Multithreaded client-server chat application

Nicolas Durat[1]

[1] *C# and .NET environment lesson, EFREI Paris, 94800 Villejuif, France**

December 10, 2020

Abstract

As part of the C# project, we have to realize a networked multi-threaded client-server chat application with a fully object-oriented design. After introducing the application, we will discuss its functionalities: the server application and the client application. Indeed, we will explain the implemented and non-implemented functions and possible improvements. Then, we will try to understand the design and implementation of the project through use case diagrams and class diagrams. Finally, we will see the execution of the application through screenshot.

Keywords: Multithreading application, Client-server, Full object oriented design, Serialization

*Electronic address : nicolas.durat@efrei.net

Contents

1 Introduction

The objective of the project is to realize a client-server application based on a multithreaded network. Multithreading in C# is a process in which several threads run simultaneously. It is a process to achieve multitasking. This saves time because several tasks are executed at the same time. Our application must be able to establish a connection between clients and a server in order to provide various services : the creation of profiles (login, password) and saving the login, the creation of topics, the display of topics, access to topics, sending messages to all users and finally the sending of private messages.

2 Application features

2.1 ServerApp

The different features of the server are: the Start button, the Close button, the list of users connected to the chat, sending a private message to a particular user, and access to the public chat.The start button connects the TCP Listener to listen for client connections on Port 5000. The Close button stops the server and disconnects users automatically. The list of connected users allows you to see who are the clients connected to the chat. Access to the public channel is possible but the developer can address a user privately by sending him a message through the private channel.

2.2 ClientApp

2.2.1 Functions implemented

The user must first register or connect to the server with the localhost address 127.0.0.1 and the server port number 5000. Each user has a TCP client that will be picked up by the server after sending the request. He connects with a username and password stored in a text file. In case the user is a new member he just has to register. If the data is already being used by another person, the user will be informed and asked to choose another non-empty username and password. After logging in or registering his account, the user will have the possibility to choose his nickname which will be displayed on the public channel. Then, he will have access to the public channel where he will first have to connect to the server via a button. Automatically, the server receives the request from the client and confirms the connection in the public channel with a message. In addition, the word "connected" in the upper left corner is displayed in green (red if disconnected). The user's nickname will appear in the list of connected clients. This can be used to send a private message. The user will be able to send public messages, see the list of topics and connected members or quit the application by clicking on the cross in the top right corner of the application. The message that is a subject will be serialized in byte table to be sent to the server that will analyze it thanks to the delegates and deserialize it in the public channel to be displayed. Note that the message will be visible on the client and server side. Finally, the user will be able to send a private message to another user. He chooses a person in the list of connected users and clicks on a button reserved for a private channel. The user then arrives on a private chat room with the person concerned. He will then be able to chat with only one user. The server will receive the information but will not display it on the public channel.

2.2.2 Possible improvements

- Create new topics
- Be able to connect to several topics
- Protect account backup
- Give the administrator the ability to delete topics or users
- Set a clock
- Having notifications
- Check that the message has been sent and read

3 Design and Implementation

3.1 Use case diagrams

3.1.1 ServerApp

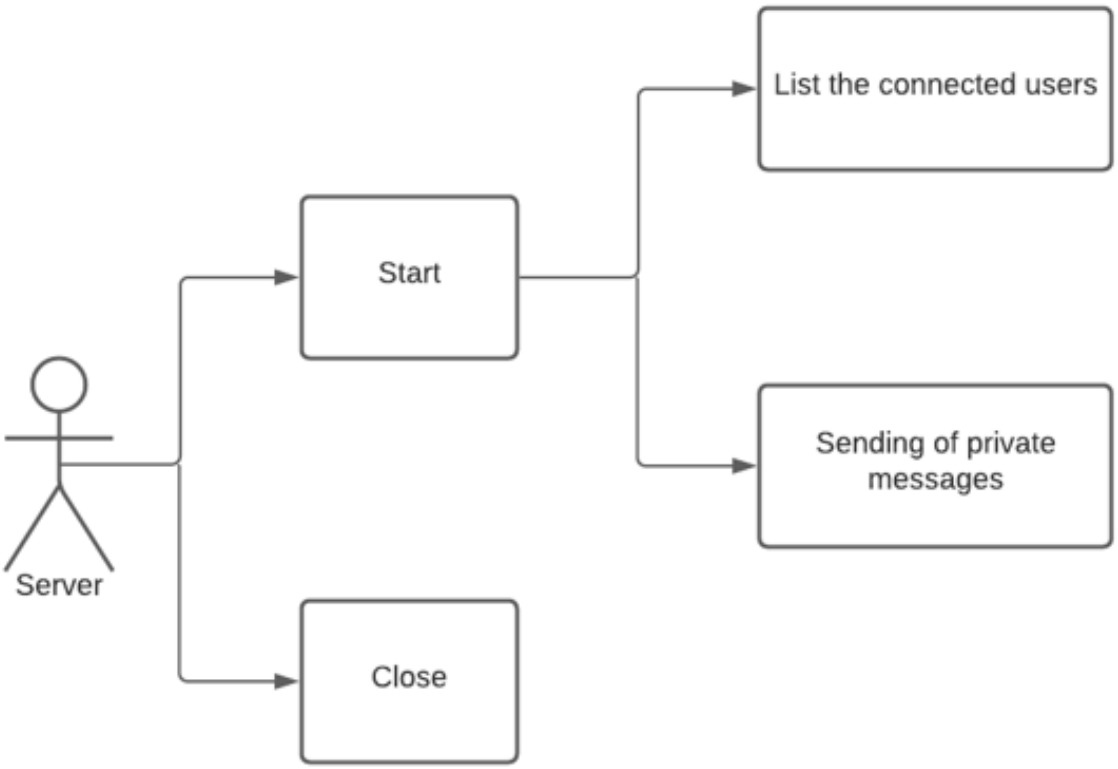

Figure 1: Use Case diagram of Server Application

3.1.2 ClientApp

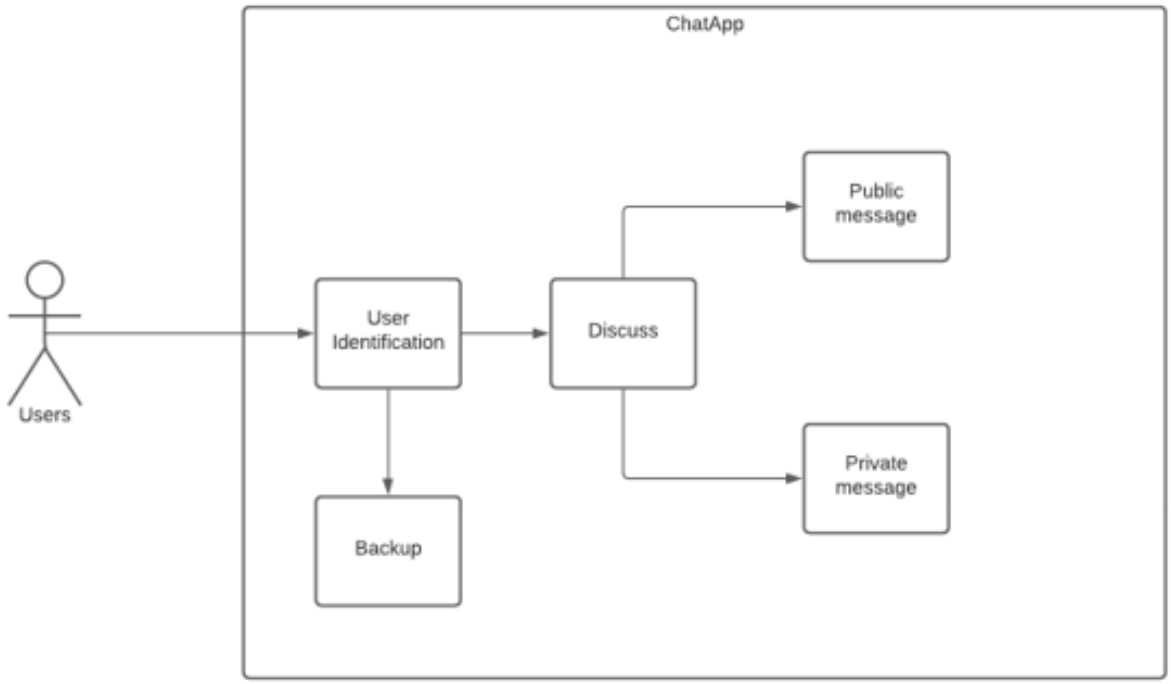

Figure 2: Use Case diagram of Client Application

3.2 Class diagrams

3.2.1 ServerApp

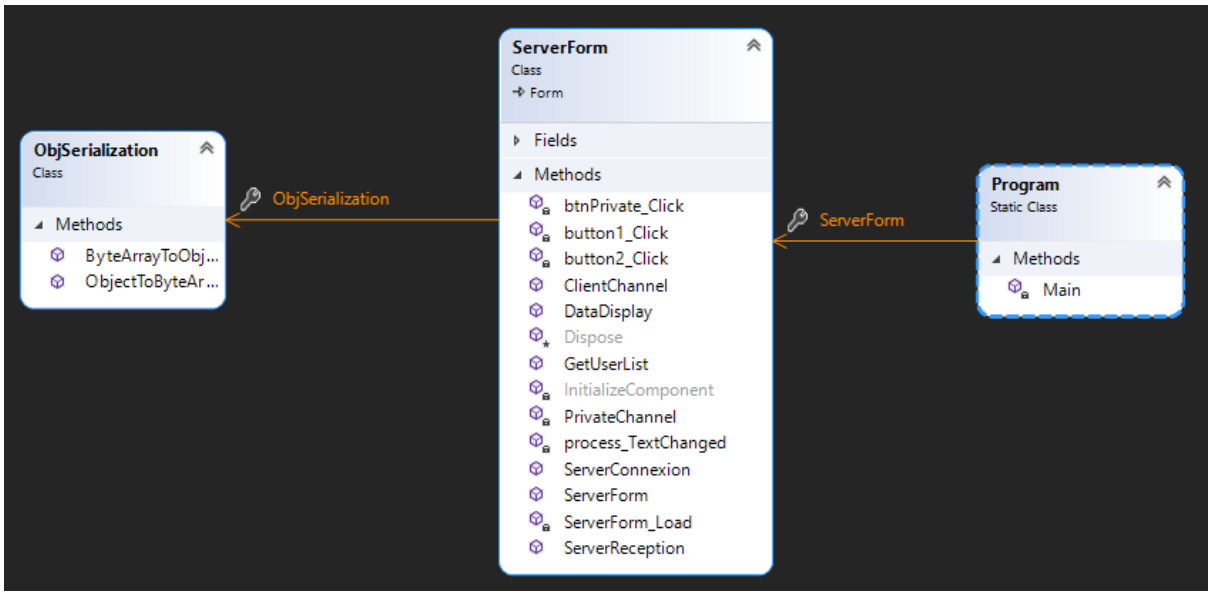

Figure 3: Class diagram of Server Application

3.2.2 ClientApp

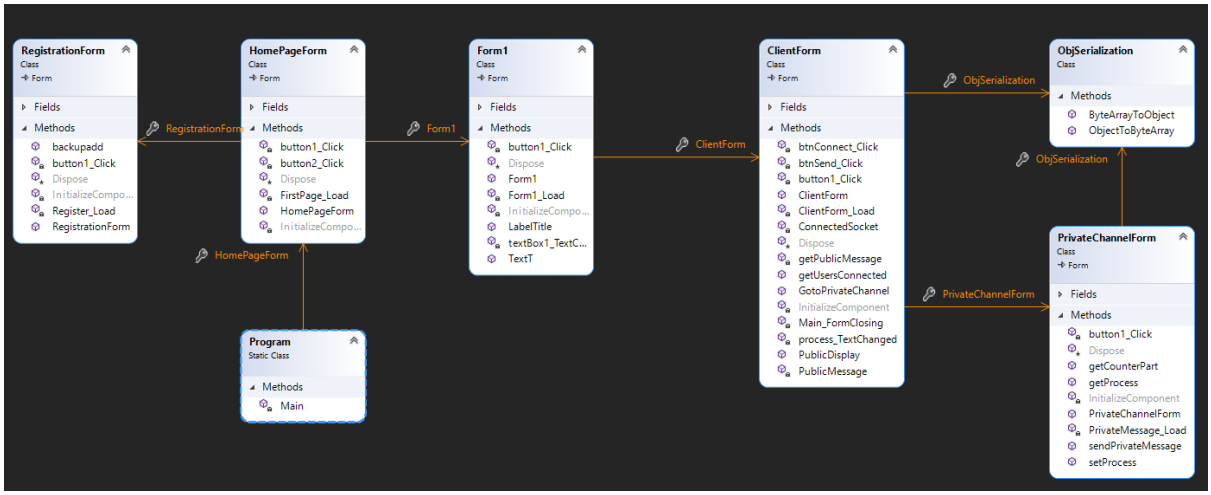

Figure 4: Class diagram of Client Application

3.3 Implementation

3.3.1 ServerApp

The server part consists of 3 classes: ObjSerialization, ServerForm and Program.

- **The ObjSerialization class processes user messages that will be displayed in the public channel on the client side and on the server side.** The ByteArrayToObject() and ObjectToByteArray() methods convert respectively a binary array into an object and an object into a binary array through serialization/deserialization.
- **The ServerForm class implements most of the features of the Server application.** It stores the IP addresses of the clients that connect to the server in the clientList dictionary, stores their messages in a list and deploys all the methods required by the project for the Server part :
  - ServerForm() : Initializes the WinForm components.
  - DataDisplay() : Receives and displays data in the public channel through a delegate.
  - ClientChannel() : Receives messages when the flag is true and displays the messages in the client's public channel using Broadcasting . Using the underlying stream returns the NetworkStream used to send and receive data. If flag is True, the message will appear directly in the client application's chat room.
  - GetUsersList() : Allows you to obtain the list of users. I add each client name in a users list that I convert into a bit array (userList). For each item in clientList I take the Broadcast value, write the data in bytes and then delete the data stream.
  - PrivateChannel() : Sends a private message to the selected client from the server as administrator. I use a Linq query to find the client in the dictionary.
  - ServerConnexion() : Connects the TCP Listener to IP Address 127.0.0.1 and port 5000 (localhost). As long as the server is connected and active, the task performed is asynchronous (runs without blocking the wire) thanks to await which allows to start a task without blocking the wire and to continue the execution once the task is finished. Thus, a connection request is accepted asynchronously. Hence the "Users Waiting" message, which confirms that the program is waiting for clients without blocking the wires. Then the users are retrieved by obtaining and reading the data stream (NetworkStream). The bytes are converted into a string. The client is added to the "ClientDico" dictionary and to the list of users. At the same time the server application and the client application indicate the arrival of the user. The task is completed within 1000 milliseconds.

- ServerReception() : Displays messages and confirmations from clients connected to the public channel of the server application. It retrieves and reads the data stream with the TCP client. Writes the data received through a delegate in the public or private channel. Finally, once the client has left, it is removed from the client dictionary.
- Button1_Click() : Button that connects the server to the localhost.
- Button2_Click() : Button to interrupt the connection and close the server. The client is automatically disconnected.
- BtnPrivate_Click() : Button that allows you to send a message to a specific person in private. The messages are put in a list. The client is found in the dictionary according to its name. The message is sent to the right client using the TcpClient. Finally thanks to NetworkStream we send and receive data streams on stream sockets in blocking mode for network access.

- **The Program class launches the application.**

3.3.2 ClientApp

The customer part is composed of 7 classes: ClientForm, HomePageForm, ObjSerialization, PrivateChannelForm, Program, PseudonymForm, RegistrationForm.

- **The HomePageForm class allows the user to login or go to the profile registration page.** It has 3 methods :
  - FirstPage_Load() : Recovers the login and password in a text file in order to classify them in two disjoint lists.
  - Button1_Click() : If username and password are recognized, open the window to choose a pseudonym.
  - Button2_Click() : Button to register in the database.
- **The RegistrationForm class allows the user to register an account.** It has 3 methods :
  - Button1_Click() : Registers, checks if the login and password already exist in the database.
  - Register_Load() : Reading, checking the text file. Recovery of login and passwords so that they are filed in two disjoint lists.
  - backupadd() : Insert the login and password in the text file using the StreamWriter.
- **The PseudonymForm class allows the user to register the pseudonym used to chat on the public channel.**
- **The ClientForm class represents the user interface to communicate, connect to the server, send public messages, go to the private messages page, see the list of connected users and the list of topics.** It has 11 methods of which the two most important are :
  - getPublicMessage() : Retrieves the different messages (private, public or user list), displays them in the right channel by converting the array of bytes into an object. If the connection fails it asks to start again.
  - ConnectedSocket() : Checks whether a user is connected to the server or not.
- **The PrivateChannelForm class allows to send a private message to a user.** It has 5 methods of which the three most important are :
  - getCounterPart() : Retrieve the name of the user with whom we communicate privately.
  - getProcess(): Returns private channel messages.
  - setProcess() : Allows us to receive (write) messages from users with whom we communicate privately through a delegate.
- **The ObjSerialization class processes user messages that will be displayed in the public channel on the client side and on the server side.** The ByteArrayToObject() and ObjectToByteArray() methods convert respectively a binary array into an object and an object into a binary array through serialization/deserialization.
- **The Program class launches the application.**

4 Application execution

4.0.1 ServerApp

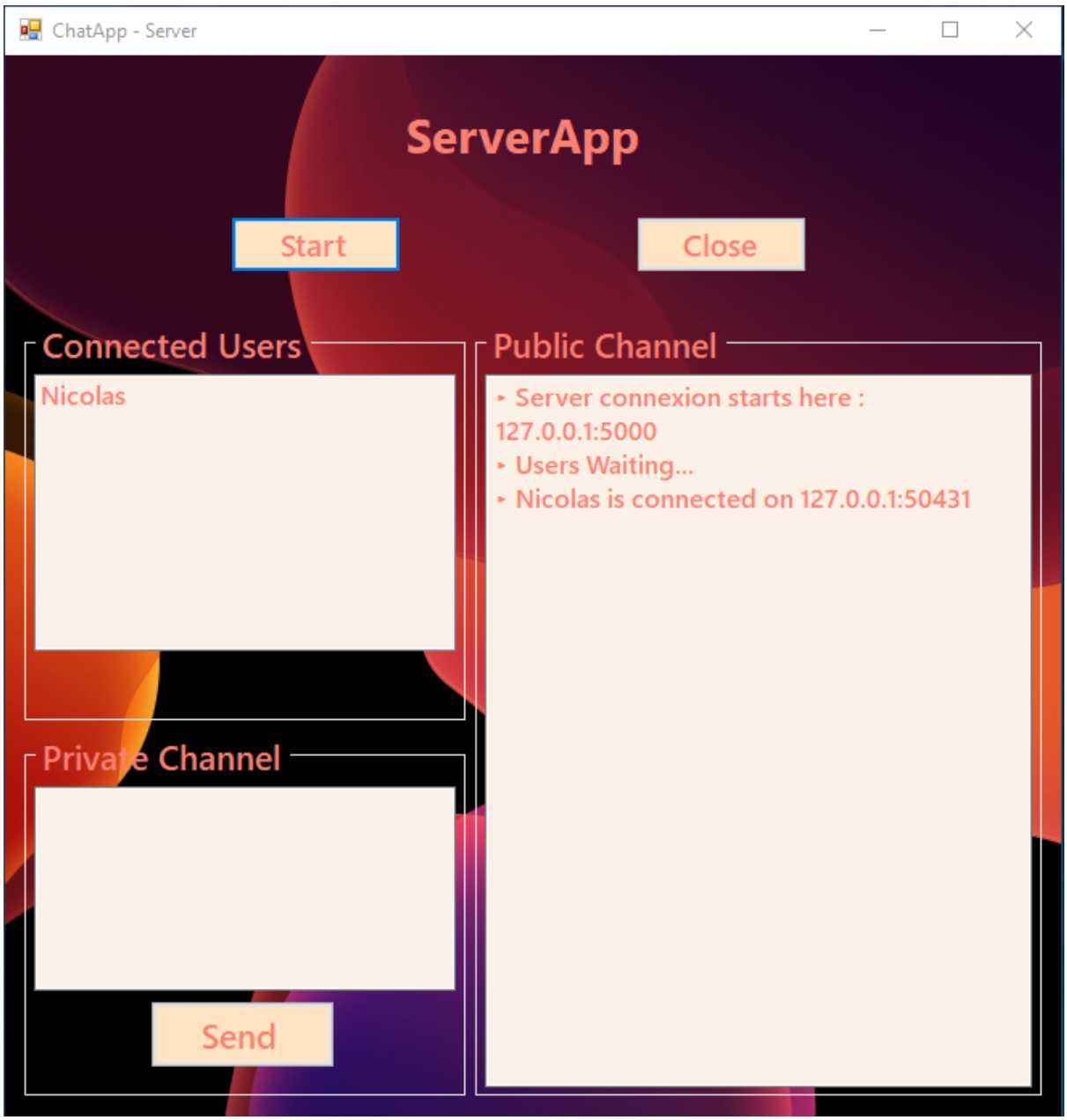

Figure 5: ServerApp Screenshot

4.0.2 ClientApp

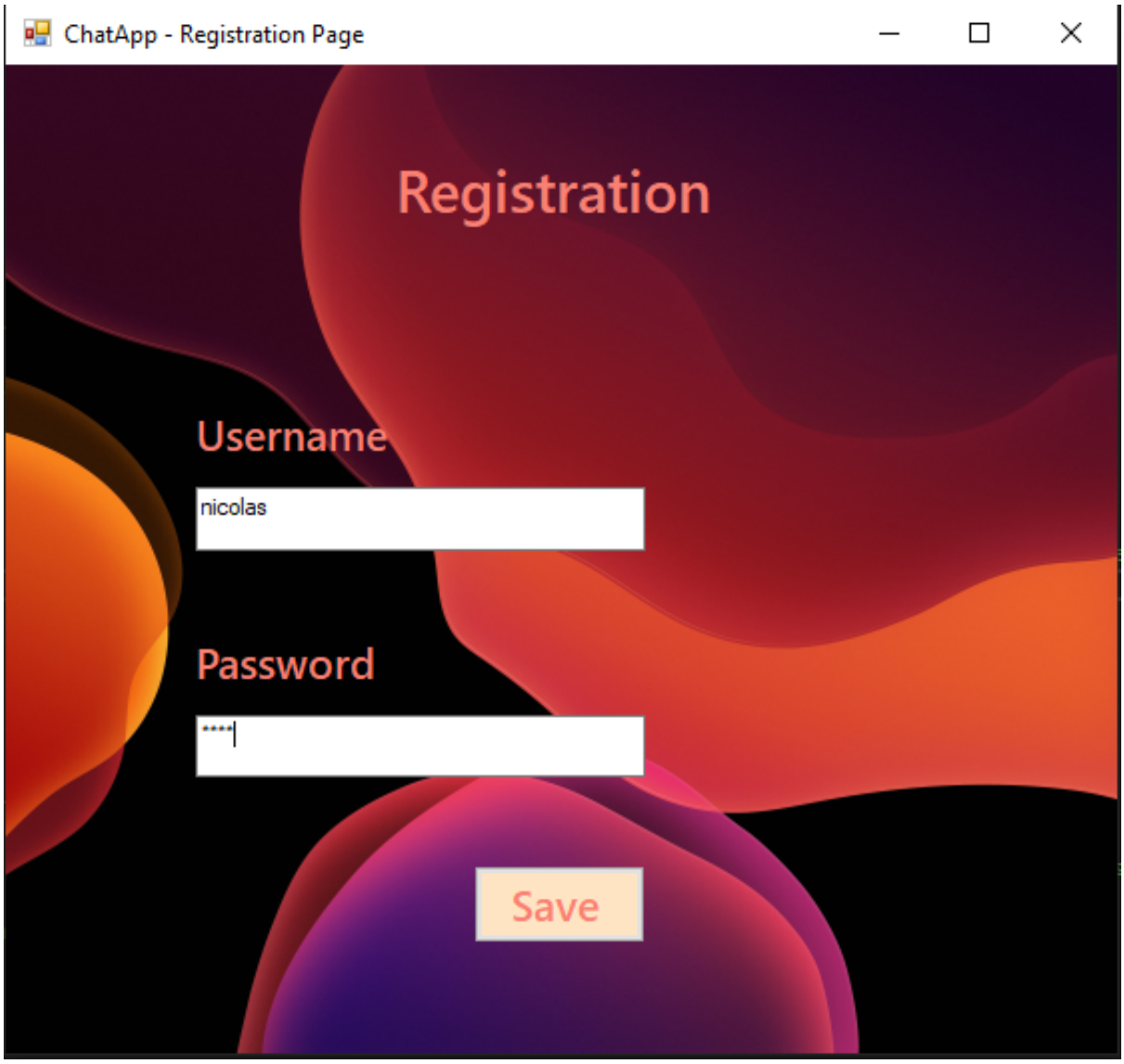

Figure 6: Registration Screenshot

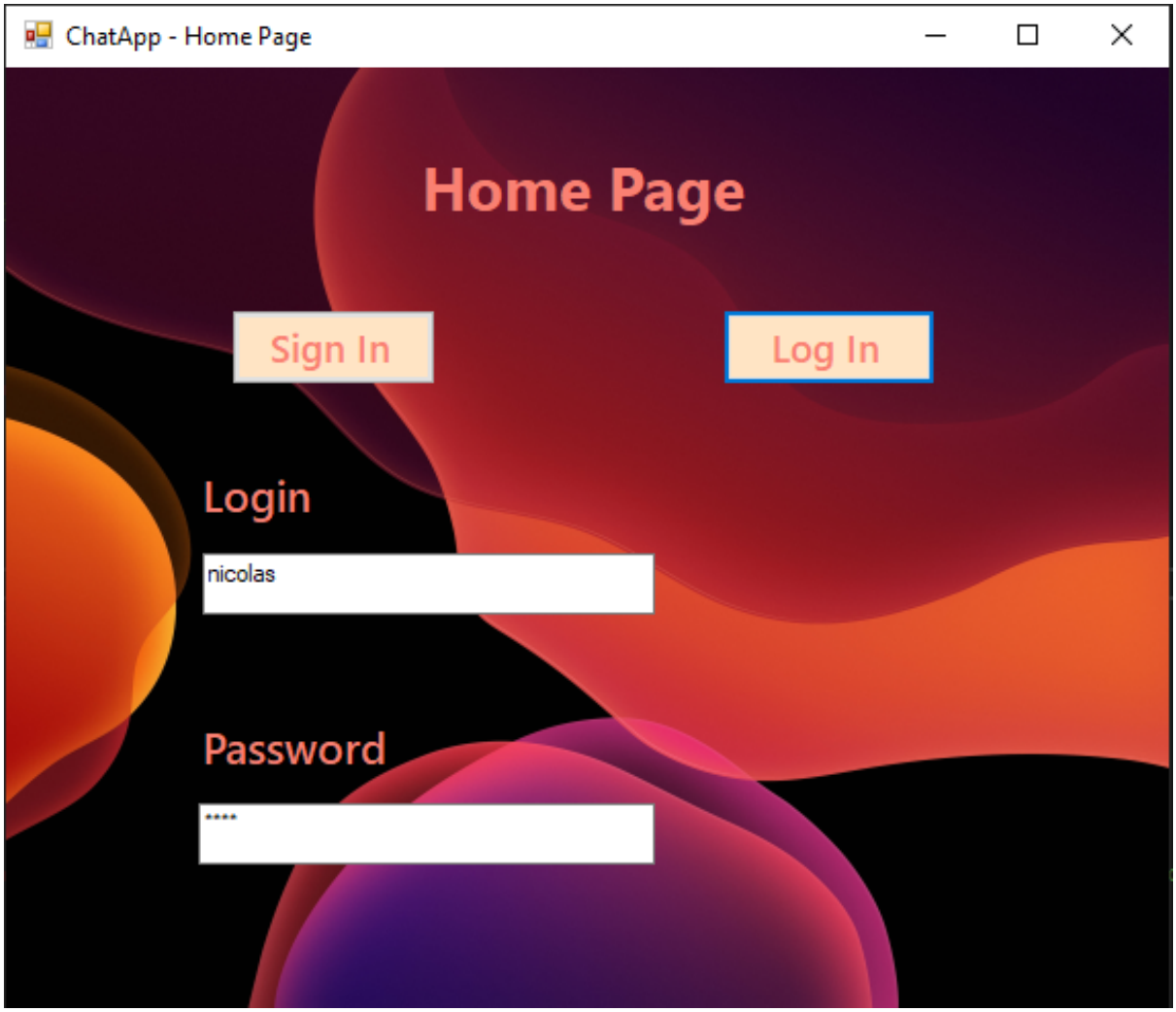

Figure 7: Login Screenshot

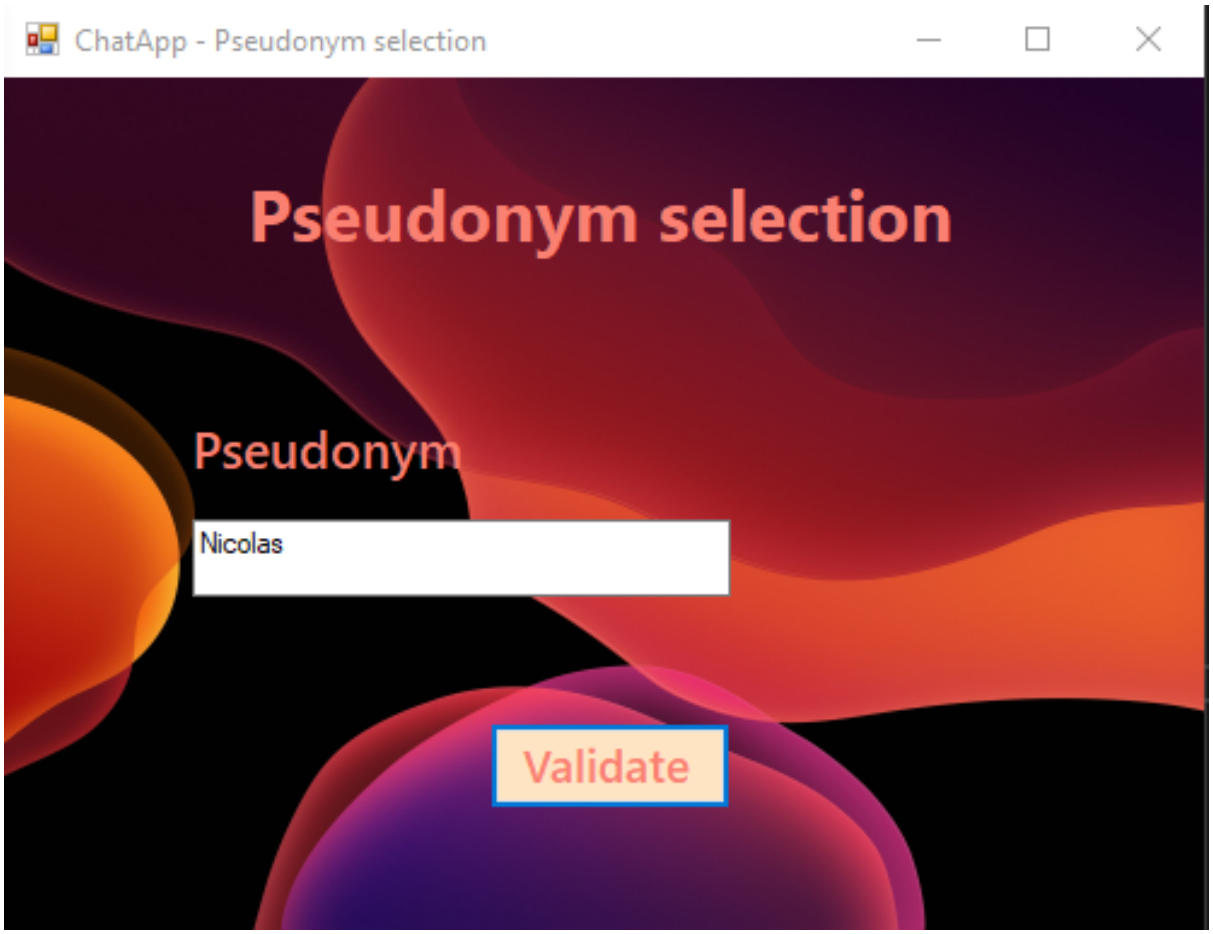

Figure 8: Pseudonym selection Screenshot

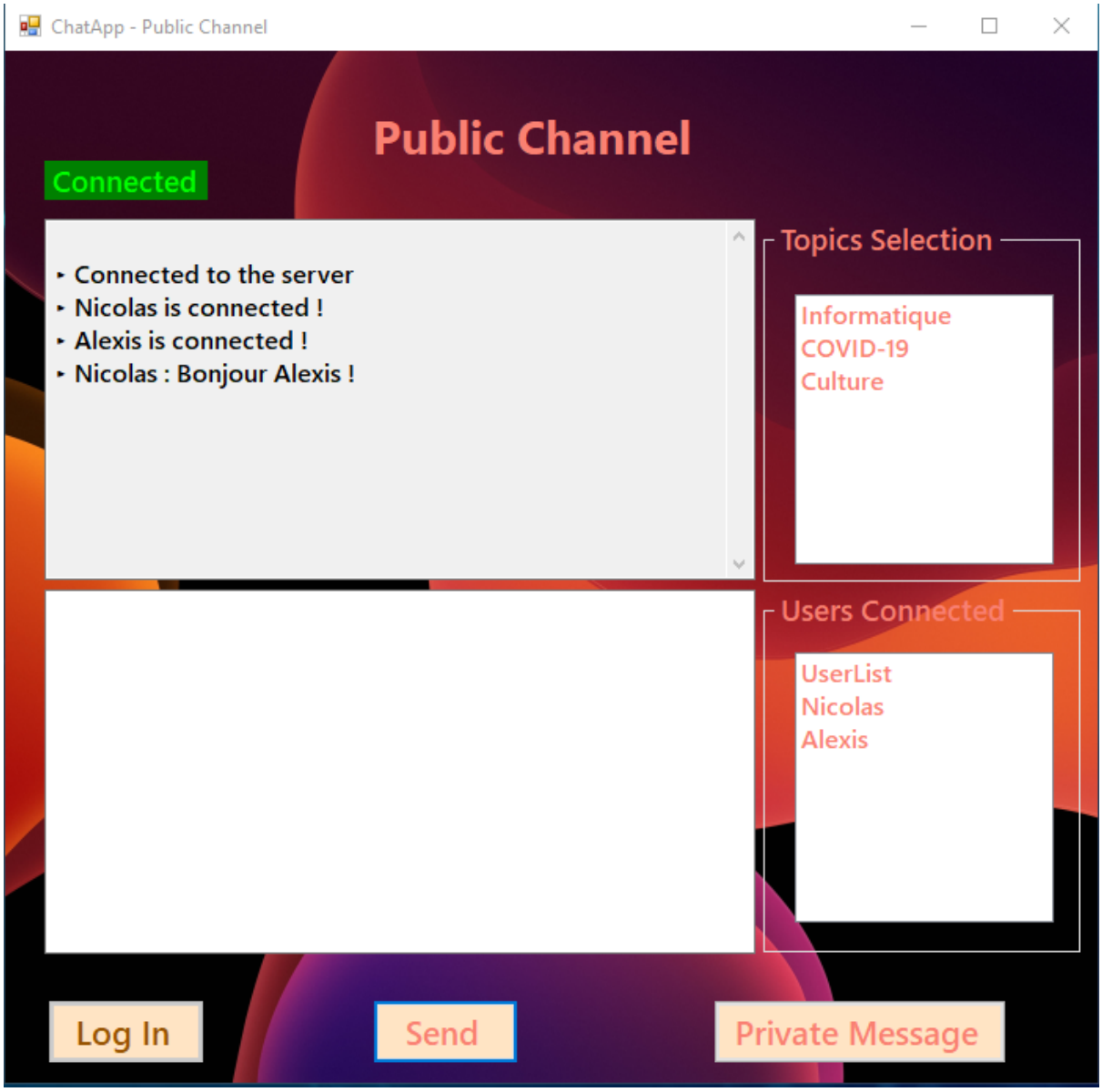

Figure 9: ClientApp Screenshot

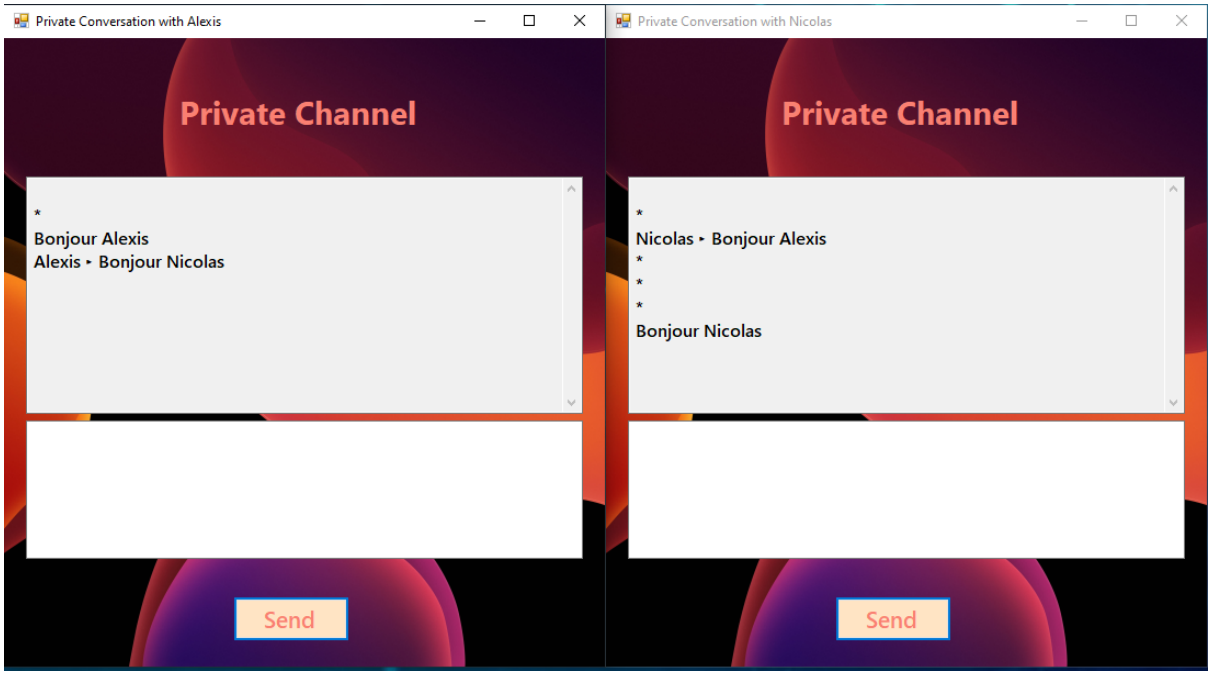

Figure 10: Private Message Screenshot