University of East London

Ain Shams University

Faculty of Engineering

AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

CREDIT HOURS ENG. PROGRAM

Computer engineering and software systems

Phase 1: CSE 351

Project

Peer-to-Peer Multi-User Chatting Application

Submitted to:

Prof. Ayman M. Bahaa-Eldin

Submitted by:

Mohamed Mostafa Bedair El Maghraby	20p7732
Malak Mohamed Mahfouz Mohamed Sadek	20P7813
Mohamed Hesham El Said Zidan	20p7579
Ahmed Saif Elsayed Ibrahim Soliman	20P7668

Contents

Project Goal:

This project aims to develop a robust, user-centric, and scalable Peer-to-Peer Multi-User Chatting Application using Python and sockets. Focused on text-based communication, to emulate features akin to platforms like Clubhouse, emphasizing secure user authentication, efficient client-server communication, and to implement versatile chat functionalities, including group and one-to-one messaging. Additionally, the project aims to implement user-friendly features such as message formatting and hyperlink support, ensuring a seamless and visually appealing user experience.

Project Scope:

Core functionalities include user authentication, basic client-server setup, chat room features, group messaging, one-to-one chat sessions, message formatting, error handling, and UI enhancements. The project will adhere to a command-line interface for simplicity, utilize color-coded messages, and focus on documentation to facilitate easy installation, configuration, and usage. Robust error handling, resilience to network interruptions, and scalability to handle increasing user loads will be integral to the application's success.

List of Functionalities:

1. User Authentication:

- User should be able to create an account using a unique username and password or log in to an existing account if one exists.
- User should have his unique usernames used as his identifier for other users.
- User should be able to change his username if desired.

2. Basic Client-Server Setup:

- Users should be able to connect to the server using a client application.

- Users should be able to see a list of all the currently active users.

3. Chat Room Functionality:

- Users should be able to create or join a chat room.
- Users should be able to see a list of available chat rooms.
- The user that creates the chat room should be the group leader/admin.
- A Chat room admin should be able to remove other participants from the chat.
- When a chat room admin leaves the chat room, the second oldest member in the chat room should become the new admin.
- A chat room admin should be able to make other participants admins.

4. Group Messaging in Chat Rooms:

- Users should be able to send/see a message to/from everyone in the chat room.
- Users should receive notifications for new messages.
- Users can turn off notifications/mute from a chat room.

5. One-to-One Chat Functionality:

- Users can initiate one-to-one chat sessions, in which users can send/receive messages with only one user.
- Users should receive notifications for new private messages.
- Users can turn off notifications/mute from a user in private chat.

6. Message Formatting and Features:

- The application supports basic text formatting (e.g., bold, italics) in messages.

- Requirement: Users can share hyperlinks in messages, which opens a browser and redirects a user that clicked on it to a certain URL.

7. Error Handling and Resilience:

- The application implements robust error handling for unexpected scenarios. (for ex: invalid command or character entered)
- User should receive meaningful error messages for troubleshooting.
- Users are automatically reconnected in case of a network interruption.

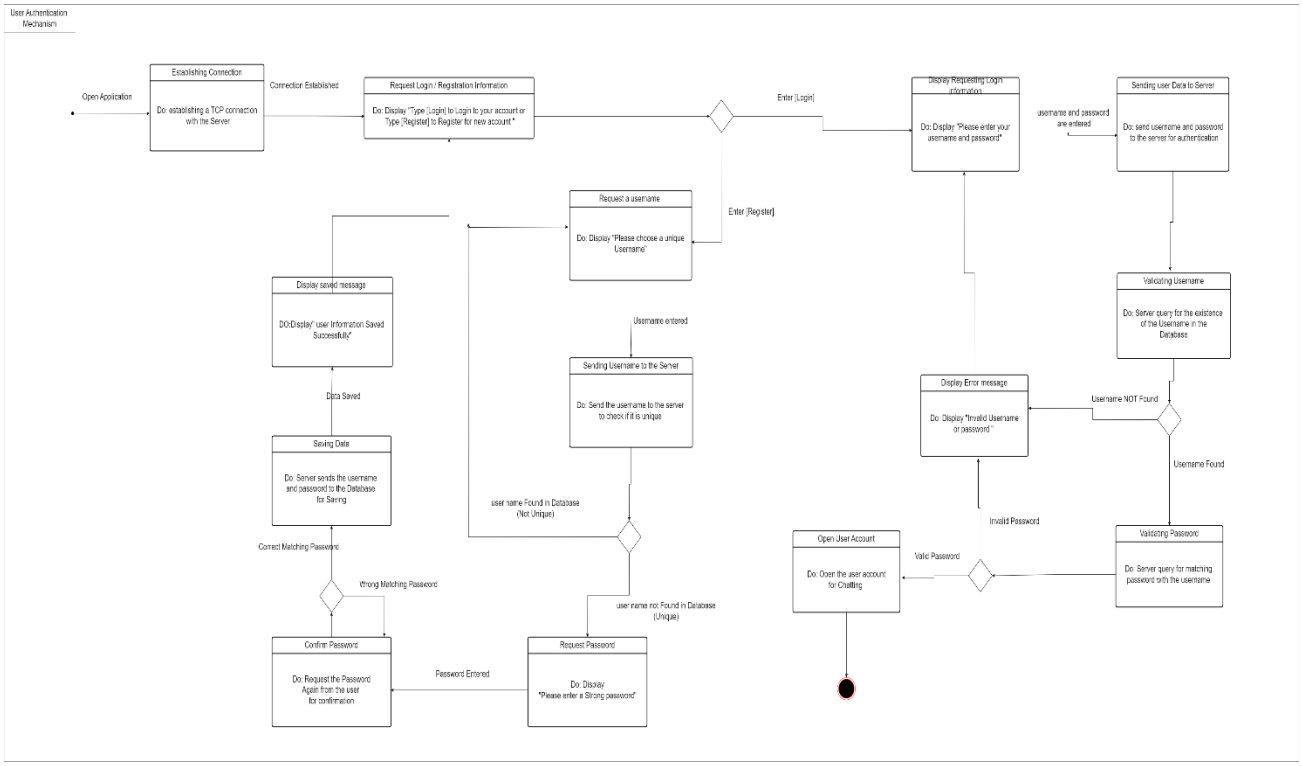
8. User Interface (UI) Enhancements:

- The application implements a clean command line user interface, utilizing color-coded messages.
- User can identify different types of messages, for example usernames appears in a certain colour, notifications appear in a different colour and font, connection errors appear in a third different way. Etc

System Diagrams

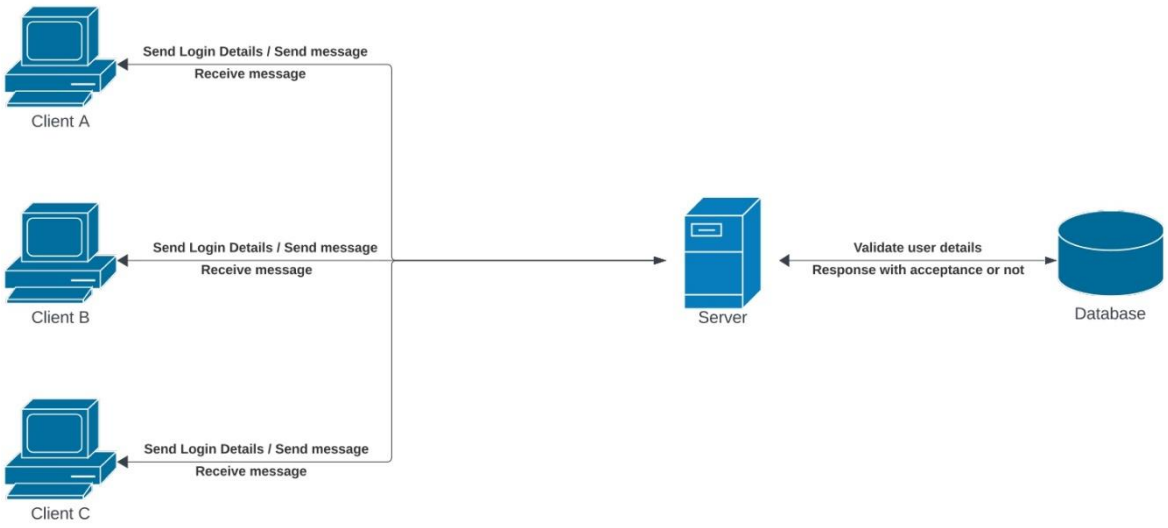
User Authentication Mechanism State Diagram:

Below is a state Diagram showing the states of the system to authenticate the user either by logging in or registration.


User Authentication Mechanism
Establishing Connection
Do: establishing a TCP connection with the Server
Open Application
Connection Established
Request Login / Registration Information
Do: Display "Type [Login] to Login to your account or Type [Register] to Register for new account "
Enter [Login]
Display Requesting Login information
Do: Display "Please enter your username and password"
username and password are entered
Sending user Data to Server
Do: send username and password to the server for authentication
Enter [Register]
Request a username
Do: Display "Please choose a unique Username"
Display saved message
DO:Display" user Information Saved Successfully"
Data Saved
Saving Data
Do: Server sends the username and password to the Database for Saving
Correct Matching Password
Wrong Matching Password
Confirm Password
Do: Request the Password Again from the user for confirmation
Password Entered
Request Password
Do: Display "Please enter a Strong password"
Username entered
Sending Username to the Server
Do: Send the username to the server to check if it is unique
user name Found in Database (Not Unique)
user name not Found in Database (Unique)
Validating Username
Do: Server query for the existance of the Username in the Database
Username NOT Found
Display Error message
Do: Display "Invalid Username or password "
Username Found
Invalid Password
Validating Password
Do: Server query for matching password with the username
Valid Password
Open User Account
Do: Open the user account for Chatting

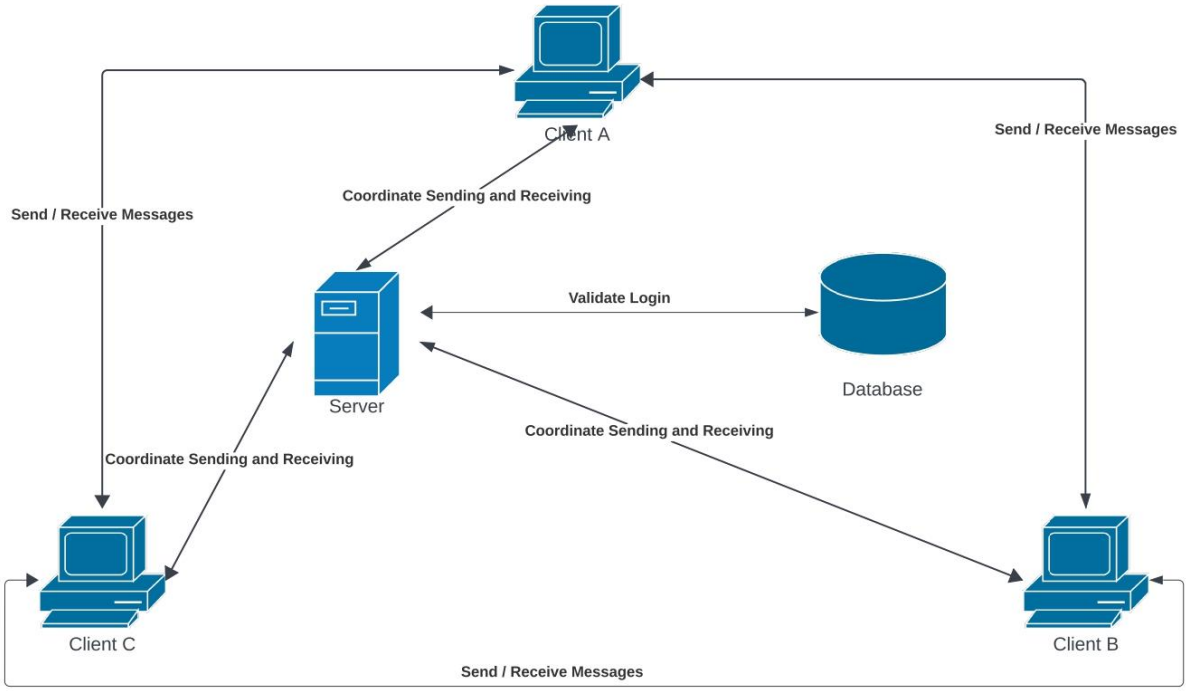
System Architecture Diagram:

1.Client server:



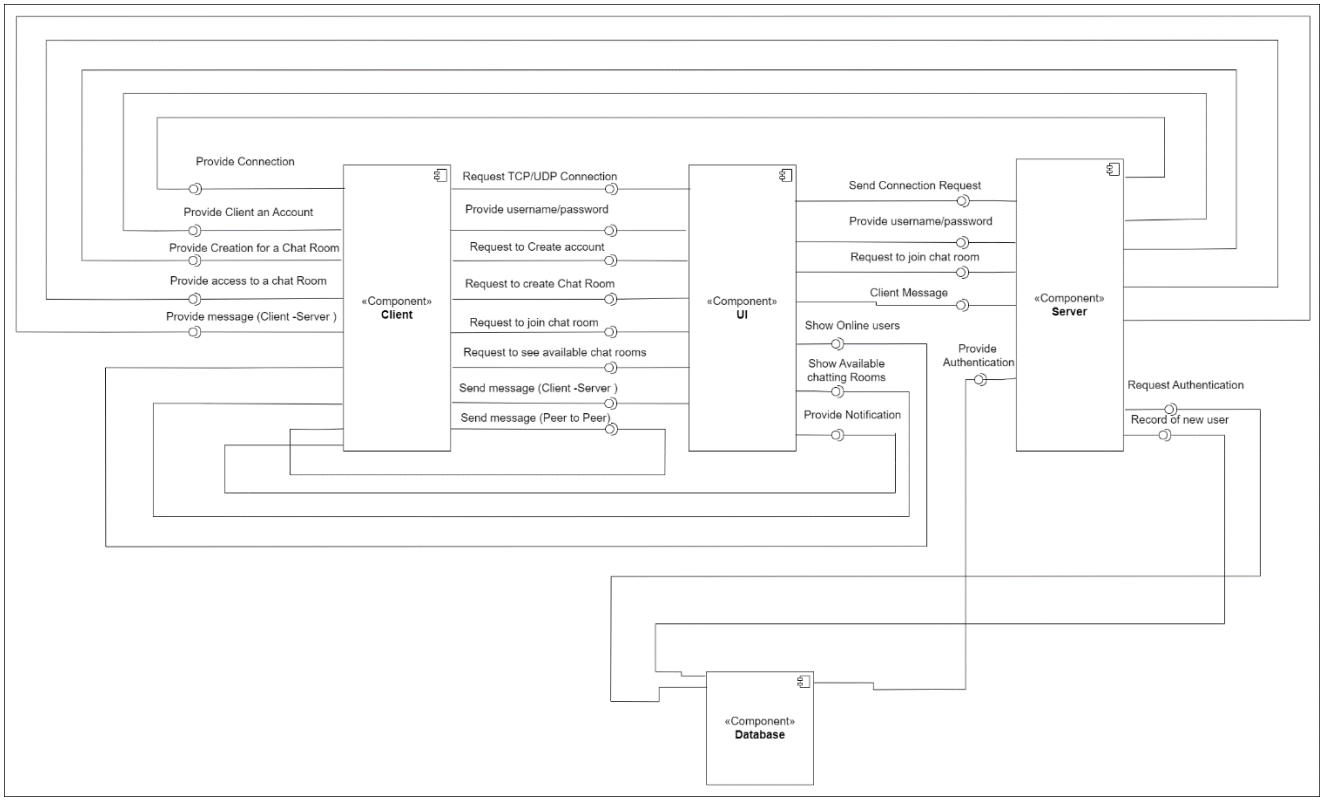
The way a server provides resources and services to one or more clients is described by the client-server model. We have clients (a, b and c) and it validates user details the same way as peer to peer but in client server the user sends the messages to the server with the receiver name that will receive the message. The connection in client server is between them and the server not between them and each other's like peer-to-peer

2.Peer to peer:



A peer-to-peer (P2P) network is created when two or more PCs are connected and share resources without going through a separate server computer. Clients (a, b and c) send requests to connect with the server to validate the user details with database then it will send and receive the messages with each other's and the server can act as a tracker to coordinate the sending and receiving (optional)

Component Diagram:



Communication Protocols:

Note: These protocols are still under development which means it can be added or modified later along the project.

We've created 12 communication protocol, 4 for requests, 4 for successful response messages, and 4 for failure response messages.

Request Protocols:

1.Login & Registration protocol

```
Title:[Login,Registration]
Connection:TCP
send From:[IP address,PortNo]
Send to:[Server IP address, PORT_NO]
Data:[Username,Password]
```

This protocol is for login & registration requests; therefore, login and registration methods are used in the field "title" and the connection used is TCP to make sure credentials are correctly sent.

Data of sender, IP address and port number, are sent to the recipient server. (Identified using its IP address and port number).

The content of the message itself is the user's credentials. (username & password)

2.Chatroom protocol

```
Title:[ChatRoom]
Connection:TCP
send From:[IP address,Port_No]
Send to:[IP address, PORT_NO]
Data:[[CREATE,CHATROOM_NAME],[JOIN,CHATROOM_NAME],SEE,QUIT]
```

This protocol is for chat room operations. Once again, TCP connection is used to make sure credentials are correctly sent.

Data of sender, IP address and port number, are sent to the recipient server. (Identified using its IP address and port number).

The content of the message itself is the desired method on chat room.

3.One-to-One protocol

```
Title:[PRIVATE]
connection:[TCP,UDP]
send From:[IP address,PortNo]
Send to:[IP address, PORT_NO]
Data:[USERNAME]
```

Connection type for this protocol is chosen by the user as TCP or UDP depending on his needs and preferences.

Data of sender, IP address and port number, are sent to the recipient server. (Identified using its IP address and port number).

The content of the message itself is the desired method on the private chat.

P2P Chatting Application Introduction

Just to Remember from phase 1 we wanted to implement Peer-to-Peer Multi-User Chatting Application that has a lot of functionalities mentioned in Phase1.

In this Phase we will discuss implementing The server and the Client and how they interact with each other and how the server handle multiple clients. In addition, we will show how the server authenticate the Client using A TCP Connection with the client through the database.

All of this will be made using python and sockets. A socket is a communications connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes.

The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP. A wide range of operating systems support socket APIs. i5/OS sockets support multiple transport and networking protocols. Socket system functions and the socket network functions are thread safe.[2]

Server

A server is a computer program or device that provides a service to another computer program and its user, also known as the client. In a data centre, the physical computer that a server program runs on is also frequently referred to as a server. That machine might be a dedicated server, or it might be used for other purposes.

In the client/server programming model, a server program awaits and fulfils requests from client programs, which might be running in the same, or other computers. A given application in a computer might function as a client with requests for services from other programs and as a server of requests from other programs.[1]

In this case we will implement the server using Socket programming using python programming language, that is capable of handling multiple clients Simultaneously.

Server functions:

Main():

First thing we must import the needed libraries for this task:

1- Socket lib -> for establishing socket connection.
2- Threading lib -> for handling multiple clients
3- Hashlib lib -> for hashing the password before sending it to the database.
4- Sqlite3 -> for connecting with the database and write SQL queries.

```
1  import socket
2  import threading
3  import hashlib
4  import sqlite3
```

Then we must specify the address of the of the server and its port number. Then we establish a TCP connection with the server using a socket and we bind the host with the port number and make the server start Listening.

```
host = '127.0.0.1'
port = 56789

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()
```

Then we make an empty dictionary for the clients to store the client object as a key and for each client object their will be a value of List contain the unique username and the nickname of the client.

```
clients = {}
```

Then we will print a “server is listening…” to know that the server is up and running.

Finally a while Loop is made and we will state inside it the **server.accept()** method so we keep accepting clients and we create a thread that will run the **Handle_Client()** function separately for each accepted client without affecting the server for accepting other clients.

```
print("Server is listening...")

while True:
    client, address = server.accept()
    threading.Thread(target=Handle_Client,args=(client,address)).start()
```

Handle_Client():

1- Function Parameters:
   a. Client object
   b. Address of the client
2- Function Description:

This function is responsible for handling all the client issues starting from logging in and registration ending with allowing the user to chat in a chatting Room or one to one chatting or even if the client suddenly disconnected from the server.

First thing the function calls **Login_or_register()** function (navigate to it's functionality through the table of contents) , to know the choice of the user whether to login or register , etc. Then it returns a username of the client to be added to the clients dictionary and be able to enjoy the functionalities of the application.

Then it takes a nickname from the client and broadcast to all the other online clients that this client is Online and has joined the server through **Broadcast()** function (navigate to it's functionality through the table of contents).

In Case of, the client disconnected in an inappropriate way the server will print that this client has disconnected and if the client was logged in it will broadcast that the client has left the server and he is now offline and then will remove him from the dictionary of the clients and the server will close the connection with this specific client. However, if the client was not logged in the server will only close the connection with this client.

3- Function snippet code:

```python
def Handle_Client(client,address):
    while True:
        try:
            Username=Login_or_register(client)

            print(f"Connected with {str(address)}")

            client.send('Choose Your Nickname'.encode('ascii'))
            nickname = client.recv(1024).decode('ascii')
            clients[client]=[Username,nickname]

            print(f'Nickname of the client is {nickname}!')
            broadcast(f'{Username} is now online! as "{nickname}"'.encode('ascii')) #e3meli loon le username , we loon lel nickname
            client.send('Connected to the server!\n'.encode('ascii'))

            Show_Menue(client)
            # thread = threading.Thread(target=handle, args=(client,))
            # thread.start()
        except:
            print(f"Lost connection with {str(address)}")
            # client may get disconnected before saving or appending his data (login)
            if client in clients:
                broadcast(f'{clients[client][0]} is now offline!'.encode())
                client.close()
                del clients[client]
                break
            else:
                client.close()
                break
```

Login_or_register():

1- Function Parameters:
    a. Client Object
2- Function Description:

This function Displays the options for the client to either login or register or Close the Application.

In Case of choosing to login, the server will request the username and password of the client and then will authenticate them using **Client_authentication()** function (navigate to it's functionality through the table of contents) , if one of them is wrong it will display a message "Wrong user name or password " and will let the user to choose the command again , however if the username and password are correct it will login the client to server and show the client the list of features that could be done in the application.

In Case of choosing to Register, The server will request a unique Username from the Client and it will check whether it is unique or not through **is_unique()** function (navigate to its functionality through the table of contents). If the username is unique, the server will the let the user to choose his password however if the password is weak, the server will request a strong password, and if it is not weak the new username and password will be saved in the database and the client will be redirected to the Login Options again to choose from it. However, if the username is not unique the server, the server will display "This username has been taken" and will let the user to enter a unique username again.

In Case of choosing an invalid command, the server will display "Please enter A valid command" and will take another command from the client.

3- Function snippet code:

```
def Login_or_register(client):
    # client.send(str(Fore.WHITE+"Welecome To the Local P2P Chatting Application\n").encode())
    client.send("Welecome To the Local P2P Chatting Application\n".encode())
    client.send("1- Enter [login] to login\n".encode())
    client.send("2- Enter [Register] if You are New!\n".encode())
    client.send("3- Enter [Close!] if You want to leave the chatting application\n".encode())
    respond = client.recv(1024).decode()

    while True:
        if respond.lower() == "login":
            client.send("Username :".encode())
            Username = client.recv(1024).decode()
            client.send("Password :".encode())
            Password = client.recv(1024).decode()  # Receive the password directly

            status = Client_authentication(Username, Password)

            if status:
                client.send("Login Successful !".encode())
                # usernames.append({Username:None})
                return Username
            else:
                client.send("Wrong UserName or Password!".encode())
                client.send("Choose Your Command again".encode())
                respond = client.recv(1024).decode()
```

```
        elif respond.lower() == "register":
            client.send("Please enter a Unique Username".encode())
            unique_username = client.recv(1024).decode()
            status = is_unique(unique_username)

            if status:
                client.send("This Username Has been Taken.".encode())
            else:
                respond = Client_Registration(client, unique_username)

        elif respond.lower() == "close!":
            pass

        else:
            client.send("Please enter A valid Command !".encode())
            respond = client.recv(1024).decode()
#-------------------------------------------------------------------------------------------
```

Client_authentication():

1- Function Parameters:
   a. Username of the Client
   b. Password of the Client
2- Function Description:

   This Function connects with the Database and hash the password and then query for the existence of this username with that specific hashed password.

   In case it is found it returns True, Otherwise Returns False.

3- Function snippet code:

```
#-------------------------------------------------------------------------------------------------------------
def Client_authentication(Username, Password):
    conn = sqlite3.connect("DataBase.db")
    cur = conn.cursor()

    # Hash the provided password before comparing
    hashed_password = hashlib.sha256(Password.encode()).hexdigest()

    cur.execute("SELECT * FROM Client_Data WHERE USERNAME = ? AND PASSSWORD = ?", (Username, hashed_password))
    if cur.fetchall():
        return True
    else:
        return False
#-------------------------------------------------------------------------------------------------------------
```

Client_Registration():

1- Function Parameters:
   a. Client object
   b. Unique Username
2- Function Description:

   This function is used after we made sure that the username the client entered is unique using **is_unique()** function (navigate to it's functionality through the table of contents) then the server requests from the user a strong password and the server validates whether the password is strong or not through **is_strong()** function (navigate to it's functionality through the table of contents) after the validation, the server saves the information of the new Account to the Database using **add_new_user()** function (navigate to it's functionality through the table of contents) and then the server takes the next command from the user and return it to **Login_or_Registration()** function .

3- Function snippet code:

```
def Client_Registration(client, unique_username):
    client.send("Please Enter a Strong Password".encode())
    New_Password = client.recv(1024).decode()
    New_Password = is_strong(client, New_Password)

    # Hash the password before storing it in the database
    hashed_password = hashlib.sha256(New_Password.encode()).hexdigest()

    add_new_user(unique_username, hashed_password)
    client.send("Congrats, a new Account has been created".encode())
    client.send("Choose Your Command again".encode())
    respond = client.recv(1024).decode()
    return respond
```

add_new_user():

1- Function Parameters:
   a. Unique Username
   b. Hashed Password
2- Function Description:

   This function connects to the Database and makes an Insert Query to it for the username and password.

3- Function snippet code:

```
def add_new_user(unique_username, hashed_password):
    connection = sqlite3.connect("DataBase.db")
    cur = connection.cursor()
    cur.execute("INSERT INTO Client_Data (USERNAME, PASSSWORD) VALUES (?, ?)", (unique_username, hashed_password))
    connection.commit()
#-----------------------------------------------------------------------------------------------------
```

is_unique():

1- Function Parameters:
   a. Username
2- Function Description:

   This Function connects with the database and query for the existence of such username in the database, if it is Found the function returns True , otherwise it returns False.

3- Function snippet code:

```
def is_unique(UserName):
    conn = sqlite3.connect("DataBase.db")
    cur = conn.cursor()

    cur.execute("SELECT * FROM Client_Data WHERE USERNAME = ?", (UserName,))
    if cur.fetchall():
        return True
    else:
        return False
```

is_strong():

1- Function Parameters:
   a. Client Object
   b. password
2- Function Description:

   This function see the length of the password, if it is less than 5 characters, the server notifies the client that this is a weak password, and the server will loop on the password until the client enter a strong one and the function then return the strong password.

3- Function snippet code:

```
def is_strong(client, password):
    while len(password) < 5:
        client.send("Weak password! Please choose a password with 5 or more characters".encode())
        password = client.recv(1024).decode()
    return password
#-----------------------------------------------------------------------------------------------------
```

Broadcast():

1- Function Parameters:
   a. Message
2- Function Description:

   This function aims on looping upon the clients a send them all the message sent to this function.

3- Function snippet code:

```
def broadcast(message):
    for client in clients:
        client.send(message)
#-------------------------------------------------------------------
```

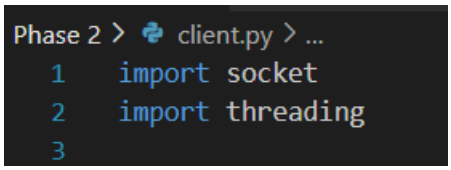
Client

Client Functions:

Main():

First thing we will import the necessary libraries for running the client code

1- Socket
2- threading



Then we make a client object using socket library establishing a TCP connection, then we specify the address and the port number of the server to be connected with.

Finally, we will make two threads, one for the **receive** function and the other one for the **write** function so they can work simultaneously and independently. Noticing that we will put all of this code inside try except statement in case the client code is running without a server to catch this exception.

Receive():

1- Function Parameters:
   a. NULL
2- Function Description:

   This function consists of a while loop containing the receiving function from the server to keep receive messages or commands from the server and print this message being Received for the Client.

3- Function snippet code:

```
# Function to handle receiving messages from the server
def receive():
    while True:
        try:
            # Receive messages from the server, decode them from ASCII
            message = client.recv(1024).decode('ascii')
            print(message)
        except:
            # If an error occurs during message reception, print an error message
            print("An error occurred")

            # Close the client socket and exit the loop
            client.close()
            break
```

Write():

1- Function Parameters:
   a. NULL
2- Function Description:

   This consists of a while loop that keeps taking input from the client and send it to the Server.

3- Function snippet code:

```
def write():
    while True:
        message=f"{input('')}"
        client.send(message.encode('ascii'))
```

Database

First of all, this code of the data base can be used only once just to initialize and create a table in the database and to create a database File. However it can be modified upon Request.

Database Functions:

Main():

First Thing we Import the necessary libraries for establishing the database.

```
Phase 2 > Database.py > ...
  1   import sqlite3
  2   import hashlib
```

Then we create an Empty Table Named "**Client_Data**" using SQL query, Having The username as its primary key as the username must be unique for each Client, and a password related to that Username.

```
cur.execute("""
CREATE TABLE IF NOT EXISTS Client_Data (
            USERNAME VARCHAR(255) PRIMARY KEY,
            PASSSWORD VARCHAR(255) NOT NULL
)
""")
```

After executing one or more SQL statements or modifications to the database (such as inserts, updates, or deletes), the changes are not immediately applied to the database. Instead, they are held in a transaction.

Calling **connection.commit()** is necessary to commit (save) the changes made during the current transaction to the database.

```
connection.commit()
```

Delete_All():

1- Function Parameters:
   a. NULL
2- Function Description:
   This function establishes a connection with the database, then run and SQL query for Deleting All the entries inside the Table.
3- Function snippet code:

```
def Delete_All():
    connection= sqlite3.connect("DataBase.db")
    cur=connection.cursor()
    cur.execute("DELETE FROM Client_Data")
```

Insert_db():

1- Function Parameters:
   a. Username
   b. Password
2- Function Description:

   This function establishes a connection with the database, then run and SQL query for inserting a new entry (Row) inside the table after hashing the password using the Hashlib Library.

3- Function snippet code:

```
def insert_db(username,password):
    connection= sqlite3.connect("DataBase.db")
    cur=connection.cursor()
    cur.execute("INSERT into Client_Data (USERNAME , PASSSWORD ) values (?,?)",(username,hashlib.sha256(password.encode()).hexdigest()))
```

Repository Link:

https://github.com/MHZDN/-P2P-Multi-User-Chatting

References

1- https://www.techtarget.com/whatis/definition/server [1]
2- https://www.ibm.com/docs/en/i/7.1?topic=communications-socket-programming [2]