Summary Report – Part 2 System overview and code structure

The system has two files: server.py and nicegui_client.py.
The server opens a TCP socket and listens for connections. Each client is handled in its own thread, so multiple clients can connect at the same time. The server keeps a mapping of connected users and also tracks private chat state: who is connected to whom, and who has a pending request.
To start a private chat, a client types a username. The server sends a request to that user, who must accept or reject. During a private chat, any message is automatically forwarded to the partner until one side ends the chat (leave) or disconnects (exit).

Updated run instructions

Open VS code -> Folder part 2

1. Open a terminal in the folder:

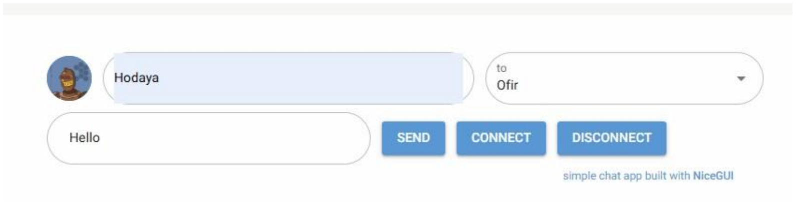
Start the server : python server.py

2.Start a client (in another terminal): python nicegui_client.py

How does a private chat start?

In the NiceGUI page, the user connects, selects recipients in the “to” field, types a message, and clicks Send. The client sends a string in the format : name client to another name client - message to the server.

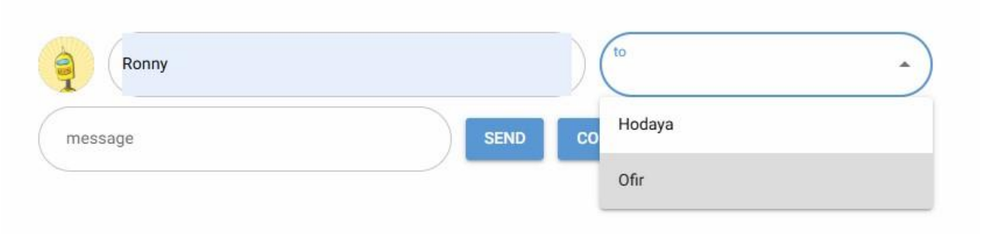
For example:



What happens during a private chat?

The server parses that format and forwards the message only to the chosen recipients.
The sender sees an immediate local echo, and the recipients see the message when it arrives from the server.

For example:



Protocol Analysis of a Multi-User TCP Chat System:

1. Application Layer

The capture reveals a custom **Chat Server** protocol operating on port 10000. The interaction follows a specific command-and-response flow:

Initialization: Upon connection, the server sends a "welcome to the chat" message and prompts the user to "enter name:".

User Identity: Four users are identified through their input: **Ofir** (Port 58197) , **Ronny** (Port 52837) , **Hodaya** (Port 51542) , and **Emily** (Port 51546).

Menu System: allows the user to first connect to the chat.
After connecting, the **"to"** field displays a dropdown menu with the list of users who are currently online. The user can select one or more recipients from this list and send messages directly to them.
In addition, the client can leave the current conversation, similar to the behavior described earlier in the system.

2. Transport Layer (TCP)

The communication relies on the **Transmission Control Protocol (TCP)** to ensure reliable data delivery.

Handshake: Each client performs a standard three-way handshake ([SYN], [SYN, ACK], [ACK]) to establish a connection with the server at 127.0.0.1:10000.

Data Pushing: Most data packets use the **[PSH, ACK]** flags, which instructs the TCP stack to pass the data to the application immediately rather than waiting for the buffer to fill.

Sequencing and Acknowledgement:

*The server tracks data using Sequence (Seq) and Acknowledgment (Ack) numbers.

*For example, after the server sends 20 bytes of "welcome" data , the client acknowledges receipt by setting its Ack number to 21.

Port Mapping:

***Server Port**: 10000.

***Client Ports**: 58197, 52837, 51542, and 51546.

3. Network Layer (IPv4)

The network layer handles the addressing and routing of packets.

***Protocol**: Internet Protocol version 4 (IPv4).

***Addressing**: All communication occurs on the **Loopback address (127.0.0.1)**. This indicates that the client and server programs are running on the same physical machine.
***Traffic Flow**:

Source: 127.0.0.1.

Destination: 127.0.0.1.

Fragmentation: There is no evidence of IP fragmentation, as the data segments (ranging from 4 to 48 bytes) are well below the standard MTU size.

Use of AI Tools in Editing the Assignment

During the preparation of this assignment, AI-based tools were used as part of the editing and processing of the materials. These tools were mainly used to improve wording, organize the structure of the text, and fix language issues, in order to better understand how the code works and how to run it.

For example- Prompts:

1.Create a CSV file Just like this script.

2. How to make our code more readable and organized?

3.Give me a better function names.

Simple Data Structures Behind the Chat App-Dictionary:

In server.py, the main dict is clients. It maps each client's socket to their name, so the server knows who's connected, can broadcast, send private messages by name, and push the user list to everyone.

In nicegui_client.py, there isn't one big dict, but each chat message is stored as a dict in state.messages (fields like text, stamp, sent, avatar, kind). That makes it easy for the UI to render messages the same way and tell system messages apart from normal chat.