**C# Chat App – Final Report**

*“In this project,*

*you are asked to create a network based multithreaded client-server chat app.”*

**Summary**

|  |  |
| --- | --- |
| **I - Functional Specifications** | |
| **Functions implemented** | 2 |
| **Function not implemented / not finished** | 2 |
| **Possible improvements** | 2 |
|  | |
| **II - My design** | |
| **Structure of the project** | 3 |
| **Representation of the database** | 4 |
| **Communication between the Server and a Client** | 5 |
|  | |
| **III - Screenshots** | |
| **Server** | 7 |
| **Client’s console app** | 8 |
| **Client’s windows form app** | 13 |

Hugues BEGEOT - 20160386

Software Engineering

**I - Functional Specifications**

**I – a : Functions implemented**

These are the required functions :

* Create a profile and save it
* Login
* List topics
* Create topics
* Join topics
* Send messages to all chatters on a specific topic
* Send private messages

All of them have been successfully implemented.

**I – b : Functions not implemented / not finished**

One could argue that the functions remain incomplete : indeed, the function do work well, but they only display the Topics / Chats / Messages at a given time, not in continue.

Hence, **the project is not automated** : **the user has to refresh the app** in order to see the new Topics / new Chats / new Messages.

**I – c : Possible improvements**

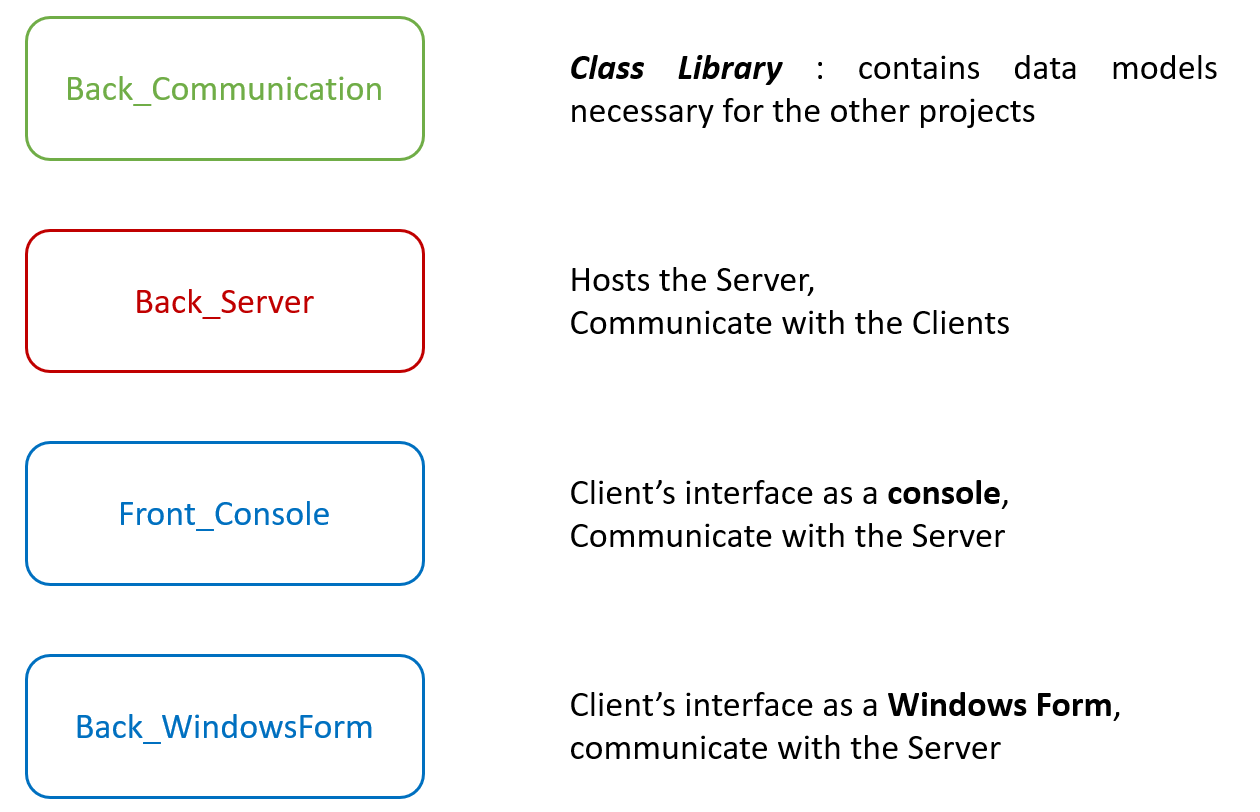
Having the methods display the Messages in continue (if a new one is created, it should be displayed instantaneously on all the users screens). Same for the Topics and Chats : once one is created, all the users should be able to see it.

Also, it would be great to implement a system of notifications : for instance, if a user receives a Message, he won’t know it until he opens his Chat and acknowledges that a new Message appeared.

**II - My design**

**II – a : Structure of the project**

The project is composed of 4 different parts :



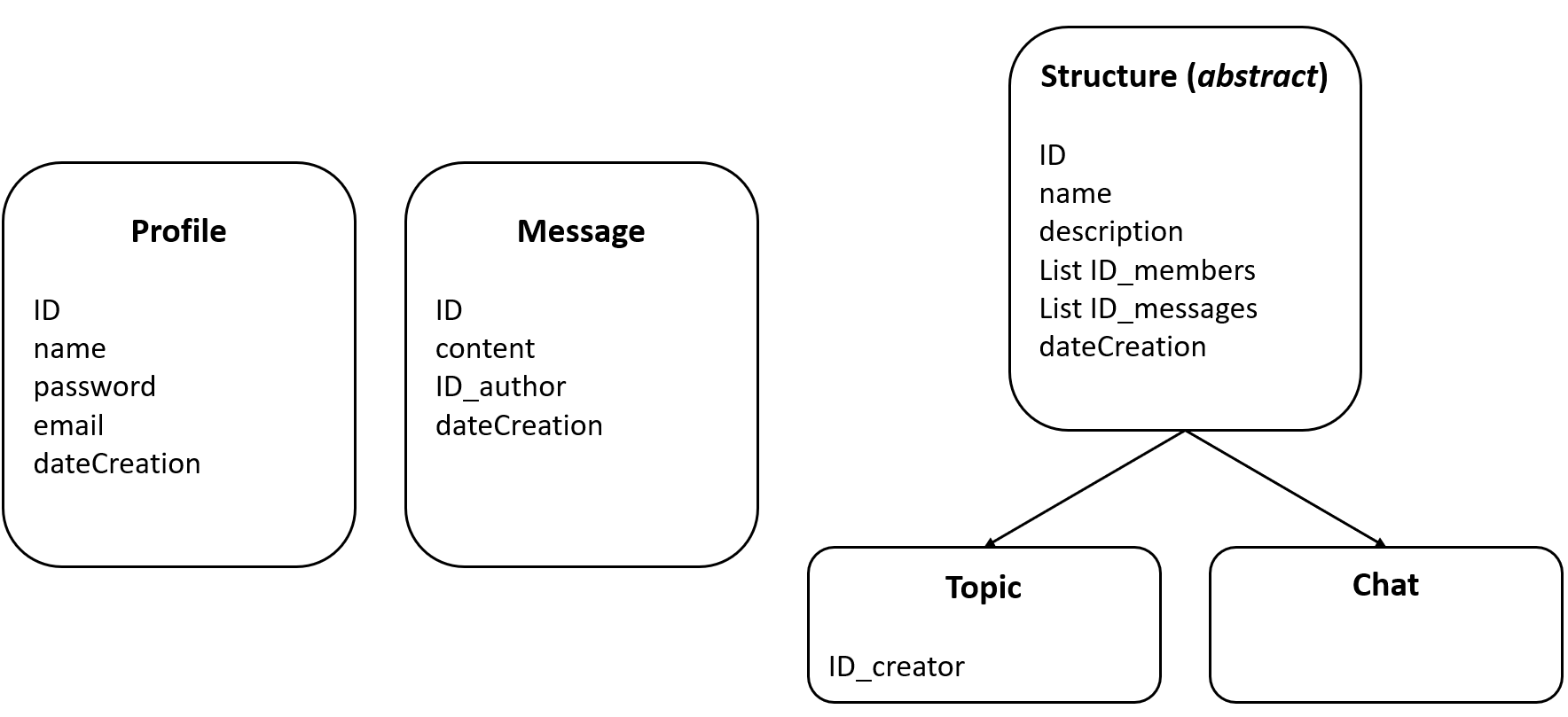
**Back\_Communication** is used *as a reference* by all the other projects.

One single instance of **Back\_Server** must be initiated in order to emulate the Server. About the clients :

* You can launch [0, n] instance(s) of **Front\_Console**
* You can launch [0, n] instance(s) of **Front\_WindowsForm**

**II – b : Representation of the database**

Here is the whole representation of the database :



Each Class from the database contains the following structure :

* A unique **Guid** *ID*, to keep the instance unique
* A **DateTime** *dateCreation*, to keep track of when the object was created
* Some complementary data representing the class

Not all rows are meant to be used : for instance, a Profile has an *email* field, but I never use it properly, nor do I verify if the email entered actually exist.

About the Topic / Chat differentiation, here is my interpretation :

* A **Topic** is a public discussion that everybody can see ; everyone can see and add Messages, and freely join or leave the Topic.
* A **Chat** is a private conversation amongst specific Profiles : only them can see and add Messages. One can join a Chat only and only if he is invited by one of its members. If a Profiles leaves a Chat, he cannot access it anymore (until he’s re-invited).

Since both Topic and Chat share a lot of similar data, they inherit from the abstract class **Structure**, which host most of these information.

Only the Topic keeps track of the Profile that created it (I thought I might use this feature sometime, but I never ended up using it).

Finally, all this database is stored in XML files at this location, in the Server’s files :

* *@\Back\_Server\bin\Debug\Database XML files*

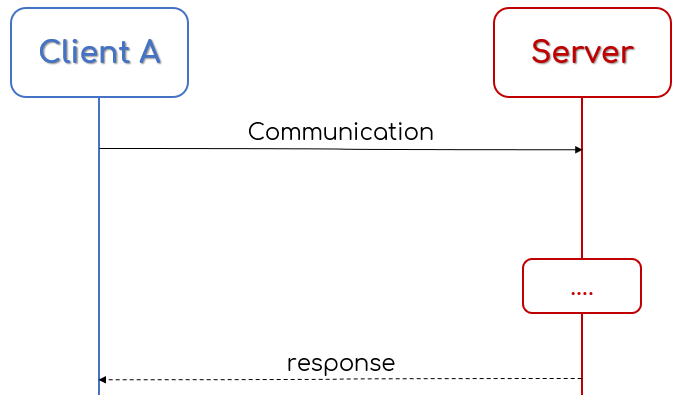
When launching the Server, it reads all the files and dynamically create a virtual database. Whenever the Server updates this virtual database, it also alters or create the corresponding XML file.

**II – c : Communication between the Server and a Client**

Whenever we launch one of the client app (either Console or Windows Form), the Server detects it : it creates a TcpClient relation with it, which allows the Server and this client instance to communicate via the serialization / deserialization of data.

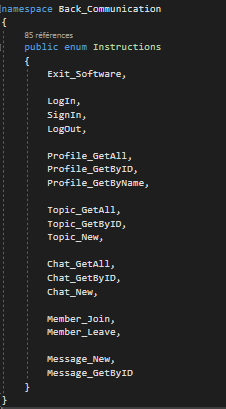
The communication between a client and the server obeys very specific rules :

* The client always sends an instance of Communication
* The server may (sometimes) return some data



This means that the Server is always in a passive state, waiting for one its Client to send a Communication through the (de)serialization, following the TCP protocol.

The class Communication is composed of the following :

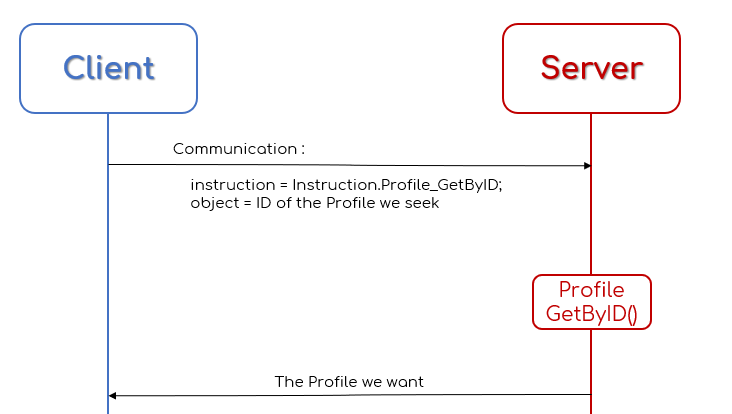
* An Instruction : a value from the enumeration
* An Object : the data that may be needed by the server

Here is the enumeration ***Instructions*** :

Both the Client and the Server have access to it.

Each value of the enumeration is unique, meaning that when the Client sends a specific value, the Server knows which one it is, and understand what it has to do.

For instance : the Client wants to get all the data about a Profile of which it knows the ID



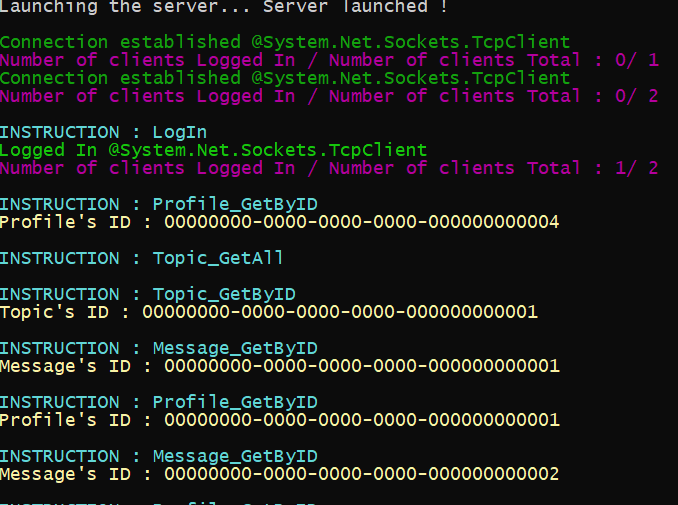
**III – Screenshots**

**II – a : Server**

There is not much to be seen in the Server’s console, so I display the various logs in different colours, each one having a specific meaning. For instance :

* LightBlue : instruction received by the server
* LightYellow : precision about the data received

Etc..



But the server’s console is meant to be scrutinized,

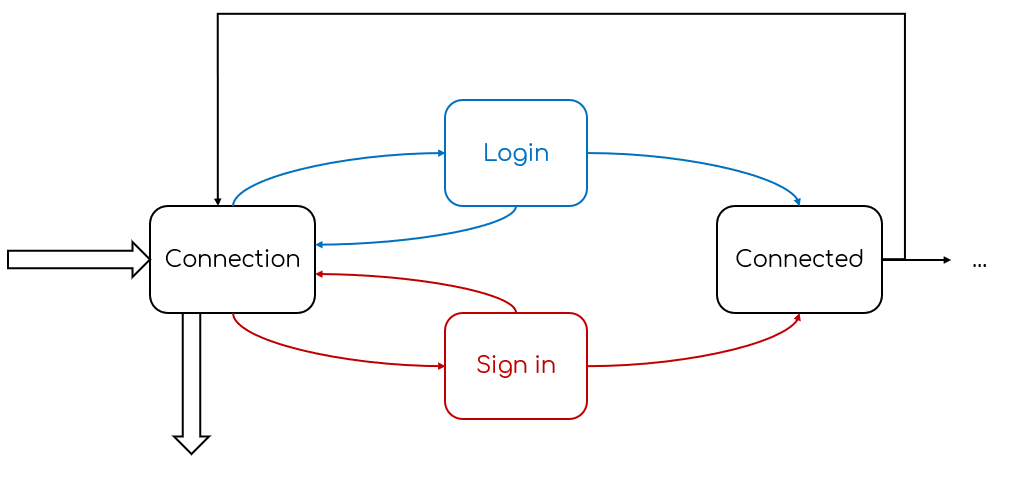
so let’s focus on the Client’s interface.

**II – b : Client’s console app**

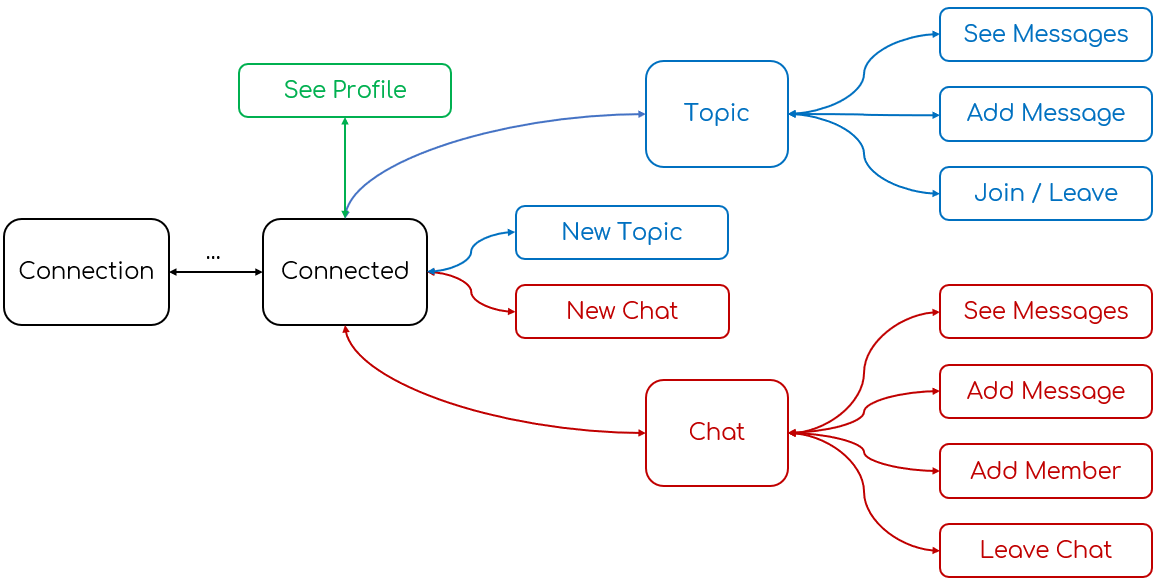
Note about the Console app : whenever the user has to enter values, he faces some input requirements. He can enter nothing in the fields :

* Any form that has empty fields understands that the user wants to go back.

Structure of the *Connection phase* :

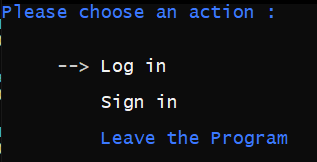


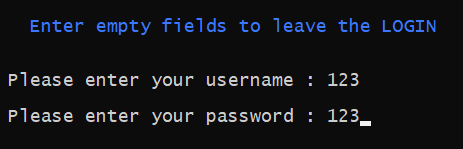
Structure of the *Connected phase* :

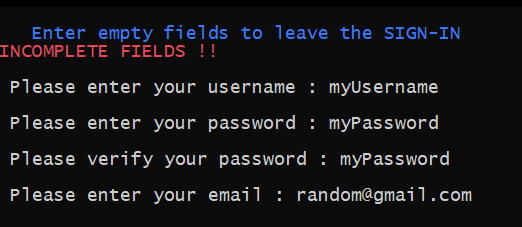


Since the Console cannot display many information at once, I use this configuration.

**Connection phase :**

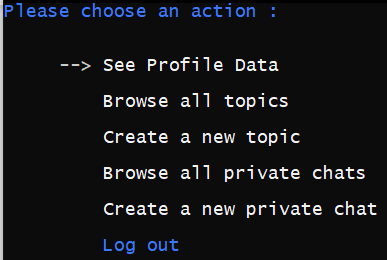




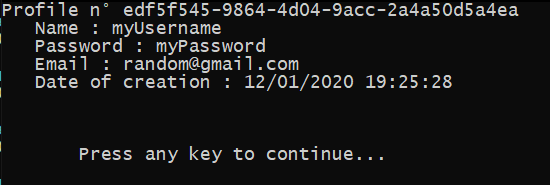


**Connected phase :**

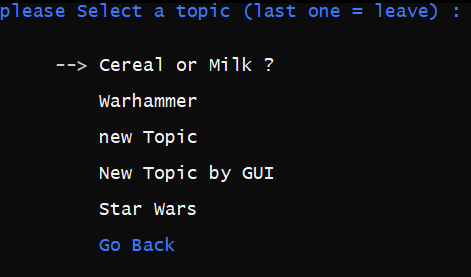
Main Menu :



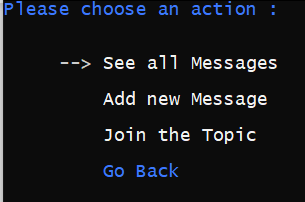
See Profile Data :



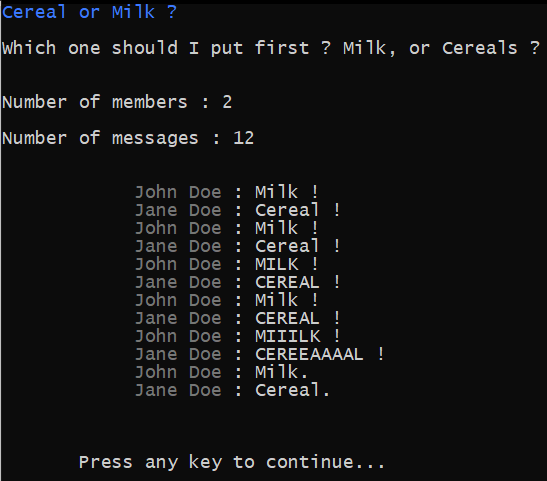
List of all the Topics :



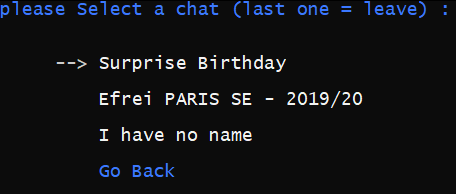
Action available for a Topic :



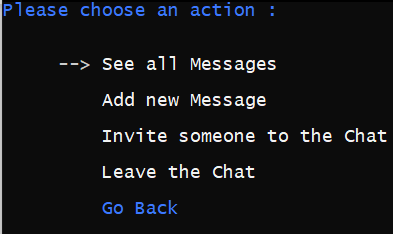
Message of a Topic :

C

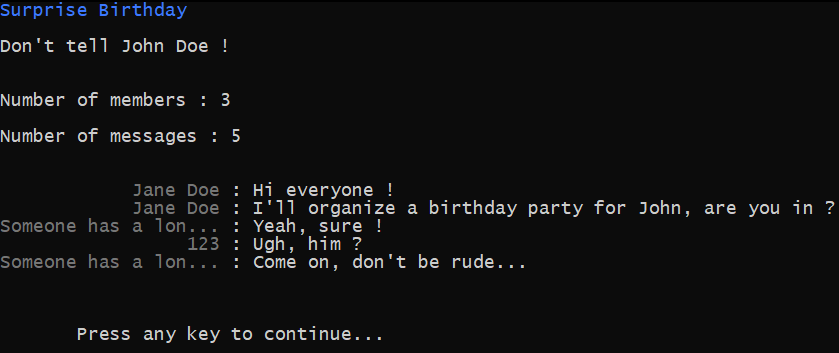
List of all the Chat (the user is connected to) :



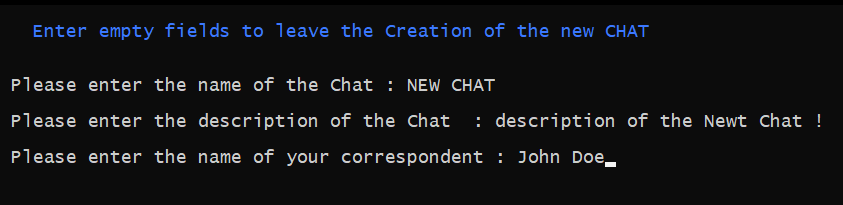
Action available for a Chat :



Messages of a Chat :



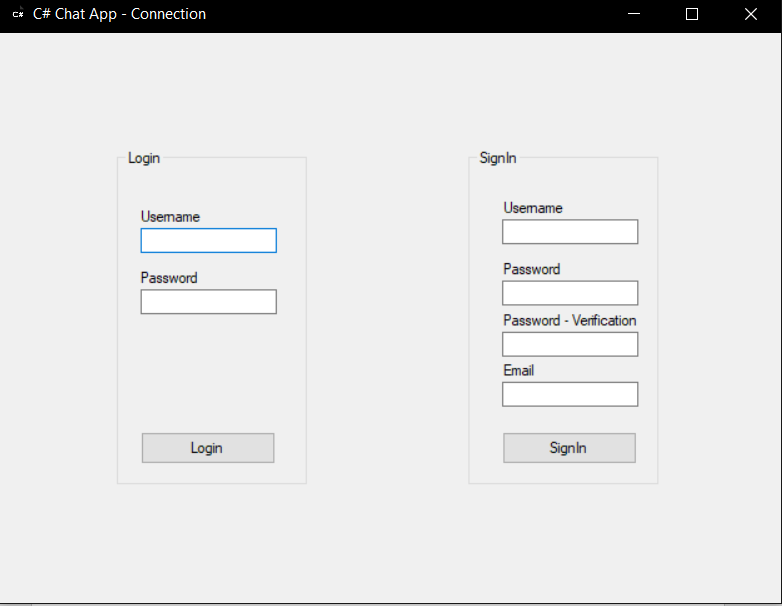
Creation of a new Chat :



**II – c : Client’s windows form app**

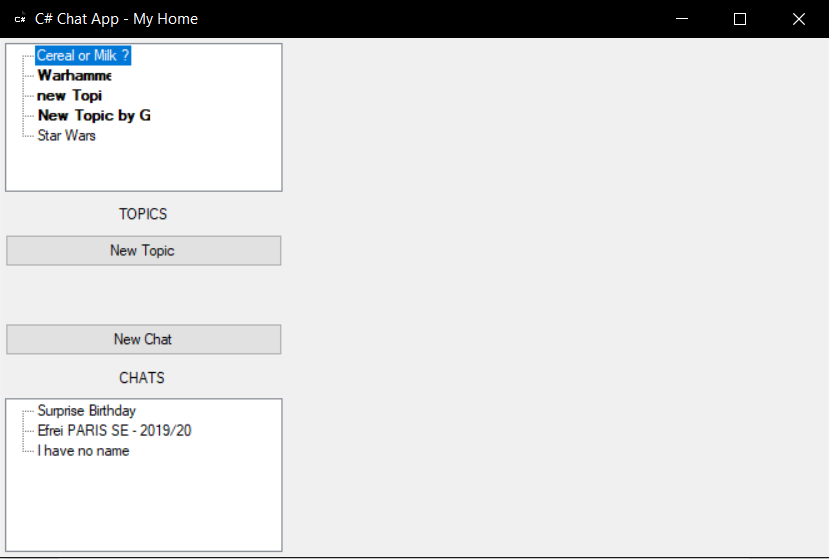
The Windows Form is much more straightforward : I don’t think it needs any schema to represent its layout, since most information are displayed.

**Connection phase :**

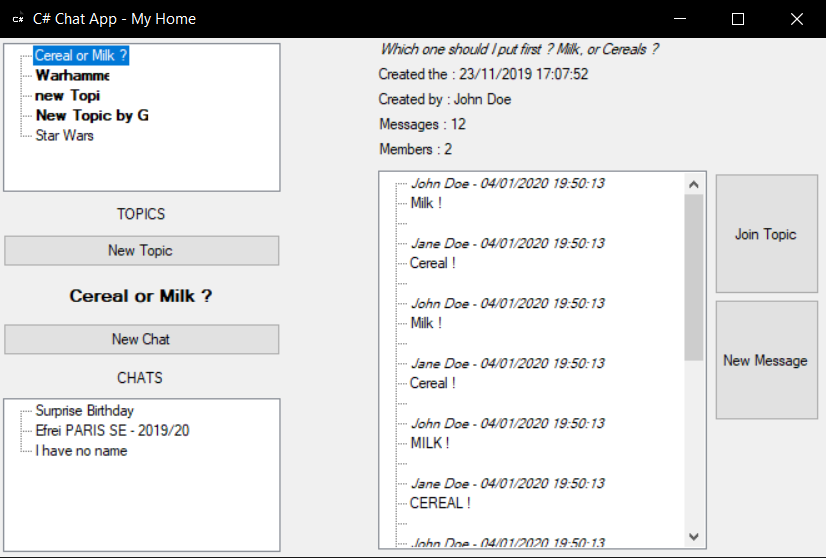


**Connected phase :**

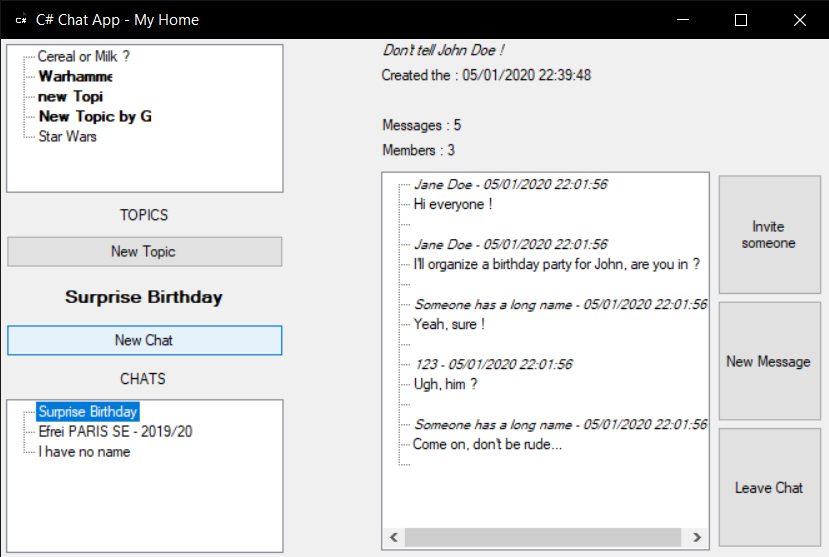
When no Topic / Chat is selected



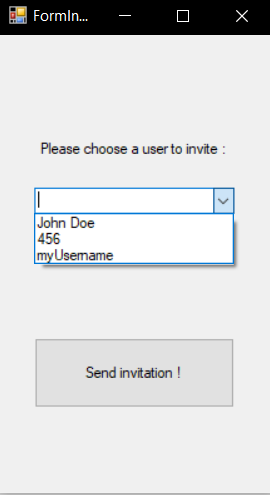
When the Topic ***Cereal or Milk ?*** is selected



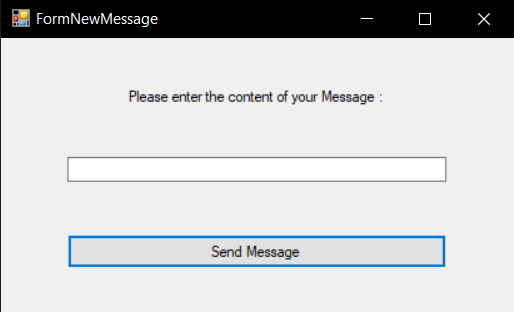
When the Chat ***Surprise Birthday*** is selected

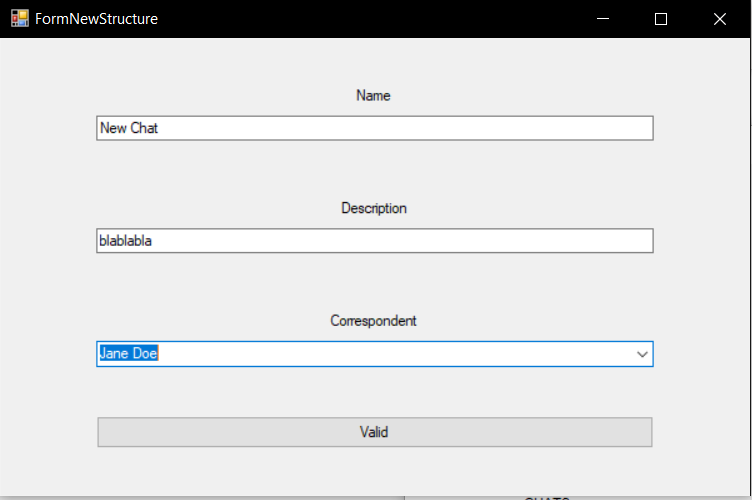


When inviting someone to a Chat



When writing a new Message



Adding a new Topic 

Adding a new Chat

