 Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Department of Computer Science

October 2019

## Documentation

##### Client - Server Messenger app using .NET Framework

## Structure of Computer Systems

## Professor Gheorghe Sebestyen

## Laboratory Teaching Assistant Zoltan Czako

## 3rd year of study, 1st semester

## Teodora Irina Mărginean

## group 30433

Table of Contents

## Introduction ………………………………………………………………………………… 3

## Objectives …………………………………………………………………………………….. 4

## Theoretical fundaments ………………………………………………………………… 5

## Implementation and Design ……………………………………………………………. 9

## Testing …………………………………………………………………………………………. 15

## Conclusions and Further Development …………………………………………… 19

## Bibliography …………………………………………………………………………………. 20

1. Introduction

#30. Demonstrative – interactive application that illustrates the communication instruments in C# and .NET Framework.

Design a client – server application that communicates through the network (a messenger type of application). The application will be accompanied by a documentation that exemplifies how to implement a client – server application in the .NET Framework.

Chat applications seem quite simple to implement, but they require a good knowledge of network communication protocols and network sockets, in order to be able to send and receive messages from different executables, which could run on different computers.

Sockets are used in order for two nodes in the network to communicate with each other. They are a one-way Client and Server setup where the Client connects to the Server, it sends it a message and the Server will know what message it was sent to them using a socket connection.

It is important for this application to implement bidirectional communication, so that the sender would get feedback from the receiver, unlike one-way communication where the message flows only from the sender to the receiver. This allows the sender to know if the receiver got the accurate message from the sender. Communication is also negotiated, which means that the sender and the receiver listen to each other.

This Chat Application will be developed as a Windows Presentation Foundation (WPF), which is a newer, better way of developing GUIs for a Windows Desktop Application, than Windows Forms that have grown old and are harder to use.

2. Objectives

The objectives of the assignment are:

* To get accustomed to the development of a client – server application
* To get used to network sockets used in .NET Framework
* To learn about network communication protocols
* To have hands-on experience in developing a chat application
* To learn how to develop WPFs in .NET Framework
* To get used to documenting an application and the code behind it

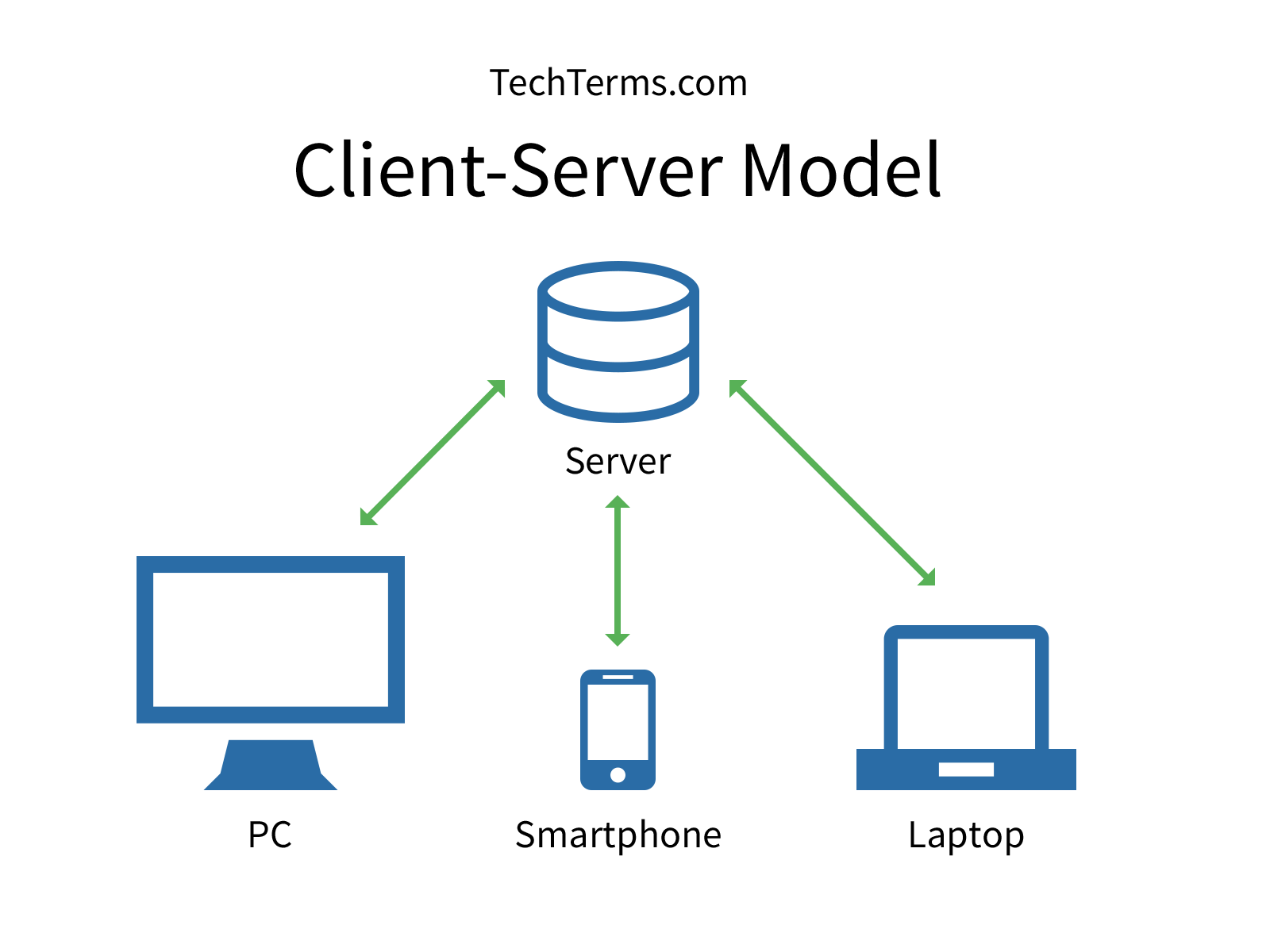
The objectives of the application:

* have a Graphical User Interface (GUI) as a Windows Presentation Foundation (WPF)
* have a Server
* have multiple Clients
* change the username of the current user (being it the server or a client) from the GUI
* change the port of the current user from the GUI
* change the host of the current user from the GUI
* start or stop the current application from the GUI
* the Server has a list on the GUI that shows the clients it has connected to
* the Server contains a log of the messages sent in the current session
* if the Server crashes, the Client will be notified of it and will not be able to send any more messages
* if the Client crashes, the Server will be notified and will be listening for other clients
* if the connection is interrupted, the Client and Server will wait for each other
* if a message was not sent, the sender would have to send it again or restore the connection between the server and the client through a button in the app

3.Theoretical fundaments

Client-Server Architecture

The structure of this project is based on the Client – Server Architecture, which is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. This type of architecture allows one or more client applications to be connected to a central server over a network or an internet connection. It also allows resources to be shared between server and clients, which are usually running on different computers (though they may be also running on the same computer).

Client - server architecture is also known as a networking computing model or client - server network because all the requests and services are delivered over a network.

It is also a producer – consumer type of architecture, where the server acts as the producer and the client as the consumer. The server hosts and provides computer-intensive services to the client on demand. These services can include application access, storage, file sharing, printer access and/or direct access to the server’s raw computing power.

Client - server architecture works when the client sends a resource or process request to the server over the network connection, which is then processed and delivered to the client. A server can manage several clients simultaneously, whereas one client can be connected to several servers at a time, each providing a different set of services.

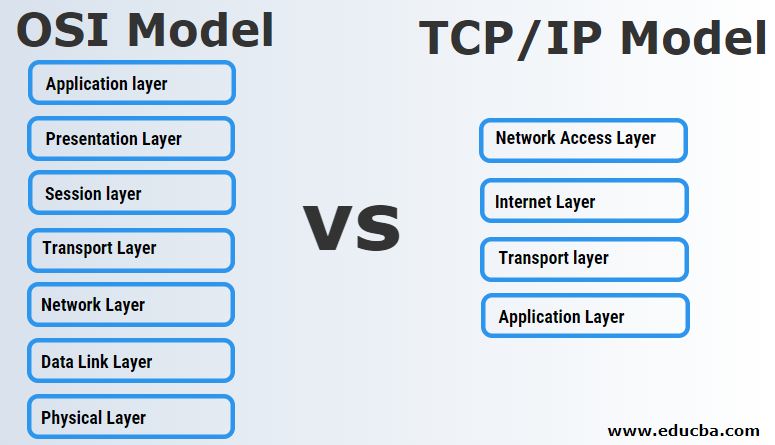
In its simplest form, the internet is also based on client - server architecture where web servers serve many simultaneous users with website data.

TCP

TCP stands for Transmission Control Protocol. It is one of the main protocols in TCP/IP networks. While the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and that packets will be delivered in the same order in which they were sent.

TCP is a connection-oriented protocol, which means that a connection is established and maintained until the application programs at each end have finished exchanging messages. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and — because it is meant to provide error-free data transmission — handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive.

In the Open Systems Interconnection (OSI) communication model, TCP covers parts of Layer 4, the Transport Layer, and parts of Layer 5, the Session Layer. TCP is considered the condensed version of the OSI Model.



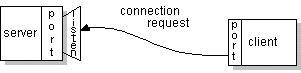
Sockets

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

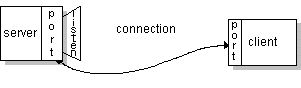
An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. In this way, it becomes possible to have multiple connections between your host and the server.

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. On the client side, the client knows the host name of the machine on which the server is running and the port number on which the server is listening.

To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and it also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

TCP allows communication between clients through the server. One client can send a message to another client by sending the command to the server, who then re-routes the message to the target client. In this way, two clients will never communicate directly, but intermediated by the server.

Three – Tier Architecture

A 3-tier architecture is a type of software architecture which is composed of three “tiers” or “layers” of logical computing. They are often used in applications as a specific type of client-server system. 3-tier architectures provide many benefits for production and development environments by modularizing the user interface, business logic, and data storage layers. Doing so gives greater flexibility to development teams by allowing them to update a specific part of an application independently of the other parts. This added flexibility can improve overall time-to-market and decrease development cycle times by giving development teams the ability to replace or upgrade independent tiers without affecting the other parts of the system.

Three - Tier architecture layers:

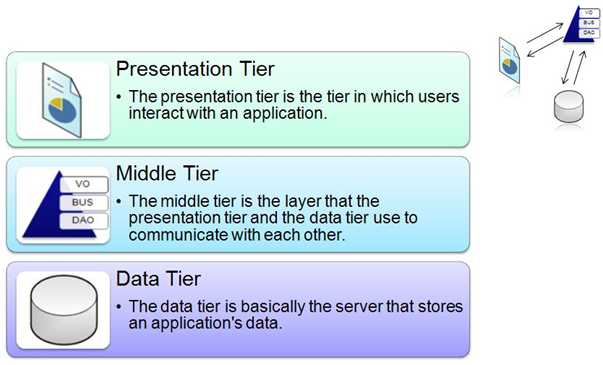
1. Application Layer or Presentation Layer

Presentation layer is the layer in which the users interact with an application. This layer contains UI part of our application. This layer mainly used for design purpose and get or set the data back and forth.

2. Business Access Layer (BAL) or Business Logic Layer (BLL)

Business layer is mainly working as the bridge between Data layer and Presentation layer. All the Data passes through the Business layer before passing to the presentation layer This layer contains business logic, calculations related with the data like insert data, retrieve data and validating the data.

3. Data Access Layer (DAL):

Data layer contents Database Tables, XML Files and other means of storing Application Data.

4. Implementation and Design

This application was implemented in the C# programming language using the .NET Framework as an WPF (Windows Presentation Foundation) project developed in Visual Studio 2017.

The design of the application is consistent with the Three – Tier Architecture presented in the Theoretical Fundaments Part of this documentation.

The entire solution was split into two projects, the first one representing the Server part and the second one serving as the Client. According to the Server – Client approach, where the Server acts as an intermediary between Clients, the development of the Data Tier in the Client was not necessary, as all the data contained in the Database is reached through the Server.

ChatServer.cs

The ChatServer class serves as the main mechanism of communication between the apps. It has multiple methods that use sockets to process messages between the server and the client

The Server communicates with the Client through 2 types of commands:

* “/setusername”
* “/msgto”

The ProcessMessages method decodes the command received to establish the usernames of the sender and the receiver, along with the message sent, then calls the SendMessage method that adds the log of the message to the Server console and then prompts the Client entity to send the message content through the socket.

private void ProcessMessages(Client client)

{

while (true)

{

if (client.IsConnected() == false)

{

Dispatcher.Invoke(() =>

{

Clients.Remove(client);

client.Dispose();

});

return;

}

var messageBytes = new byte[1024];

var receivedBytes = client.Socket.Receive(messageBytes);

var messageReceived = Encoding.Unicode.GetString(messageBytes);

if (receivedBytes <= 0) continue;

if (messageReceived.Substring(0, 8) == "/setname")

{

var newUsername = messageReceived.Replace("/setname ", "").Trim('\0');

client.Username = newUsername;

}

else if (messageReceived.Substring(0, 6) == "/msgto")

{

var data = messageReceived.Replace("/msgto ", "").Trim('\0');

var targetUsername = data.Substring(0, data.IndexOf(':'));

var messageToSend = data.Substring(data.IndexOf(':') + 1);

Dispatcher.Invoke(new Action(() =>

{

SendMessage(client, targetUsername, messageToSend);

}

), null);

Dispatcher.Invoke(new Action(() =>

{

SendMessage(client, client.Username, messageToSend);

}

), null);

}

}

}

The WaitForConnections method listens to the Socket for new clients, and if a new client is connected, it starts a new thread to process incoming messages from the new client, and then adds it to the list of current clients connected to the Server.

private void WaitForConnections()

{

while (true)

{

if (Socket == null)

{

return;

}

Client client = new Client

{

Id = ClientIdCounter,

Username = "Client" + ClientIdCounter,

Socket = Socket.Accept()

};

client.Thread = new Thread(() => ProcessMessages(client));

Dispatcher.Invoke(new Action(() =>

Clients.Add(client)), null);

client.Thread.Start();

ClientIdCounter++;

}

}

The Start method creates a new socket and binds it to the endpoint established by the user. Then, it creates a new thread that listens for new clients.

public void Start()

{

if (IsActive)

{

return;

}

Socket = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

Socket.Bind(IpEndPoint);

Socket.Listen(5);

Thread = new Thread(WaitForConnections);

Thread.Start();

Clients.Add(new Client() {Id = 0, Username = Username});

IsActive = true;

OnPropertyChanged(nameof(IsActive));

OnPropertyChanged(nameof(IsNotActive));

}

The Stop method erases the list of connected clients and disconnects then deletes the socket.

public void Stop()

{

if (!IsActive)

{

return;

}

while (Clients.Count != 0)

{

Client client = Clients[0];

Clients.Remove(client);

}

Socket.Dispose();

Socket = null;

IsActive = false;

OnPropertyChanged(nameof(IsActive));

OnPropertyChanged(nameof(IsNotActive));

}

ChatClient.cs

The ChatClient class is similar with the ChatServer class in the way that it Starts and Stops the client similarly to the Server.

The ReceiveMessages method is the communication mechanism between the Server and the Client. It receives an array of bytes from the Server through the socket and then decodes the message and adds it to the Client Console.

private void ReceiveMessages()

{

while (true)

{

byte[] informationBytes = new byte[1024];

try

{

if (!IsActive)

{

Dispatcher.Invoke(() => Disconnect());

return;

}

int receivedBytes = Socket.Receive(informationBytes);

if (receivedBytes > 0)

{

string message = Encoding.Unicode.GetString(informationBytes).Trim('\0');

Dispatcher.Invoke(() => Messages.Add(message));

}

}

catch (Exception)

{

Dispatcher.Invoke(() => Disconnect());

return;

}

}

}

The Send command method is the way the Client sends the two types of commands to the Server through the Socket.

private void SendCommand(string command)

{

Socket.Send(Encoding.Unicode.GetBytes(command));

}

ServerConnection.cs

The ServerConnection class manages the socket connection mechanism for the server and was introduced in order to respect the Single Responsibility Principle.

public class ServerConnection

{

private IPAddress IpAddr { get; set; }

public string IpAddress { get; set; }

public int Port { get; set; }

private IPEndPoint IpEndPoint => new IPEndPoint(IpAddr, Port);

public Socket Socket { get; private set; }

public bool IsActive { get; private set; }

public ServerConnection()

{

IpAddress = "127.0.0.1";

IpAddr = IPAddress.Parse(IpAddress);

Port = 5960;

IsActive = false;

}

public void Start()

{

if (IsActive)

return;

Socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

Socket.Bind(IpEndPoint);

Socket.Listen(5);

IsActive = true;

}

public void Stop()

{

if (!IsActive)

return;

Socket.Dispose();

Socket = null;

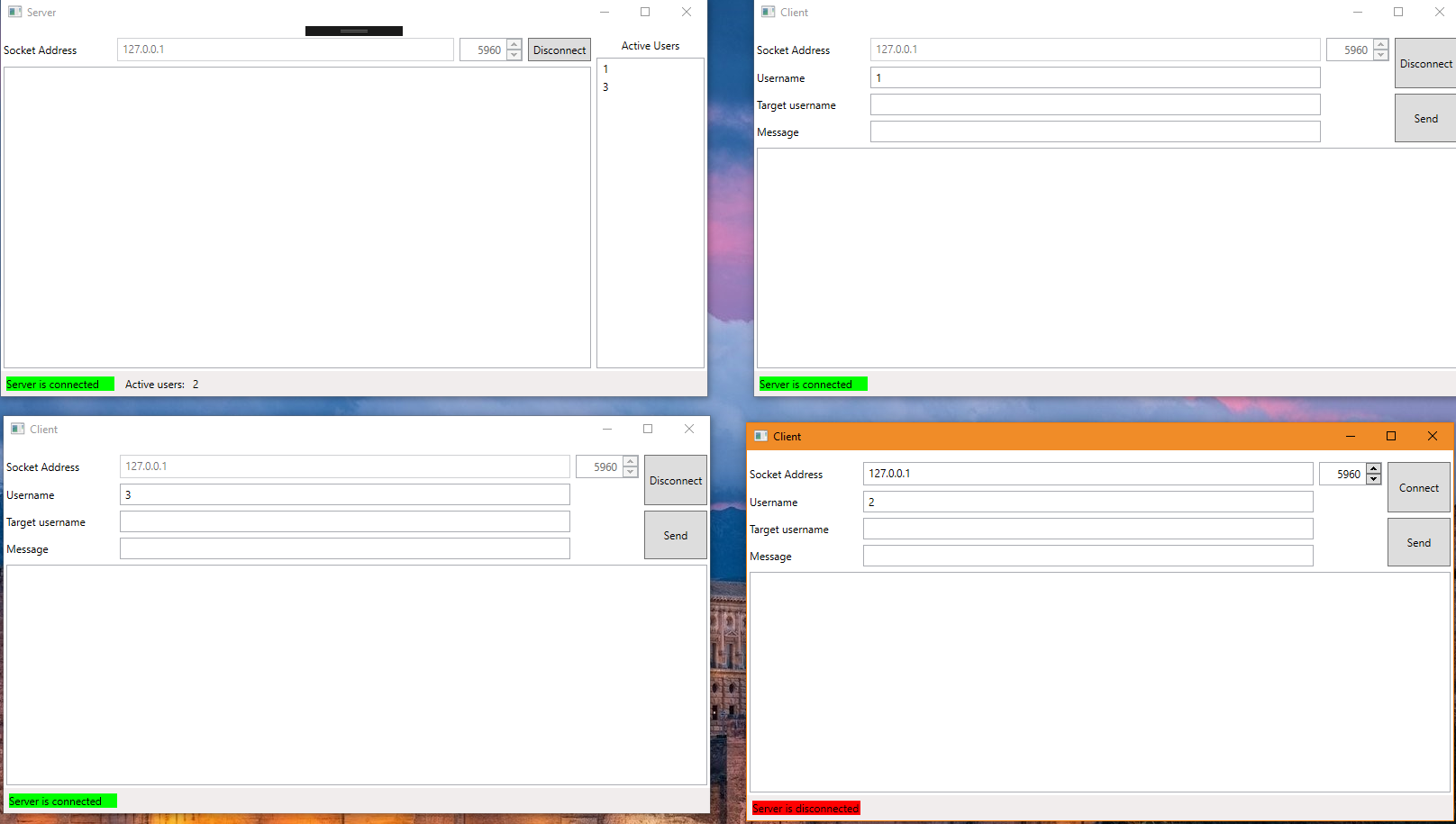
IsActive = false;

}

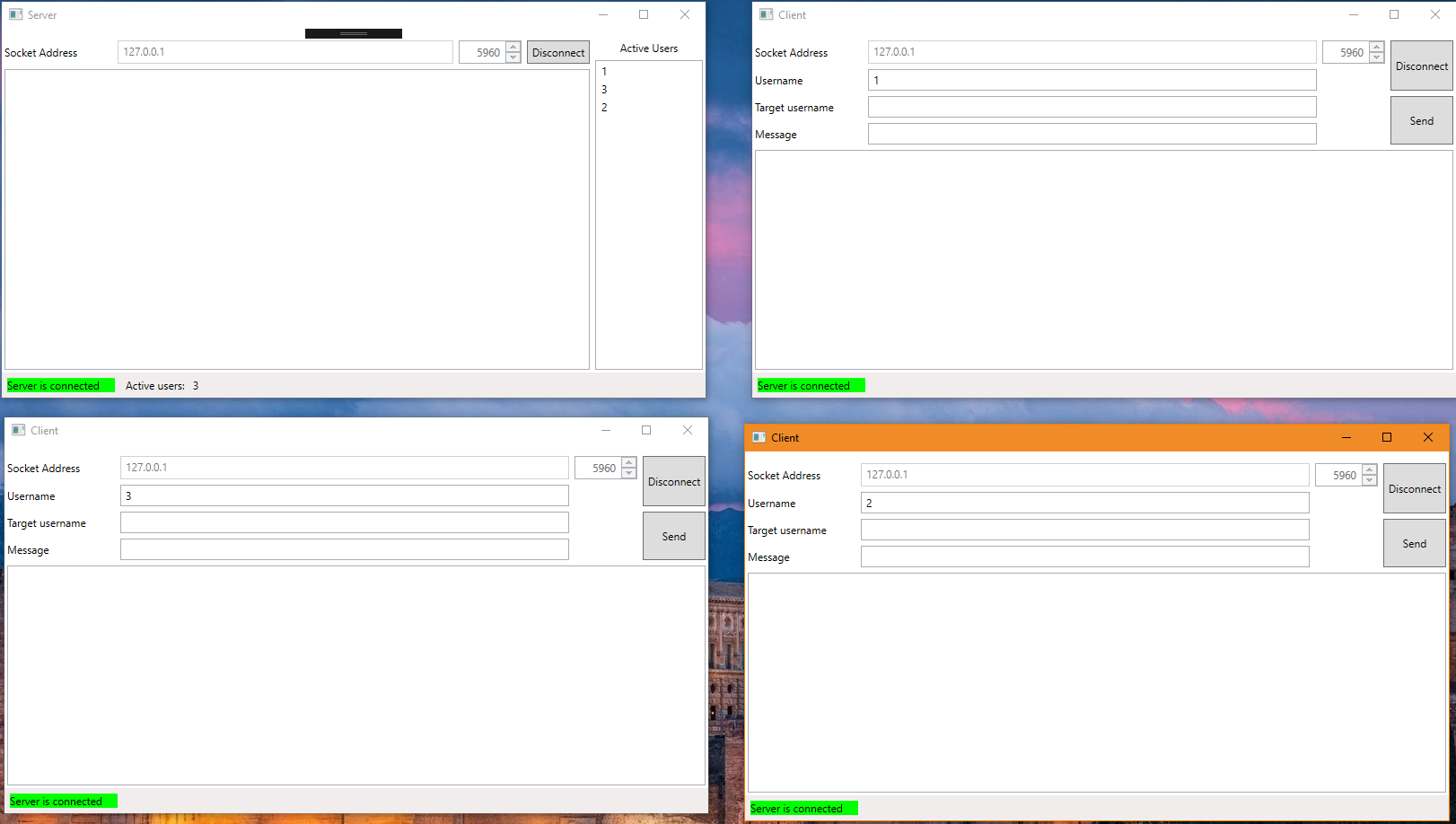
}

5. Testing

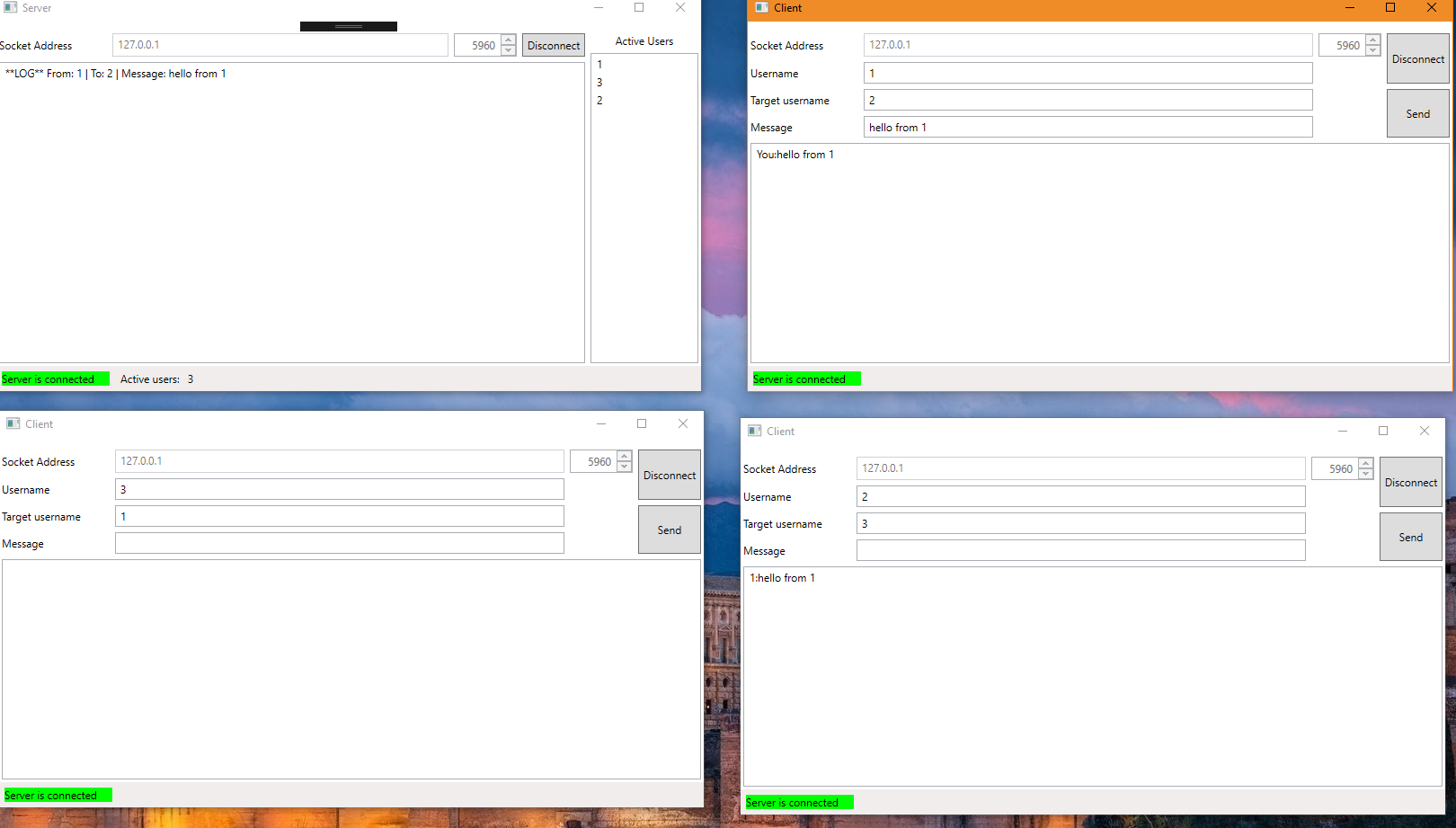
Happy flow Server – Client(s) connection testing, where one Server is active, three Clients are open, two of which are active:

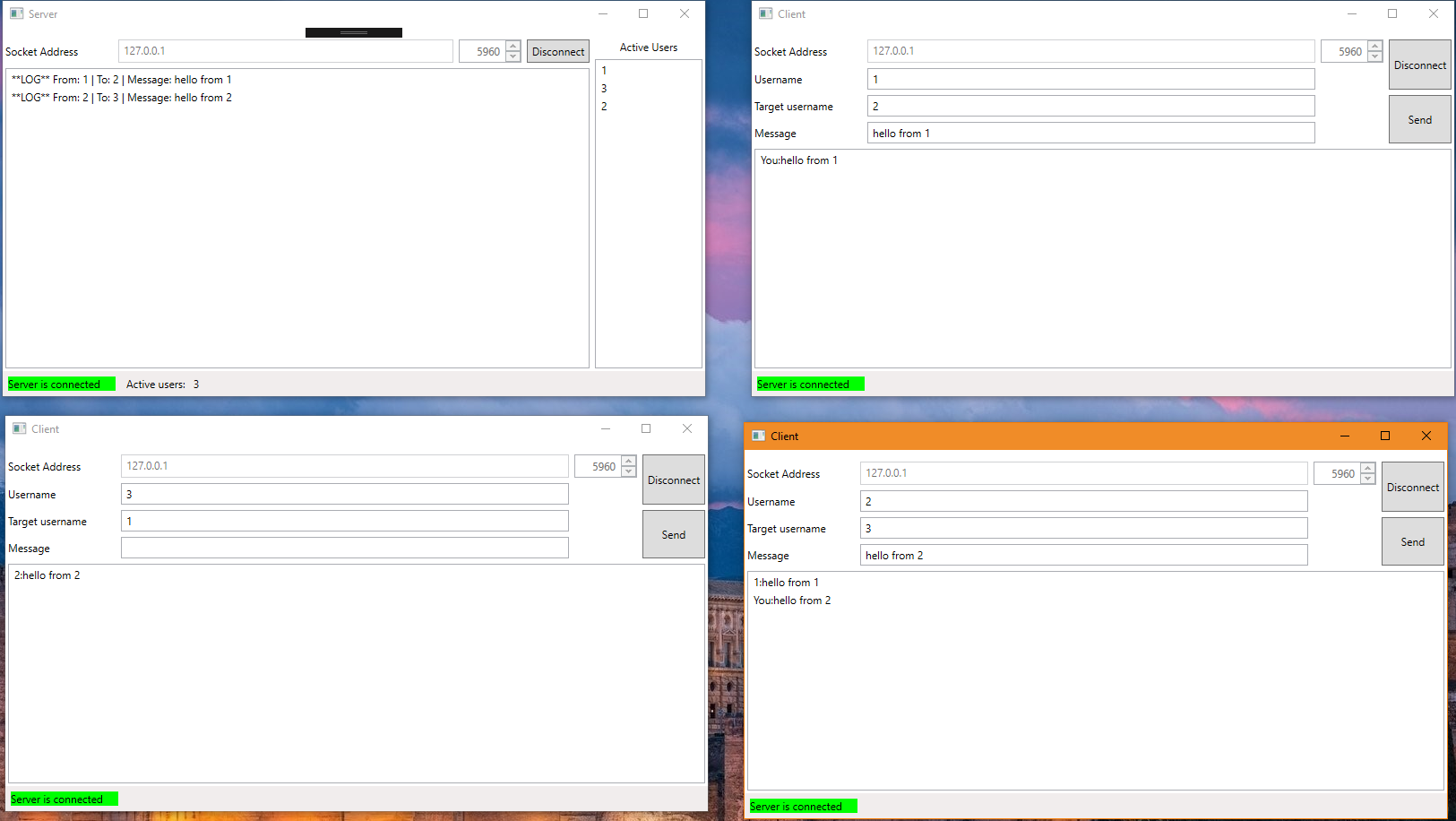


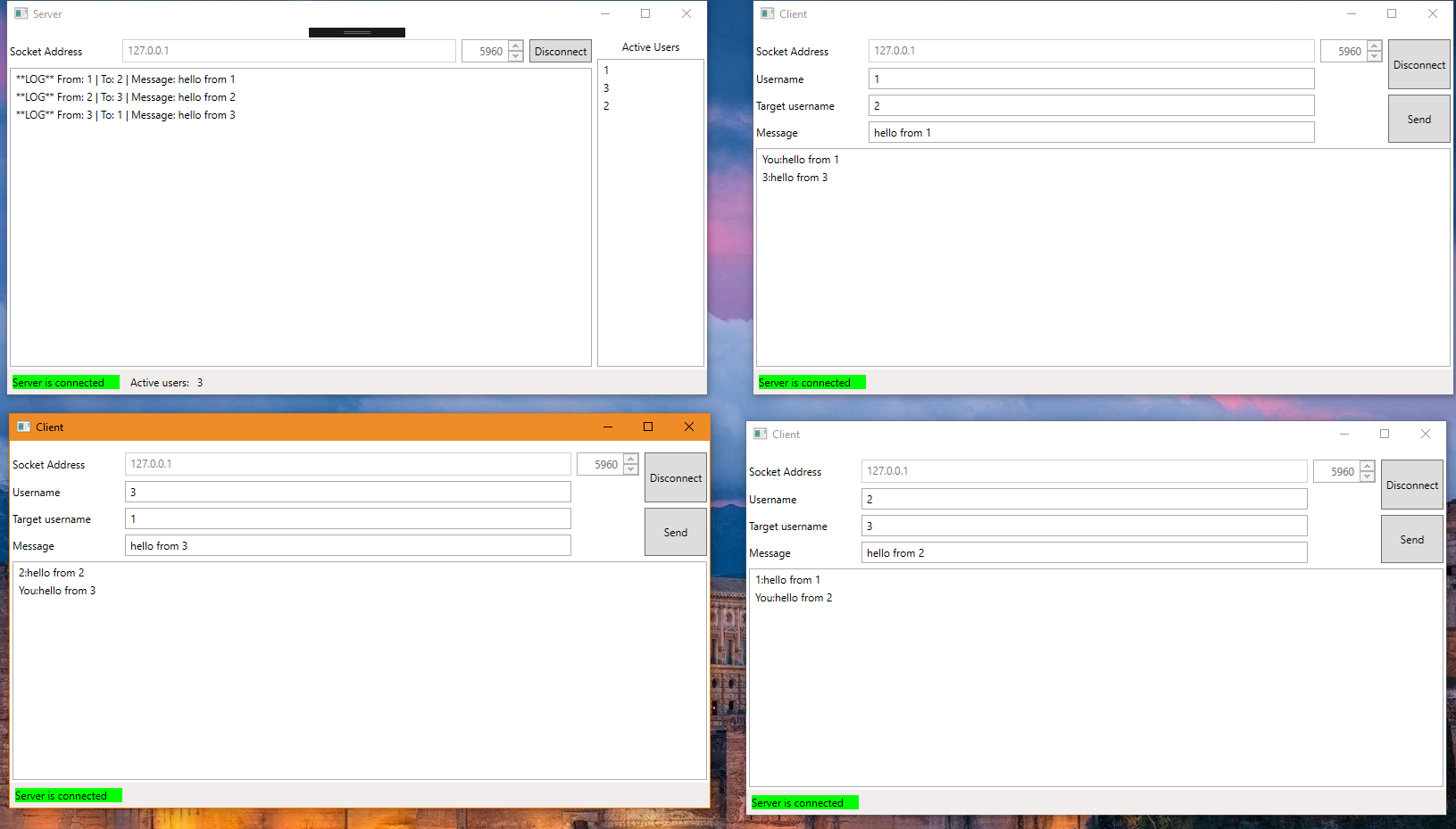
Happy flow Server – Client(s) connection testing, where one Server is active, three Clients are open, all of them being active:



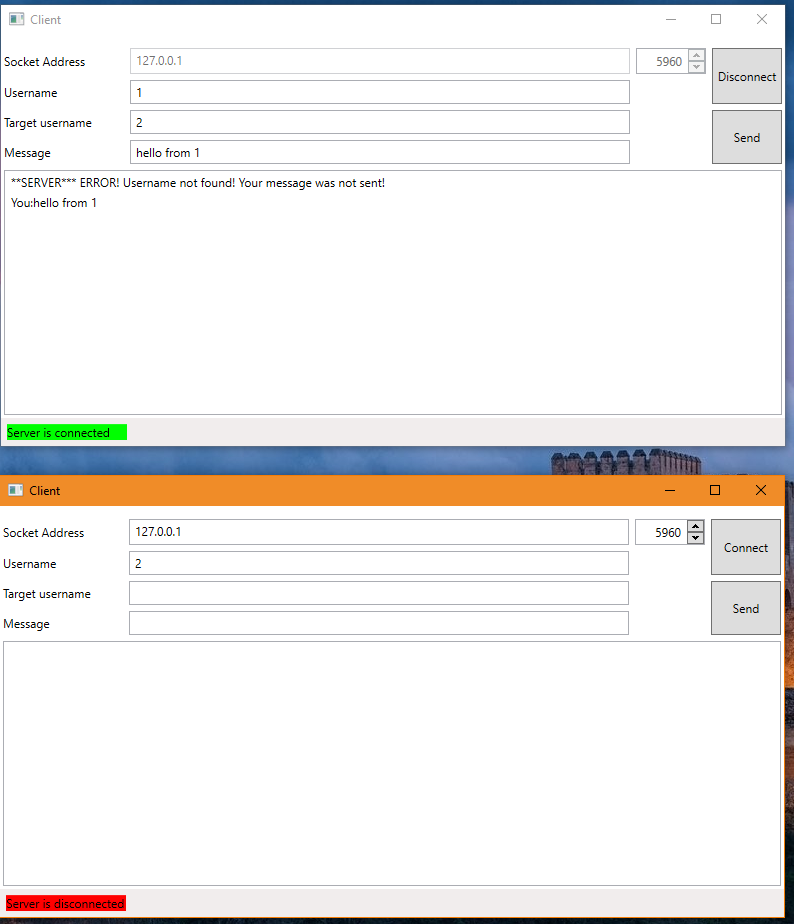
Happy flow, a clients sending messages to each other, while the Server shows message logs:



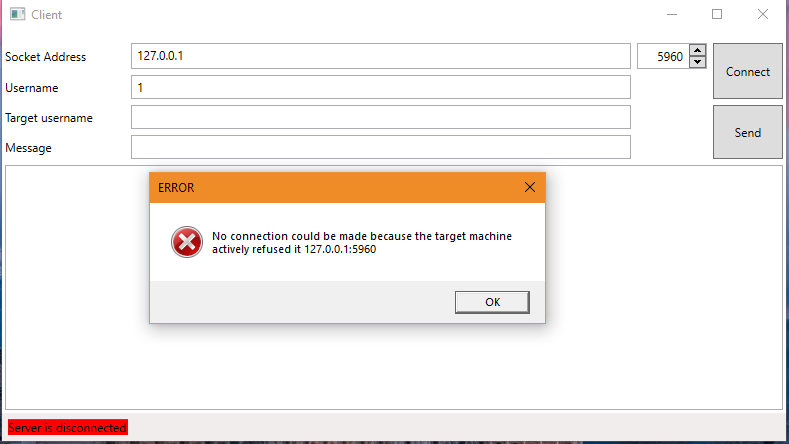




Error case, sending a message to an inexistent user testing:



Error case, trying to connect client when no server is active testing:



6. Conclusions And Further Development

Conclusions

To conclude, this application offered the opportunity to develop a client server type of program in C# and .NET Framework, and to get used to creating a Graphical User Interface for the Desktop Application, using WPF Framework, available for C# programs.

Further Development

This messenger application could be developed in the future to include a series of features to make it seem closer to a real-life scenario, which involves a significant amount of work and surpasses the purposes of this assignment. They include:

* The user entity
* User registration
* User login
* Saving messages and users in a database
* Loading a history of previous conversations

7. Bibliography

<http://www.networkcomms.net/how-to-create-a-client-server-application-in-minutes/>

<http://www.networkcomms.net/creating-a-wpf-chat-client-server-application/>

<https://www.winsocketdotnetworkprogramming.com/clientserversocketnetworkcommunication8b.html>

<https://www.c-sharpcorner.com/UploadFile/201fc1/creating-a-serversharp47client-application-using-only-tcp-prot/>

<https://www.geeksforgeeks.org/socket-programming-in-c-sharp/>

<https://www.geeksforgeeks.org/tcp-ip-model/>

<https://www.geeksforgeeks.org/layers-of-osi-model/>

<https://www.geeksforgeeks.org/basics-computer-networking/>

<https://www.techopedia.com/definition/438/clientserver-architecture>

<https://wpf-tutorial.com/about-wpf/what-is-wpf/>

<https://searchnetworking.techtarget.com/definition/TCP>

<http://loopcoder.com/pages/page/Understanding-Layer-architecture-in-C-SHARP>