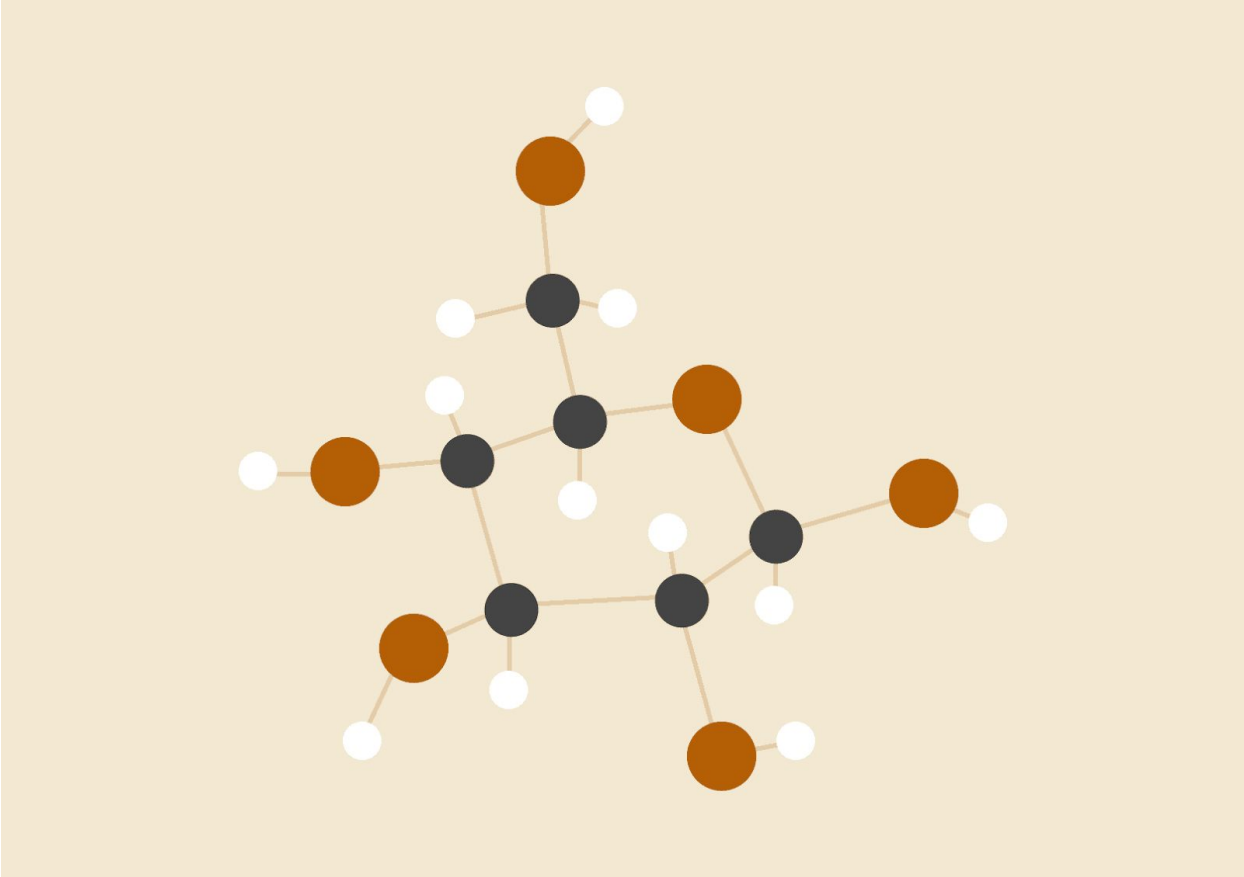
C# Project report



Amine Ben Slama

07/01/2021

M1 - Software Engineering - 20170939

TABLE OF CONTENT

Introduction

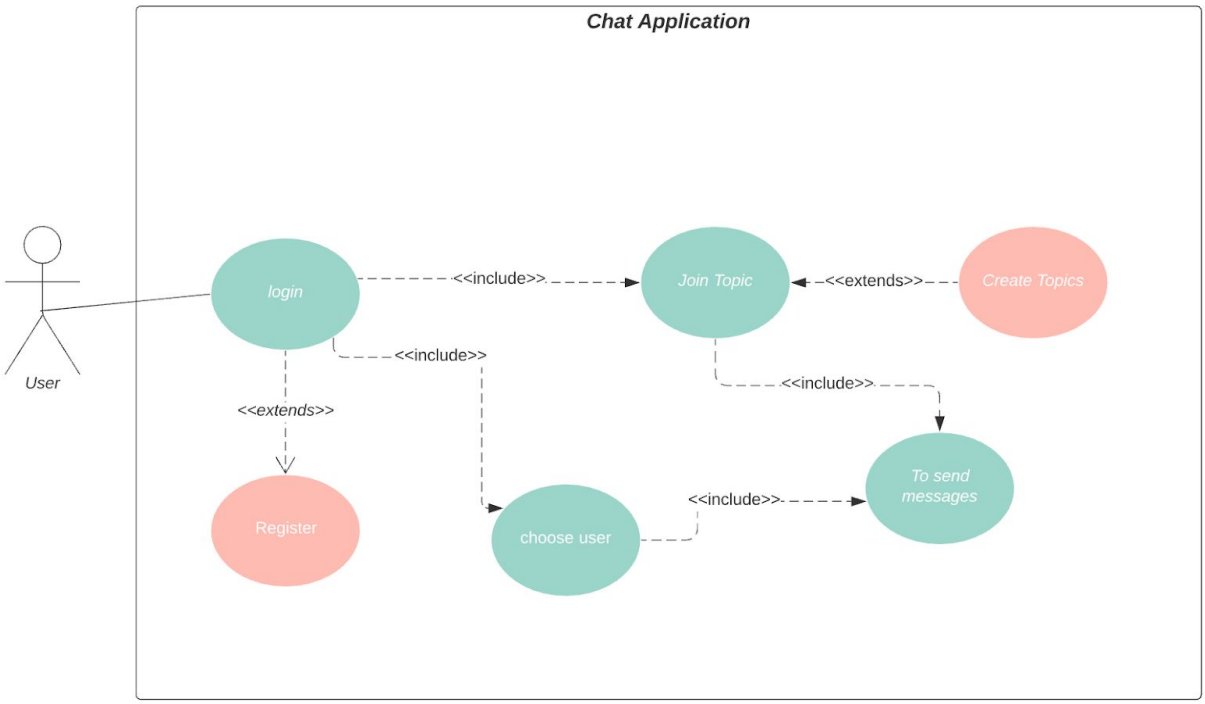
In this project, we were asked to develop a network based multithreaded client-server chat application. We had to implement seven features which are the following :

- register/login
- create Topics
- List all the topics created
- Send messages in a topic
- Send messages between users

The application is in console form with all the features except the one for sending messages between users.

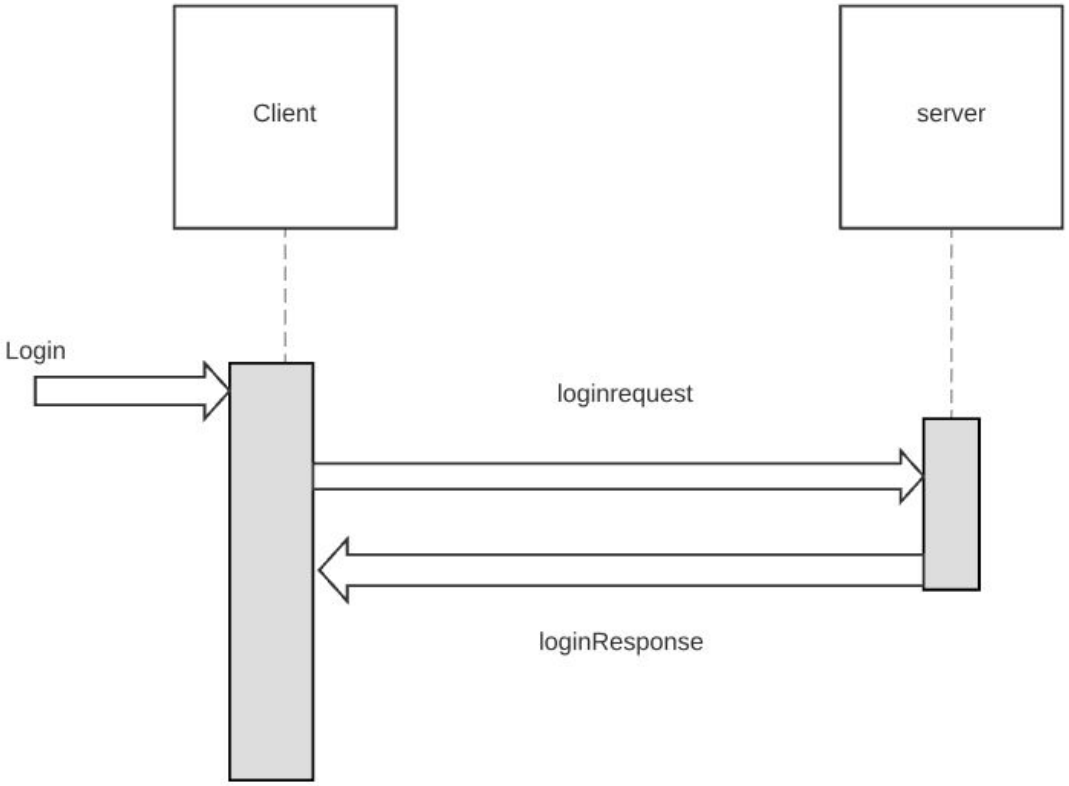
I didn't implemented the last feature which is about sending messages between users.

Use case diagram



Sequence Diagram

Each request implements the Message interface, they all work on the following model :



The server side performs different processing depending on the type of message it receives, and returns the corresponding response.

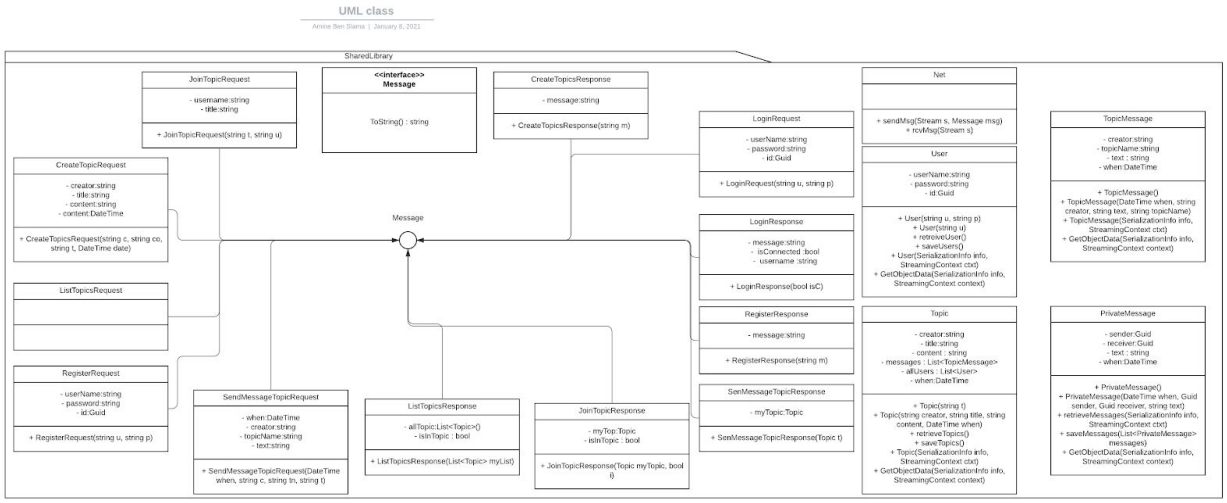
The server side processes the request as follows :

```
if (type == typeof(LoginRequest))
{

    User myUser = new User(((LoginRequest)msg).UserName, ((LoginRequest)msg).Password);
    List<User> allUser = User.retrieveUsers();
    bool safe = false;
    bool isConnected = false;


    foreach (User u in allUser)
    {
        if (u.Username == myUser.Username && u.Password == myUser.Password)
        {
            isConnected = true;
            LoginResponse myResp = new LoginResponse(isConnected);
            myResp.Username = myUser.Username;

            Net.sendMsg(ns, myResp);
        }
    }
    if (!isConnected) {
        Net.sendMsg(ns, new LoginResponse(isConnected));
    }


}
```

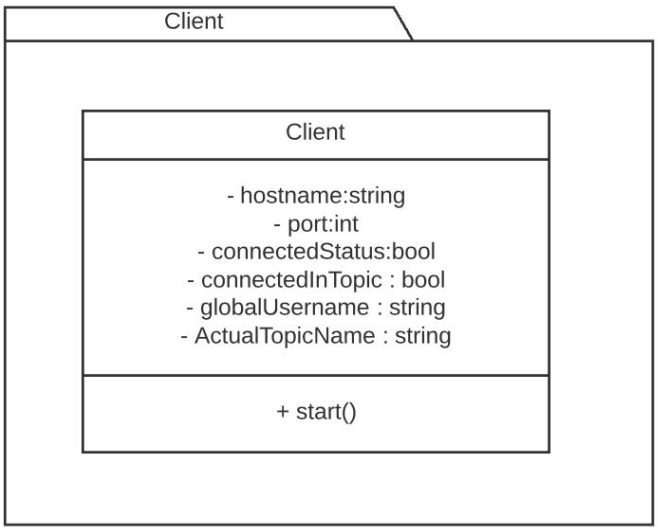
Class Diagram

The application contains three packages which are :

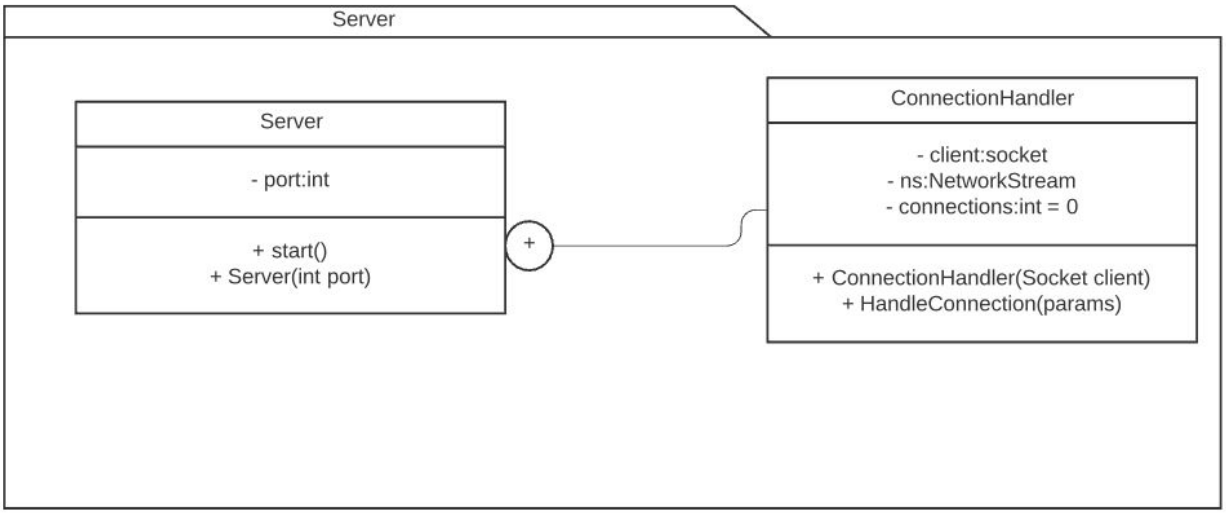
- The shared library package



- The client package



- The server package



For the server class, I implemented an inner class ConnectionHandler.

The main purpose of this project was to create a multi-client server, for this, I used a thread class. We can use a thread to allow a server to maintain more than one session with multiple clients at the same time. In order to be able to set up this system I have divided my server class into two parts, a connection acceptor and a second one which manages them.

The part that accepts connections is the main application that runs in an infinite loop waiting for a client connection.

Each time a client connects; the “connection acceptor” accepts the client and then do the following:

- creates an instance of the “connection handler”
- creates a separate thread, associating it with the connection handler instance and
- starts the thread.

The connection handler instance is responsible for the communication between the client and the server.

The connection handler is an inner class of the server class, which has a method that matches the signature of the ThreadStart delegate:

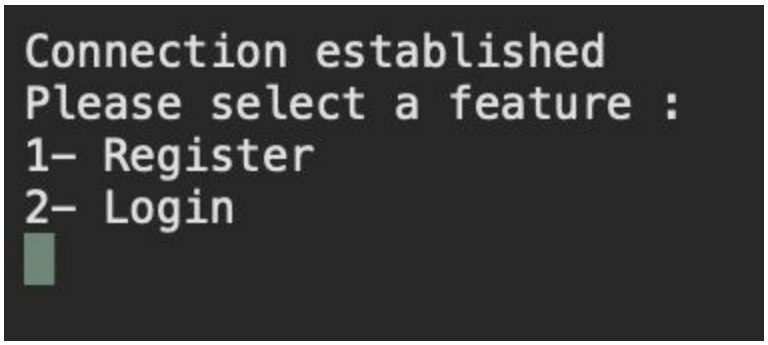
void HandleConnection()

```
public void start()
{
    Socket server = new Socket(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp);
    IPEndPoint endpoint = new IPEndPoint(IPAddress.Any, port); // Contains host information
    server.Bind(endpoint);

    server.Listen(10); // Up to 10 users in a queue
    Console.WriteLine("Waiting for clients on port " + port);

    while (true)
    {
        try
        {
            Socket client = server.Accept();
            ConnectionHandler handler = new ConnectionHandler(client);

            Thread thread = new Thread(new ThreadStart(handler.HandleConnection));
            thread.Start();
        }
        catch (Exception)
        {
            Console.WriteLine("Connection failed on port " + port);
        }
    }
}
```

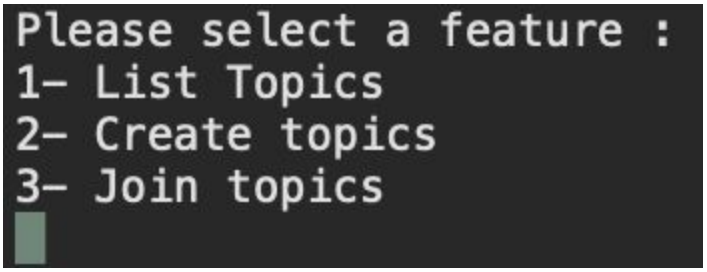
```
public void HandleConnection()
{
    try
    {
        ns = new NetworkStream(client);
        connections++;
        Console.WriteLine("New client accepted: {0} active connections", connections);

        Console.WriteLine("Welcome to my server");


        while (true)
        {
            Message msg = Net.rcvMsg(ns);

            Type type = msg.GetType();

            Console.WriteLine(msg.GetType());

            if (type == typeof(LoginRequest))
            {

                User myUser = new User(((LoginRequest)msg).UserName, ((LoginRequest)msg).Password);
                List<User> allUser = User.retrieveUsers();
                bool safe = false;
                bool isConnected = false;


                foreach (User u in allUser)
                {
                    if (u.Username == myUser.Username && u.Password == myUser.Password)
                    {
                        isConnected = true;
                        LoginResponse myResp = new LoginResponse(isConnected);
                        myResp.Username = myUser.Username;

                        Net.sendMsg(ns, myResp);
                    }
                }
                if (!isConnected) {
                    Net.sendMsg(ns, new LoginResponse(isConnected));
                }
```

User Guide

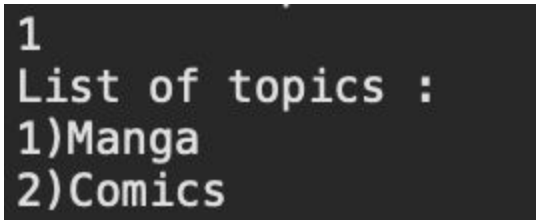
1) You have the choice of register or log yourself to the application



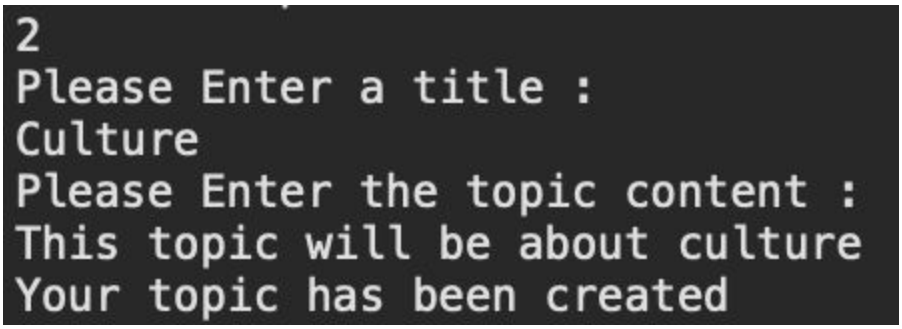
2) Once you're logged you can choose one of the following features :



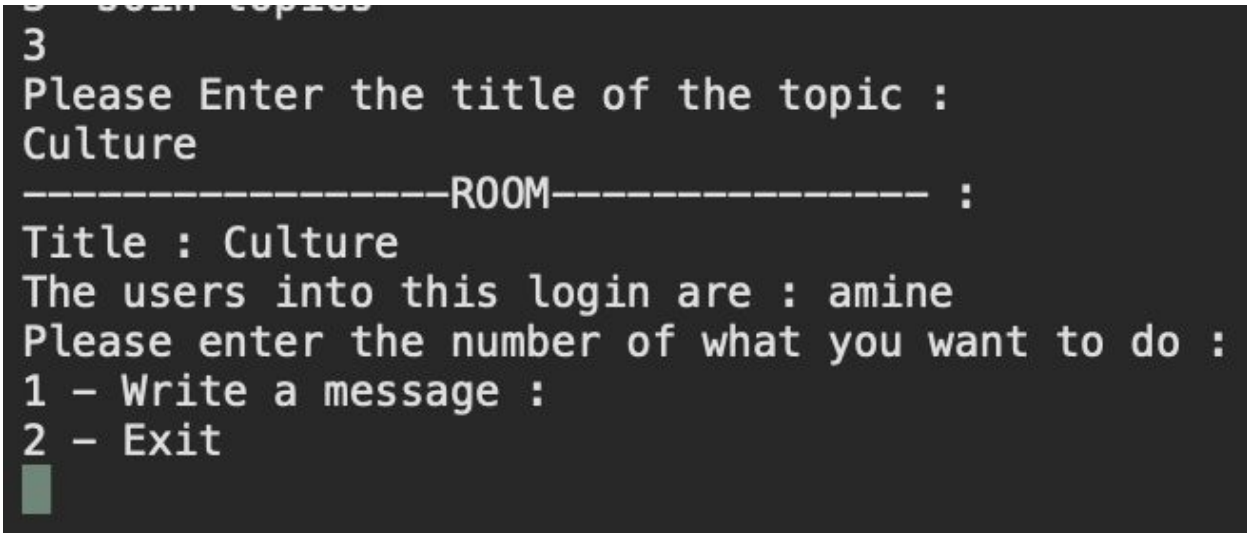
3) If you choose the first feature, you will be able to see all the topics created



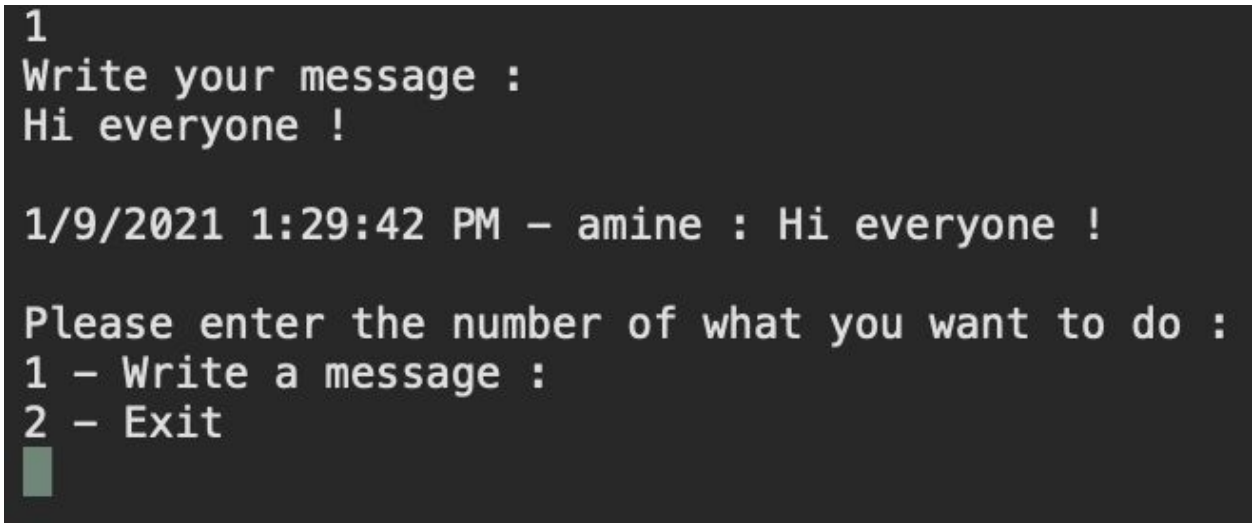
4) If you choose the second one, you will be able to create a topic and fill a title and a description of it.



5) If you choose the third one, you will be able to join a topic and choose to write a message or to exit the topic.



6) If you choose to write a message :



Conclusion

This project allowed me to use everything I learnt during the tp in a wider aspect. The main difficulty I encountered was the implementation of networking and multi-tasking. I would like to thank Mr. Khoury for helping me understand fundamental aspects of C# programming, which helped me to come up with a project that links them all together.