CHAT ROOM

Computer Networks

Team Members

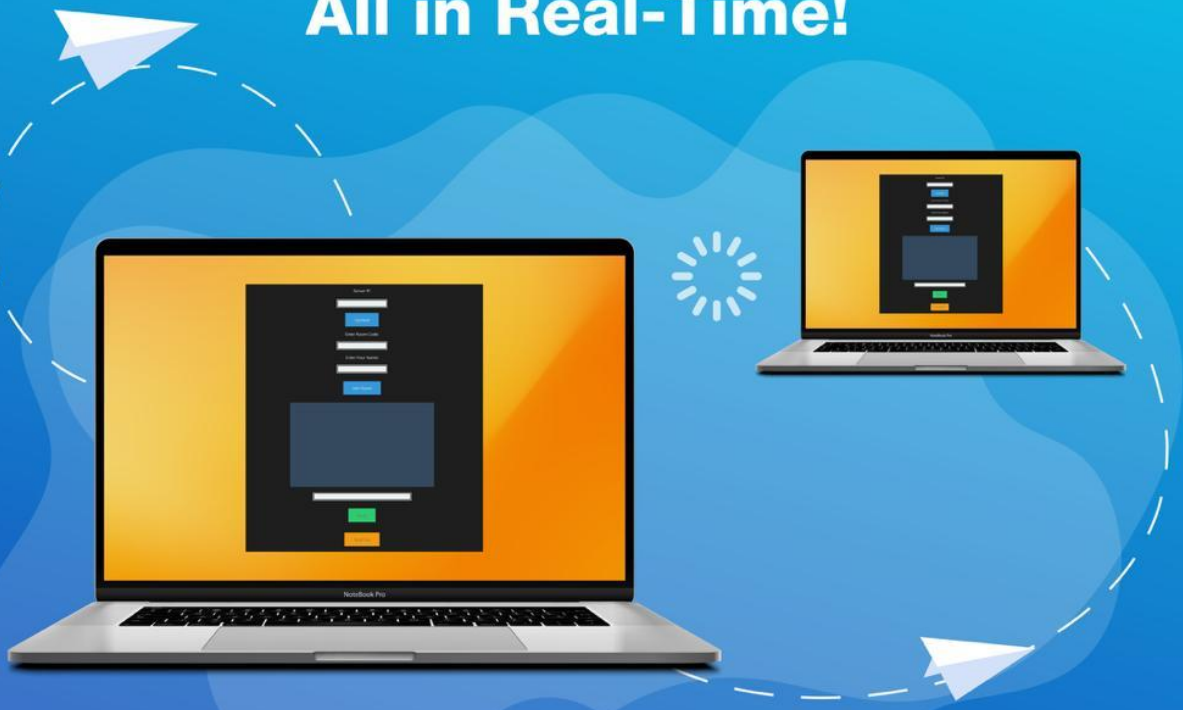
1. Mahmoud Hany (Team Leader)
2. Adham Hesham
3. Shehab Tawfik
4. Adam Mohammed

JANUARY 3, 2025

AAST

CHAT ROOM

Chat..Share..Connect..
All in Real-Time!



- Real-Time Messaging : Instant communication at your fingertips.
- File Sharing Made Easy : Share files, images, and videos seamlessly.
- Private Rooms : Connect with friends in exclusive spaces.
- User-Friendly Interface : Modern design for effortless chatting.
- Cross-Device Compatibility : Works on any computer!

chat.room.com 222 8881188 chat.room

Chat Room

1.Server:

Application Description

This is a basic **chat server** implemented using Python. The server facilitates multiple clients joining "rooms" and communicating with each other within the respective room. Below are the core aspects of the application:

1. **Protocols Used**:
    a. **TCP (Transmission Control Protocol)**: The server uses sockets to create reliable connections between the server and clients for data exchange.
2. **Frameworks and Libraries**:
    a. **`socket`**: A Python library for low-level networking. It provides the foundation for creating the server and client connections.
    b. **`threading`**: Used to manage multiple client connections simultaneously by assigning a separate thread for each client.
3. **Models**:
    a. **Client-Server Model**: The server is the central hub that listens for incoming connections, and clients connect to it to interact with other clients in chat "rooms."
    b. **Room Model**: Clients are grouped into rooms, identified by a unique `room_code`. Messages are broadcast only to the clients in the same room.

Functions and Their Roles

Global Variables

- **clients**:

    - A list intended to keep track of connected clients (though not utilized in the current implementation).
- **rooms**:
    - A dictionary mapping room_code to a list of tuples, where each tuple contains a client's socket and name.
    - This structure organizes clients by their respective rooms.

handle_client(client_socket, addr)

- **Purpose**: Handles all interactions with a single client from connection to disconnection.
- **Steps**:
    - **Client Initialization**:
        - Receives the room code and username from the client.
        - Adds the client to the corresponding room in the rooms dictionary.
    - **Notification**:
        - Notifies all clients in the room about the new user's entry.
    - **Message Handling**:
        - Continuously listens for messages from the client.
        - Broadcasts received messages to all clients in the same room, including the sender.
    - **Disconnection**:
        - If the client disconnects or an error occurs, removes the client from the room and closes their connection.
- **Relevance**:
    - This function is the core of the server's operation, ensuring smooth communication and client management.

start_server()

- **Purpose**: Sets up and starts the chat server.
- **Steps**:
    - Creates a socket object for the server.
    - Binds the server to a specified IP (0.0.0.0, which allows connections from any network interface) and port (5555).
    - Listens for incoming connections.

    - For every new connection, spawns a new thread running handle_client() to manage the client.
- **Relevance**:

    - This function establishes the server and ensures that it can handle multiple clients concurrently

How the Application Works

1. **Server Initialization**:
   The server starts by calling `start_server()`, which begins listening for connections.
2. **Client Connection**:
   A client connects to the server, sending its room code and username.
3. **Room Management**:
   The server assigns the client to the appropriate room, creating the room if it doesn't exist.
4. **Communication**:
   Messages sent by a client are broadcast to all other clients in the same room.
5. **Disconnection**:
   When a client disconnects, it is removed from the room.

Considerations

- **Scalability**:
    - The application handles concurrency using threads, which may become a bottleneck with many clients.
    - Switching to an asynchronous framework like **asyncio** could improve performance.

- **Error Handling**:
    - The code lacks robust error handling for unexpected cases like malformed client messages or server overload.

- **Security**:
    - The application does not use encryption (e.g., SSL/TLS) for data transmission, making it vulnerable to eavesdropping.

2.Client:

Application Overview

This is a **client-side chat application** with a graphical user interface (GUI) built using **Tkinter**. It allows users to connect to a server, join chat rooms, and exchange messages with other participants in real time. The GUI is designed to be modern and user-friendly.

Frameworks and Protocols Used

1. **Frameworks**:

    a. **Tkinter**:
        i. Python's standard GUI toolkit is used for building the application's interface.
        ii. The interface elements (e.g., text boxes, buttons, and labels) are styled to provide a modern appearance.
    b. **Threading**:
        i. Ensures that the GUI remains responsive while the application receives messages from the server in the background.
2. **Protocols**:
    a. **TCP (Transmission Control Protocol)**:
        i. Used for reliable communication between the client and server.
        ii. Ensures that messages are delivered in the correct order without loss.

Functions and Their Roles

receive_messages(client_socket)

- **Purpose**: Continuously listens for incoming messages from the server and displays them in the chat box.
- **Key Operations**:
    - Decodes and displays the received message.
    - Updates the chat box dynamically and ensures it scrolls to show the latest message.
    - Handles errors gracefully if the server disconnects or an issue arises.

send_message(client_socket, message)

- **Purpose**: Sends a user-generated message to the server.
- **Key Operations**:
    - Encodes the message as a UTF-8 string and sends it through the client socket.

connect_to_server(ip, port)

- **Purpose**: Establishes a connection to the chat server.
- **Key Operations**:

    - Creates a socket object.
    - Connects the client to the specified IP address and port (default is 5555).

join_room()

- **Purpose**: Sends the room code and user name to the server to join a chat room.
- **Key Operations**:
    - Sends the room_code and name to the server.
    - Disables the "Join Room" button and name entry to prevent rejoining.
    - Enables the message entry field for chat functionality.

send_chat_message()

- **Purpose**: Captures the user's message from the text entry field and sends it to the server.
- **Key Operations**:
    - Clears the entry field after sending the message.

start_client()

- **Purpose**: Initiates the client connection process and starts a thread for receiving messages.
- **Key Operations**:
    - Calls connect_to_server() to connect to the server.
    - Starts a background thread to execute receive_messages() without blocking the main GUI thread.
    - Displays an error dialog if the connection fails.

GUI Components

1. **Labels**:
    a. Provide instructions and contextual information for fields like "Server IP," "Room Code," and "Your Name."
2. **Entry Fields**:
    a. Allow users to input server IP, room code, and their name.
    b. A dedicated message entry field for chat input.

3. **Buttons**:
    a. **"Connect"**: Establishes the connection with the server.
    b. **"Join Room"**: Sends room code and username to the server and enables chat functionality.
    c. **"Send"**: Sends a message to the chat room.
4. **Chat Box**:
    a. A `Text` widget displays received messages.
    b. Set to read-only mode (`state=tk.DISABLED`) to prevent user modifications.

Styling and Usability Features

1. **Modern Appearance**:
    a. Fonts, colors, and button styles give a sleek, modern look.
    b. Inspired by flat UI design principles.

2. **Dynamic Behavior**:
    a. Chat box auto-scrolls to show the latest messages.
    b. GUI remains responsive even when receiving messages, thanks to threading.

3. **Error Handling**:
    a. Graceful error dialogs are shown if the connection fails.

How the Application Works

1. **Connecting to the Server**:
    a. User enters the server's IP address and clicks "Connect."
    b. The client attempts to establish a TCP connection to the server.

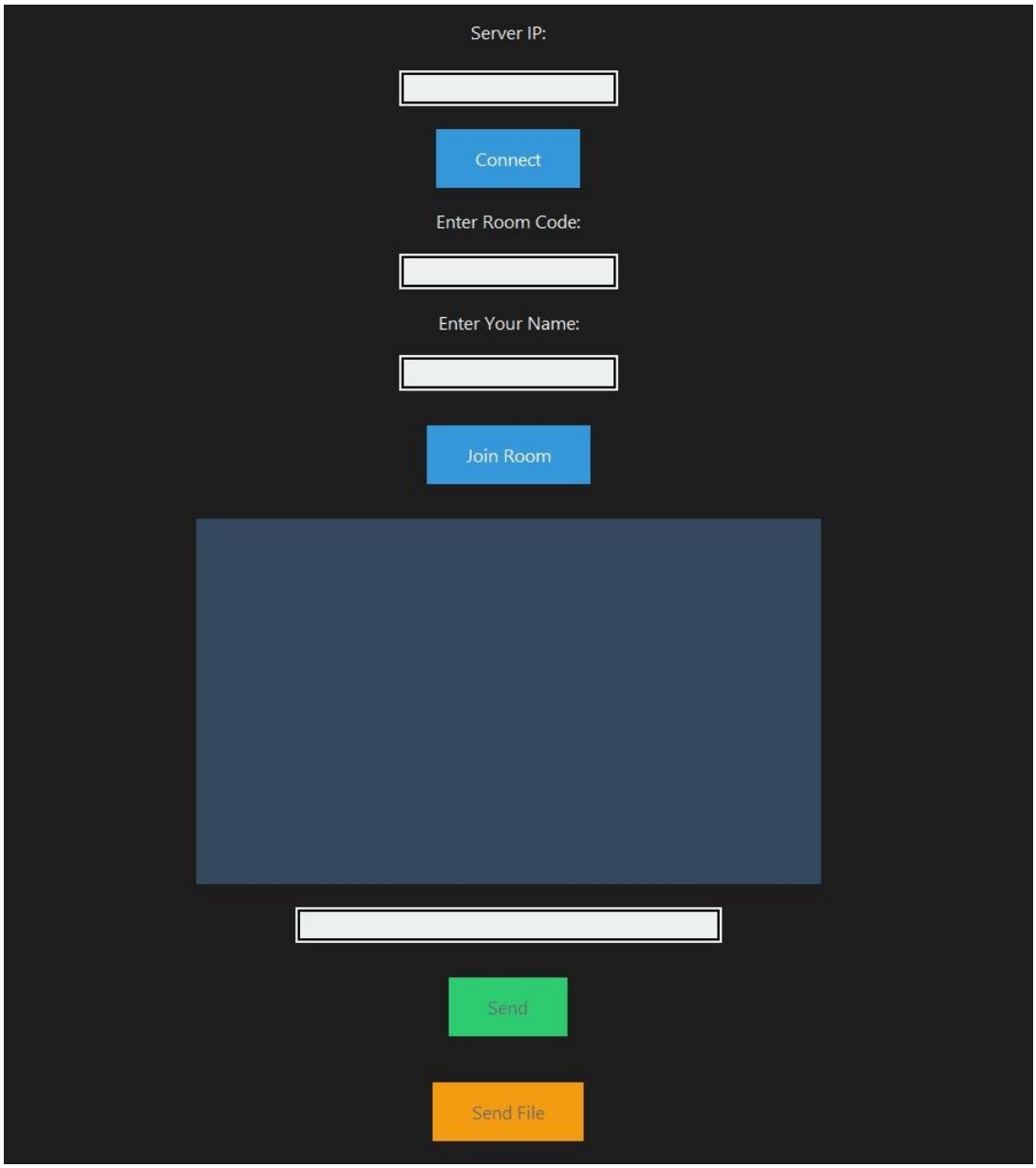
2. **Joining a Chat Room**:

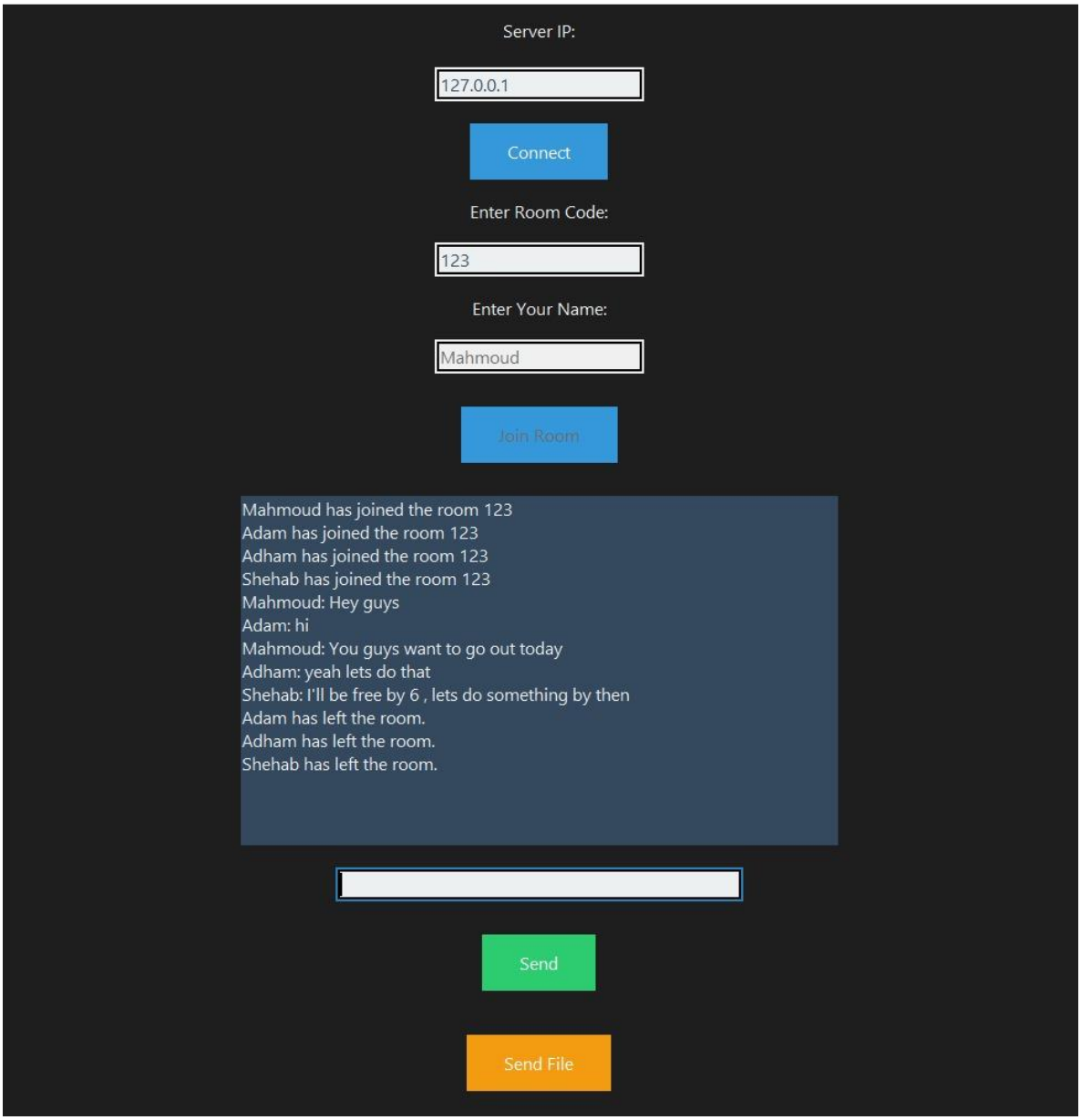
a. User enters a room code and their name, then clicks "Join Room."
b. The client sends this information to the server to associate the user with a room.

3. **Messaging**:
    a. Users type messages in the entry field and click "Send."
    b. Messages are displayed in the chat box and transmitted to the server.

4. **Receiving Messages**:
    a. Messages from the server are continuously displayed in the chat box.

Screenshots:




Server IP:
127.0.0.1
Connect
Enter Room Code:
123
Enter Your Name:
Mahmoud
Join Room
Mahmoud has joined the room 123
Adam has joined the room 123
Adham has joined the room 123
Shehab has joined the room 123
Mahmoud: Hey guys
Adam: hi
Mahmoud: You guys want to go out today
Adham: yeah lets do that
Shehab: I'll be free by 6 , lets do something by then
Adam has left the room.
Adham has left the room.
Shehab has left the room.
Send
Send File

Group Work Distribution

1. **Mahmoud Hany (Team Leader) :** Socket Programming, GUI Design, Report and Poster final touches
2. **Adham Hesham :** Demo Video, GUI Design
3. **Shehab Tawfik :** Report, GUI Design
4. **Adam Mohammed :** Poster, GUI Design

The GUI Design was developed through collaborative efforts by all team members. The report and poster were reviewed and evaluated by the team leader for final approval.