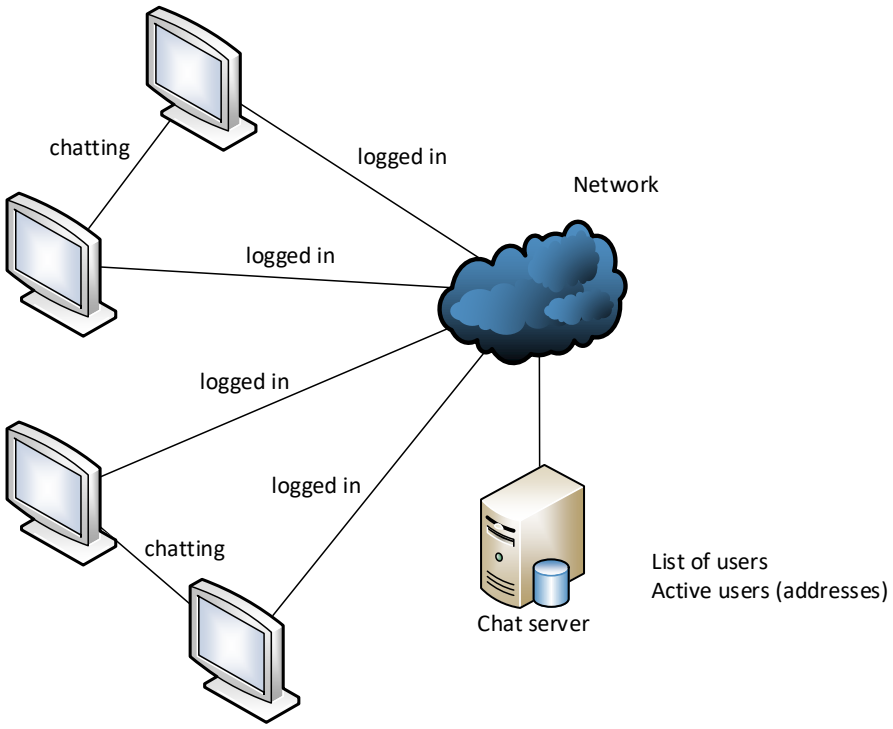
Distributed and Integration Technologies (TDIN)

Assignment # 1 – Remote objects (.NET Remoting)

2019-20

Scenario

Some company needs to develop an IRC (*internet relay chat*) based on remote objects architectural style (.NET Remoting) and with a client graphical user interface (GUI). Each user should be registered in a common server, supplying a *nick name* (a single word), his *real name* and a *password*. All users have the same client application to initiate the *chat* sessions.



Operation

After registering (only one time) and running the client application, users must *login*, supplying the *nick name* and *password*.

After a successful *login* the server supplies to this client the list of all other active users (the users with a login already done). That list should be refreshed in real time, meaning that whenever a new user logs in or a current user logs out, the others should see the changes immediately.

After *login*, and from the list of active users, this user can ask another for a *chat*. If that *chat* is accepted by the other user, a new window should appear for each of the chatting users. This window should contain at least an editable line of text, for writing new messages, and a button, for sending the message. Also, a text box, containing the exchanged messages (identified by the *nick names* or different colors, for instance) should be present in that window.

Any user can terminate the *chat* closing that window (which automatically causes the other user correspondent chat window to close also).

Finally, when a user does not want to continue chatting, he should do a *logout* operation (or simply can close the client window, causing that *logout*).

Interactions

The *login*, *logout* and *register* (only once per user) operations are carried out between each user and the server application.
The server, whenever there is a change in the list of active users (logged in), should notify all other users of that change, allowing them to refresh their lists of active users.
Conversations in chats should be done **directly** between the *chatting* users. For that, the server sends also to each user's client the 'address' of each other client (together with the list of active users or when there is a change).
With several clients in the same machine, their remote objects should listen in different ports, automatically attributed when the client starts running.
The server application should persist the information regarding all *registered* users.

Implementation

Students should implement a demonstrator of this scenario using .NET Remoting and developing the needed applications and object, through the best practices available.
The applications should have a simple, practical, and intuitive user interface with one of the GUI technologies available in .NET.
Other convenient functionalities can be added, like, for instance, allowing each user to maintain several separated chats, at the same time (in different windows), logging chat conversations, sending other types of information (files, pictures, …), creation of chat rooms involving more than two users, etc.

Report

A report containing your architecture specification (applications, modules, remote objects, events, and interactions), list of included functionalities, functionality tests performed, and working modes (with screen captures of the main flows), should be delivered together with the project code. Also, instructions on how to build, configure, and use of the demonstrator should be provided.