()

        print(f"Successfully connected to client {address[0]} {address[1]}")

        threading.Thread(target=client\_handler, args=(client, )).start()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Client

On the client site the client knows the hostname of the machine on which the server is running and the port number on which the server is listening.

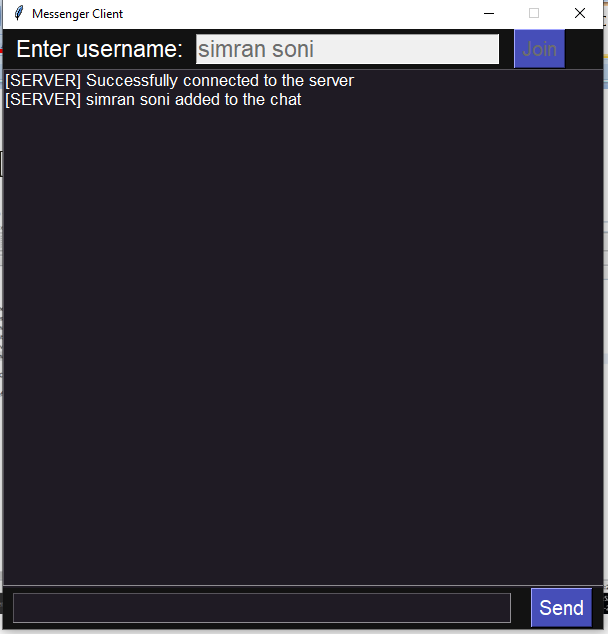
To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

The model used for this project is the single server –

Single client models.

The following specifications must be implemented:

THE CLIENT SCREEN



Client.py:

import socket

import threading

import tkinter as tk

from tkinter import scrolledtext

from tkinter import messagebox

HOST = '127.0.0.1'

PORT = 1234

DARK\_GREY = '#121212'

MEDIUM\_GREY = '#1F1B24'

OCEAN\_BLUE = '#464EB8'

WHITE = "white"

FONT = ("Helvetica", 17)

BUTTON\_FONT = ("Helvetica", 15)

SMALL\_FONT = ("Helvetica", 13)

# Creating a socket object

# AF\_INET: we are going to use IPv4 addresses

# SOCK\_STREAM: we are using TCP packets for communication

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

def add\_message(message):

    message\_box.config(state=tk.NORMAL)

    message\_box.insert(tk.END, message + '\n')

    message\_box.config(state=tk.DISABLED)

def connect():

    # try except block

    try:

        # Connect to the server

        client.connect((HOST, PORT))

        print("Successfully connected to server")

        add\_message("[SERVER] Successfully connected to the server")

    except:

        messagebox.showerror("Unable to connect to server", f"Unable to connect to server {HOST} {PORT}")

    username = username\_textbox.get()

    if username != '':

        client.sendall(username.encode())

    else:

        messagebox.showerror("Invalid username", "Username cannot be empty")

    threading.Thread(target=listen\_for\_messages\_from\_server, args=(client, )).start()

    username\_textbox.config(state=tk.DISABLED)

    username\_button.config(state=tk.DISABLED)

def send\_message():

    message = message\_textbox.get()

    if message != '':

        client.sendall(message.encode())

        message\_textbox.delete(0, len(message))

    else:

        messagebox.showerror("Empty message", "Message cannot be empty")

root = tk.Tk()

root.geometry("600x600")

root.title("Messenger Client")

root.resizable(False, False)

root.grid\_rowconfigure(0, weight=1)

root.grid\_rowconfigure(1, weight=4)

root.grid\_rowconfigure(2, weight=1)

top\_frame = tk.Frame(root, width=600, height=100, bg=DARK\_GREY)

top\_frame.grid(row=0, column=0, sticky=tk.NSEW)

middle\_frame = tk.Frame(root, width=600, height=400, bg=MEDIUM\_GREY)

middle\_frame.grid(row=1, column=0, sticky=tk.NSEW)

bottom\_frame = tk.Frame(root, width=600, height=100, bg=DARK\_GREY)

bottom\_frame.grid(row=2, column=0, sticky=tk.NSEW)

username\_label = tk.Label(top\_frame, text="Enter username:", font=FONT, bg=DARK\_GREY, fg=WHITE)

username\_label.pack(side=tk.LEFT, padx=10)

username\_textbox = tk.Entry(top\_frame, font=FONT, bg=MEDIUM\_GREY, fg=WHITE, width=23)

username\_textbox.pack(side=tk.LEFT)

username\_button = tk.Button(top\_frame, text="Join", font=BUTTON\_FONT, bg=OCEAN\_BLUE, fg=WHITE, command=connect)

username\_button.pack(side=tk.LEFT, padx=15)

message\_textbox = tk.Entry(bottom\_frame, font=FONT, bg=MEDIUM\_GREY, fg=WHITE, width=38)

message\_textbox.pack(side=tk.LEFT, padx=10)

message\_button = tk.Button(bottom\_frame, text="Send", font=BUTTON\_FONT, bg=OCEAN\_BLUE, fg=WHITE, command=send\_message)

message\_button.pack(side=tk.LEFT, padx=10)

message\_box = scrolledtext.ScrolledText(middle\_frame, font=SMALL\_FONT, bg=MEDIUM\_GREY, fg=WHITE, width=67, height=26.5)

message\_box.config(state=tk.DISABLED)

message\_box.pack(side=tk.TOP)

def listen\_for\_messages\_from\_server(client):

    while 1:

        message = client.recv(2048).decode('utf-8')

        if message != '':

            username = message.split("~")[0]

            content = message.split('~')[1]

            add\_message(f"[{username}] {content}")

        else:

            messagebox.showerror("Error", "Message recevied from client is empty")

# main function

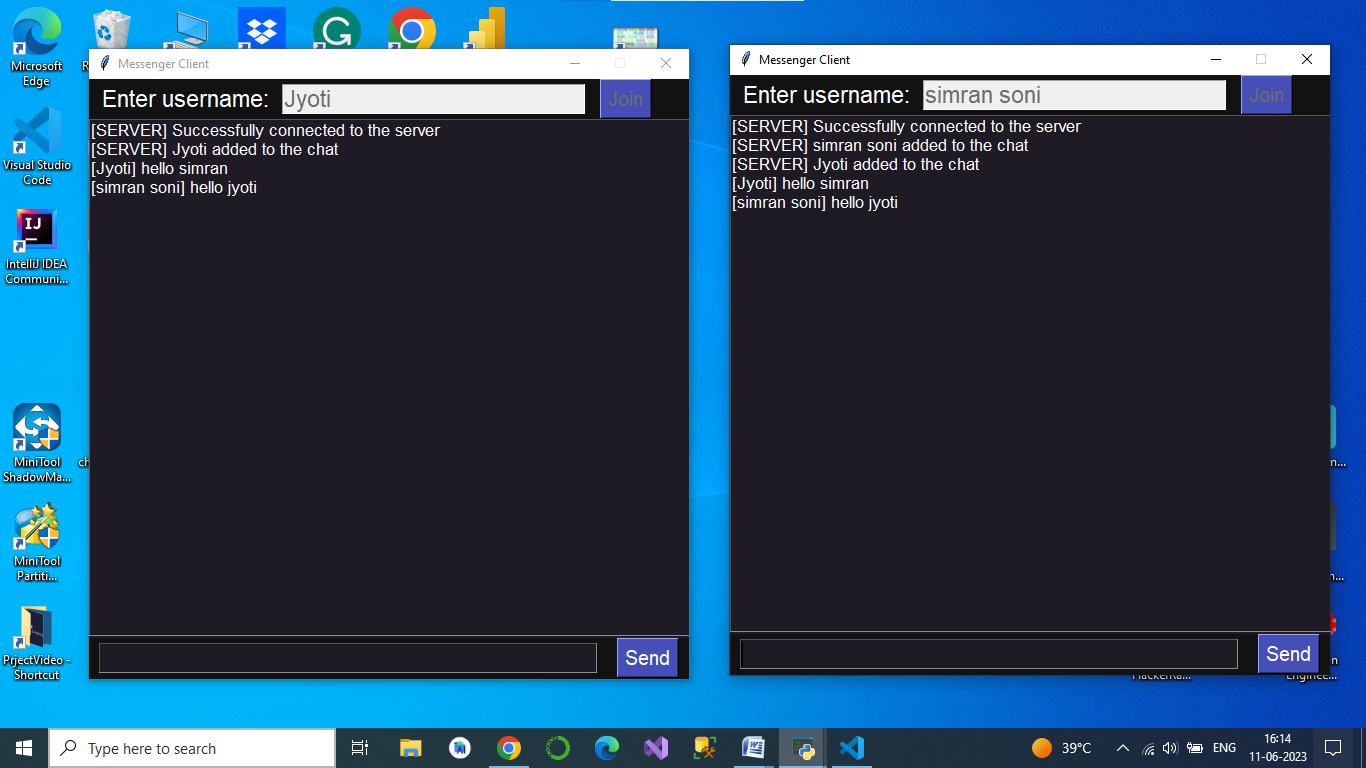
def main():

    root.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**CHAPTER-9: Result/ Output**



# CHAPTER-10: Operational Concepts and Scenarios

Operation of the application based on the inputs given by the user:

When the run button is clicked then the chat form is initialized with a connection

∙between the host and the client machine.

Note: server must be started at first before a client start.

∙ Contains a rich textbox which send messages from one user to another

∙ Contains a textbox for messages to be written that is sent across the network.

∙ Contains a Send button

.∙ When the sent button is clicked, in the background, the text in the textbox is encoded

∙and sent as a packet over the network to the client machine. Here this message is decoded and is shown in the rich textbox.

There is always a room for improvements in any software package, however good and efficient it may be done. But the most important thing should be flexible to accept further modification. Right now we are just dealing with text communication. In future this software may be extended to include features such as:

Files transfer: this will enable the user to send files of different formats to others via the chat∙ application.

Voice chat: this will enhance the application to a higher level where communication will be∙ possible via voice calling as in telephone.

Video chat: this will further enhance the feature of calling into video communication.∙

This server can be set up on a local area network by choosing any on the computer to be a server node, and using that computer’s private IP address as the server IP address.

For example, if a local area network has a set of private IP addresses assigned ranging from 192.168.1.2 to 192.168.1.100, then any computer from these 99 nodes can act as a server, and the remaining nodes may connect to the server node by using the server’s private IP address. Care must be taken to choose a port that is currently not in usage. For example, port 22 is the default for ssh, and port 80 is the default for HTTP protocols. So these two ports preferably, shouldn’t be used or reconfigured to make them free for usage.

However, if the server is meant to be accessible beyond a local network, the public IP address would be required for usage. This would require port forwarding in cases where a node from a local network (node that isn’t the router) wishes to host the server. In this case, we would require any requests that come to the public IP addresses to be re-routed towards our private IP address in our local network, and would hence require port forwarding.

# CHAPTER-11: LIMITATION AND PROJECT FUTUTE SCOPE

Moving forward, more brands will look to capitalize on messaging and establish a clear and concise phone-based relationship with consumers. Similarly, consumers are not going to want to give up the convenience of messaging with brands, which has become commonplace during the pandemic.

On the brand side, messaging cannot simply be mass information that is released on all platforms. Even though a consumer has opted in to text messaging, that does not mean they will stay opted in. Brands cannot just simply add messaging to their list of channels but need to keep consumers wanting to stay engaged and hear more**.**Consumers are smart and know when information is not authentic. Personalization should be at the forefront. Consumers will expect highly curated and specific-to-them information to be shared, such as a new product that coincides with their recent purchase.

Brand messaging will also develop into a more conversational form. Two-way conversations will become the norm, especially in regard to customer support to help consumers feel as though they have an engaging relationship with the brand and the sender.

Additionally, it is not enough for brands to simply rely on SMS, but they will also need to move into chat apps and similar channels as they work to create these more intimate relationships with consumers. Chat apps offer a unique opportunity, and brands will need to establish this increased level of omnichannel messaging in all aspects of their business.

Given the well-rounded experience of email, consumers will also be looking for the same look and feel of content within text messages, creating a more unified customer experience. Email can offer news, pictures, videos, and links, and consumers are expecting this level of information and engagement to transfer to messages. This sentiment is also similar to other technologies such as chat apps and Google Verified SMS as more integrated and advanced features are being added and brand adoption increases.

In the future, consumers can expect messaging to be even more dynamic as video, voice, and high-quality images increasingly become a part of the rich and standard messaging experience. Imagine a world where consumers’ shopping orders are updated in real time via messaging and they are also able to check on different orders or modify the order from the same aesthetically pleasing screen rather than having to follow a link to a separate website. Compared to a standard text message, this integrated and personalized content will create an increased level of engagement and ROI.

While brands continue to innovate with how they use messaging, however, technology teams shouldn’t overlook ensuring the foundational elements of messaging system delivery are also in place. Consumers will expect safe and reliable messaging delivery from all businesses, possibly turning away from those that don’t meet their expectations, so traffic stability and delivery speed are all factors for consideration.

Messaging offers too lucrative of a prospect for brands to ignore. Through the evolution of messaging from simply personal use to now being a regular occurrence in the brand-consumer relationship, the future opportunity is unmatched. It is vital that brands implement a rich, reliable, and innovative messaging strategy so they continue to build and grow their relationships with both their current and future customers.

**CHAPTER-12: CONCLUSION**

I Developed network applications in Python by using sockets, threads, and Web services.

These software is portable, efficient, and easily maintainable for large number of clients. Our developed web-based chatting software is unique in its features and more importantly easily customizable. The Python.net package provides a powerful and flexible set of classes for implementing network applications. Typically, programs running on client machines make requests to programs on a server Machine. These involve networking services provided by the transport layer. The most widely used transport protocols on the Internet are TCP (Transmission control Protocol) and UDP (User Datagram Protocol).

TCP is a connection-oriented protocol providing a reliable flow of data between two computers. On the other hand, UDP is a simpler message-based connectionless protocol which sends packets of data known as datagrams from one computer to another with no guarantees of arrival.

**REFERENCES**

* <https://codinginfinite.com/python-chat-application-tutorial-source-code/>
* <https://www.thepythoncode.com/article/make-a-chat-room-application-in-python>
* <https://www.youtube.com/watch?v=pGolbRsvgnA>
* <https://www.geeksforgeeks.org/simple-chat-room-using-python/>
* <https://github.com/topics/python-chat-application>
* <https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwjqiNjf-7r_AhUVmWYCHVtpDQoYABAAGgJzbQ&ohost=www.google.com&cid=CAESbOD2zFVnfv08fxb6HCauk1rFysx2ypc6Z-_1NxMYqUEN3p7oZ2sE68-RcOFKS3Ur-GmrSVdfERU7o5z6tfuBGtKprS-K5vJPshXFN6Ej2VeTTOD8HcZS1i2C7zf0WnUym3kM3_A28ji9BuE2Zw&sig=AOD64_2j17aLxonI3u3cwz0scOpzyd2Fgg&q&adurl&ved=2ahUKEwjlv9Lf-7r_AhUsR2wGHSSgDKMQ0Qx6BAgIEAE>