VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER NETWORK (CO3094)

Assignment 1

CHAT APPLICATION

Advisor:	Nguyễn Phương Duy	
Students:	Nguyễn Hữu Trùng Dương	2052929 - CC05
	Nguyễn Mạnh Phú Quý	2052680 - CC05
	Huỳnh Tuấn Kiệt	2052561 - CC05
	Nguyễn Phước Nguyên Phúc	2053342 - CC05

HO CHI MINH CITY, DECEMBER 2022

Member list & Workload

No.	Fullname	Student ID	Task
1	Nguyễn Hữu Trùng Dương	2052929	Assignmet 2
2	Nguyễn Mạnh Phú Quý	2052680	- User functions, GUI, Class Diagram, Report
3	Nguyễn Phước Nguyên Phúc	2053342	- Admin functions, GUI, Architecture, Report
2	Huỳnh Tuấn Kiệt	2052561	Assignment 2

Contents

1 Introduction

In this Assignment, we build a chat application according to the TCP / IP protocols.

1.1 Requirements

1. The application allows two or more users on different machines to chat with each other
2. The chat client is built on the hybrid model between the Client-Server and P2P models. The system has a central server for user registration and online user management, but clients chat directly with each other
3. One person has a list of friends retrieved from the central server. He/she can start some chats with different people at the same time. Each conversation is a P2P connection.
4. Application allows file transfer during chats between clients through P2P connection.
5.

2 Application design

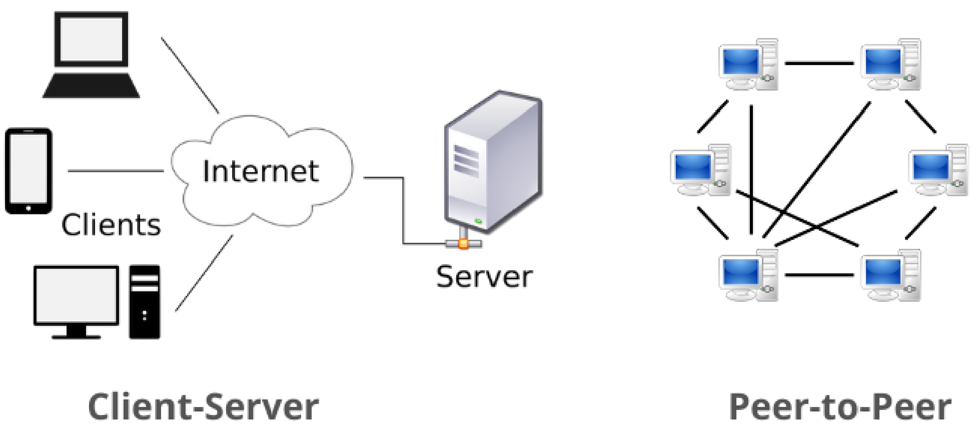


Figure 1: Client-server architecture

2.1 Based Architecture for Client-Server Model

1. Server

- Always-on host
- Permanent IP addres
- Often in data centers, for scaling

2. Client

- Contact, communicate with server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other

2.2 Based Architecture for P2P Model

In a p2p network, all computers on the network are considered equal, with each workstation offering access to resources and data.

This means that each node in the p2p network model can both request for services from the other peers or offer services to the other peers. Each node can be both a client and a server.

2.3 Hybrid of client-server and P2P

The chat client is built on the hybrid model between the Client-Server and P2P models. The system has a central server for user registration and online user management, but clients chat directly with each other.

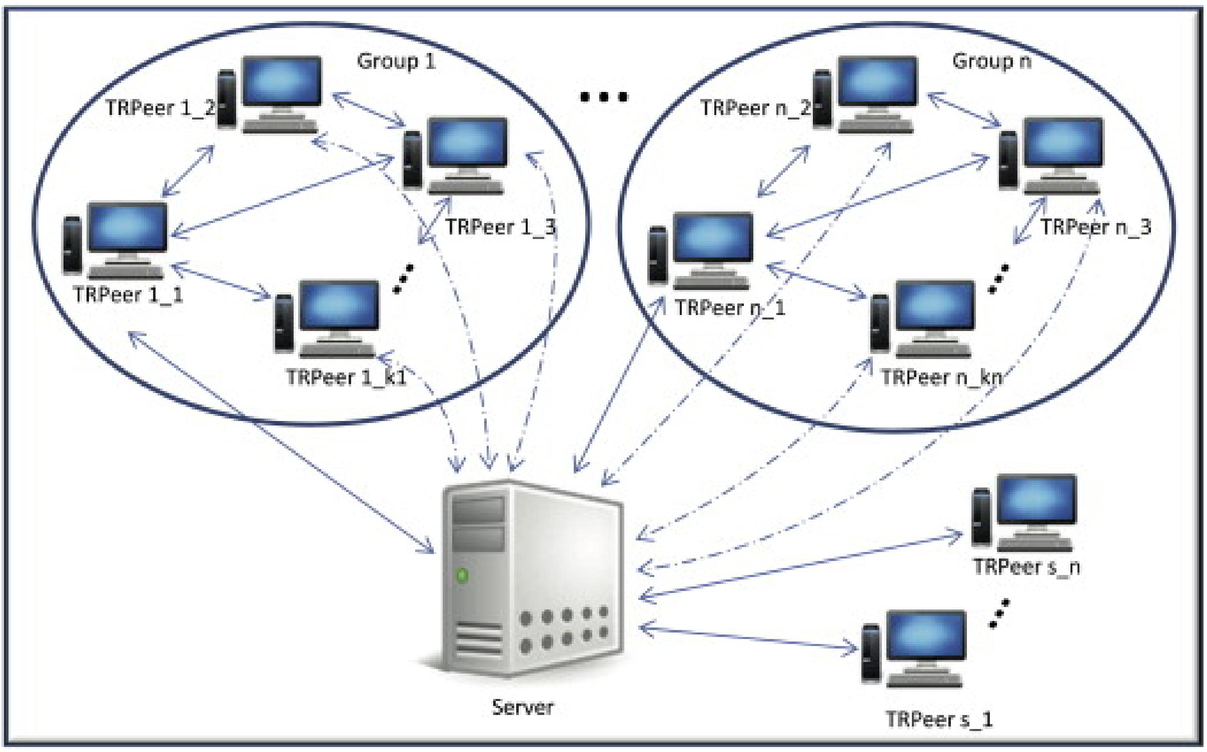


Figure 2: Client-server architecture

Architecture Description:

1. As above, Clients chat directly with each other- P2P model, this mean each User has both Server and Client Role. This is beacause:

- Has all the feature of Client and P2P model
- Because a User is a Server itself, it can always listen and accept other Client directly
- An User is also a Client, so it can Connect to other Server

2. We also need to Central Server as Admin to Handle User Registration and Authentication. Admin Role:

- If a User registers, Admin will add this User to the User List, also save it's IP address and Port
- Display list of Online User
- When a User want to chat with others on the Online List, it will request it's wanted User's IP and Port from Admin

Use-Case Diagram :

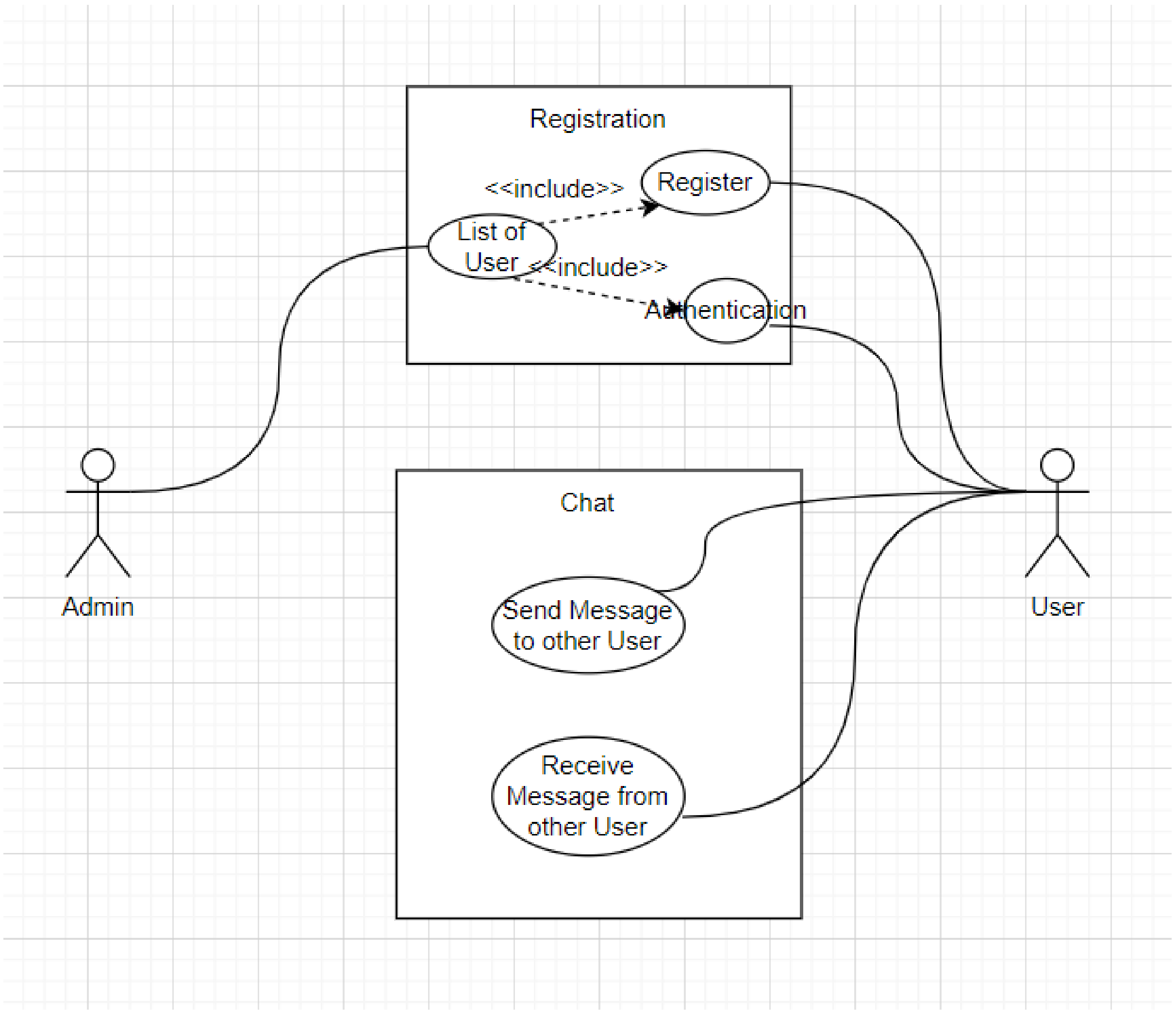


Figure 3: Client-server architecture

2.4 Class diagram for Chat Application

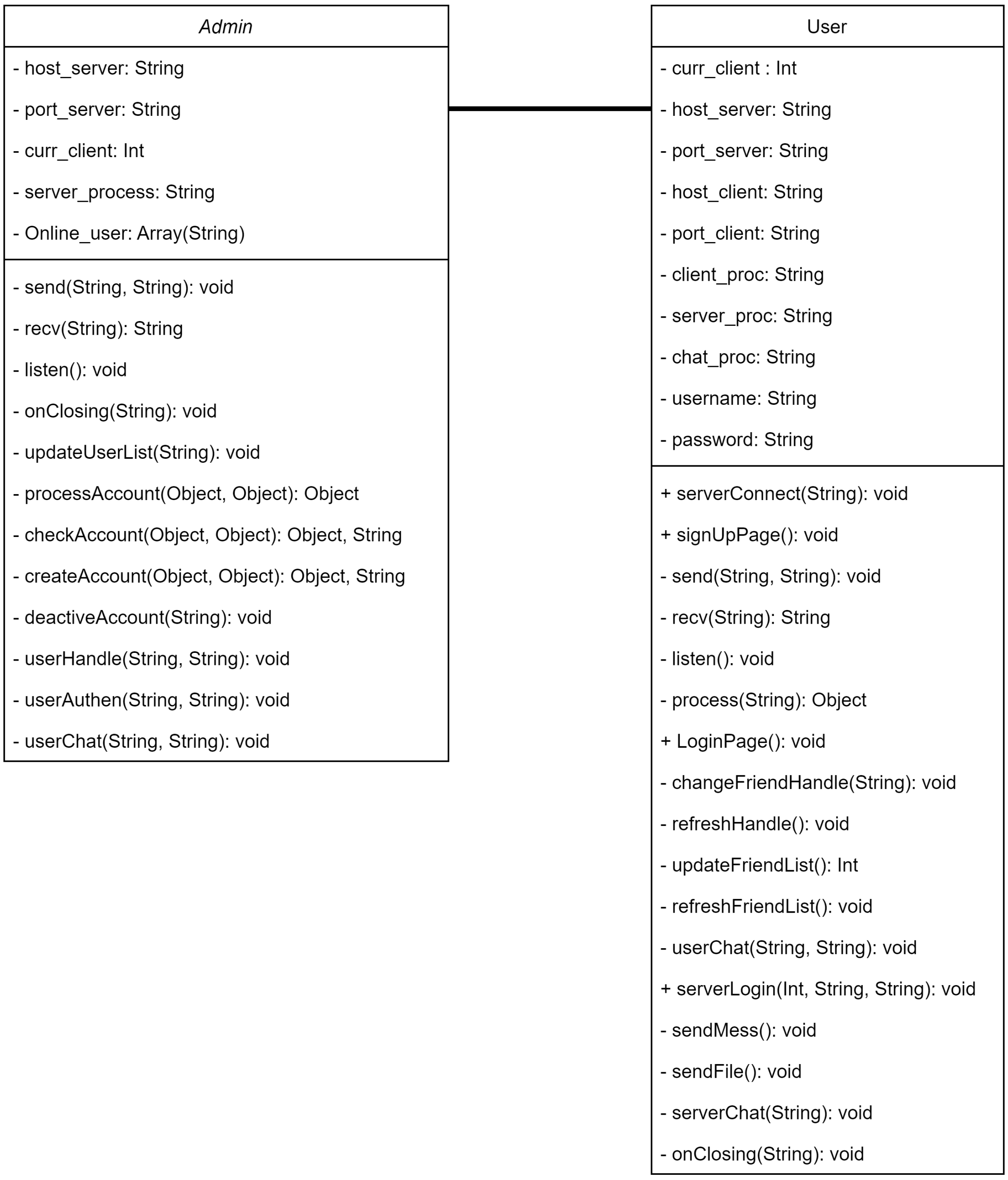

Admin
- host_server: String
- port_server: String
- curr_client: Int
- server_process: String
- Online_user: Array(String)
- send(String, String): void
- recv(String): String
- listen(): void
- onClosing(String): void
- updateUserList(String): void
- processAccount(Object, Object): Object
- checkAccount(Object, Object): Object, String
- createAccount(Object, Object): Object, String
- deactiveAccount(String): void
- userHandle(String, String): void
- userAuthen(String, String): void
- userChat(String, String): void
User
- curr_client : Int
- host_server: String
- port_server: String
- host_client: String
- port_client: String
- client_proc: String
- server_proc: String
- chat_proc: String
- username: String
- password: String
+ serverConnect(String): void
+ signUpPage(): void
- send(String, String): void
- recv(String): String
- listen(): void
- process(String): Object
+ LoginPage(): void
- changeFriendHandle(String): void
- refreshHandle(): void
- updateFriendList(): Int
- refreshFriendList(): void
- userChat(String, String): void
+ serverLogin(Int, String, String): void
- sendMess(): void
- sendFile(): void
- serverChat(String): void
- onClosing(String): void

Figure 4: Class diagram of Chat App

3 Phase 1

3.1 Define the communication protocols used for each function:

3.2 Define and describe the functions of the application

3.2.1 For admin:

1. Socket() : create socket object
2. Bind (): Binding the server to the host and port.
3. Listen (): server-process listen to client
4. Accept() : accept new client and create new thread
5. Receive_mess (): server-process receive message from other
6. Send(): send mess to other
7. UserHandle(): Handle User
8. Close ()

3.2.2 For User:

1. Socket() : create socket object
2. Bind (): Binding the server to the host and port.
3. Listen (): server-process listen to client
4. Connect () : Connect the clients to the server
5. Accept () : accept new client and create new thread
6. Receive_mess (): server-process receives message from other
7. Send(): send mess to other
8. Close (): close the connection

4 Phase 2

4.1 Describe each specific function of the application:

4.1.1 Admin's Function:

1. **socket.gethostbyname(socket.gethostname())**:
   - The Python function socket.gethostname() returns the host name of the current system under which the Python interpreter is executed.
   - This Python function can be combined with socket.gethostbyname()to get the IP address of the local host.

2. **server_process = socket.socket(socket.AF_INET, socket.SOCK_STREAM)**
   - The socket function returns a new socket object. A socket is an end point of communication. A socket returned by this method can be made either a server socket or a client socket by further customizing it.
   - AF_INET : For protocols based on IP addresses using IPv4. An IPv4 address consists of four numbers each ranging from 0 to 255 and each of them separated by a dot. IPv4 is a 32-bit number and supports up to 2 to the power 32 IP addresses.
   - socket.SOCK_STREAM: Specifies a stream socket (TCP). Needs to be provided when a streaming connection based client or server is needed which will be used for reliable communication.

3. **server_process.bind((self.host_server, self.port_server)):**
   - The bind() method of Python's socket class assigns an IP address and a port number to a socket instance. The bind() method is used when a socket needs to be made a server socket. As server programs listen on published ports, it is required that a port and the IP address to be assigned explicitly to a server socket.

4. **server_process.listen(10) :**
   - Calling listen() makes a socket ready for accepting connections. We can pass parameter to this function to define the maximum connections.

5. **listen_to_client(self):**
   - The function listens for new clients and creates a new thread for each client.
   - Function **accept()** is called in this function, the accept() method of Python's socket class, accepts an incoming connection request from a TCP client.

6. **receive_message(self, channel):**
   - It receives a message from the client
   - param channel: the channel that the message is being received from.
   - In this function, we use **recv()** : the recv() function of Python's socket module can be used to receive data from both TCP and UDP sockets.

7. **send(self, channel, message) :**
   - Server-process send message to other .

- In this function, we call sendall(): Gửi dữ liệu thông qua giao thức TCP.

8. **UserHandle(self,channel, client):**
   The function userHandle() is called when a user connects to the server. It calls the user-Authen() function to authenticate the user, then calls the updateUserList() function to update the user list, and finally calls the userChat() function to allow the user to chat

   :param channel: The channel that the user is in
   :param client: The client object that is connected to the server

9. **onClosing(self):**
   - In this function, we call **server_process.close()** to close the server

4.1.2 User's Function:

1. **serverConnect(self, serverAddress)**:
   - This function will get the server IP address and make it as the user IP address to make a peer.
   - In this function, both of the user's IP address and port number will connect to the server by using Python's connect() function

2. **signUpPage(self) and logInPage(self)**:
   - Enter the sign up screen and login screen for the user.

3. **userChat(self, channel, client)**:
   - This function is in the user's server process that can communicate to other users by using chat messages and files.
   - In this function, if the user is not directly chatting with specify user and that specify one send messages, the server will display them in the notification box.

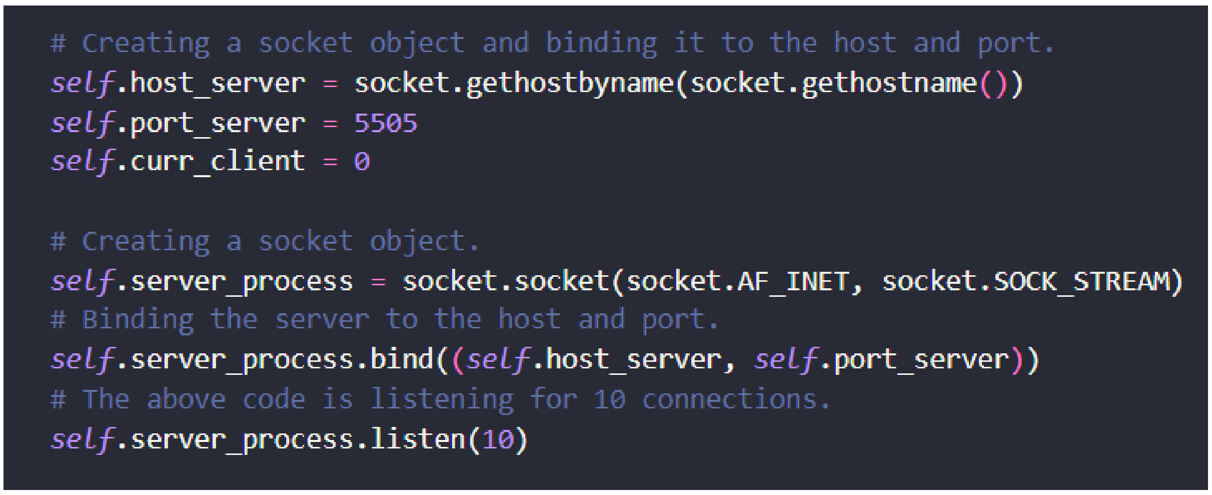
4. **serverLogin(self, mode, name, pssd)**:
   - Before given ability to communication with other, normal user has to send information to admin user this step i called login/sign in.
   - Execute Authentication follow the server instruction. This function also ensures that the server can receive messages in order. After that, the user can start to chat.

5. **sendMess(self) and sendFile(self)**:
   - Allows users to start communication with others

6. **onClosing(self)**:
   - In this function, we call **server_process.close()** to close the server and the application will be closed too.

7. **serverChat(self, name)**:
   - This function will help user to enter the chat room with other user.
   - This function use socket.socket() to create socket, then the chat server will connect the IP address and port of user we want to chat.

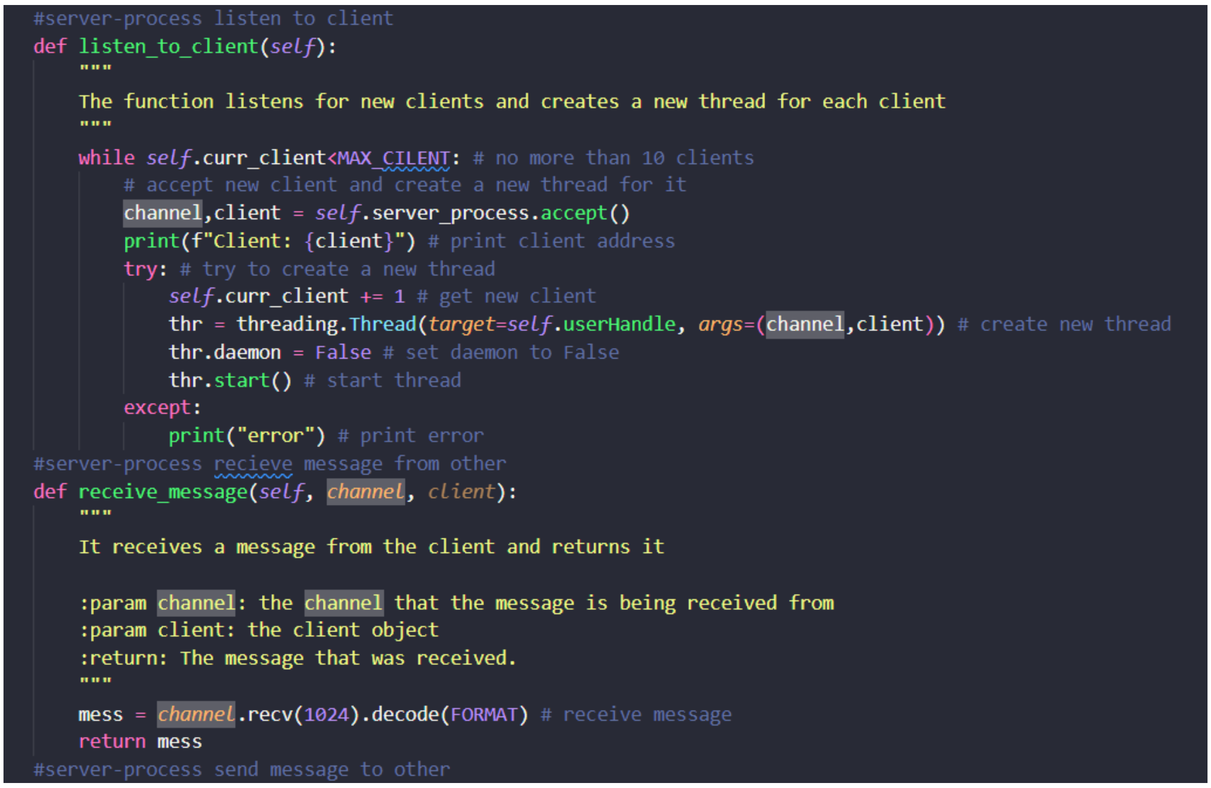
8. **send(self, channel, message):**
   - Server-process send message to other .
   - In this function, we call sendall(): Send messages through TCP protocol



```python
# Creating a socket object and binding it to the host and port.
self.host_server = socket.gethostbyname(socket.gethostname())
self.port_server = 5505
self.curr_client = 0

# Creating a socket object.
self.server_process = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Binding the server to the host and port.
self.server_process.bind((self.host_server, self.port_server))
# The above code is listening for 10 connections.
self.server_process.listen(10)
```

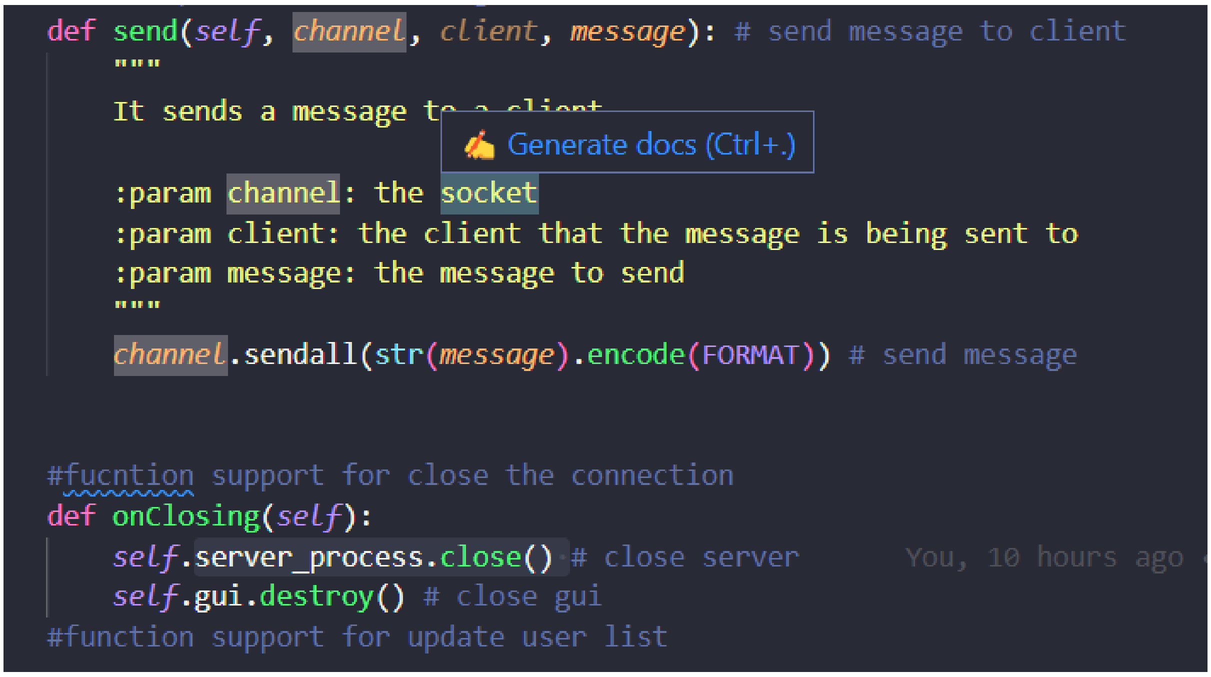
Figure 5: Admin functions



```python
#server-process listen to client
def listen_to_client(self):
    """
    The function listens for new clients and creates a new thread for each client
    """
    while self.curr_client<MAX_CILENT: # no more than 10 clients
        # accept new client and create a new thread for it
        channel,client = self.server_process.accept()
        print(f"Client: {client}") # print client address
        try: # try to create a new thread
            self.curr_client += 1 # get new client
            thr = threading.Thread(target=self.userHandle, args=(channel,client)) # create new thread
            thr.daemon = False # set daemon to False
            thr.start() # start thread
        except:
            print("error") # print error
#server-process recieve message from other
def receive_message(self, channel, client):
    """
    It receives a message from the client and returns it

    :param channel: the channel that the message is being received from
    :param client: the client object
    :return: The message that was received.
    """
    mess = channel.recv(1024).decode(FORMAT) # receive message
    return mess
#server-process send message to other
```

Figure 6: Admin functions

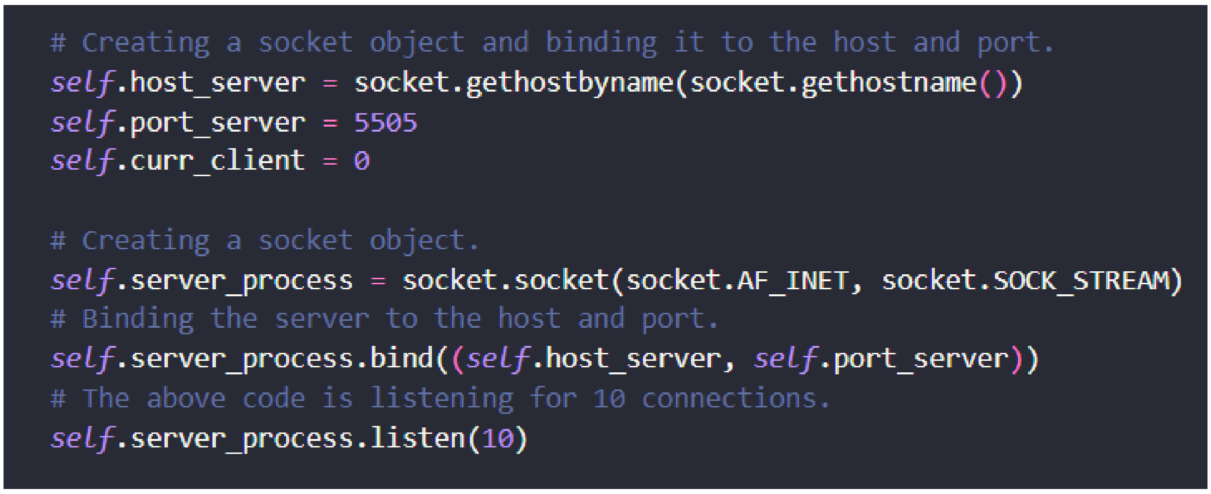


```python
def send(self, channel, client, message): # send message to client
    """
    It sends a message to a client

    :param channel: the socket
    :param client: the client that the message is being sent to
    :param message: the message to send
    """
    channel.sendall(str(message).encode(FORMAT)) # send message


#fucntion support for close the connection
def onClosing(self):
    self.server_process.close() # close server      You, 10 hours ago
    self.gui.destroy() # close gui
#function support for update user list
```

Figure 7: Admin functions



```
# Creating a socket object and binding it to the host and port.
self.host_server = socket.gethostbyname(socket.gethostname())
self.port_server = 5505
self.curr_client = 0

# Creating a socket object.
self.server_process = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Binding the server to the host and port.
self.server_process.bind((self.host_server, self.port_server))
# The above code is listening for 10 connections.
self.server_process.listen(10)
```

Figure 8: Admin functions

5 Extension functions of the system in addition to the requirements specified in section 2

5.1 Admin extension function:

For admin, beside base functions already described in Section 1, we need additional functions - for support User Management, such as:

- **CheckAccount():** If the name and password in the jsonObject match the name and password in the jsonFile, then update the address, port, and isAct fields in the jsonFile.
- **createAccount():** It takes a json file, and a json object, and appends the json object to the json file
- **DeactiveAccount():** It opens the json file, finds the account with the name that matches the username parameter, sets the isAct value to 0, and then dumps the json file
- **User_Authentication() :** It receives a message from the client, then sends a message back to the client to ensure that the client receives the message in order

5.2 User extension function:

For user, we can optimize the functions that we have given in the above sections by using some extended one - especially for manage friend list, their room and authentication.

- **refreshHandle()**: This function will execute after the users press the "Refresh" button, then the message box will display to warn the user if they want to leave all the conversation to update the friend list
- **process()**: The list account of friends will be converted into JSON file.
- **changeFriendHandle()**: If users press button of their online friend's name, then users will be announced that they will leave the current chat to enter that friend's room.
- **updateFriendlist()**: The friend list that server sent to user will be converted to JSON file and also the ID of users will be returned if in this list contains their username.
- **refreshFriendList()**: After refreshHandle() function, the client process will notify to other users that a new friend has entered the chat. And the user will connect to socket with their IP address and port number.