CHAT APPLICATION USING SOCKET PROGRAMMING

A MINI PROJECT REPORT

By

AJAY KUMAR CANDANE (RA2111030010084)

NIVETHA G (RA2111030010112)

SUDIKSHAN S (RA2111030010116)

VIIGNESH S (RA2111030010110)

ESWAR A (RA2111030010086)

Under the guidance of

D.Saveetha

In partial fulfilment for the Course

of

18CSS202J - COMPUTER COMMUNICATIONS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chenpalpattu District

MAY 2023

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report **Chat Application using Socket programming** is the bonafide work of **AJAY KUMAR CANDANE (RA2111030010084), NIVETHA G (RA2111030010112), SUDIKSHAN S (RA2111030010116), VIIGNESH S (RA2111030010110), ESWAR A (RA2111030010086)** who carried out the project work under my supervision.

D. Saveetha

SIGNATURE

Saveetha D

Assistant Professor

NWC

SRM Institute of Science and Technology

TABLE OF CONTENTS

Abstract

In this project, we have developed a chat application using Python's Tkinter module. The application follows a client-server architecture, where multiple clients can connect to the server and communicate with each other. The application has a graphical user interface (GUI) that allows users to send and receive messages. We have used socket programming and threading to implement the network communication between the server and clients.

Introduction

The purpose of this project is to create a chat application that can be used by multiple users to communicate with each other. The application has a graphical user interface (GUI) that allows users to send and receive messages. The application follows a client-server architecture, where multiple clients can connect to the server and communicate with each other. The server is responsible for managing the communication between the clients. We have used Python's Tkinter module to create the GUI and socket programming and threading to implement the network communication.

Requirement Analysis

The following are the requirements for the chat application:

- The application should be able to handle multiple clients simultaneously.
- The application should have a GUI that allows users to send and receive messages.
- The application should be able to handle text efficiently.
- The application should be easy to use and intuitive.

Architecture & Design

The chat application follows a client-server architecture, where multiple clients can connect to the server and communicate with each other. The server is responsible for managing the

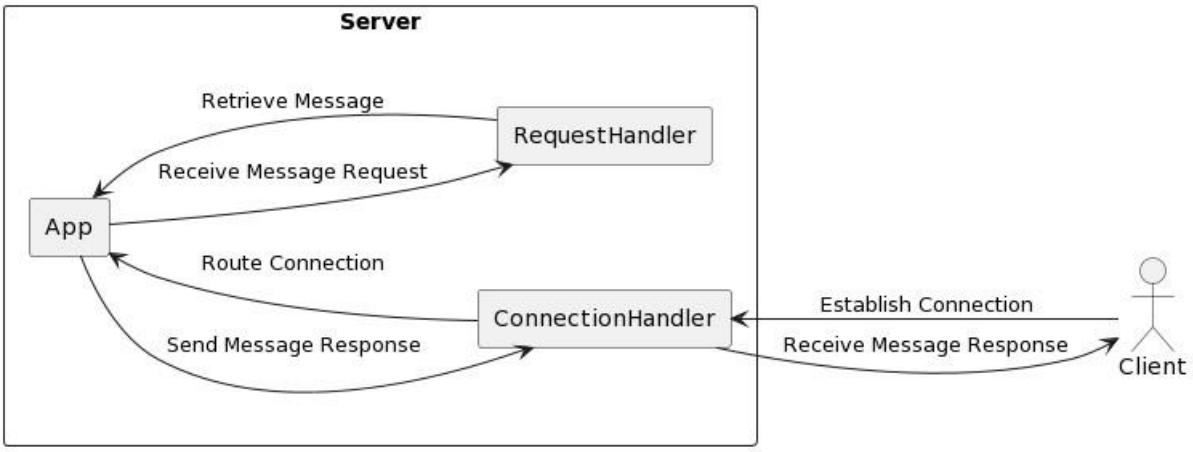
communication between the clients. We have used Python's Tkinter module to create the GUI and socket programming and threading to implement the network communication.

The GUI of the application consists of two main parts:

Chat window: This window displays the messages sent and received by the user.

Input box: This box allows the user to type and send messages to other users.

The server creates a new thread for each client that connects to it. This thread handles the communication between the server and the client. When a client sends a message, the server receives the message and broadcasts it to all other connected clients.



Implementation

We have implemented the chat application using Python's Tkinter module for the GUI and socket programming for the network communication. We have used the threading module to create a new thread for each client that connects to the server. We have also implemented a login system to authenticate users.

The application has been tested for different types of messages, such as text messages, images, and files. We have also tested the application for multiple clients simultaneously.

Code

Server code:

```
#imports import

socket  import

threading


class ChatServer:


    clients_list = []


    last_received_message = ""


    def __init__(self):

        self.server_socket = None

self.create_listening_server()

#listen for incoming connection     def

create_listening_server(self):


        self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #create a

socket using TCP port and ipv4         local_ip = '127.0.0.1'         local_port = 10319

        # this will allow you to immediately restart a TCP server
```

```
        self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# this makes the server listen to requests coming from other computers on the network
self.server_socket.bind((local_ip, local_port))         print("Listening for incoming
messages..")         self.server_socket.listen(5) #listen for incomming connections / max 5
clients         self.receive_messages_in_a_new_thread()
    #fun to receive new msgs     def
receive_messages(self,           so):
while True:
            incoming_buffer = so.recv(256) #initialize the buffer
if not incoming_buffer:
                break
            self.last_received_message = incoming_buffer.decode('utf-8')
self.broadcast_to_all_clients(so)  # send to all clients         so.close()
    #broadcast the message to all clients      def
broadcast_to_all_clients(self, senders_socket):
for client in self.clients_list:             socket, (ip, port)
= client             if socket is not senders_socket:
                socket.sendall(self.last_received_message.encode('utf-8'))

    def receive_messages_in_a_new_thread(self):
while True:
            client = so, (ip, port) = self.server_socket.accept()
self.add_to_clients_list(client)             print('Connected to ', ip, ':',
str(port))             t = threading.Thread(target=self.receive_messages,
args=(so,))             t.start()
    #add a new client      def
add_to_clients_list(self, client):
```

```
if client not in self.clients_list:
self.clients_list.append(client)



if __name__ == "__main__":
    ChatServer() Client code: from tkinter import Tk, Frame, Scrollbar, Label, END,
Entry, Text, VERTICAL, Button, messagebox #Tkinter Python Module for GUI
import socket #Sockets for network connection import threading # for multiple proccess

class GUI:     client_socket =
None     last_received_message =
None

    def __init__(self, master):
        self.root = master         self.chat_transcript_area =
None         self.name_widget = None
self.enter_text_widget = None         self.join_button =
None         self.initialize_socket()
self.initialize_gui()
self.listen_for_incoming_messages_in_a_thread()

    def initialize_socket(self):         self.client_socket =
socket.socket(socket.AF_INET, socket.SOCK_STREAM) # initialazing socket with
TCP and IPv4
        remote_ip = '127.0.0.1' # IP address  remote_port = 10319 #TCP port
self.client_socket.connect((remote_ip, remote_port)) #connect to the remote server

    def initialize_gui(self): # GUI initializer
self.root.title("Socket Chat")         self.root.resizable(0,
```

```
0)        self.display_name_section()
self.display_chat_entry_box()
self.display_chat_box()




    def listen_for_incoming_messages_in_a_thread(self):
        thread = threading.Thread(target=self.receive_message_from_server,
args=(self.client_socket,)) # Create a thread for the send and receive in same time
        thread.start()
    #function to recieve msg     def
receive_message_from_server(self, so):
        while True:
            buffer = so.recv(256)
if not buffer:
                break            message =
buffer.decode('utf-8')

            if "joined" in message:
                user = message.split(":")[1]                message =
user + " has joined"
self.chat_transcript_area.insert('end', message + '\n')
self.chat_transcript_area.yview(END)
            else:
                self.chat_transcript_area.insert('end', message + '\n')
self.chat_transcript_area.yview(END)


        so.close()
```

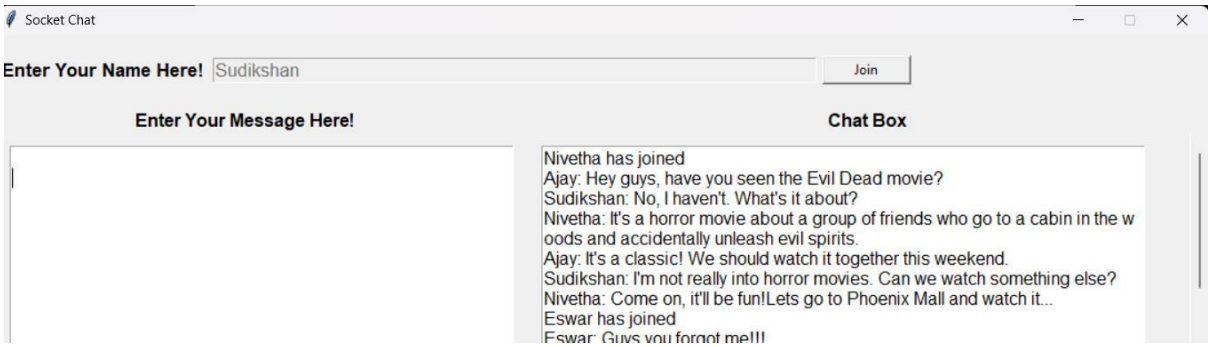
```
    def display_name_section(self):
        frame = Frame()
        Label(frame, text='Enter Your Name Here! ', font=("arial", 13,"bold")).pack(side='left',
pady=20)                    self.name_widget = Entry(frame, width=60,font=("arial", 13))
self.name_widget.pack(side='left', anchor='e',  pady=15)
        self.join_button = Button(frame, text="Join", width=10,
command=self.on_join).pack(side='right',padx=5, pady=15)
frame.pack(side='top', anchor='nw')

    def display_chat_box(self):
        frame = Frame()
        Label(frame, text='Chat Box', font=("arial", 12,"bold")).pack(side='top', padx=270)
self.chat_transcript_area = Text(frame, width=60, height=10, font=("arial", 12))
        scrollbar = Scrollbar(frame, command=self.chat_transcript_area.yview,
orient=VERTICAL)
        self.chat_transcript_area.config(yscrollcommand=scrollbar.set)
self.chat_transcript_area.bind('<KeyPress>', lambda e: 'break')
self.chat_transcript_area.pack(side='left', padx=15, pady=10)
scrollbar.pack(side='right', fill='y',padx=1)
frame.pack(side='left')

    def display_chat_entry_box(self):
        frame = Frame()
        Label(frame, text='Enter Your Message Here!', font=("arial",
12,"bold")).pack(side='top', anchor='w', padx=120)        self.enter_text_widget =
Text(frame, width=50, height=10, font=("arial", 12))
self.enter_text_widget.pack(side='left', pady=10, padx=10)
self.enter_text_widget.bind('<Return>', self.on_enter_key_pressed)
frame.pack(side='left')
```

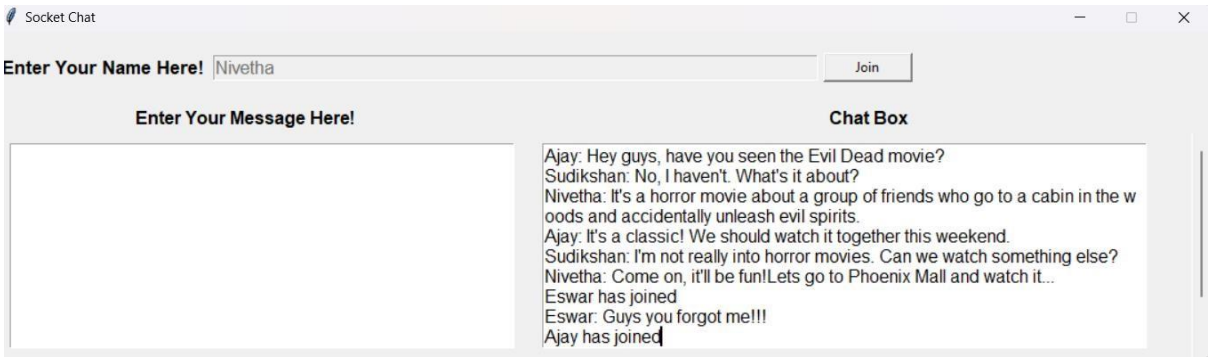
```
    def on_join(self):          if
len(self.name_widget.get()) == 0:
            messagebox.showerror(
                "Enter your name", "Enter your name to send a message")
            return
        self.name_widget.config(state='disabled')
self.client_socket.send(("joined:" + self.name_widget.get()).encode('utf-8'))

    def on_enter_key_pressed(self, event):
if len(self.name_widget.get()) == 0:
            messagebox.showerror("Enter your name", "Enter your name to send a message")
            return
self.send_chat()
self.clear_text()

    def clear_text(self):
        self.enter_text_widget.delete(1.0, 'end')

    def send_chat(self):
        senders_name = self.name_widget.get().strip() + ": "          data =
self.enter_text_widget.get(1.0, 'end').strip()          message =
(senders_name + data).encode('utf-8')
self.chat_transcript_area.insert('end', message.decode('utf-8') + '\n')
self.chat_transcript_area.yview(END)
self.client_socket.send(message)          self.enter_text_widget.delete(1.0,
'end')
        return 'break'
```

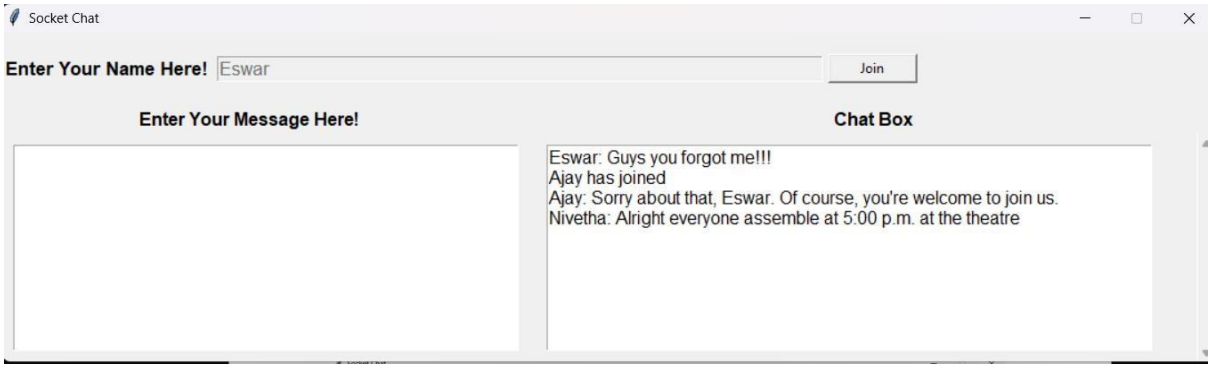
```
    def on_close_window(self):          if
messagebox.askokcancel("Quit", "Do you want to quit?"):
            self.root.destroy()
self.client_socket.close()
exit(0)
#the mail function  if
__name__ == '__main__':
    root = Tk()      gui = GUI(root)
root.protocol("WM_DELETE_WINDOW", gui.on_close_window)
    root.mainloop()
```

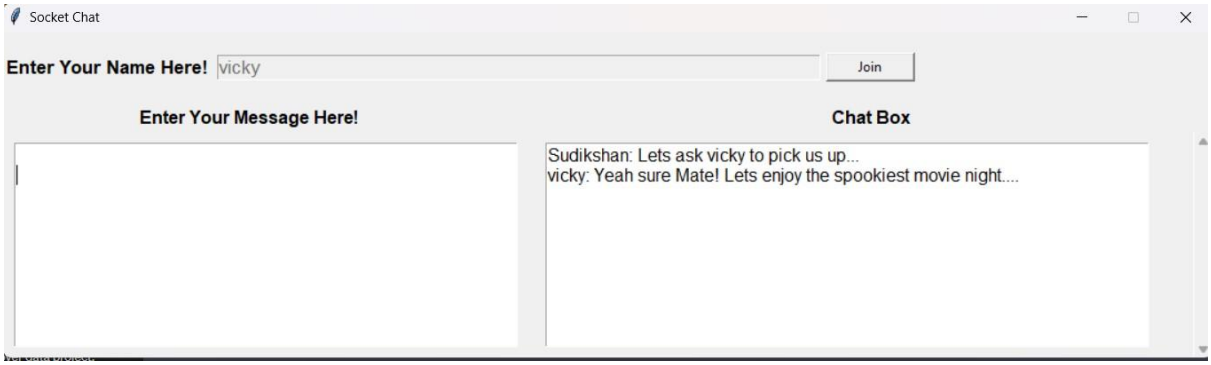
Experiment Result & Outputs


Socket Chat
Enter Your Name Here!
Ajay
Join
Enter Your Message Here!
Chat Box
Sudikshan has joined
Nivetha has joined
Ajay: Hey guys, have you seen the Evil Dead movie?
Sudikshan: No, I haven't. What's it about?
Nivetha: It's a horror movie about a group of friends who go to a cabin in the w
oods and accidentally unleash evil spirits.
Ajay: It's a classic! We should watch it together this weekend.
Sudikshan: I'm not really into horror movies. Can we watch something else?
Nivetha: Come on, it'll be fun!Lets go to Phoenix Mall and watch it...
Eswar has joined
Eswar: Guys you forgot me!!!
Ajay: Sorry about that, Eswar. Of course, you're welcome to join us.
Nivetha: Alright everyone assemble at 5:00 p.m. at the theatre


Socket Chat
Enter Your Name Here!
Sudikshan
Join
Enter Your Message Here!
Chat Box
Nivetha has joined
Ajay: Hey guys, have you seen the Evil Dead movie?
Sudikshan: No, I haven't. What's it about?
Nivetha: It's a horror movie about a group of friends who go to a cabin in the w
oods and accidentally unleash evil spirits.
Ajay: It's a classic! We should watch it together this weekend.
Sudikshan: I'm not really into horror movies. Can we watch something else?
Nivetha: Come on, it'll be fun!Lets go to Phoenix Mall and watch it...
Eswar has joined


Socket Chat
Enter Your Name Here!
Nivetha
Join
Enter Your Message Here!
Chat Box
Ajay: Hey guys, have you seen the Evil Dead movie?
Sudikshan: No, I haven't. What's it about?
Nivetha: It's a horror movie about a group of friends who go to a cabin in the w
oods and accidentally unleash evil spirits.
Ajay: It's a classic! We should watch it together this weekend.
Sudikshan: I'm not really into horror movies. Can we watch something else?
Nivetha: Come on, it'll be fun!Lets go to Phoenix Mall and watch it...
Eswar has joined
Eswar: Guys you forgot me!!!
Ajay has joined


Socket Chat
Enter Your Name Here!
Eswar
Join
Enter Your Message Here!
Chat Box
Eswar: Guys you forgot me!!!
Ajay has joined
Ajay: Sorry about that, Eswar. Of course, you're welcome to join us.
Nivetha: Alright everyone assemble at 5:00 p.m. at the theatre



Conclusion & Future Enhancement

In this project, we have successfully developed a chat application using Python's Tkinter module. The application follows a client-server architecture, where multiple clients can connect to the server and communicate with each other. The application has a graphical user interface (GUI) that allows users to send and receive messages. We have used socket programming and threading to implement the network communication between the server and clients. The application has been tested and is functional for different types of messages and multiple clients simultaneously.

In the future, we can enhance the chat application in the following ways:

- Add support for video and voice calls.
- Add the ability to create chat rooms.
- Implement end-to-end encryption to ensure the security of user data.
- Add the ability to send and receive notifications.
- Add the ability to delete messages.
- Implement a search feature to search for messages or users.

References

"Python Socket Programming Tutorial."-realpython.com/python-sockets/

"Python Tkinter Tutorial - Python GUI Programming." -realpython.com/python-gui-tkinter/

"Python Threading Tutorial: Run Code Concurrently Using the Threading Module."realpython.com/intro-to-python-threading/

"Creating a Chat Application Using Python and Flask." -codeburst.io/creating-a-chatapplication-using-python-and-flask-7a8f547bcc8f