**Messaging App / Messenger**

* **Project Overview:** This Messenger utilizes 2 python scripts to function properly. One of the scripts (Server Script) is run on only one system which is kept on during the entirety of the whole conversation and the other script (Client Script) runs only on the system whose user wants to join the conversation and can be turned off anytime that individual wants to leave. Whenever someone wants to join the conversation, he will simply run the Client Script, enter his name and then communicate with those who are already present there. All The Conversations and the name input prompt that a person gets when he runs client script takes place of a GUI that we created with the help of a built-in python library (tkinter) inside the client script.
* **Server Script (code):**

|  |
| --- |
| # Libraries  import socket # Used To Setup A Secure Connection  import threading # Used To Handle Multiple Clients Simultaneously  # Connection Route  ip = "192.168.0.102" # Host Ip  port = 55555 # Connection Port (Host)  server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # Start The Socket  server.bind((ip, port)) # Connect The Socket To Ip And Port Specified Above  server.listen() # Look For Incoming Connection Requests  clients = [] # Socket ID Of People Connected  names = [] # Name Of People Connected  def send\_message(message, client): # Recieves Message From The 'recieve\_message' Function And Sends It To All Connected Clients  for c in clients:  if c != client:  try:  c.send(message)  except:  remove(c)  def remove(client): # Removes A Person Who Closes Their Client File On Their System And Notifies Those Who Are Still Connected  if client in clients:  index = clients.index(client)  clients.remove(client)  client.close()  name = names[index]  send\_message(f"{name} has left the chat.".encode("utf-8"), None)  names.remove(name)  def recieve\_message(client): # Recieve Messages From Senders And Sends Them To 'send\_message' Function  while True:  try:  message = client.recv(1024)  send\_message(message, client)  except:  # Remove the client if unable to receive a message  remove(client)  break  def connect(): #Establishes A Secure Connection When A Client First Enters After Entering His Name, And Also Notifes Other Connnected Clients That Xyz Has Joined.  while True:  client, address = server.accept()  print(f"Connection established with {str(address)}")  name = client.recv(1024).decode("utf-8")  names.append(name)  clients.append(client)  join\_message = f"JOIN:{name} has joined the chat!".encode("utf-8")  send\_message(join\_message, None)  thread = threading.Thread(target=recieve\_message, args=(client,))  thread.start()  # Start the server  print("Server Initiated, Looking For Connections ..........")  connect() |

* **Server Script (overview):** Server Script is the code that helps us initiate the server so that it starts searching for clients that are trying to connect to it. It uses built-in python library (Socket) to establish a secure path where connection can be setup and clients can share their messages, the path that is is used is the IP address of the device where server script is run and any port number ranging from 0 to 65,535, In our case we used port 5555 and the IPV4 address of our host device was 192.168.0.102 (IPV4 address will be different for all devices).Another library (Threading) is used to handle multiple clients on our application at a single time and prevent it from crashing.
* **Server Script Working (Functions):**
* **‘send\_message’ function** receives message from **‘receive\_message’ function** and broadcasts the message to all connected clients.
* **‘remove’ function** removes an individual from the messanger once he closes the client script and it also sends out a message on the messanger that ‘xyz’ disconnected.
* **‘receive\_message’ function** receives messages from senders and sends them to **‘send\_message’** function so that message can be sent to the connected clients.
* **‘connect’ function** establishes a connection between the client who just connected and the server after the client inputs his name, after this the server also sends out a message to the messanger that ‘xyz’ connected.
* **Client Script (code):**

|  |
| --- |
| # Libraries  **import** **socket**  **import** **threading**  **import** **tkinter** **as** **tk** # Used To Create GUI For Messanger  **from** **tkinter** **import** simpledialog  ip = "192.168.0.102" # Host Ip  port = **55555** # Connection Port (Host)  client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # Setup A Socket  client.connect((ip, port))  name = simpledialog.askstring("Input", "Enter your name", parent=tk.Tk()) # Clients Name (Input)  client.send(name.encode("utf-8")) # Send Clients Name To The Server  # Messanger Interface Settings (Graphics)  window = tk.Tk()  window.title("Chat App")  window.configure(bg="#2C2C2C")  name\_label = tk.Label(window, text=f"Logged in as: {name}", bg="#2C2C2C", fg="white")  name\_label.pack()  # Display Area For Messages Configuration  message\_box = tk.Text(window, height=**15**, width=**50**, bg="#2C2C2C", fg="white", insertbackground="white")  message\_box.pack(pady=**10**)  message\_box.config(state=tk.DISABLED)  # Messages Typing Area Configuration  message\_entry = tk.Entry(window, width=**50**, bg="#404040", fg="white", insertbackground="white")  message\_entry.pack(pady=**10**)  **def** **send\_message**(event=**None**): # Send messages to the server  message = message\_entry.get()  **if** message:  formatted\_message = f"{name}: {message}"  client.send(formatted\_message.encode("utf-8"))  message\_entry.delete(**0**, tk.END)  # Display the sent message in the client's chat window  update\_message\_box(formatted\_message)  window.bind("<Return>", send\_message) # Setting Up Enter Key To Send Message  # Making An Extra Send Button To Send Messages  send\_button = tk.Button(window, text="Send", command=send\_message, bg="#404040", fg="white")  send\_button.pack()  **def** **receive**(): # Receive Messages From The Server And Show Them On The Message Display Screen We Setup Earlier  **while** **True**:  **try**:  message = client.recv(**1024**).decode("utf-8")    **if** message.startswith("JOIN:"): # It Check if the message is a join message  update\_message\_box(message[**5**:])  **else**:  update\_message\_box(message)  **except**:  print("An error occurred while receiving messages.")  client.close()  **break**  **def** **update\_message\_box**(message): # Used to update the message box  message\_box.config(state=tk.NORMAL)  message\_box.insert(tk.END, message + "**\n**")  message\_box.config(state=tk.DISABLED)  message\_box.see(tk.END)  receive\_thread = threading.Thread(target=receive) # Initiates A Thread To Receive Messages  receive\_thread.start()  # Run the GUI  window.mainloop() |

* **Client Script (Overview):** Client Script is the piece of code through which we configure the GUI to make it look more appealing and useful for the clients using the tkinter library. To start the connection we put the correct IP address and port number of the host device since we want to connect to it. In our case IPv4 adress was ‘192.168.0.102’ and port number was ‘55555’, it could be different for everyone. After the connection is established we communicate with other clients using the function in this script. Through the use of tkinter we also change the background colour and text color of our application and different sizes e.g the text box area, the chat screen area.
* **Client Script Working (Functions):**
* **send\_message** function is used to send messages to server so that it can broadcast it to other clients.
* **receive\_message** function is used to receive broadcasted messages from server and then send these messages to **update\_message\_box** function.
* **update\_message\_box** function receives messages from the **receive\_message** function and then shows these messages on clients text screen.