Chat APP in C# with Visual Studio

KOHL Aurélien[*]

[*] IT for Finance, Efrei Paris, Id : 20170211

I. Introduction

In this project, you are asked to create a network based multithreaded client-server chat app.

The Visual Studio solution (Console version) of the chat application is separated into 3 parts, a first project named "Communication" which will manage the communication between the server and the users, a second named "ConsoleAppClient" which will regroup the elements. of the client interface and a third named "ConsoleAppServer" which will manage the server.

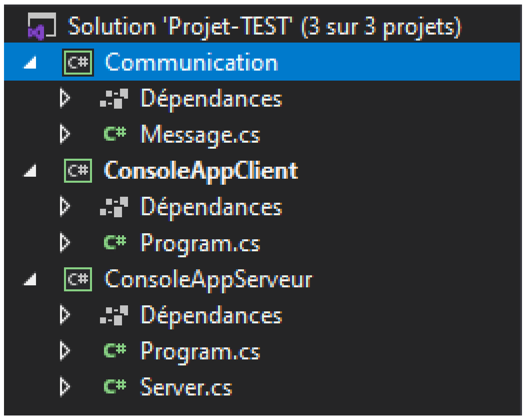


Figure 1- Project solution, console version

II. Functions implemented, Functions not implemented or not finished

- All the function are implemented on the server, with can test it with ConsoleAppClient.exe
    - Create a profile (login, password) and save it
    - login
    - List topics
    - create topics
    - join topics (only one)
    - send messages to all chatters on a specific topic
    - send private messages
- The GUI is still in progress

The next UML diagrams show a type of utilisation of this chat app and the principal functions.

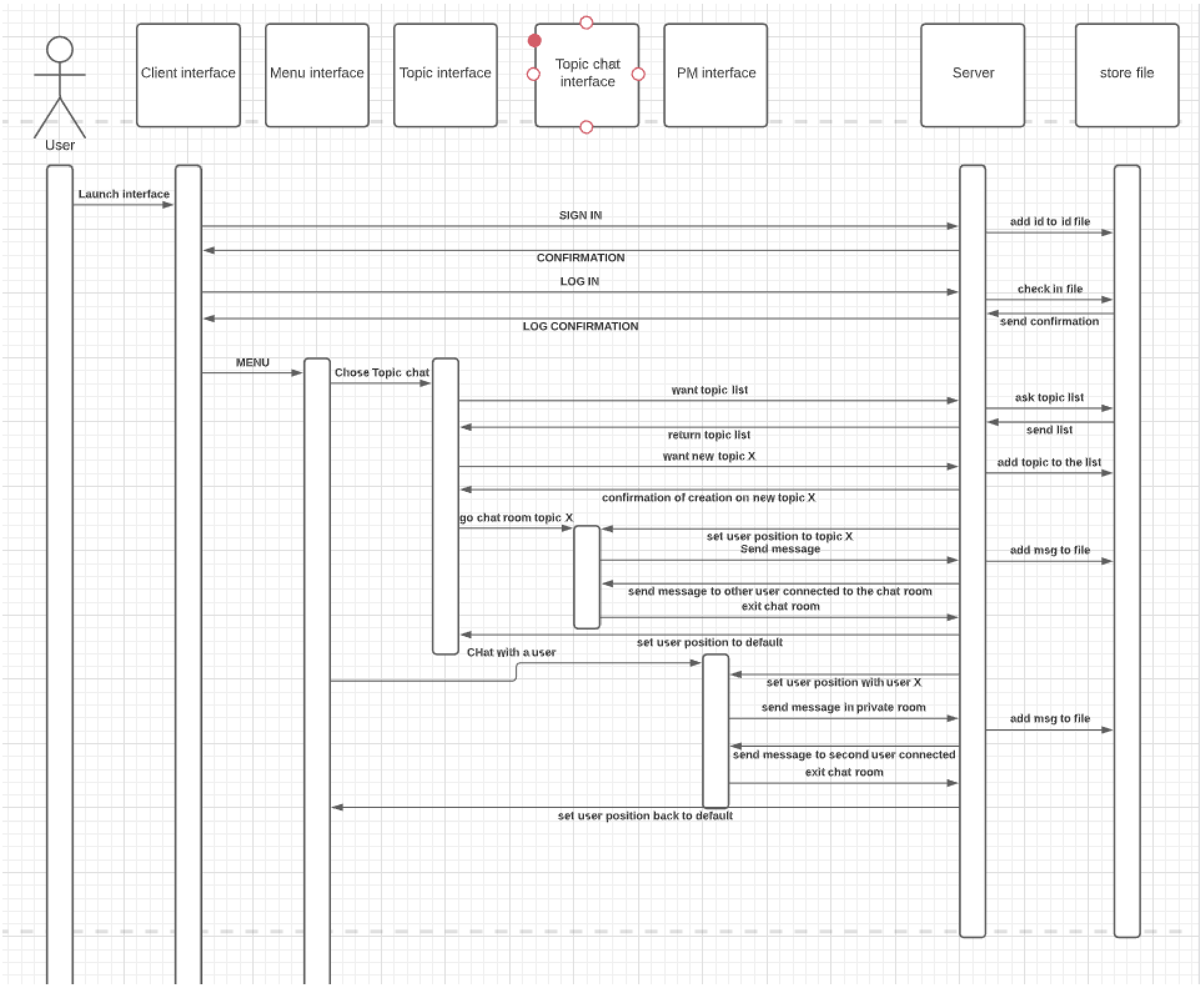


Figure 2- UML Sequence diagram of the ChatApp

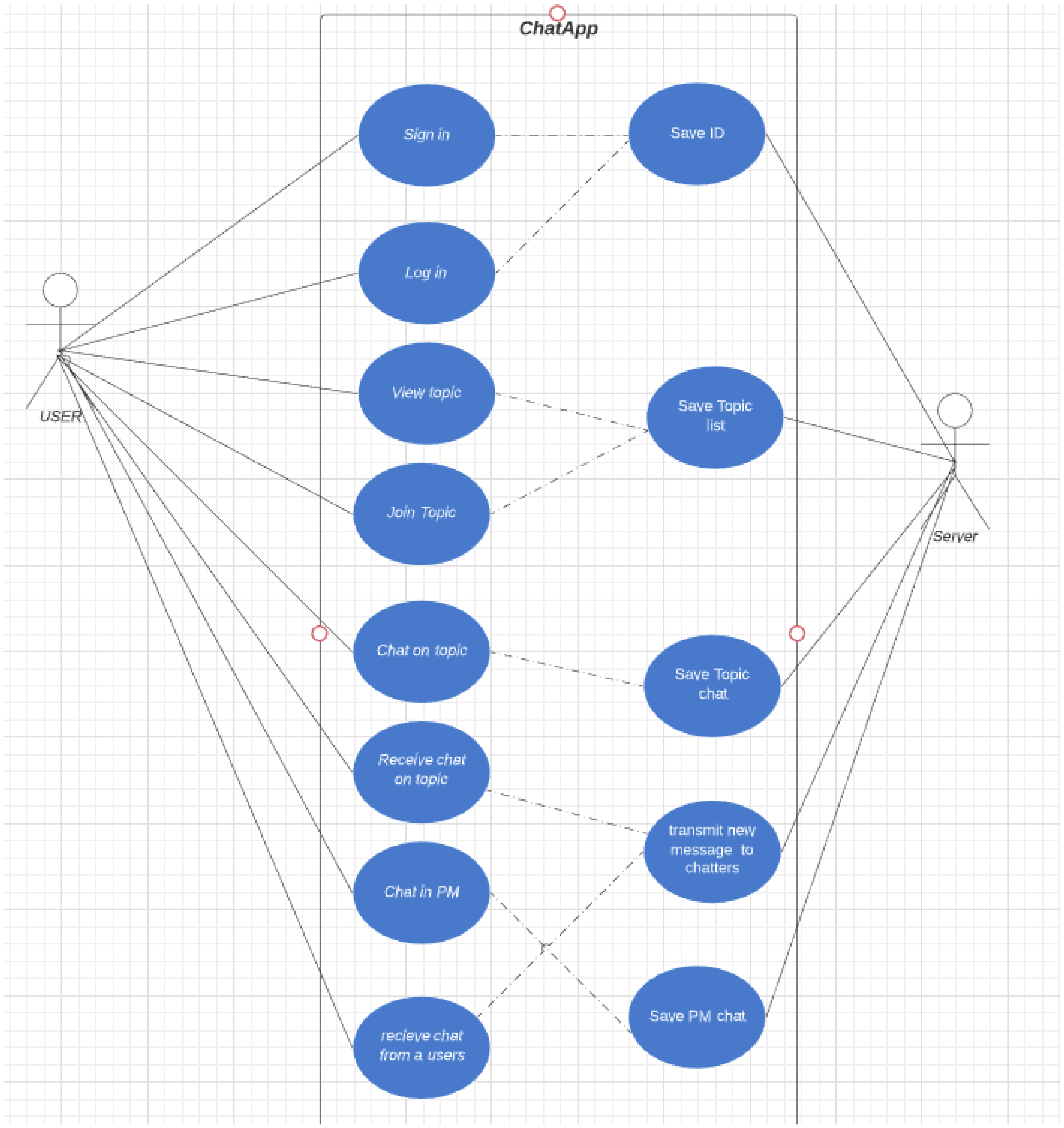
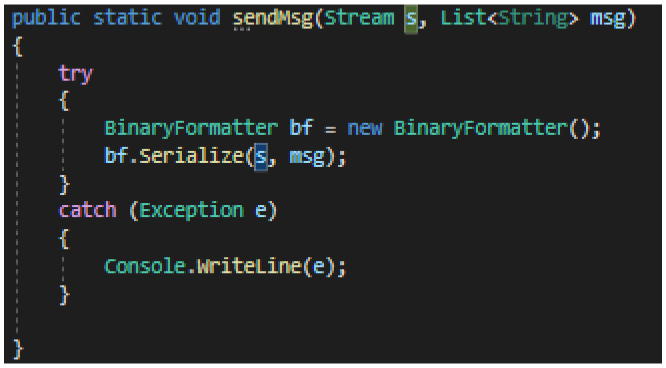


Figure 3 - UML Use Case Diagram for ChatApp

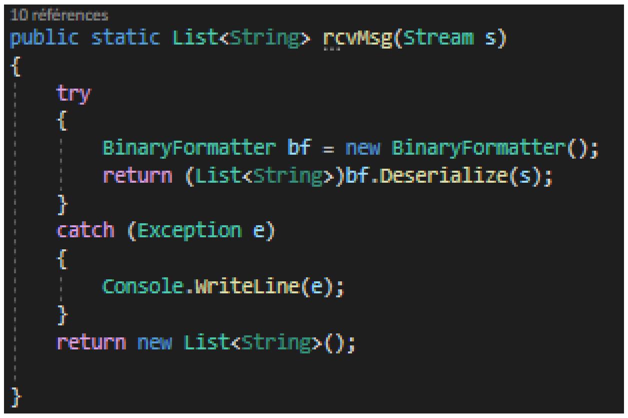
III. DESIGN AND IMPLEMENTATION

→ The most important part of this project is the **communication** capacity between the different parties, therefore between the server and the connected users. So we will therefore first talk about the part (VisualStudio project) "Communcation". This part presents only one C# class named "Message". And this class contains 2 objects, a first named "sendMsg" which will manage the format of the messages sent and a second "rcvMsg" for (rcv means "receive") which will manage the format of the messages received.



```
public static void sendMsg(Stream s, List<String> msg)
{
    try
    {
        BinaryFormatter bf = new BinaryFormatter();
        bf.Serialize(s, msg);
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }

}
```

The "sendMsg" objects take 2 arguments, an S stream (this is the stream from the TcpClient) and a list which will regroup the elements of the message. After having retrieved this it will serialize our list and send it to its recipient according to the given stream.



```
10 références
public static List<String> rcvMsg(Stream s)
{
    try
    {
        BinaryFormatter bf = new BinaryFormatter();
        return (List<String>)bf.Deserialize(s);
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
    return new List<String>();

}
```

The "rcvMsg" objects take only one argument, the stream S for the receiver, He check the messages sent on his stream, and when he receives something he saves it in a list, and then returns the created list.

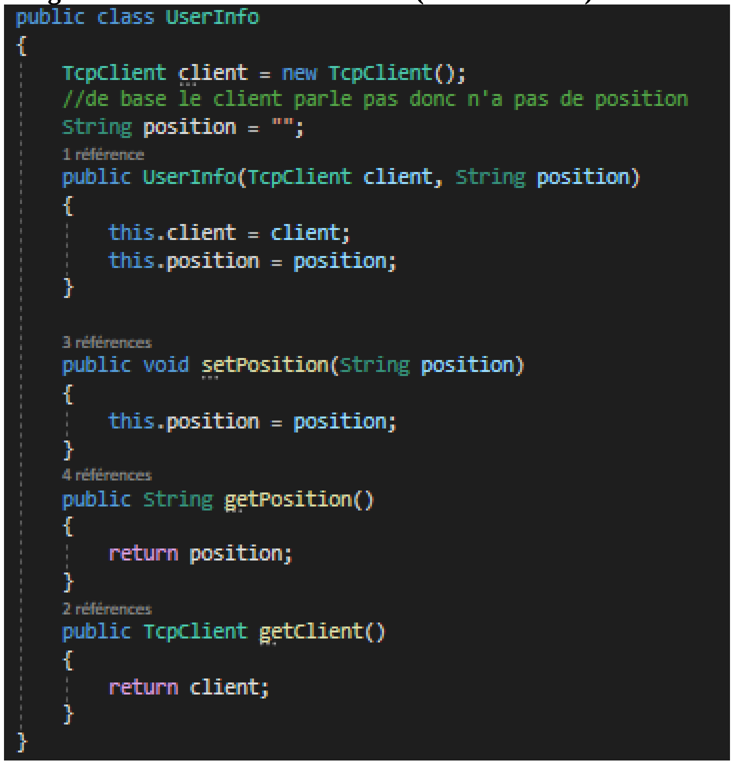
Here is how the system for sending and receiving messages works.

→ Now we can look at how our server works **, ConsoleAppServer**. The Server is also the basis of everything because it will manage all our functionality but also all our backup files. Our server is based on TCP-IP technology.

Starting the server : (server class)

It needs an Ip address and a port to listen to messages from users. For the door we choose it when launching the "server" class from the "program" class, it will be 123. But for the ip, I had to perform a "Get Local IP" function and no matter where we start the server, the function will retrieve the local address. Once the server is started, it opens and accepts all connection requests from new users (the user interface by the user itself (not yet password and username). Every time a new user interface connects to the server, it will be noted in the server console. And the server will then launch a thread grouping the functions of the server for this particular user, each user has his own thread.

Management of the connected Users : (UserInfo class)



```
public class UserInfo
{
    TcpClient client = new TcpClient();
    //de base le client parle pas donc n'a pas de position
    String position = "";
    1 référence
    public UserInfo(TcpClient client, String position)
    {
        this.client = client;
        this.position = position;
    }

    3 références
    public void setPosition(String position)
    {
        this.position = position;
    }
    4 références
    public String getPosition()
    {
        return position;
    }
    2 références
    public TcpClient getClient()
    {
        return client;
    }
}
```

This class will handle user interfaces that have successfully logged in with valid credentials. Each user will have an associated "UserInfo" object, in this one we record its TcpClient and its position. The position is just a character string, where we note the position of the user, for example his topic or his private chat.

The main variables : (Receiver class)

The semaphores of the various functions of the app, they are used not to overload the server and to let it save the data modified by one user before processing that of the next.

`TcpClient comm ,` this is info from the connected user interface.

`List<Identifiant> IdentifiantList,` this is a liste from identifiant object (identifiant + password)

`List<string> TopicList,` this is a list of string, just the different topic name

`Dictionary<String, UserInfo> users_list,` this is a dictionary in the server class, it will create a dictionary with a numeric key and the different users, to simplify the choice during private messages, you will understand later

`Identifiant` objects group together a username and password.

The sub- elementary object and function : (Receiver class)

Before going to the function of our chat App we must see under functions / objects, they are the bases of the essential functions.

`serializeID()` This function is used to serialize all `Identifiant` saved in the file `Identifiant.dat`

`deserializeID()` This function is used to recover all the serialized `Identifiant` from the file `Identifiant.dat` `serializeTOPIC()` This function is used to serialize the `TopicList` saved in the file `Topic.dat`

`deserializeTOPIC()` This function is used to recover the serialized `TopicList` from the file `Topic.dat`

`SerializeChatTOPIC(List<String> TopicChat, String nameTopic)` This function is used to serialize the chat of a specific topic in a file named `nameTopic.dat`

`List<String> DEserializeChatTOPIC(String nameTopic)` This function is used to recover the serialized chat from a specific topic chat room.

`SerializePM(List<String> conversation, String nameUser1, String nameUser2)` This function is used to serialize the chat of a specific private message chat room between user1 and user 2 in a file named `nameUser1andnameUser2.dat`

`List<String> deSerializePM(String nameUser1, String nameUser2)` This function is used to recover the serialized private message from a specific private chat room..

The main function of the chat App : (`doOperation()`**)**

The doOp function loops while waiting for a new message from the user it is following. the messages received are the lists, the list is in the following form, in the first box, the type of action requested is entered and then the rest of the message if necessary. When the server receives an instruction it notes "instructions received" in the server console. Then he looks at the request in the first box of the list, A switch is then launched to compare the request received with a possible function.

The possible cases are:

- `"logout"`, this is for login out one client from the server and not kill the server but it doesn't work.
- `"login"`, this is for login the client, the server retrieves from the message list, the identifier and the password sent, it will then check if they are valid in the deserializer `IdentifiantList`. if they are good, then the server sends a validation message and adds this user to the user list (dictionary with username as key and userinfo as element) logged on to the server. If they are not good, the server will reject the request and send a rejection message to the client. Then starts a count of 3 attempts for this client tcp. If he makes 3 false attempts then this client is banned.
- `"signin"`, this is for add a new `Identifiant` chosen by the user in the `IdentifiantList`, after the server serializes the new list. the server then sends a confirmation message to the client.
- `"listTopic"`, this is for sending to the client the list of the topic saved into the file. The server deserialize the file and put in a list all the different topic and send bac this list to the client.
- `"addTopic"`, this is for adding a new topic to the topic list, the server deserialize the TopicList, then add the new topic, check if it doesn't already exist and after send back the confirmation to the user.
- `"joinTopic"`, this is for placing a user in a specific topic channel, the server browse in the user list dictionary to find the good username key, it must be equal to the identifier of this tcp window saved as a variable. We will then, in the userInfo, position this user at the position: username + "enter the room" + topic name received with the `.setPosition() function.`
- `"readTopic"`, this is for sending the saved message to the user, the server deserialized the .dat file in question, put all the element in a list and send the list back to the client.
- `"msgTopic"`, this is for adding a message to the saved message, the server recup the list send by the user, she's composed, in place 1, the topic, in Place 2, the message. The server deserialized the saved message of this topic, add this new message and serialized the new list.

After that, the server will browse the user dictionary and send the message to users connected to the same topic

- `"listPrivate"`, this is for sending back to the user , the list of all the other user connected on the server.
- `"joinPrivate"`, this is for placing a user in a specific private message channel, the server browse in the user list dictionary to find the good username key, it must be equal to the identifier of this tcp window saved as a variable. We will then, in the userInfo, position this user at the position: `username + "and" + name of the front user` with the `.setPosition() function.`

- `"readPrivate"`, this is for sending the saved message of this PM conversation to the user, the server deserialized the .dat file in question, put all the element in a list and send the list back to the client.

- `"msgPrivate"`, this is for adding a message to the saved message, the server recup the list send by the user, she's composed, in place 1, the topic, in Place 2, the message. The server deserialized the saved message of this conversation, add this new message and serialized the new list.
  After that, the server will browse the user dictionary and send the message to the front user if this one is connected to the private chat room, if not , the front user will see the new msg when he will connect.

- `"/Exit"`, this is for changing the user position to default when this one is leaving a topic chat room or a private chat room.

→Now we will see how the user interface works**, ConsoleAppClient**

This interface is very simple, it results in a sequence of menus.

- The Start() menu

When starting, the start function is launched, it displays the connection menu, 2 choices are then possible, to connect or to register.
Each choice will launch a different function:
  - The `LogIn()` function, this function connect the client to the server, after she ask to the user the id and password, and it sends the server a "login" request, if the request is accepted, the program open the MenuApp(), if not, the program loops for 3 times.
  - The `SignIn()` function is running , this function connect the client to the server , after she ask to the user the id and password, and it sends the server a "signin" request, if the request is accepted, the program send back user to Start() menu to log in.
  - The `LogOut(TcpClient client)` function,

- The MenuApp() menu

This menu offer 3 possibility, chat on topic, by private message or go back.
Each choice will launch a different function:
  - The `Topic(TcpClient client)` function, open the topic choice menu
  - The `PrivateMessage(TcpClient client)` function, open the PM choice menu

- The Topic() menu

This menu offer 3 possibility, chat on topic, add topic or go back. To have the update topic list, the client ask to server the topic list and after, the interface show the topic list.
Each choice will launch a different function:
  - The `NewTopic(TcpClient client)` function, client interface send a `"addTopic"` request to the server, depending on the response from the server, the interface displays the success or failure of the creation.
  - The `TopicMessage(TcpClient client, String nameTopic)` function, open the menu for chatting on a specific topic and send a `"joinTopic"` instruction to the server to change the position of user a be aware of new message.

- The `TopicMessage(TcpClient client, String nameTopic)` function, on this menu, the user can type new message or /exit to go back. Before the program has recup the saved chat by sending a `"readTopic"` request to the server
  - The `UpdateTopicMessage(TcpClient client, String nameTopic)` function, is in a thread launch on the beginning of topic function , this function loops on itself, it waits for new message, two possibilities are possible, either the message is classic then it displays it, it is a message from other users. Either the msg coming from the server announces that the user has asked to leave the server, then the thread stops and the topic menu loop too, we are leaving to the main menu
  - Otherwise the user can type messages, they will be sent to the server, the server will send them to different users

- The PrivateMessage() menu

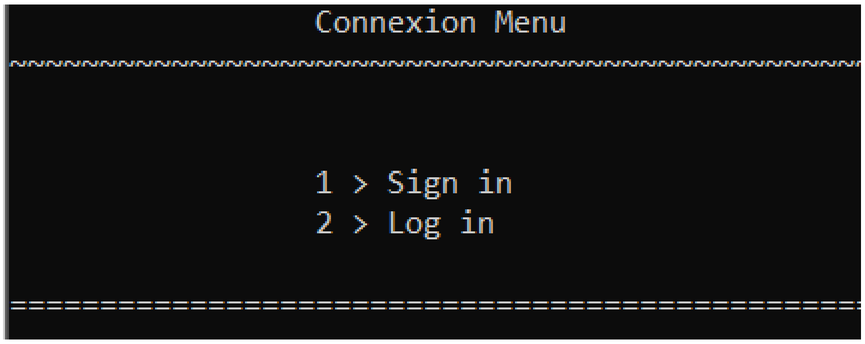
This menu offer 3 possibility, chat with a connected user or go back. To have the user list, the client ask `"listPrivate"` to server and after, the interface show the user list.
Each choice will launch a different function:

  - The `writePrivateMessage(TcpClient client, String ChatterUsername)` function, open the private message chat room. This function send to the server the new position of the user.

- The `writePrivateMessage(TcpClient client, String ChatterUsername)` function, on this menu, the user can type new message or /exit to go back. Before the program has recup the saved chat by sending a `" readPrivate "` request to the server.

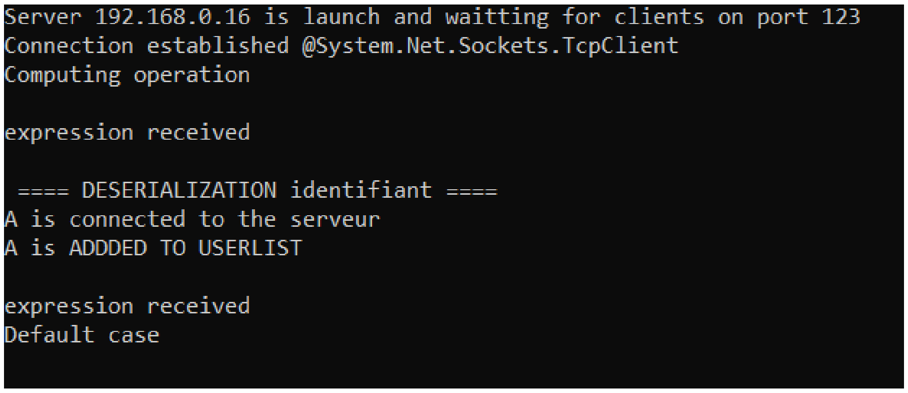
  - The `UpdatePMessage(TcpClient client, String ChatterUsername)` function, is in a thread launch on the beginning of `writePrivateMessage` function , this function loops on itself, it waits for new message, two possibilities are possible, either the message is classic then it displays it, it is a message from the other user. Either the msg coming from the server announces that the user has asked to leave the server, then the thread stops and the topic `writePrivateMessage` loop too, we are leaving to the main menu.
  - Otherwise the user can type messages, they will be sent to the server, the server will send them to front user.

IV. Screenshots from the app



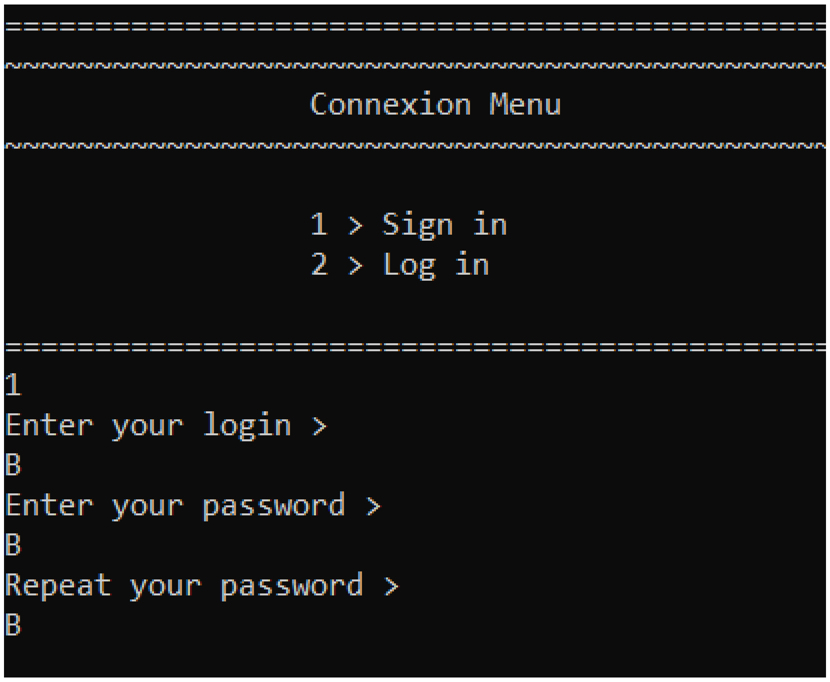
```
                    Connexion Menu
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



                    1 > Sign in
                    2 > Log in


===========================================================
```

Figure 4 - Client interface login menu



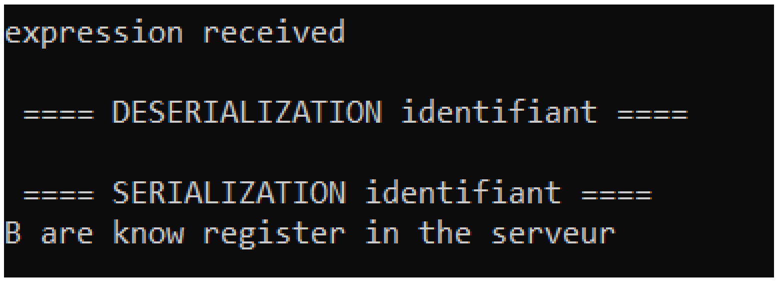
```
Server 192.168.0.16 is launch and waitting for clients on port 123
Connection established @System.Net.Sockets.TcpClient
Computing operation

expression received

 ==== DESERIALIZATION identifiant ====
A is connected to the serveur
A is ADDDED TO USERLIST

expression received
Default case
```

Figure 5 - Server window



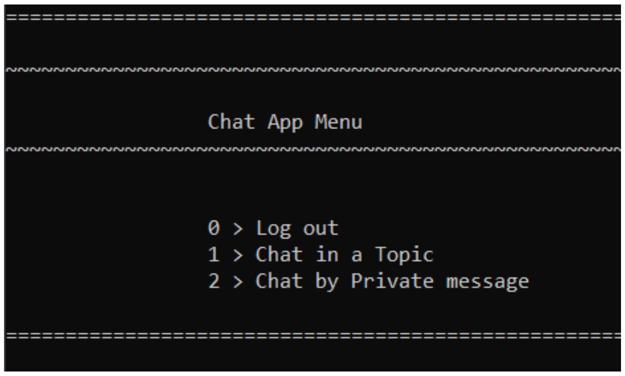
```
===========================================================
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                    Connexion Menu
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


                    1 > Sign in
                    2 > Log in


===========================================================
1
Enter your login >
B
Enter your password >
B
Repeat your password >
B
```

Figure 6 – Client interface, new user registration (Console Version)



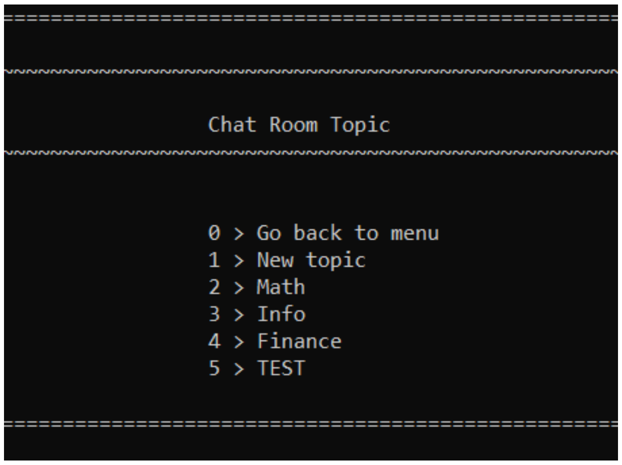
```
expression received

 ==== DESERIALIZATION identifiant ====

 ==== SERIALIZATION identifiant ====
B are know register in the serveur
```

Figure 7- Server Interface new user registration



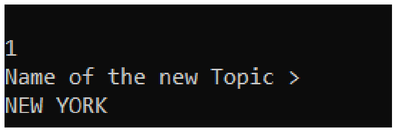
```
======================================================
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                 Chat App Menu
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


                 0 > Log out
                 1 > Chat in a Topic
                 2 > Chat by Private message

======================================================
```

Figure 8 - Client interface , main menu (Console Version)



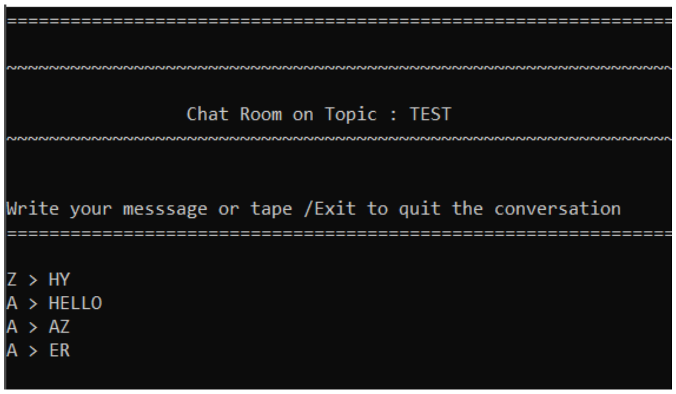
```
======================================================
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                 Chat Room Topic
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


                 0 > Go back to menu
                 1 > New topic
                 2 > Math
                 3 > Info
                 4 > Finance
                 5 > TEST

======================================================
```

Figure 9 - Client interface, Topic menu (Console Version)



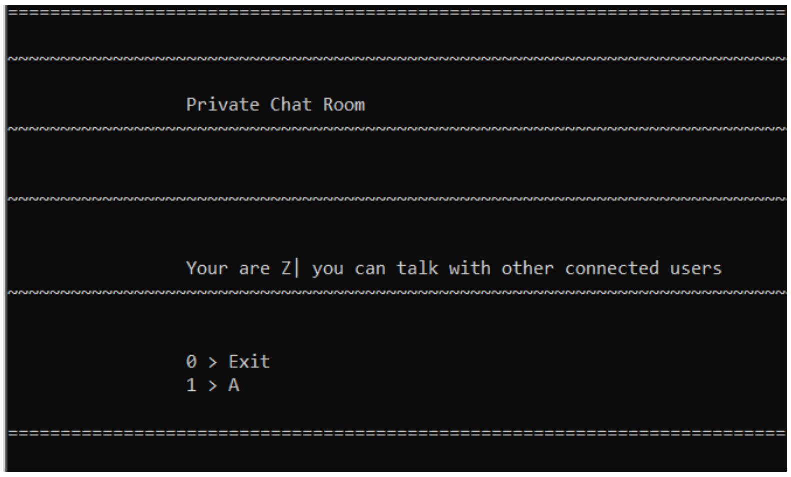
```
1
Name of the new Topic >
NEW YORK
```

Figure 10 - Client interface, new topic registration (Console Version)



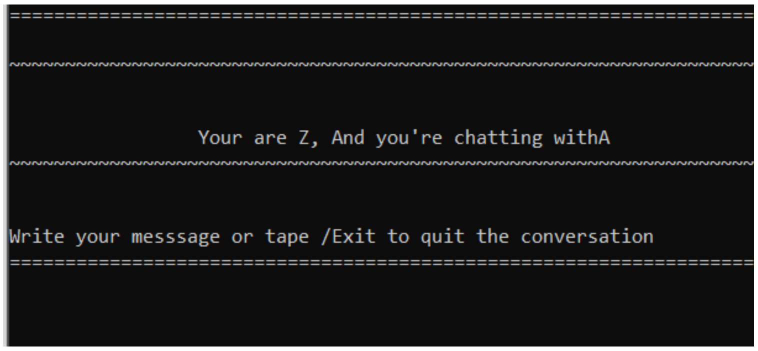
```
===============================================================
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                 Chat Room on Topic : TEST
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Write your messsage or tape /Exit to quit the conversation
===============================================================

Z > HY
A > HELLO
A > AZ
A > ER
```

Figure 11 - Client interface, Chat Room for a specific topic (Console Version)



```
==========================================================================
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

                 Private Chat Room
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


                 Your are Z| you can talk with other connected users
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


                 0 > Exit
                 1 > A

==========================================================================
```

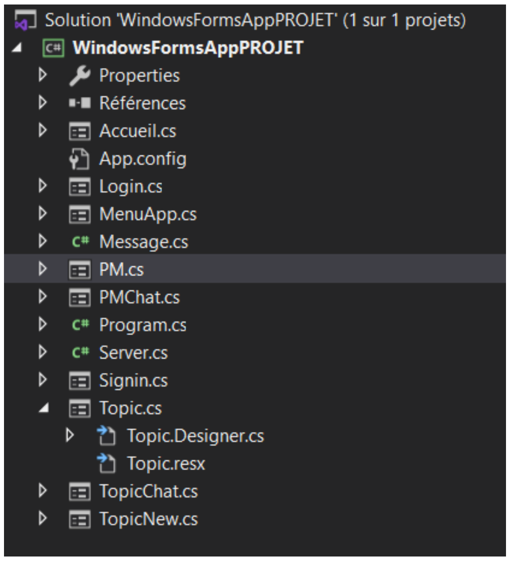
Figure 12 - Client interface, private Chat Room selection menu (Console Version)



```
======================================================================
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


                  Your are Z, And you're chatting withA
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Write your messsage or tape /Exit to quit the conversation
======================================================================
```

Figure 13 - Client interface, Private chat Room (Console Version)

V. GRAPHICAL USER INTERFACE (G.U.I.)



```
Solution 'WindowsFormsAppPROJET' (1 sur 1 projets)
  WindowsFormsAppPROJET
    Properties
    Références
    Accueil.cs
    App.config
    Login.cs
    MenuApp.cs
    Message.cs
    PM.cs
    PMChat.cs
    Program.cs
    Server.cs
    Signin.cs
    Topic.cs
      Topic.Designer.cs
      Topic.resx
    TopicChat.cs
    TopicNew.cs
```

I'm not sure why but after a change in my files and my computer, I can't get the GUI to work properly.
Attached are screenshots of the various windows, and the code is also available but probably will not work.

Even this possibly does not work, the principle is very simple, we juggle from window to window by sending the server the same commands as for the project without GUI. It will only be necessary to pass the client TCP data from window to window to properly keep the client signature.

you will then have to display the server's responses on the windows and no longer in the console.

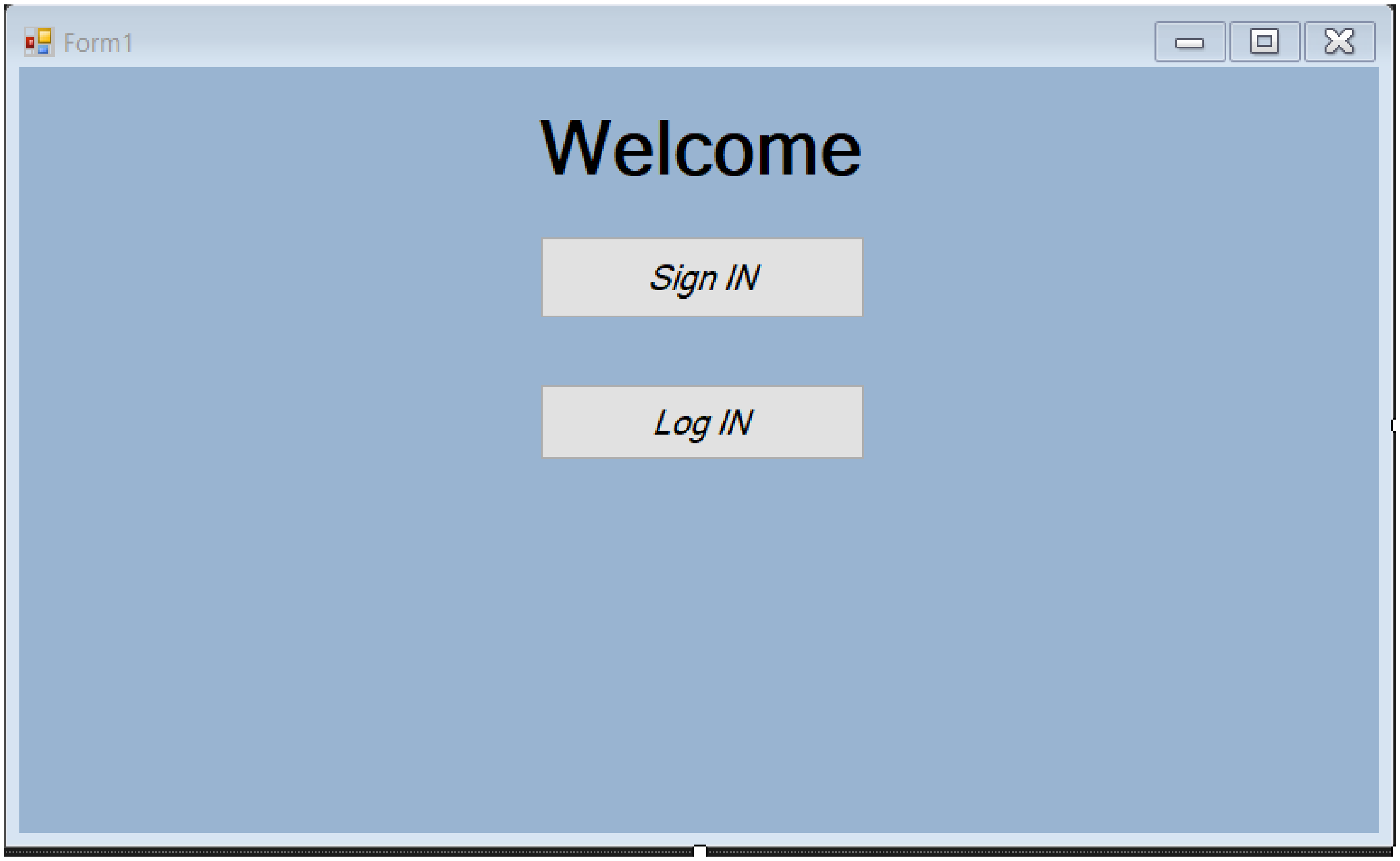


Figure 14 - GUI Form welcome

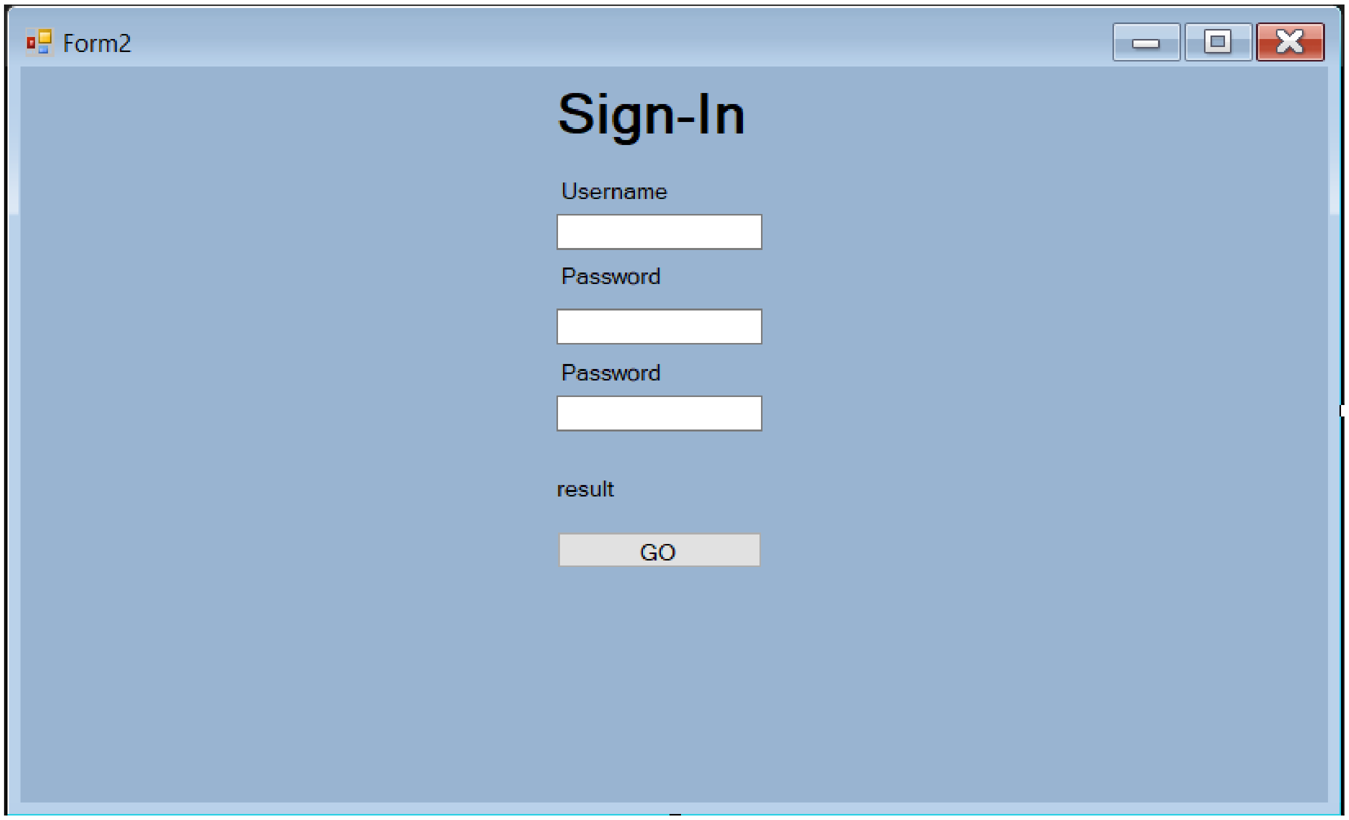


Figure 15 - GUI Form Sign new user on the server

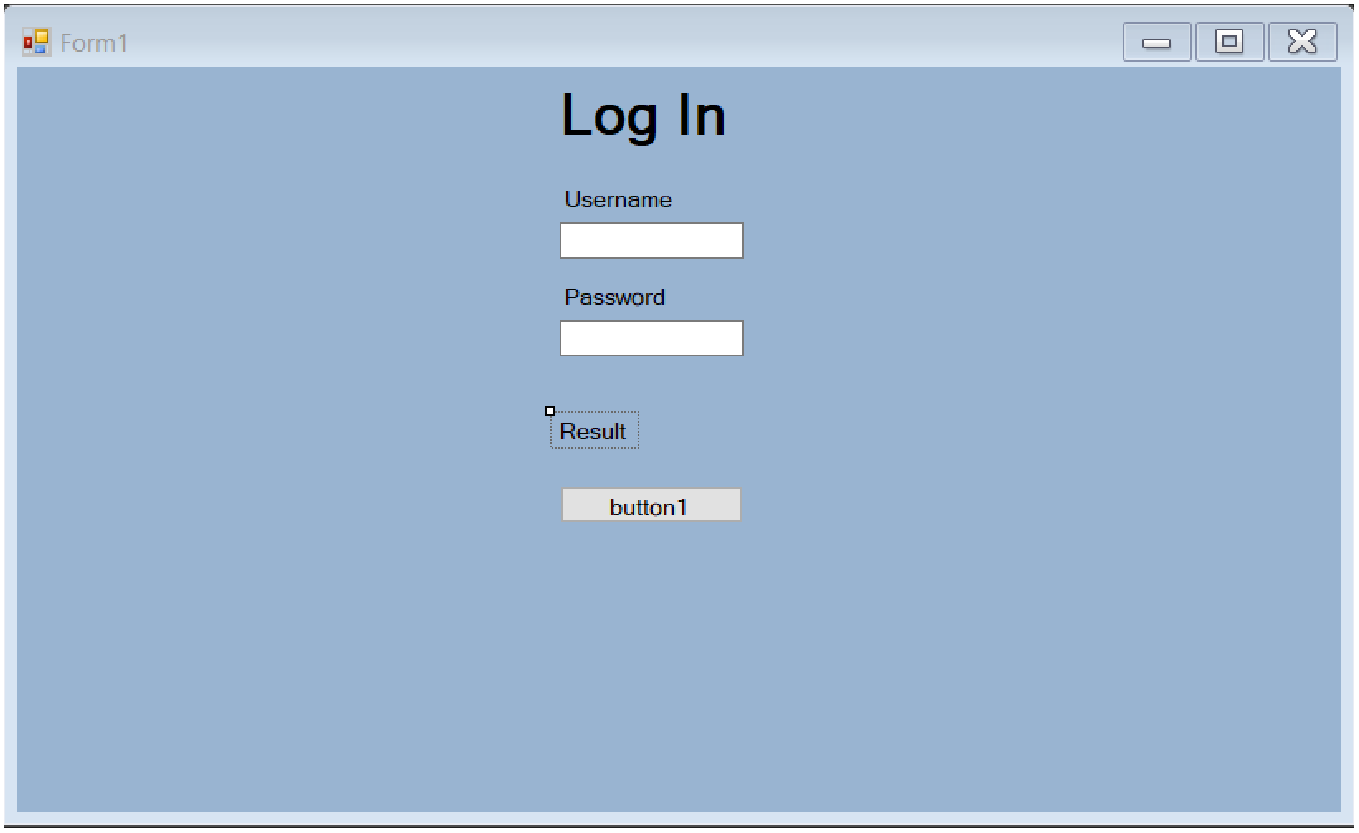


Figure 16 - GUI Form Log to the server

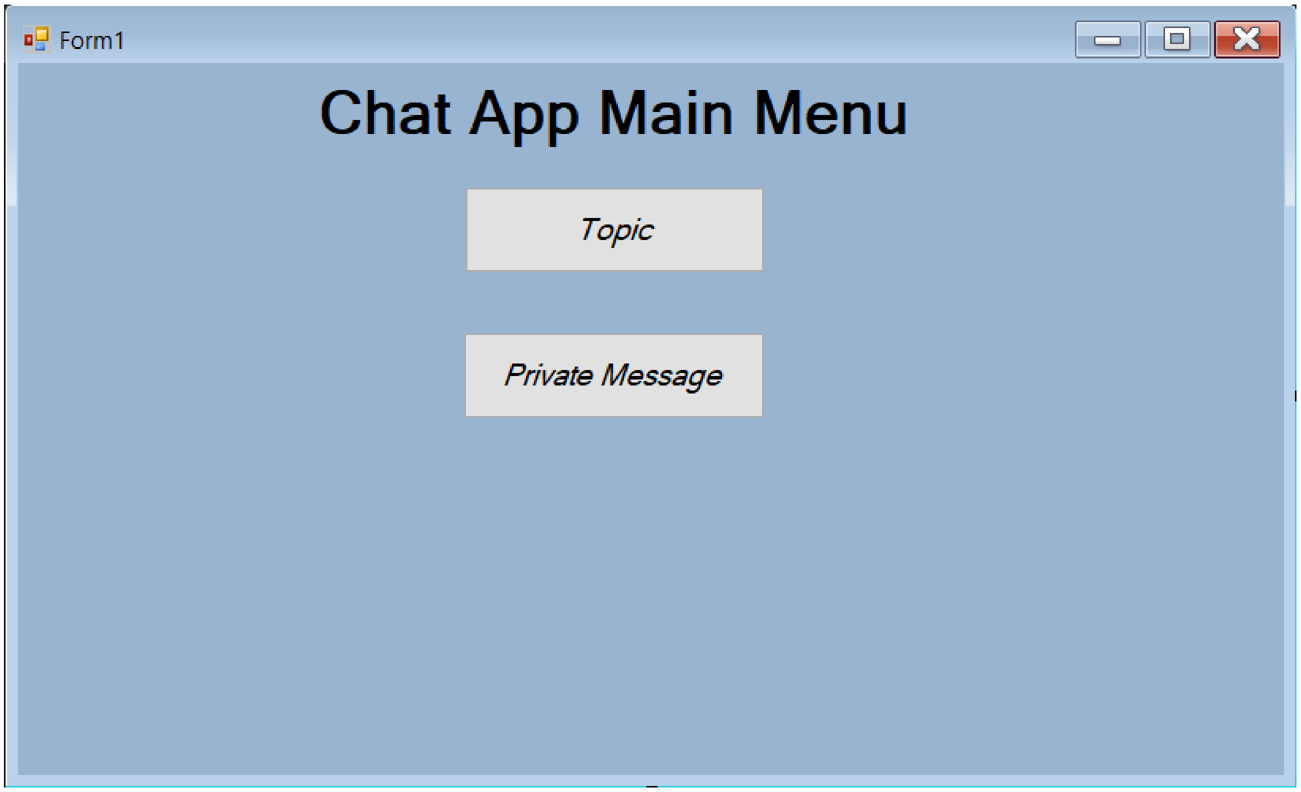


Figure 17 - GUI Form Main menu, topic or PM

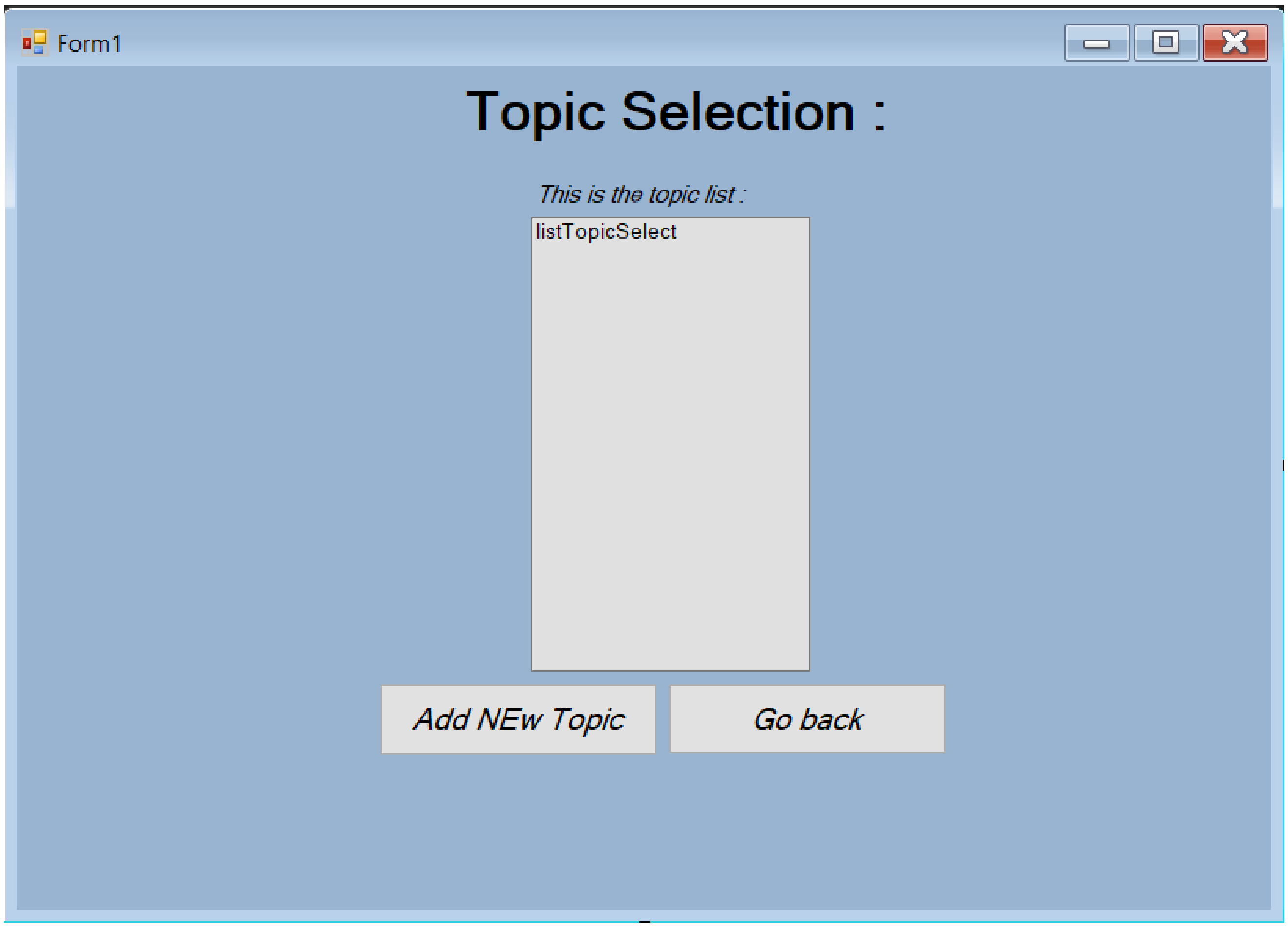


Figure 18 - GUI Form topic selection

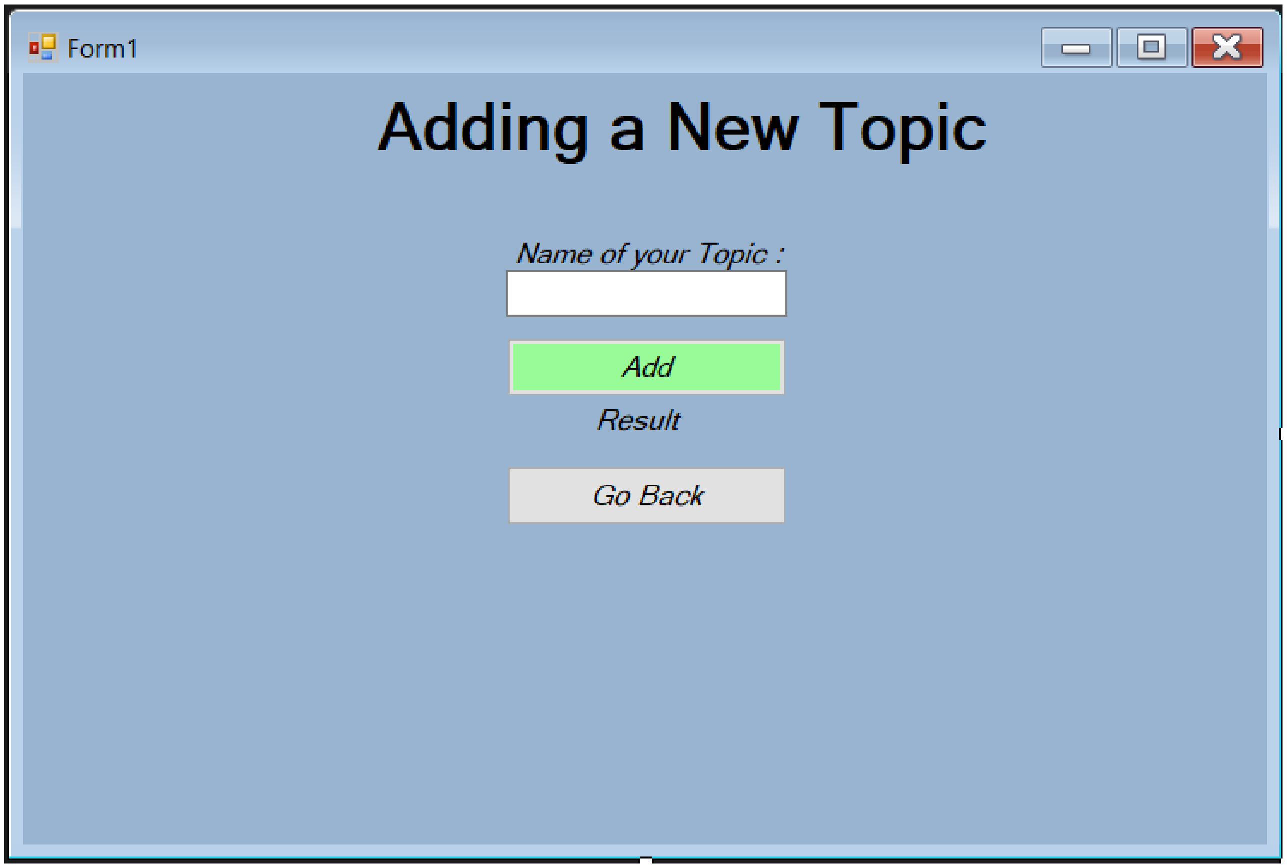


Figure 19 - GUI Form new topic creation

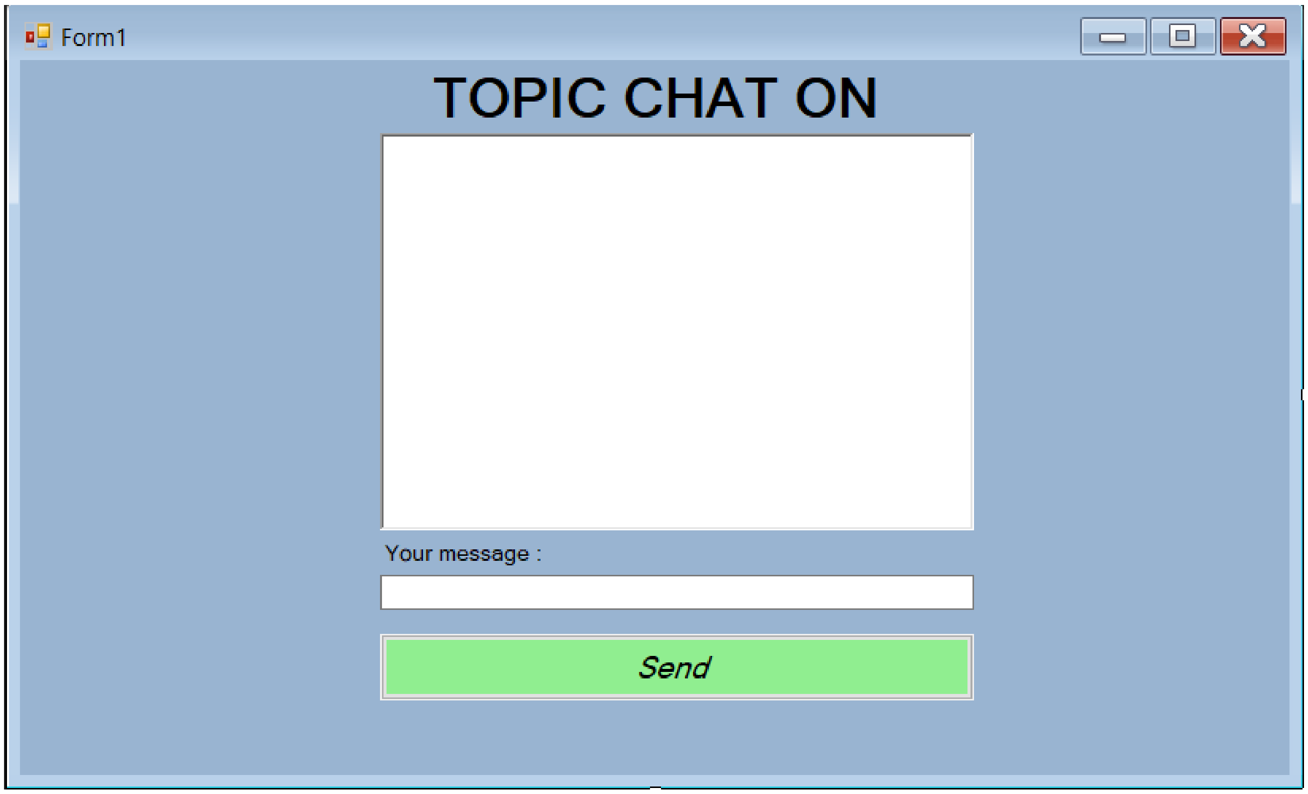


Figure 20 - GUI Form Topic Chatroom

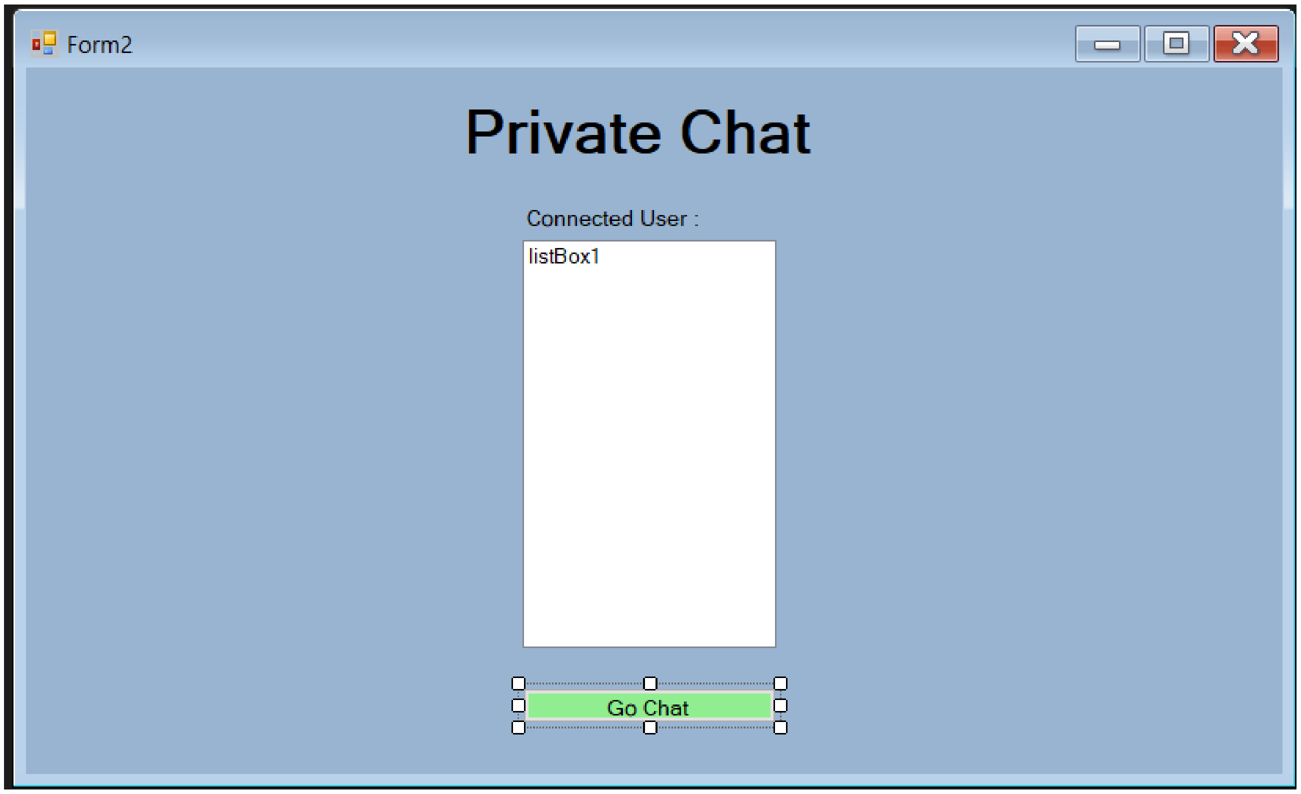


Figure 21 - GUI Form Selection of user for PM

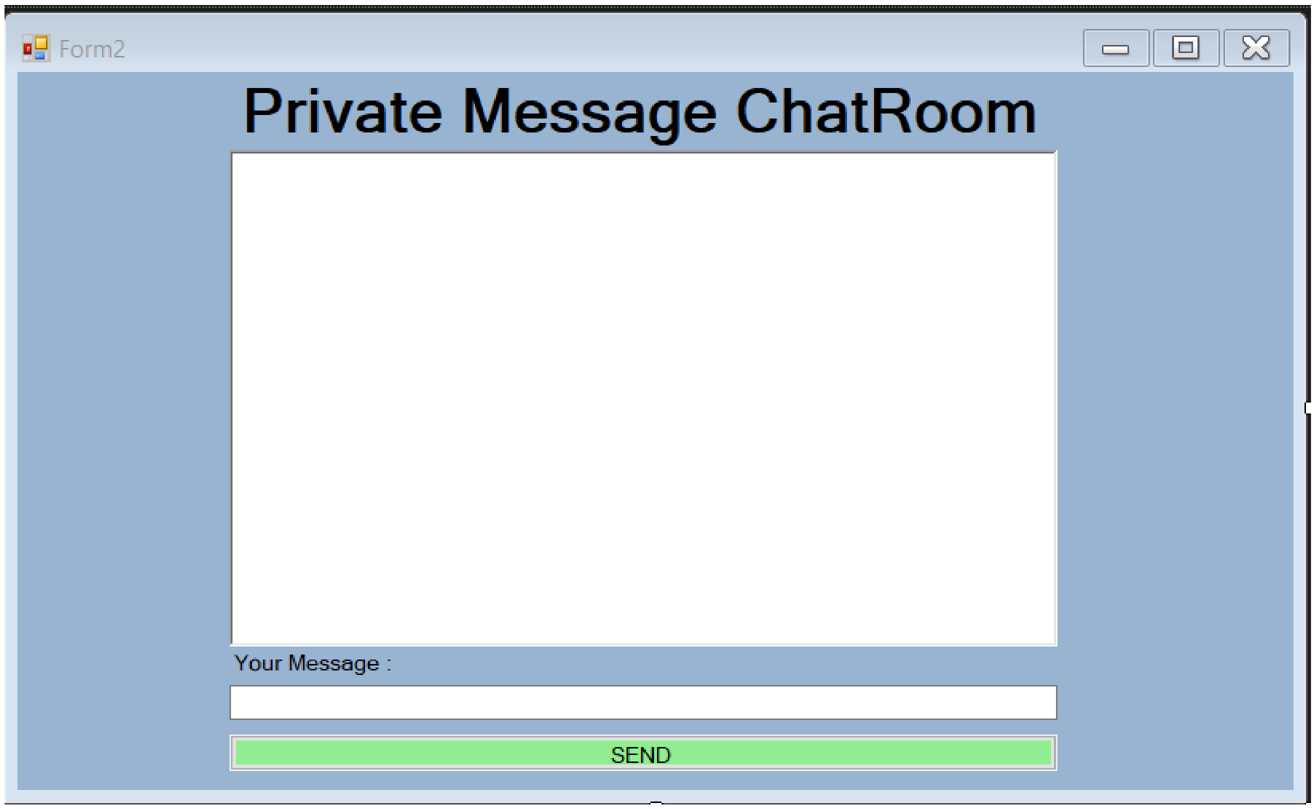


Figure 22 - GUI Form PM chat room

[1]   .