Project Report

Name	Awais Ali (2020-EE-376) Hasnain Malik (2020-EE-419)
Subject	Computer Networks
Supervisor	Sir Azeem Iqbal

University Of Engineering and Technology, Lahore, Faisalabad Campus

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

Project Report: Chat Application

Abstract:

The Real-Time Chat Application is a client-server system designed for seamless communication between users over a network. Utilizing socket programming and threading, the application enables concurrent connections, allowing users to engage in real-time messaging, private conversations, file sharing etc.

Introduction:

1. **Background:**
   In the era of digital communication, the need for real-time messaging applications is evident. This project aims to address this need by developing a chat application using Python.

2. **Objectives:**
   The primary objectives of this project include creating a user-friendly chat application with features such as real-time messaging, private conversations, file sharing and dynamic user lists.

System Architecture:

1. **Client-Side:**

   The client-side architecture is designed to provide an intuitive and user-friendly interface. It utilizes the Tkinter library for creating graphical elements, including chat boxes, user lists, and input fields. The main components on the client side include:

   Graphical User Interface (GUI): Tkinter is employed to build the GUI, allowing users to interact with the application seamlessly. The GUI provides a platform for users to view and send messages, initiate private conversations, and share files.

   Socket Programming: The client utilizes socket programming to establish and maintain a connection with the server. This connection serves as the communication channel through which messages and other data are exchanged between the client and the server.

   Threading: Threading is implemented to ensure a responsive user experience. By using threads, the client can handle multiple tasks simultaneously, such as sending and receiving messages without freezing the GUI.

2. **Server-Side:**

   The server-side architecture is responsible for managing multiple client connections, authenticating users, and facilitating communication between clients. The main components on the server side include:

Socket Programming: The server employs socket programming to listen for incoming client connections. Each accepted connection spawns a new thread, allowing the server to handle multiple clients concurrently.

Threading: Threading is crucial on the server side to handle multiple client connections simultaneously. Each connected client is assigned a dedicated thread, ensuring that the server can respond to multiple requests concurrently.

User Management: User management is implemented to handle user authentication and keep track of connected users. The server maintains a list of connected usernames and ensures the uniqueness of each username.

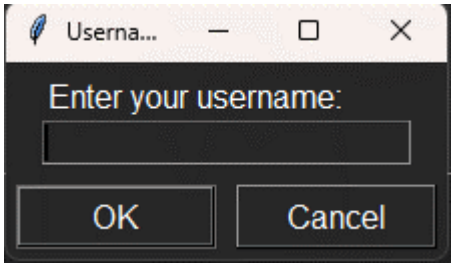
Broadcasting Messages: The server is responsible for broadcasting messages received from one client to all connected clients. This ensures that real-time messaging is achieved, and all users are kept updated with the ongoing conversations.

Implemented Features

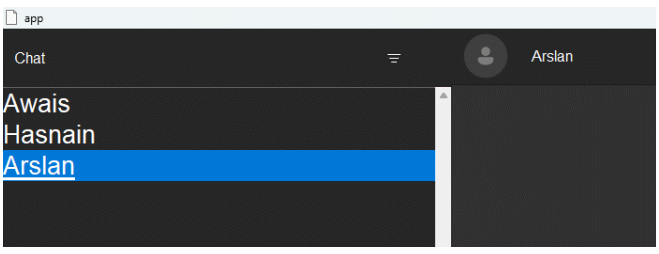
- **Client Connection:** The client initiates a connection to the server using socket programming.

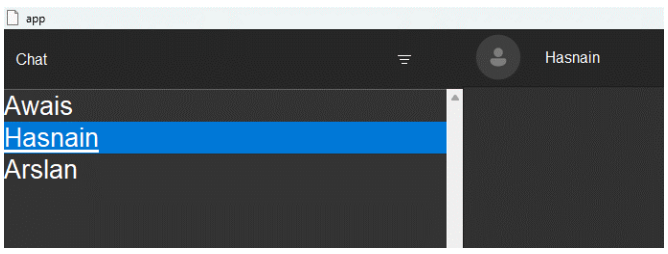
```
Server listening on 192.168.86.191:6500
Connection from: ('192.168.86.191', 55497)
```

- **User Authentication:** The application ensures user authentication by prompting users to enter a unique username. Conflicts are handled gracefully, ensuring a seamless user experience.



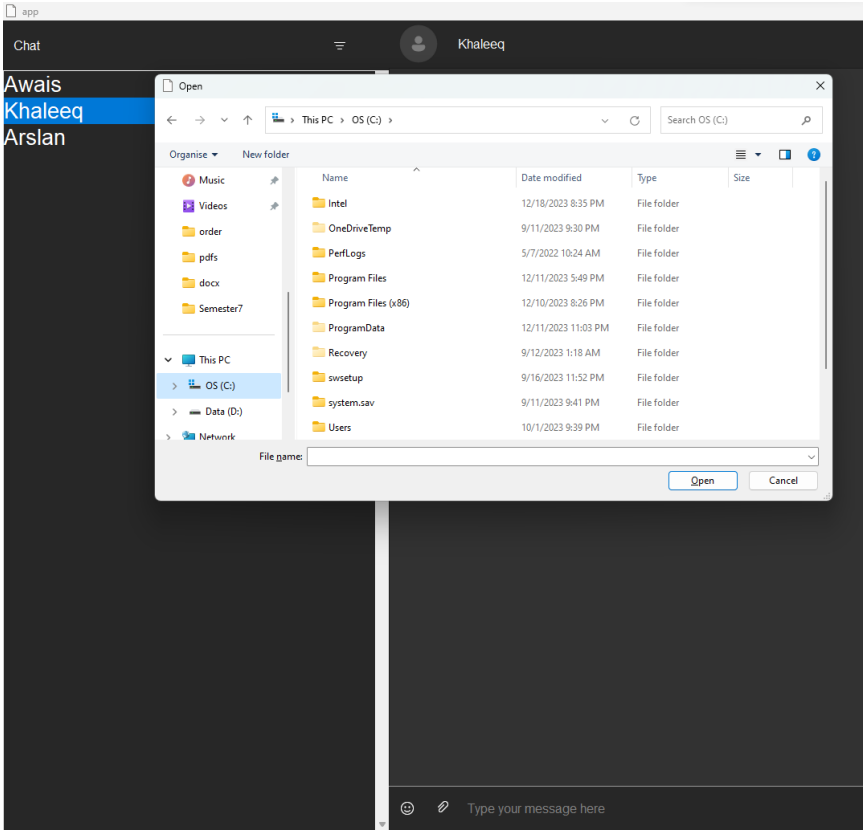
- **Real-time Messaging:** The heart of the application lies in its real-time messaging feature. The `send_message` and `receive_messages` functions facilitate instant communication between connected users.
- **Private Conversations:** Users can initiate private conversations with others, enhancing the application's versatility and facilitating one-on-one interactions.



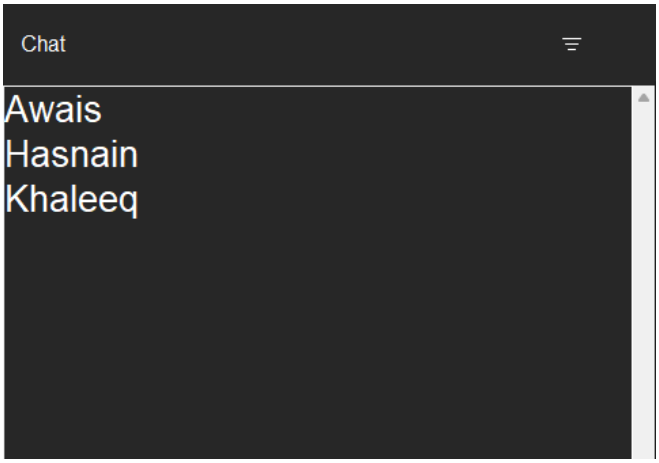


Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

- **File Sharing:** A notable feature is the ability to share files between users. The application leverages the `send_file` and `receive_file` functions, employing pickle for efficient file serialization.



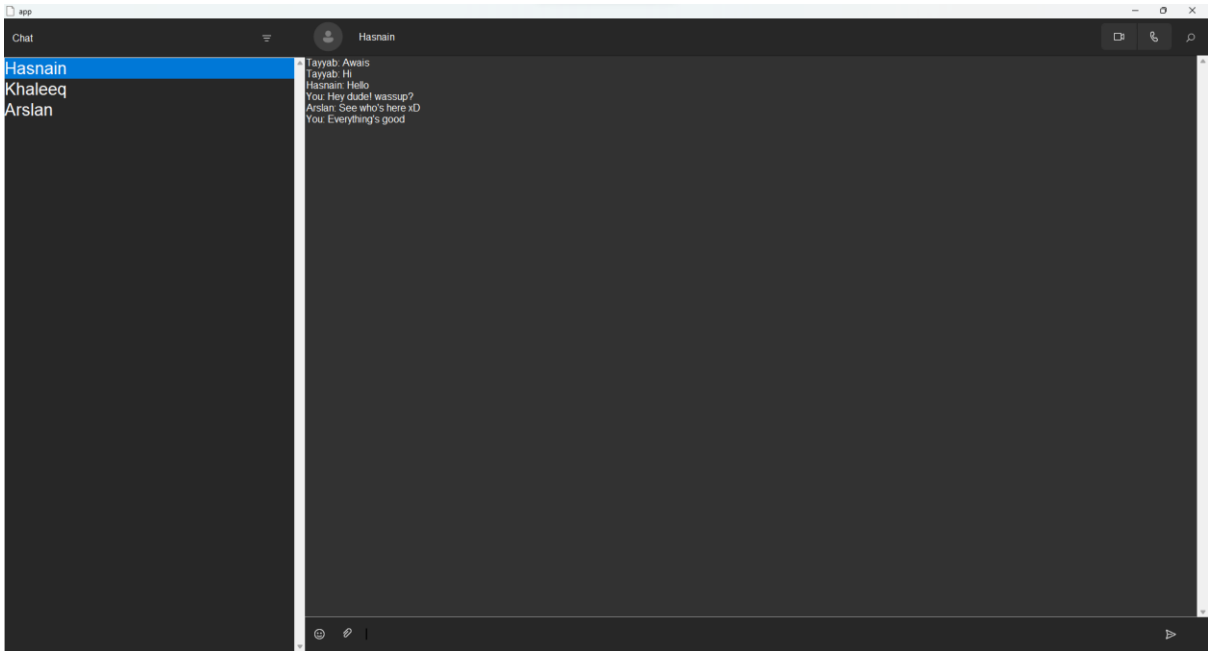
- **Dynamic User List:** The user list dynamically updates to reflect connected users. The `refresh_user_list` function plays a key role in maintaining an accurate and current user list.



- **Voice and Video Call Request** (Note: This feature is not fully implemented yet.) The application lays the groundwork for voice call requests, allowing users to initiate and respond to voice call requests. This feature can be further developed to enhance real-time communication.

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

- **Responsive GUI:** The application have responsive and attractive graphical user interface inspired by the dark mode of WhatsApp.



Code:

Server.py

```
import socket
from socket import error as SocketError
import threading

connected_users = {}
client_sockets = {}
connected_users_lock = threading.Lock()
BUFFER_SIZE = 1024

def handle_client(client_socket):
    try:
        username = client_socket.recv(1024).decode()

        with connected_users_lock:
            if username in connected_users:
                client_socket.send("UsernameNotAccepted".encode())
            else:
                connected_users[username] = client_socket
                client_sockets[username] = client_socket
                client_socket.send("UsernameAccepted".encode())


        while True:
            data = client_socket.recv(1024).decode('utf-8')
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

```
            if not data:
                break

            if data.startswith("StartConversation:"):
                start_conversation(username, data[len("StartConversation:"):])
            elif data == "GetConnectedUsers":
                print(f"{username} is requesting the connected users data")
                connected_users_str = ",".join(connected_users.keys())
                response = f"Connected Users:{connected_users_str}"
                client_socket.send(response.encode())
            elif data.startswith("FileShare:"):
                filename = data.split(":")[1]
                receive_file(client_socket, filename)
                print(f"{filename} received successfully")
            elif data.startswith("VoiceCall:"):
                initiate_voice_call(username, data[len("VoiceCall:"):])
            else:
                broadcast_message(username, data)

    except ConnectionResetError:
        print(f"Connection with {username} reset.")
    except (SocketError, ConnectionResetError) as e:
        print(f"Error in handling client {username}: {e}")
    finally:
        print(f"{username} disconnected.")
        del connected_users[username]
        if username in client_sockets:
            del client_sockets[username]

def initiate_voice_call(sender_username, recipient_username):
    recipient_socket = connected_users.get(recipient_username)

    if recipient_socket:
        recipient_socket.send(f"VoiceCallRequest:{sender_username}".encode())

def receive_file(client_socket, filename):
    try:
        with open(filename, 'wb') as file:
            while True:
                data = client_socket.recv(BUFFER_SIZE)
                if not data:
                    break
                file.write(data)
            print(f"Received file: {filename}")
    except Exception as e:
        print(f"Error receiving file: {e}")

def start_conversation(sender_username, recipient_username):
```

```
    sender_socket = connected_users.get(sender_username)
    recipient_socket = connected_users.get(recipient_username)

    if sender_socket and recipient_socket:
        with connected_users_lock:
            sender_socket.send(f"{recipient_username}".encode())
            recipient_socket.send(f"{sender_username}".encode())

        recipient_socket.send(f"StartedConversation:{sender_username}".encode(
))

    else:
        print(f"Either sender {sender_username} or recipient
{recipient_username} not found.")
        with connected_users_lock:
            del connected_users[recipient_username]
            del connected_users[sender_username]


def broadcast_message(sender_username, message):
    for user, user_socket in connected_users.items():
        if user != sender_username:
            user_socket.send(f"{sender_username}: {message}\n".encode('utf-
8'))

def server_program():
    host = "192.168.86.191"
    port = 6500

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(100)
    print("Server listening on " + host + ":" + str(port))

    while True:
        client_socket, address = server_socket.accept()
        print("Connection from: " + str(address))
        client_handler = threading.Thread(target=handle_client,
args=(client_socket,))
        client_handler.start()

if __name__ == '__main__':
    server_program()
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

Client.py

```
from tkinter import *
from tkinter import scrolledtext, font, LabelFrame
from tkinter.simpledialog import askstring
from tkinter import messagebox
import socket
import threading
import os
import tkinter.filedialog
import pickle

client_socket = None
username = None
connected_users = []
connected_users_lock = threading.Lock()
BUFFER_SIZE = 1024

def get_username():
    global username

    while True:
        username = askstring("Username", "Enter your username:")
        if not username:
            root.quit()
            break

        if username in connected_users:
            messagebox.showerror("Error", "Username already exists. Please try
another username.")
        else:
            client_socket.send(username.encode())

            try:
                response = client_socket.recv(1024).decode()
                if response == "UsernameAccepted":
                    break
            except ConnectionAbortedError:
                messagebox.showerror("Error", "Connection to the server was
unexpectedly closed.")
                root.quit()
                break

def update_chat_box(message):
    chat_box.config(state=NORMAL)
    chat_box.insert(END, message)
    chat_box.config(state=DISABLED)
    chat_box.see(END)
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

```
def receive_messages():
    while True:
        try:
            data = client_socket.recv(1024).decode()
            if not data:
                print("Connection closed by the server.")
                messagebox.showinfo("Info", "Connection closed by the
server.")
                break

            if data.startswith("StartedConversation:"):
                handle_private_conversation(data[len():])
            elif data.startswith("Connected Users:"):
                handle_user_list_response(data[len("Connected
Users:"):].encode())
            elif data.startswith("FileShare:"):
                handle_file_share(data[len("FileShare:"):])
            elif data.startswith("Emoji:"):
                handle_emoji(data[len("Emoji:"):])
            else:
                root.after(0, lambda: update_chat_box(f"{data}"))

        except ConnectionResetError:
            print("Connection with the server reset.")
            break
        except Exception as e:
            print(f"Error in receiving messages: {e}")
            break

def handle_private_conversation(recipient_username):
    print(f"Started a private conversation with {recipient_username}")

def send_message(message):
    if not message.strip():
        return
    global client_socket, username
    parts = message.split(':', 1)
    if len(parts) == 2:
        recipient_username, _ = parts
        if recipient_username in connected_users:
            chat_box.config(state=NORMAL)
            chat_box.insert(END, f"{username} {message}\n")
            chat_box.config(state=DISABLED)
            client_socket.send(f"Message:{message}".encode())
        else:
            print(f"Recipient {recipient_username} not found.")
    else:
        chat_box.config(state=NORMAL)
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

```
        chat_box.insert(END, f"You: {message}\n")
        chat_box.config(state=DISABLED)
        client_socket.send(f"{message}".encode())
    entry.delete(0, END)

def populate_user_list(user_list):
    user_list.delete(0, END)
    for user in connected_users:
        if username == user:
            continue
        user_list.insert(END, user)
    user_list.bind("<Enter>", on_enter)
    user_list.bind("<Leave>", on_leave)


def refresh_user_list():
    print("Sent request for connected users.")
    client_socket.send("GetConnectedUsers".encode())

def handle_user_list_response(response):
    global connected_users
    usernames = response.decode().split(',')
    with connected_users_lock:
        connected_users = usernames.copy()
    print("Updated connected users:", connected_users)
    populate_user_list(user_list)


def send_file(client_socket, file_path):
    BUFFER_SIZE = 1024
    try:
        if os.path.exists(file_path):
            filename = os.path.basename(file_path)
            with open(file_path, 'rb') as file:
                file_data = file.read()
                file_info = {"filename": filename, "data": file_data}
                client_socket.send(f"FileShare:{pickle.dumps(file_info)}".enco
de())
                print(f"{filename} file sent successfully")
        else:
            print(f"Error: File '{file_path}' does not exist.")
    except Exception as e:
        print(f"Error sending file: {e}")


def handle_file_share(data):
    try:
        filename, file_data = data.split(":")[1], data.split(":")[2].encode()
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

```
        file_path = f"received_files/{filename}"
        with open(file_path, "wb") as file:
            file.write(file_data)

        messagebox.showinfo("File Share", f"{current_recipient.get()} sent
file '{filename}' successfully!")
    except Exception as e:
        print(f"Error handling file share: {e}")



def handle_emoji(emoji_code):
    update_chat_box(f"Received Emoji: {emoji_code}\n")
def send_emoji():
    emoji_code = entry.get()
    if not emoji_code:
        return
    try:
        entry.delete(0, END)
        message = f"Emoji:{emoji_code}"
        client_socket.send(message.encode())
    except Exception as e:
        print(f"Error sending emoji: {e}")

def add_placeholder(entry, placeholder_text):
    entry.insert(0, placeholder_text)
    entry.config(fg="grey")
    def on_entry_click(event):
        if entry.get() == placeholder_text:
            entry.delete(0, "end")
            entry.config(fg="white")
    def on_focus_out(event):
        if not entry.get():
            entry.insert(0, placeholder_text)
            entry.config(fg="grey")
    entry.bind("<FocusIn>", on_entry_click)
    entry.bind("<FocusOut>", on_focus_out)

def handle_user_selection(event):
    selected_items = user_list.curselection()
    if selected_items:
        selected_user = user_list.get(selected_items[0])
        if selected_user == "All Users":
            current_recipient.set("All Users")
        else:
            client_socket.send(f"{selected_user}".encode())
            handle_private_conversation(selected_user)
            current_recipient.set(selected_user)
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

```
            selected_user_label = Label(right_header, text=f"{selected_user}",
width=130, bg="#272727", fg="white", anchor="w")
            selected_user_label.grid(row=0, column=1, sticky="w", padx=10)
            chat_box.config(state=NORMAL)
            chat_box.delete(1.0, END)
            chat_box.config(state=DISABLED)

def on_closing():
    global client_socket
    if messagebox.askokcancel("Quit", "Do you want to quit?"):
        root.destroy()
        client_socket.close()

def set_background_image(root, image_path):
    return
def on_enter(event):
    user_list.config(bg="#343434")
def on_leave(event):
    user_list.config(bg="#272727")

def client_program():
    global client_socket, username
    host = "192.168.86.191"
    port = 6500
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        client_socket.connect((host, port))
    except Exception as e:
        print(f"Error connecting to the server: {e}")
        return

    get_username()
    refresh_user_list()

    receive_thread = threading.Thread(target=receive_messages)
    receive_thread.daemon = True
    receive_thread.start()

    root.protocol("WM_DELETE_WINDOW", on_closing)
    root.mainloop()

if __name__ == '__main__':
    root = Tk()
    root.title("app")
    root.iconbitmap("images/icon.png")
    root.option_add('*background', '#272727')
    root.option_add('*foreground', '#ffffff')
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

```
    roboto_font = font.Font(family="Roboto")
    root.option_add("*Font", roboto_font)

    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()
    left_frame_width = screen_width // 4
    right_frame_width = screen_width - left_frame_width

    left_frame = Frame(root)
    left_frame.grid(row=0, column=0, sticky="nsew")
    root.grid_columnconfigure(0, weight=1)
    left_frame.config(bd=0)
    root.grid_columnconfigure(0, minsize=left_frame_width, weight=1)

    left_header = Frame(left_frame, bg="#272727", height=30)
    left_header.grid(row=0, column=0, columnspan=2, sticky="nsew")

    chat_label = Label(left_header, text="Chat", bg="#272727", fg="white",
width=40, anchor="w")
    chat_label.grid(row=0, column=0, sticky="w", padx=10)

    refresh_image = PhotoImage(file="images/refresh.png")
    refresh_button = Button(left_header, image=refresh_image,
command=refresh_user_list, bg="#272727", fg="white", bd=0)
    refresh_button.grid(row=0, column=2, sticky="e")

    user_list = Listbox(left_frame, selectmode=SINGLE, exportselection=0,
font=(roboto_font, 20), bg="#272727", fg="white", bd=0)
    user_list.grid(row=1, column=0, columnspan=2, sticky="nsew")
    user_list_scroll = Scrollbar(left_frame, orient=VERTICAL,
command=user_list.yview)
    user_list_scroll.grid(row=1, column=1, sticky="nes")
    user_list.config(yscrollcommand=user_list_scroll.set)

    right_frame = Frame(root)
    right_frame.grid(row=0, column=1, sticky="nsew")
    root.grid_columnconfigure(1, weight=3)
    root.grid_columnconfigure(1, minsize=right_frame_width, weight=3)
    right_frame.config(bd=0)

    right_header = Frame(right_frame, bg="#272727", height=30)
    right_header.grid(row=0, column=0, columnspan=2, sticky="nsew")

    profile_image = PhotoImage(file="images/profile.png")
    profile_image_label = Label(right_header, image=profile_image, bd=0)
    profile_image_label.grid(row=0, column=0, padx=10)
```

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git

```
    selected_user_label = Label(right_header, text="Start the Conversation",
width=130, bg="#272727", fg="white", anchor="w")
    selected_user_label.grid(row=0, column=1, sticky="w", padx=10)

    video_call_image = PhotoImage(file="images/video.png")
    video_call_button = Button(right_header, image=video_call_image,
bg="#272727", fg="white", bd=0)
    video_call_button.grid(row=0, column=2, sticky="e")

    voice_call_image = PhotoImage(file="images/voice.png")
    voice_call_button = Button(right_header, image=voice_call_image,
bg="#272727", fg="white", bd=0)
    voice_call_button.grid(row=0, column=3, sticky="e")

    search_image = PhotoImage(file="images/search.png")
    search_button = Button(right_header, image=search_image, bg="#272727",
fg="white", bd=0)
    search_button.grid(row=0, column=4, sticky="e")

    chat_box = scrolledtext.ScrolledText(right_frame, state=DISABLED,
bg="#323232", fg="white", wrap="word")
    chat_box.grid(row=1, column=0, columnspan=2, sticky="nsew")
    bg_image_path = "images/bg.png"
    set_background_image(root, bg_image_path)

    footer_frame = Frame(right_frame, bg="#272727", padx=5, pady=5, bd=0,
highlightbackground="#1c1c1c")
    footer_frame.grid(row=2, column=0, columnspan=2, sticky="nsew")

    emoji_share_image = PhotoImage(file="images/emoji.png")
    emoji_share_button = Button(footer_frame, image=emoji_share_image,
command=send_emoji, bg="#272727", fg="white", bd=0)
    emoji_share_button.grid(row=0, column=0, sticky="w")

    file_share_image = PhotoImage(file="images/file.png")
    send_file_btn = Button(footer_frame, image=file_share_image,
command=lambda: send_file(client_socket,
tkinter.filedialog.askopenfilename()), bg="#272727", fg="white", bd=0)
    send_file_btn.grid(row=0, column=1, sticky="w")


    entry = Entry(footer_frame, width=140, font=("Arial", 12), bg="#272727",
fg="white", bd=0)
    entry.grid(row=0, column=2, sticky="ew", padx=5, pady=5,)
    placeholder_text = "Type your message here"
    entry.bind("<Return>", lambda event: send_emoji())
    add_placeholder(entry, placeholder_text)

    send_button_image = PhotoImage(file="images/send.png")
```

```
    send_button_right = Button(footer_frame, image=send_button_image,
command=lambda: send_message(entry.get()), bg="#272727", fg="white", bd=0)
    send_button_right.grid(row=0, column=3, sticky="e")

    current_recipient = StringVar()
    entry.bind("<Return>", lambda event: send_message(entry.get()))

    user_list.bind("<<ListboxSelect>>", handle_user_selection)

    # Configure grid weights for root
    root.grid_rowconfigure(0, weight=1)
    root.grid_columnconfigure(0, weight=1)
    root.grid_columnconfigure(1, weight=3)

    # Configure grid weights for left frame
    left_frame.grid_rowconfigure(1, weight=1)
    left_frame.grid_columnconfigure(0, weight=1)
    left_frame.grid_columnconfigure(1, weight=1)

    # Configure grid weights for right frame
    right_frame.grid_rowconfigure(1, weight=1)
    right_frame.grid_rowconfigure(2, weight=0)
    right_frame.grid_columnconfigure(0, weight=1)
    right_frame.grid_columnconfigure(1, weight=1)

    client_program()
```

Challenges and Solutions

Username Conflicts:

Handling username conflicts is crucial for a smooth user experience. The application addresses this challenge by prompting users to choose a different username in case of a conflict.

File Transmission:

Ensuring reliable file transmission posed challenges, but the implementation of the `send_file` and `receive_file` functions overcame these obstacles, providing a robust file-sharing feature.

Robustness:

The application incorporates robust error handling mechanisms to handle unexpected scenarios, ensuring stability and a graceful termination process.

Future Improvements:

Potential enhancements for the future include implementing voice and video call functionality, introducing additional chat features, and refining the user interface for an even more polished user experience.

Conclusion:

In conclusion, the chat application successfully fulfills its objectives by providing a feature-rich and dynamic platform for real-time communication. The implemented features showcase the application's versatility and set the stage for further improvements and expansions. The Real-Time Chat Application exemplifies core computer network principles:

- TCP Connection: Ensures reliable and secure communication.
- Threading: Enables concurrent handling of client connections for responsiveness.
- User Authentication: Secures unique usernames for authenticated access.
- Real-Time Messaging (TCP): Facilitates prompt and reliable communication.
- Private Conversations: Enhances privacy and personalized communication.
- File Sharing (TCP): Enables efficient and secure file transmission.
- Dynamic User List: Reflects real-time active connections for effective network management.
- Voice Call (Feature in Progress): Lays the groundwork for potential future enhancements.

Get fully functional app by running this command:
git clone https://github.com/Awais15151/Tkinter_Socket_CN_Lab_App.git