Chat room System Documentation

***STUDENT NAME**: Samuel Asuoha*
***STUDENT NUMBER**: C00305107*

TABLE OF CONTENTS

1. Overview

This project is a simple chat room system that allows multiple users to:

- Authenticate (login or register).
- Create and join chat rooms.
- Send and receive messages in real-time.
- View chat history.

The system consists of a **server** that manages chat rooms and clients and a **client** program that allows users to interact with the server.

2. Server Side

2.1 Purpose

The server handles:

- User authentication (registration and login).
- Creation and management of chat rooms.
- Broadcasting messages between clients in the same chat room.
- Logging chat history for each chat room.

2.2 How It Works

1. **Authentication:**
    - Users log in with existing credentials or register a new account.
    - Credentials are stored in a text file (*users.txt*).
2. **chat room Management:**
    - Users can view a list of available chat rooms.
    - Users can join an existing chat room or create a new one.
3. **Message Broadcasting:**
    - Messages are time stamped and sent to all clients in the same chat room.
    - Messages are logged into the respective chat room's history file.
4. **Threading:**
    - Each client is handled in a separate thread, enabling multiple concurrent users.

2.3 Key Functions

a. **Authentication**
    - *authenticate(client_socket)*: Handles user login or registration.
    - *validate_credentials(username, password)*: Verifies user credentials.

- *register_user(username, password)*: Adds a new user to the system.

b. **chat room Management**
   - *list_chat rooms()*: Lists all available chat rooms.
   - *create_chat room(chat room_name)*: Creates a new chat room.
   - *join_chat room(client_socket, username, chat room_name)*: Adds a user to a chat room and displays chat history.

c. **Messaging**
   - *broadcast_message(chat room_name, message, username=None)*: Sends messages to all users in a chat room and logs them.

d. **Utilities**
   - *send_message(client_socket, message)*: Sends a message to a specific client.

2.4 Server Setup

- The server runs on *127.0.0.1* (localhost) and listens on port *5000*.
- The main function *start_server()* initializes the server and spawns a thread for each new client.

2.5 Server Files

- **Users File**: *users.txt* contains registered usernames and passwords in the format *username:password*.
- **chat room Files**: Stored in the chat rooms directory, each chat room has a *.txt* file to log chat history.

3. Client Side

3.1 Purpose

The client provides an interface for users to:

- Connect to the server.
- Authenticate via login or registration.
- Interact with the chatroom system by creating or joining chat rooms, sending messages, and exiting.

3.2 How It Works

1. The client connects to the server at *127.0.0.1* on port *5000*.
2. A background thread listens for messages from the server, ensuring real-time updates.
3. The user interacts with the server via text-based input, allowing them to send commands and messages to the server.

3.3 Key Functions

a. **Message Handling**
   - *receive_messages(client_socket)*: Continuously receives and displays messages from the server.

b. **Client Setup**
   - *start_client()*: Establishes the connection to the server and manages user interactions.

3.4 User Commands

- **chat room Selection:**
  - *create <chat room_name>*: Creates a new chat room.
  - *<chat room_name>*: Joins an existing chat room.
- **Chat Messaging:**
  - Type a message to send it to the chat room.
  - *exit*: Leave the chat room and disconnect.

4. Usage Instructions

4.1 Running the Server

1. Save the server script as *server.py*.
2. Run the script
3. The server will start and listen on port *5000*.

4.2 Running the Client

1. Save the client script as *client.py*.
2. Run the script
3. Follow the on-screen instructions to log in, register, or join a chat room.

5. Features

5.1 Authentication

- Ensures only valid users can access the system.
- Supports user registration with unique usernames.

5.2 chat room System

- View available chat rooms.
- Create a new chat room.

- Join an existing chat room and view its history.

5.3 Real-Time Messaging

- Messages are time stamped and broadcasted to all users in a chat room.
- Chat history is saved to a file for persistence.

5.4 Multi-Client Support

- Multiple users can connect and interact simultaneously.
- Each client runs in its own thread.

6. Limitations

- Chat room names must be unique.
- Credentials are stored in plain text (consider encrypting passwords for production use).
- The server is designed for local use and may require modifications for deployment over a network.

7. Future Enhancements

- **Enhanced Security**: Implement password encryption and secure connections using SSL.
- **Private Messaging**: Allow users to send direct messages.
- **GUI**: Develop a graphical interface for better user experience.
- **File Sharing**: Enable file transfers within chat rooms.
- **Admin Controls**: Add administrative features to manage chat rooms and users.

8. Conclusion

This chat room system is a reliable application for real-time text-based communication. By leveraging Python's networking and threading capabilities, it supports multiple users interacting concurrently. It is a foundational project that can be further developed into a robust communication platform.