PEER2PEER

Name: Danielle Frenklakh

ID: 322403155

School: Hakfar Hayarok

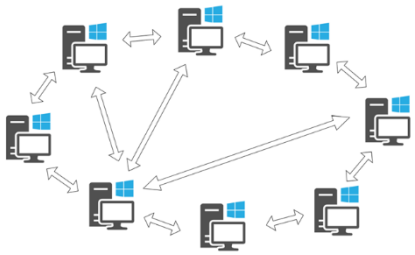
Year: 2018

Mentor: Yehuda Or

Table of Contents

Reflection

Throughout this year, I have decided to create a project, using technologies and languages I haven't used before. While deciding the final idea for the project, I thought about the concept of the project, and what are the features and main technologies I'm willing to learn and extend my knowledge in.
First, I decided to specify my ideas to networking between computers. After a couple ideas, I decided to make an application similar to team viewer, which already exists. The idea of this app set some questions in my mind, and I wanted to answer myself, by making such program.
The name of my application is p2p. The reason for this name is simple. P2p is short for peer to peer, which is a type of networking that is connection directly two computers, without no server or anything else in the middle. In my project, there is a direct connection between two peers, or computers, using peer to peer networking.



Basically, my project allows connecting two computers, by sharing screen and controlling mouse and keyboard events.

I am very happy by the product I have made, and the knowledge I gained while doing it.

Work organizing

In order to complete a project as major as this, it is very important to work organized, step by step. To do so, I have divided the work into iterations, and completed one after the other, in an organized way. The iteration I have divided to issues, posted in GitLab, each issue had a do date.

First iteration - server – handling data base.

- Research the options and the language to write the server.
- After finding, learn about the language.
- Deciding and creating a data base.
- Adding register, log in and sign out. (when registering and connecting, send ip address)
- Checking credentials while doing the register and log in features.

Second iteration – GUI- graphical user interface

- Make the basic user interface.
- Learn and make http requests to the server – register log in and sign out.

Third iteration – handling connection.

- Adding the option to connect – enter username and password of the user you are willing to connect.
- Make an option to connect in server – if the user exists, send back his ip address.
- Allowing connection option

Fourth iteration – mouse and keyboard events control

- Find the best way to capture the events in my computer.
- Make a basic connection using sockets between two computers.
- Send and receive events in the connected computers.

Fifth iteration – screen sharing

- Find what is the best way to share screen
- FFmpeg library was the best way – find how to use it in the project

Sixth iteration – connecting all the pieces together

- Open the mouse and keyboard event control that was written in a different project in c++, from the GUI
- Use the FFmpeg in the GUI.

Initiation document

General description:

For the project I decided to create a program which will enable controlling another computer from distance, though located in the same network area (LAN = local area network). The controlling includes two aspects - mouse events controlling and keyboard events. In addition, screen capturing and sharing among the computers.

Features and main processes:

- Graphical user interface.
- Connection with the server using HTTP requests.
- Database handling in the server.
- Home page that includes registration and login, while checking the credentials.
- The ability to make a first connection or allowing connection.
- Screen capturing and sharing with another PC (personal computer).
- Capturing and sending mouse and keyboard events.
- Handling from code with mouse and keyboard events.

Main technologies:

Node JS in order to write the server, while handling sqlite3 database.

C++ peer to peer connection in order to handle the mouse and keyboard events.

GUI (graphical user interface) in C#.

Gained knowledge

Before I started this project, from the technologies used in it, I faced only client-server networking in c++. After writing and finishing the project I gained a big amount of knowledge and experience in different technologies, programming languages and much more. While working on the project I have faced some difficulties in the way, but have managed to outcome them and succeed in making this project work.
First of all, the server that is written in Node JS, a language that I haven't met before I started working on the project. Before I decided to write the server as it is written now, I thought of many ways to complete this job. The biggest idea I had is to use django which is a free open source web framework, written in python, that its main use is to ease the creation of complex, database-driven websites. The idea was to create a website that handles a database, and constantly send http requests from the GUI (written in c#) in order

to post and get information. This caused security problems and I was unable to work with this technology, considering I wanted to make a windows application and not a website. Using Node JS in order to write the server was the easiest and the most effective way to complete the job.
To sum up, while writing the server I faced and learned two different and new technologies, which in the end only one of them was useful for this project. Furthermore, I focused and understood much more how the internet works, using http requests and different function such as GET and POST.

In addition, I used c# language in order to write the GUI (graphical user interface). Before working on this project, I faced this language a limited number of times.

Specification document

Features and main processes:

- *Graphical user interface* – The program is accessible, easy and comfortable to use. There are suitable text boxes to enter the username and password, and all the details needed to use this program.
- *Username and password* – every user who is interested to register and use the program, must register with a username and a password, each different username.
- Connection with the server using HTTP requests - Every time that it is required to handle the database: Register new user, while checking if a user with the same name does not exist and adding it to the database; Log in the system, while checking if the username exists and the password is suitable with the username; allow connection, which means changing the 'is_active' table in the DB from 0 to 1; make a connection, while checking if the user exists and returning his IP address in case it does; Log out, which means changing the 'is_active' table in the DB from 1 to 0, and disabling making a connection with him.
- Database handling in the server - The server is handling with sqlite3 database, for saving new users and their IPs', checking credentials while logging in, registering and trying to connect to another pc and allowing connection.
- Screen capturing and sharing - With the use of FFmpeg library, a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams, using SRTP protocol (Secure Real-time Transport Protocol) I am able to share my screen or capture the screen of the other computer.
- Mouse and keyboard events - The keyBoard and the mouse events are sent and received in a while(true) loop (until the program is shut). For example, while computer A is taking over computer B, the program in computer A captures his mouse and keyboard events and sends via socket to computer B, that handles and changes its mouse and keyboard events.
    - Mouse events- Computer A is sending it's mouse coordinates via socket to computer B, that is changing according to it his mouse coordinates using SetCursorPos function. If computer A pressed on the right or the left click in the mouse, it sets the same event in computer B using SendInput function.
    - KeyBoard events- If a key was pressed in computer A, its ascii value is sent via socket to computer B, that sets this key event to press on the same key using SendInput function.

Architecture document

Following is the explanation of the whole project and its architecture.

The server

The server is written in Node JS and it's function is to deal with the HTTP requests sent from the user interface, and handle the database.

Registration

First, in order to register to the system, the user must fill out the username he wishes and the password (twice for confirmation). If the two passwords match, the system sends HTTP request to the server. The server that is written in Node JS checks in the DB if a user with such a name exists. If it does, it sends to the user a response that it is not possible to register with this user. Else, if a user with such a name does not exist, it adds it to the DB.

Log in

In order to log in to the system, the user fills out his information, and the system sends a HTTP request to the server. The server opens the DB and checks whether the username is suitable with the password. If it does, the server responses to the user and allows him to log in.

Sign out

If the user is willing to stop the ability of other users connect to him and control him, he presses on the sign out button. While pressing on this button, HTTP request is sent to the server, and the 'is_active' field is turning back to 0.

DB

The dataBase contains four columns; The username, password, ip_Address and is_active, which means if the user allows other users to "control" him. When the user first registers or logs in to the system, the is_active field is set to 0. Only if the user pressed the 'allow connection' button in the graphical user interface, the is_active field is set to 1, and means that the user is ready to be controlled by another user.

Connecting

In order to make a connection between two computers, and cntrolling one the other, the user fills out the other computer's details and sends an HTTP request to the server with its details. The server checks if the user is active and if it does, sends back his ip address.

Screen sharing

The screen sharing is succeeding with the use of FFmpeg library. FFmpeg is a free software project, the product of which is a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams. In my project, After the connection

has been set, and the server confirmed the connection, A hidden cmd is opened in predominant computer that is receiving the streaming and in the controlled computer that is sending the streaming.

Event capturing and controlling

After a connection has made between the PCs', the networking between the two becomes peer to peer. The 'server' side is the controlled computer, that receives the events and changes them in his computer from code. The 'client' side is the predominant computer, that captures and sends via socket the mouse and keyboard events.

Sign out

If the user is willing to stop the ability of other users to connect to him and control him, he presses on the sign out button. While pressing on this button, HTTP request is sent to the server, and the 'is_active' field is turning back to 0.

The project

The project includes 3 main parts:

1. The server in Node JS.
2. Controlling mouse and keyboard events in c++.
3. Graphical user interface in c#.

Basic knowledge

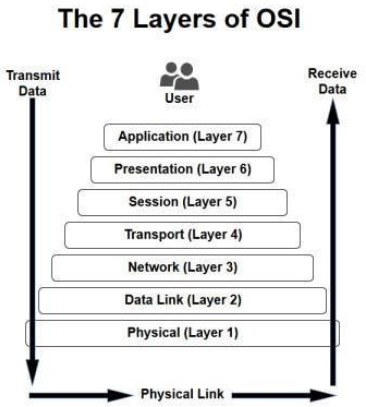
object-oriented programming (OOP)

object oriented programming or OOP is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as *methods.* A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated.

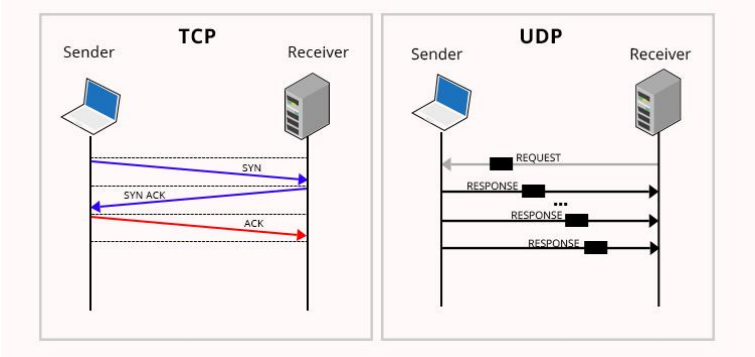
High and Low-level programming languages

Through different programming languages we are exposed to two different categories, high level and low level. Low-level languages are considered to be closer to computers. In other words, their prime function is to operate, manage and manipulate the computing hardware and components. Programs and applications written in a low-level language are directly executable on the computing hardware without any interpretation or translation. In contrast to this, High-level languages are designed to be used by the human operator or the programmer. They are referred to as "closer to humans". Their programming style and context is easier to learn and implement than low-level languages, and the entire code generally focuses on the specific program to be created. A high-level language does not require addressing hardware constraints when developing a program. However, every single program written in a high-level language must be interpreted into machine language before being executed by the computer.

OSI Model



Open Systems Interconnection model, a conceptual framework that describes the functions of a networking or telecommunication system. First, the seventh layer Is the layer closes to the user. This layer is the applications that interact directly with the user, such as we browsers and more. Next, the sixth layer, presentation layer. It represents the preparation or translation of application format to network format. In other words, the layer "presents" data for the application or the network. A good example is encrypting or decrypting data to secure in. next, session layer. This layer is used to allow servers or application to "speak" to each other. An example for a use of this layer is setting the time a system should wait for response, and more. The fourth layer in the transportation layer, this layer I will discuss more deeply. The transportation layer, as the name, deal with data transportation from the source to the destination. Defines How much data to send, at what rate, where it goes, etc. there are two known protocols used in this layer – UDP and TCP.
TCP – transmission control protocol - is a secure and safe protocol to transfer data, in contrast to UDP – user datagram protocol – which is less secure protocol.



What makes TCP more secure is that every time a packet is sent, it checks either there were complications in the way, while UDP is sending the data without checking the status of the data. Though UDP is less secure then TCP, it is faster, so that is an issue that must be considered when deciding what protocol to use.

Next network layer. this layer is responsible for packet forwarding, including routing through different routers. For example, if a computer in Israel is willing to connect to a computer in Australia, this layer routs the packets through the fastest rout to get to the destination. Fooling is the data link layer, mostly handles error correction from the physical layer. The last and first layer is the physical layer. This layer represents the electrical and physical representation of the system. This can include everything from the cable type, radio frequency link and more.

The server

Java Script - JavaScript, or in short JS, is a lightweight, interpreted compiled programming language. Most well-known as the scripting language for Web pages, many non-browser environments also use it, such as node.js, which I use in the project. JS is supporting object-oriented programming.
The server is written in Node JS which is an open source platform that executes java script server-side code, while usually java script is used for writing client side code, such as webpages and more. The main role of node JS is running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.

Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionalities, such as networking. Interesting fact about Node JS is that the commands are running concurrently or parallel. (concurrently = is a method in which commands are executed together instead of one after one).
In order to run the server, should open a command line, direct it to the place where the code is in and write app node.

The main part of the server in my project is handling the data base. Generally, a data base is an organized way to store related data. The data is stored in a table, using rows and columns. The data base I am using in my project is sqlite3. Why using a data base is better than just storing all the data in one file? The answer is simple. Loading data from the data base is much faster and easier. When using a file, in order to find the needed data, the application must load the entire file and holding it in memory, instead of just loading the needed data from the data base.

As said before, the server handles the data base, following http requests it gets from the client. Http is a protocol designed to enable communications between clients and servers, and works as a request-response between the two sides. The two main functions are GET and POST. GET is used in order to request information from the server and POST is in order to send and update data in the server.

About the data base

the db contains four columns; username, password, IP address and is active.

Username and password, as it is, are the username and password entered by the user in the register and are the details a user needs to enter when logging in.

Every time a user is logging into the system the program is sending the IP address of its computer and updates the IP column, in order to send it if a connection is made.

Every time a user I logging in, the ‘is active’ column is set to false. If the user presses the ‘allow connection’ button, the ‘is active’ column is set to true, means the connection is allowed.

Functions and commands is the server

The first function in the server is handling a register request. When a user is aiming to connect to the application, there are a few steps in the "background" in order to do so.

When the server receives the data, that contains the username, the password and the ip address, that the user chose, using the 'check_ register_ credentials' function, it checks either it is possible to register or not. The conditions to register a new user to the system is that the username is not used yet. The server is checking either the username that was registered was used before, using the command 'db.get("SELECT * FROM users WHERE username = ? ", [user_data[0]], function (err, rows)', when userdata[0] is the username. 'Db.get()' function is a way to handle and communicate with the data base from the server. This command returns in rows the object of the username. If !rows, means no user with such name exists, the function sends the user the character "T", that the user translates it to a permission to register, and the server is sending the user to 'insert_user()' function, to insert the user into the DB, db.run(`INSERT INTO users(username, password, ip_address, is_active) VALUES(?, ?, ?, ?)`, user_data that handles the DB and adds the new user with its values.

The following main function is the log in function. Basically, the functionality is the same as register, only in this part, the server is checking with the DB if the user exists and matches the password entered. If so, the server updates the 'ip_address' column, in case the user entered to the application from a different Wi-Fi area means the ip address was changed.

Next, allowing connection. In order to connect between two computers, the computer who is willing to be controlled should allow the connection. When a request as that is set, the server simply updates the 'is_active' column to true.

In order for a computer to connect to other, he must enter the details of the partner's user. Once a request to connect was sent to the server, the server checks either the username entered is suitable with the password, using the same command used in the register function, and if so, sends the ip address of the user to the one who asked for it, using 'res.send(row.ip_address)', command to enable the connection.

Finally, in order to disable the ability to connect to your PC, user must press sign out. Once a request to sign out is sent to the server, it updates the 'is_active' column to false.

The code

```
const express = require('express')

var net = require('net');

const app = express()

var sqlite3 = require('sqlite3').verbose()

var is_connected = false;

let db = new sqlite3.Database('./db/myDB.db', (err) => {

  if (err) {

        return console.error(err.message);

  }

});

var bodyParser = require('body-parser');

app.use(bodyParser.urlencoded({ extended: true }));

//handles post request to register

app.post('/register', function (req, res) {

        var user_data = [req.body.username,req.body.password, req.body.ip_address, false]//setting the user data to enter if confirmed to db

        console.log(user_data);//not necessary! Only to check if the user is the correct user entering to db

         check_credentials_register(req, res, user_data);//)//sends the user and the details from the request to check the credentials and the ability to register

})

//handles post request to login

app.post('/login', function (req, res) {

        console.log(req.body.username);////not necessary! Only to check if the user is the correct

          var user_data = [req.body.username,req.body.password, req.body.ip_address, false];//setting the user data to check in the db

        check_credentials_login(req, res, user_data);////sends the user and the details from the request to check the credentials and allow or not to the user to sign in

})
```

```
app.post('/allowConnection', function (req, res) {
	let data = [true, req.body.username];
	let sql = `UPDATE users SET is_active = ? WHERE username = ?`;//?`;//set the command to change the is_active field to 1 where the username is the user name sending the request
	db.run(sql, data, function(err) {////runs the command on the db
	if (!err) {
		res.send("cool")}
	})
})
//connecting to partner
app.post('/connect', function(req, res){
	 var user_data = [req.body.username,req.body.password];//set the user data of the user willing to connect to
	check_connection_allow(req, res, user_data);////sending to function to check if it is possible to connect to the user
	});
app.post('/logout', function (req, res) {
	console.log("disconnect");
	let data = [false, req.body.username];////sets values to the db command
	let sql = `UPDATE users SET is_active = ? WHERE username = ?`;////sets the db command to change the field is_active to 0, means not allowing to connect to this user
	db.run(sql, data, function(err) {//runs the command on the db
	if (!err) {
		res.send("cool")
	}



	})
```

```
function check_credentials_login(req, res, user_data){
  //check the user login data
  /*sends the command to the db and checks selects the user with the username and password sent*/
  db.get("SELECT * FROM users WHERE username = ? AND password = ?", [user_data[0], user_data[1]],function (err, rows) {
        if (!rows ){//if no user with this data
                console.log("No such user");
                res.send("F");//sends back to the user's interface and does not allow the connection
    }
        else {
                console.log("the username exists, u are allowed to login");
                res.send("T");//sends back to the user's interface and allow the connection
                let data = [user_data[2], user_data[0]];
                let sql = `UPDATE users SET ip_address = ? WHERE username = ?`;
                db.run(sql, data, function(err) {////runs the command on the db
                if (!err) {
                        res.send("cool")
                }})}});
}function check_credentials_register(req, res, user_data){
        //check the user register data
          db.get("SELECT * FROM users WHERE username = ? ", [user_data[0]], function (err, rows) {/*send the db a command to check if such user exists in it*/
        if (!rows){//if does not exist-- which is what we want
                console.log("user does not exist");
                res.send("T");//sending a confirmation to register and log in the system
                insert_user(user_data);//inserts to the database the user
        }
         else{//if such user exists
                console.log("user exist");
                res.send("F");//sending the user interface that the registration did not succeed
         }
    });
```

```
function check_connection_allow(req, res, user_data){/*checks if the user that we get in the request
is active and exists in order to connect to him*/
        db.get("SELECT * FROM users WHERE username = ? AND password = ? AND is_active = ?", [user_data[0],
        user_data[1], true],function (err, rows){
        /*sends the command to the db to check if such user exists and the password matches the username and if the
        user is allowing the connection*/
        if (!rows ){
                console.log("No such user");
                res.send("F");//sends to the user interface that the connection is failed
         }
        else {
                console.log("the username exists, u are allowed to connect");
                db.each('SELECT ip_address FROM users WHERE username=? AND password=?
                ',[user_data[0],user_data[1]], (err, row)=>{
                 //sends a command to the db to get the ip suitable to the user
                if(err){
                        throw err;
                }
                else{
                        res.send(row.ip_address);//sends to the user interface the ip of the user
                }
        })})}
/*This functions inserts the given user to the db with all its data: username, password, ip address and 0(because not
active yet)*/
function insert_user(user_data){
  db.run(`INSERT INTO users(username, password, ip_address, is_active) VALUES(?, ?, ?, ?)`, user_data, function(err) {
   if (err) {
     return console.log(err.message);
   }
 });}
```

Mouse and keyboard events control

The mouse and keyboard events control are written in c++/cpp is a high level programming language. This programming language combines the features of C language, such as direct memory management, byte manipulation, structures, headers, etc. and the features and advantages of high-level programming language, such as OOP, templates, overriding, etc.

First, in order to understand the functionality and the process in this part of the project, there are a couple of terms that must be marked.

Inheritance: Inheritance is one of the features in OOP programming language, that states a connection between two classes- a 'father' class and a 'son' class, when the child inherits some of the father's properties and functions. The usage of inheritance in code may minimize the code lines in a project and make it much more arranged.

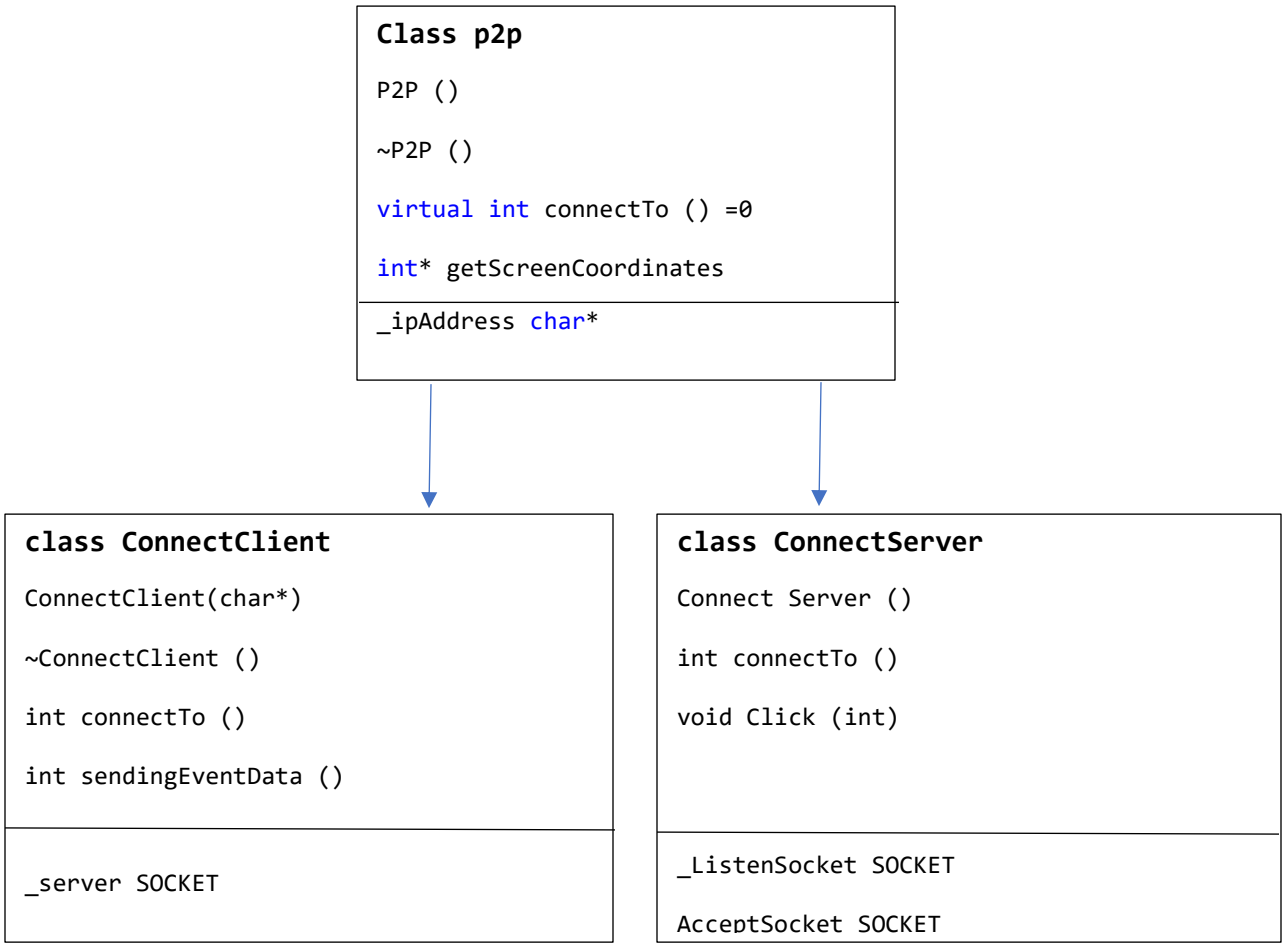
Virtual function: A virtual function is a member function which is declared within base class and is re-defined (Overridden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

Overriding: This method is taking place when a child class is declaring the same function as the father, means overriding it. The child's class will use the function declared in his class and not the father's.

Functions classes and commands

The connection between two computers in general is peer to peer, meaning the computers are connecting directly with each other, while the program is running. In order to explain the functionality, I will relate to one side as the server and the other as the client.

Classes



The program contains few classes. First, class p2p which is the father or the parent. It contains the virtual function connectTo(), that actualizes in the ConnectServer class and ConnectClient class, that are the "sons" of this class. This feature is called inheritance, when the children are taking the properties of this class. The fields that is defined in this class is char* _ipAdress which is the ip used to connect one side with the other.

connectServer class is the 'server' side of the connection. The class is overriding the connectTo() function that is defined in the p2p class, and the fields in this class are SOCKET _ListenSocket, which

is the socket that listens to connection in the server and SOCKET AcceptSocket that is the socket after the connection was accepted.
This object opens on the side of the controlled computer. Its main function is receiving in a while(true) loop the coordinates of the mouse, and the different events such as right and left click, and keyboard events from the predominant computer. Furthermore, it controls and changes the keyboard and the mouse events according to the input. The ConnectClient class is the 'client' side of the program. The field in this class is SOCKET _server that connects to the server. This object opens in the predominant computer. Its main function's act is to capture the mouse and keyboard events and send them to the 'server'. Later, will be discuss each class in particular.

Client

The client side is the predominant side. The mission of the client is to constantly send the mouse events, meaning the coordinates of it all the time, right and left click events, and the keyboard events, meaning the character that were pressed, to the server side. Following I will explain all the process "behind the scenes" from the simple connection between the computers, to the actual sending of the data.

It all stars as the predominant gets its permission to control the computer from the server and gets the controlled pc ip address. When calling the main function in this side we are receiving the ip using 'int main (char argc, char** argv)'. Argc and argv are the way to pass arguments from command line to main in c++. Argc means argument count and it is the amount of arguments being passed into the program from the command line and argv means argument vector, the array of the arguments (argc and argv are just the conventional names). As simply understood, the ip address is stored in argv [1].

Following the program creates a ConnectClient object in order to apply the client's functionality. The client is the side that connects to the server using his ip address. First, initializes the socket that will connect in the future to the server side '_server = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);' the socket is connected using TCP protocol which is a protocol in the transportation layer, designed to transport data securely and safely. Next, I define addr variable, that contains the details about the connection, such as the ip address and the port 'connect(_server, (SOCKADDR *)&addr, sizeof(addr));'. If the function has not failed, the connection is made.

Just before the actual sending data begins, there is a need to send the computer's coordinates, in order for the server to set the coordinates of the mouse in the correct proportion to the screen size. The problem in this procedure was that through a socket we are able to pass an array of chars, while the coordinates are Integer. The problem was the integer size is 4 bytes, while char in 1. In order to still manage to pass the integer through the socket, we are converting the integer to char.

Now, the actual data transportation starts. The sendingEventData function is the function that handles the data transportation. It runs in a loop that runs always if an event of mouse or keyboard on the computer has accrued. 'While (WaitForMultipleObjects (2, handles, FALSE, INFINITE))'. The next function is the function that reads the event from the computer. 'ReadConsoleInput (hStdInput, ir, 128, &nRead);'

Next, using switch case conditions, I set the functionality of each event, and the data transport. The first act whenever entering a case, is sending to the server 0 if key event or 1 if mouse event, in order for the server to relate to each case differently. bf is the buffer eventually sent to the server with the correct case pressed on the keyboard. 'Bf [0] = ir[i]. Event.KeyEvent. uChar.AsciiChar;' After this command, the value in bf [0] will be the ascii char pressed on the keyboard. Next, simply sending the char to the server.

In case of a mouse event, I define three different buffers, although it is not necessary and there is an option to send all at once, bufferX- the horizontal value, bufferY- the vertical value and bufferPress- the buffer that defines if mouse pressed or not - contain 0 if not pressed, 1 if left click and 2 if right click. The coordinates are captured using 'ir[i]. Event.MouseEvent. dwMousePosition.X/Y' and mouse press using 'ir[i]. Event. MouseEvent. dwButtonState & 0x07; '

Server

The server side is the controlled side. This class's main functionality is to receive the events of the client side and change them in his computer.

First, the server waits until a client connects to him. It runs since the user pressed allow connection button in the UI (user interface). Using '_ListenSocket = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);' The program creates a listening socket, to coming connections, in protocol TCP that I discussed in the client-side explanation. The variable serverAddr initializes the socket's properties, such as ip type and used port. The following function in the way to the connection is bind- associates a local address with a socket 'bind (_ListenSocket, (SOCKADDR*) & serverAddr, sizeof(serverAddr));' Next listens to the connection with the function listen 'listen (_ListenSocket, SOMAXCONN);' And accepts a socket using, creating a new accept socket that connects the two sides. SOCKET 'AcceptSocket = accept (_ListenSocket, (SOCKADDR*) &client, &clientSize);' Now the connection is done.

Next, data receiving and handling. In order to handle the coordination for the mouse movement, first we get the client's coordinates to set the correct proportions to the screen size. Next, the while(true) loop starts. The first receive is getting the type of the event. Now, we are handling each event differently.

Case keyboard event:

After getting the key that was pressed, now we need to set this event in the server's pc. In order to do so we are using the sendInput () function, that Synthesizes keystrokes, mouse motions, and button clicks. In this part I ran into some difficulties. First, when sent a character via socket, the character was printed twice, probably once the key is down and once up. In order to fix this problem, I have created a map that the key is a letter and the field is one or zero. Each time a character is pressed, if it was pressed the first time, the letter is set as a key and the field is one, or if the letter was set before, it changes into one. If the letter received the second time, meaning the field of this character is one, nothing changes except the field is becoming 0 again.

Second problem was sending different characters via socket and using them in code. This part of the code was taken directly from the web and was set to suit this project's values. This part was copies from two different websites, while one is working properly for letters and the second for other characters such as numbers, enter key and more. This is the reason why there are two cases, the first if the character is the letter and if not.

```
ip. Type = INPUT_KEYBOARD;
ip.ki. wScan = 0; // hardware scan code for key
ip.ki. time = 0;
ip.ki. dwExtraInfo = 0;
ip.ki. wVk = keyBoard [0]; // put the key event pressed in order to change the
event in the computer
INPUT input [2];
/*The computer sending the data is sending two characters when one pressed- one
for key up and one for key down. We need to print it in our computer once and that
is why we check if it is the first time or the second
using the map*/
ip.ki. dwFlags = 0; // 0 for key press
// This structure will be used to create the keyboard input event
ip.ki. dwFlags = 0;
if ((keyBoard [0] >= '0' && keyBoard [0] <= '9') || (keyBoard [0] >= 'A' &&
keyBoard z[0] <= 'Z') || (keyBoard [0] >= 'a' && keyBoard [0] <= 'z')){//if
letters

        input [0]. Type = INPUT_KEYBOARD; //define the event as keyboard event
        input [0].ki. wVk = 0;
        input [0].ki. wScan = keyBoard [0]; //the event that was pressed
        input [0].ki. dwFlags = KEYEVENTF_UNICODE;
        input [0].ki. time = 0;
        input [0].ki. dwExtraInfo = GetMessageExtraInfo ();
        input [1]. Type = INPUT_KEYBOARD;
        input [1].ki. wVk = 0;
        input [1].ki. wScan = keyBoard[0];
        input [1].ki. dwFlags = KEYEVENTF_UNICODE | KEYEVENTF_KEYUP;//means the key
        was pressed and up after pressing
```

```
            input [1].Ki. Time = 0;
            input [1].ki. dwExtraInfo = GetMessageExtraInfo ();

      SendInput((UINT)2, input, sizeof(*input)); //function that defines the event and
      changes it in the computer, now it's done!
}
else//if other chars
{
      SendInput (1, &ip, sizeof(INPUT));
}
```

Case mouse event:

The mouse events are divided into three different cases; mouse movement, left key press and right key press. The mouse movement is set using the function `SetCursorPos` that gets the coordinates from the client side and sets it in the correct proportion in contrast to the client's coordinates.

```
SetCursorPos (bufferX [0] * (horizontal / ((int)mouseCoord [0])), bufferY [0] * (vertical
/ ((int)mouseCoord [1]))) ;//set the mouse coordinates according to the correct ratio
```

The right and left clicks are set using the sendInput () function we met in case of a key event. Following is the function that handles the left and right clicks. in case of a left click the function gets 1, other is right click event.

```
INPUT    Input = { 0 };
                    // Create our input.
If (num == 1)
{
      Input.type = INPUT_MOUSE;
      // Let input know we are using the mouse.
      Input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN;
      // We are setting left mouse button down.
      SendInput(1, &Input, sizeof(INPUT));
      // Send the input.

      ZeroMemory(&Input, sizeof(INPUT));
      // Fills a block of memory with zeros.
      Input.type = INPUT_MOUSE;
      // Let input know we are using the mouse.
      Input.mi.dwFlags = MOUSEEVENTF_LEFTUP;
      // We are setting left mouse button up.
      SendInput(1, &Input, sizeof(INPUT));
      // Send the input.
}
else{
      Input.type = INPUT_MOUSE;
      // Let input know we are using the mouse.
      Input.mi.dwFlags = MOUSEEVENTF_RIGHTDOWN;
      // We are setting left mouse button down.
      SendInput(1, &Input, sizeof(INPUT));
      // Send the input.

      ZeroMemory(&Input, sizeof(INPUT));
      // Fills a block of memory with zeros.
      Input.type = INPUT_MOUSE;
      // Let input know we are using the mouse.
.
```

```
Input.mi.dwFlags = MOUSEEVENTF_RIGHTUP;
	// We are setting left mouse button up.
SendInput(1, &Input, sizeof(INPUT));
	// Send the input.
	}
```

The code

p2p.h

```
#ifndef P2P_H
#define P2P_H
#pragma comment(lib, "Ws2_32.lib")

#include <stdio.h> //C library to perform Input/Output operations
#include <iostream> //provides basic input and output services for C++ programs
#include <Windows.h> //contains declarations for all of the functions in the Windows API
#include <winsock.h> //Windows Sockets API
#include <stdexcept> //defines a set of standard exceptions that  the library and
programs can use to report common errors
#include <cstring> //defines several functions to manipulate C strings and arrays
#include <memory> //defines general utilities to manage dynamic memory
#include <map> //allows to use a map object and its functions

#include "gdiplus.h" //enables applications to use graphics and formatted text on both
the video display and the printer.
using namespace Gdiplus;
using namespace Gdiplus::DllExports;

#define PORT 3344 //defines the port uses to c onnect the two computers


class P2P{
public:
	P2P(); //constructor
	~P2P(); //diconstructor
	virtual int connectTo() = 0; //defines a virtual function that later will be
	actualized in the sons
	int* getScreenCoordinates();//gets the coordinates of the computer

protected:
	char* _ipAddress; //the ip of the 'server' side that the program is willing to
	connect (getting from the user interface
};


#endif
```

p2p.cpp

```
#include "P2P.h"
P2P::P2P(){
	_ipAddress = "";//initializes the ip address
}

P2P::~P2P() {}
int* P2P::getScreenCoordinates()//this function is later on showing the principle of
polymorphism because it sons are using this function
{
	int coordinates[2] = {};//the array that holds the coordinates in the end abd returns
	from the function
	int horizontal = 0;//the size of the horizontal
	int vertical = 0;//the size of the vertical
	RECT desktop; // this object stores the upper-left corner, width, and height of a
	rectangle
	const HWND hDesktop = GetDesktopWindow(); //  Get a handle to the desktop window
	//Get the size of screen to the variable desktop
	GetWindowRect(hDesktop, &desktop);
	// The top left corner will have coordinates (0,0)
	// and the bottom right corner will have coordinates (horizontal, vertical)
	horizontal = desktop.right;
	vertical = desktop.bottom;
	coordinates[0] = horizontal;
	coordinates[1] = vertical;
	return coordinates;

}
```

ConnectServer class

connectServer.h

```
#ifndef CONNECT_SERVER_H
#define CONNECT_SERVER_H

#include "P2P.h" //includes his father's header in order to inherit from him and get his
fields, function and includes
#include <sstream> //Header providing string stream classes
#include <cstdlib> //defines several general purpose functions, including dynamic memory
management, random number generation, communication with the environment, integer
arithmetics, searching, sorting and converting.

class ConnectServer :P2P{
public:
	ConnectServer();//constructor
	int connectTo();//overrides the virtual function of the father
	void LeftClick();//handles left click events
	void RightClick();//handles right click events
private:
	SOCKET _ListenSocket;//the 'servers' socket that listens to connections on the
specific port
	SOCKET AcceptSocket; //the socket after the connection was set
};
#endif
```

ConnectServer.cpp

```
#include "ConnectServer.h"
using namespace std;
#define IP "127.0.0.1"
ConnectServer::ConnectServer(){
	_ListenSocket = INVALID_SOCKET;//initializes the socket
	AcceptSocket = INVALID_SOCKET;}
int connectClientActivate(){
	return 0;}
//overrides the p2p virtual function and makes the basic connection between the two peers
int ConnectServer::connectTo()
{
	map<char, int> keyBoardEventMap;//defining the map in order to notice that the keyboard events
	are printed once
	// Initialize Winsock
	WSADATA wsaData;
	int iResult = 0;//In order to check errors in socket functions
	_ListenSocket = INVALID_SOCKET;
	sockaddr_in serverAddr;

	iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);// initiates use of the Winsock DLL by a process.
	if (iResult != NO_ERROR) {
		wprintf(L"WSAStartup() failed with error: %d\n", iResult);
		return 1;
	}
	_ListenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);    // Create a SOCKET for listening
	for incoming connection requests.
	if (_ListenSocket == INVALID_SOCKET) {
		wprintf(L"socket function failed with error: %ld\n", WSAGetLastError());
		WSACleanup();
		return 1;
	}
	// The sockaddr_in structure specifies the address family, IP address, and port for the socket
	that is being bound.
	serverAddr.sin_family = AF_INET;//can communicate with internet Protocol v4 addresses
	serverAddr.sin_addr.s_addr = INADDR_ANY;//the socket is bound to all local interfaces
	serverAddr.sin_port = htons(PORT);//Initializes to the port defined for the connection
	iResult = bind(_ListenSocket, (SOCKADDR*)& serverAddr, sizeof(serverAddr)); //This function
	associates a local address with a socket.
	if (iResult == SOCKET_ERROR) {
		wprintf(L"bind function failed with error %d\n", WSAGetLastError());
		iResult = closesocket(_ListenSocket);
		if (iResult == SOCKET_ERROR)
			wprintf(L"closesocket function failed with error %d\n", WSAGetLastError());
		WSACleanup();
		return 1;
	}
	// Listen for incoming connection requests on the created socket
	if (listen(_ListenSocket, SOMAXCONN) == SOCKET_ERROR){
		wprintf(L"listen function failed with error: %d\n", WSAGetLastError());
		closesocket(_ListenSocket);
		WSACleanup();
		return 1;
	}
	// Create a SOCKET for accepting incoming requests.
	SOCKET AcceptSocket;
	sockaddr_in client;
```

```
//----------------------
	// Accept the connection.
	int clientSize = sizeof(client);

	AcceptSocket = accept(_ListenSocket, (SOCKADDR*)&client, &clientSize);//accepting incoming
	requests
	if (AcceptSocket == INVALID_SOCKET)
	{
		wprintf(L"accept failed with error: %ld\n", WSAGetLastError());
		closesocket(_ListenSocket);
		WSACleanup();
		return 1;
	}
	else
	{
		wprintf(L"Client connected.\n");
	}
	//define and get the coordinates of the screen
	int horizontal = 0;
	int vertical = 0;
	int coordinates[2] = {};
	horizontal = getScreenCoordinates()[0];
	vertical = getScreenCoordinates()[1];
	char mouseCoord[8];
	long  client_horizontal;
	long  client_vertical;
	iResult = recv(AcceptSocket, mouseCoord, 8, 0);//receiving the clients coordinates
	if (iResult == SOCKET_ERROR) {
		printf("recv function failed with error: %d\n", WSAGetLastError());
		closesocket(_server);
		WSACleanup();
		return 1;
	}
	else
	{
		client_horizontal = mouseCoord[0] + (mouseCoord[1] >> 8);

		client_vertical = mouseCoord[4] + (mouseCoord[5] << 8) + (mouseCoord[6] << 16) +
		(mouseCoord[7] << 24);
	}

	char keyBoard[2];
	char check[2];//if keyboard event or mouse event
	INPUT ip;

	stringstream stream;
	while (true){ //loop for receiving events and handeling them
		iResult = recv(AcceptSocket, check, 2, 0);//check if the event is keyboard ot mouse
		if (iResult == SOCKET_ERROR) {
			printf("recv function failed with error: %d\n", WSAGetLastError());
			closesocket(_server);
			WSACleanup();
			return 1;
		}
		if (check[0] == '0')//if keyboard case
		{
			iResult = recv(AcceptSocket, keyBoard, 2, 0);//recieve the key pressed in the
			kyeboard
			if (iResult == SOCKET_ERROR) {
				printf("recv function failed with error: %d\n", WSAGetLastError());
				closesocket(_server);
				WSACleanup();
				return 1;}
			}
```

```
else
{

	if (keyBoardEventMap.count(keyBoard[0]) != 1)//if the key event was not defined yet in the map
	{
		keyBoardEventMap[keyBoard[0]] = 0;//set the letter as a key in the map
	}
	int key_code;//the hex keycode

	stream << (int)keyBoard[0];
	stream >> hex >> key_code;

	// Set up a generic keyboard event.
	ip.type = INPUT_KEYBOARD;
	ip.ki.wScan = 0; // hardware scan code for key
	ip.ki.time = 0;
	ip.ki.dwExtraInfo = 0;

	ip.ki.wVk = keyBoard[0];//put the key event pressed in order to change the event in the computer

	INPUT input[2];
	/*The computer sending the data is sending two characters
	when one pressed- one for key up and one for key down. we need to print it in our
	computer once and that is why we check if it is the first time or the second using the map*/
	if (keyBoardEventMap[keyBoard[0]] == 0)//check if the letter is sent the first time
	{
		ip.ki.dwFlags = 0; // 0 for key press
		// This structure will be used to create the keyboard input event
		ip.ki.dwFlags = 0;
		if ((keyBoard[0] >= '0' && keyBoard[0] <= '9') || (keyBoard[0] >= 'A' && keyBoard[0] <=
		'Z') || (keyBoard[0] >= 'a' && keyBoard[0] <= 'z'))//if letters or numbers
		{
			input[0].type = INPUT_KEYBOARD;//define the event as keyboard event
			input[0].ki.wVk = 0;
			input[0].ki.wScan = keyBoard[0];//the event that was pressed
			input[0].ki.dwFlags = KEYEVENTF_UNICODE;
			input[0].ki.time = 0;
			input[0].ki.dwExtraInfo = GetMessageExtraInfo();
			input[1].type = INPUT_KEYBOARD;
			input[1].ki.wVk = 0;
			input[1].ki.wScan = keyBoard[0];
			input[1].ki.dwFlags = KEYEVENTF_UNICODE | KEYEVENTF_KEYUP;//means the key was
			pressed and up after pressing
			input[1].ki.time = 0;
			input[1].ki.dwExtraInfo = GetMessageExtraInfo();
			SendInput((UINT)2, input, sizeof(*input));//function that defines the event and
			changes it in the computer, now its done!
		}
		else//if other chars
		{
			// input event.
			SendInput(1, &ip, sizeof(INPUT));
		}

		keyBoardEventMap[keyBoard[0]] = 1;//define that the character was printed once
	}
	else
	{
		keyBoardEventMap[keyBoard[0]] = 0;//the character skipped the printing and the next time
		it should
	}

}}
```

```
Else//mouse event(x,y)
{
	char bufferX[2];//the x coordinates
	char bufferY[2];//the y coordinates
	char bufferPress[2];//If mouse was pressed
	iResult = recv(AcceptSocket, bufferX, 2, 0);//Receive the x coordinates
	if (iResult == SOCKET_ERROR) {
		printf("recv function failed with error: %d\n", WSAGetLastError());
		closesocket(_server);
		WSACleanup();
		return 1;
	}
	iResult = recv(AcceptSocket, bufferY, 2, 0);//Receive the y coordinates
	if (iResult == SOCKET_ERROR) {
		printf("recv function failed with error: %d\n", WSAGetLastError());
		closesocket(_server);
		WSACleanup();
		return 1;
	}
	iResult = recv(AcceptSocket, bufferPress, 2, 0);//Receive the press event
	if (iResult == SOCKET_ERROR) {
		printf("recv function failed with error: %d\n", WSAGetLastError());
		closesocket(_server);
		WSACleanup();
		return 1;
	}

	SetCursorPos(bufferX[0] * (horizontal / ((int)mouseCoord[0])), bufferY[0] * (vertical /
	((int)mouseCoord[1])));//set the mouse coordinates according to the correct ratio

	if (bufferPress[0] == 001)//if got a click from the computer controlling
	{
		cout << "left click";
		Click(1);
	}
	if (bufferPress[0] == 002)//if got a click from the computer controlling
	{
		cout << "right click";
		Click(2);


	}

}}
iResult = closesocket(_ListenSocket);//closing the socket in the end
if (iResult == SOCKET_ERROR) {
	wprintf(L"closesocket function failed with error %d\n", WSAGetLastError());
	WSACleanup();
	return 1;
}
else
{
	std::cout << "\nClient disconnected" << std::endl;
}
WSACleanup();
return 0;
}
```

```
//the function enables the click events on the mouse
void ConnectServer::Click(int num)
{
	INPUT    Input = { 0 };// Create our input.
	if (num == 1)
	{
		Input.type = INPUT_MOUSE;   // Let input know we are using the mouse.
		Input.mi.dwFlags = MOUSEEVENTF_LEFTDOWN;  // We are setting left mouse button down.
		SendInput(1, &Input, sizeof(INPUT));// Send the input.
		ZeroMemory(&Input, sizeof(INPUT)); // Fills a block of memory with zeros.
		Input.type = INPUT_MOUSE;// Let input know we are using the mouse.
		Input.mi.dwFlags = MOUSEEVENTF_LEFTUP;// We are setting left mouse button up.
		SendInput(1, &Input, sizeof(INPUT));// Send the input.
	}
	else{
		Input.type = INPUT_MOUSE;// Let input know we are using the mouse.
		Input.mi.dwFlags = MOUSEEVENTF_RIGHTDOWN; // We are setting left mouse button down.
		SendInput(1, &Input, sizeof(INPUT));// Send the input.

		ZeroMemory(&Input, sizeof(INPUT)); // Fills a block of memory with zeros.
		Input.type = INPUT_MOUSE;// Let input know we are using the mouse.
		Input.mi.dwFlags = MOUSEEVENTF_RIGHTUP;// We are setting left mouse button up.
		SendInput(1, &Input, sizeof(INPUT));// Send the input.
	}
}
```

ConnectClient class

connectClient.h

```
#ifndef CONNECT_CLIENT_H
#define CONNECT_CLIENT_H
#include <string> //allows functions on strings
#include "P2P.h" //includes his father's header in order to inherit from him and get his fields,
function and includes
class ConnectClient : P2P
{
public:
	ConnectClient(char*);
//	~ConnectClient();
	int connectTo();//overrides the virtual function of the father
	int sendingEventData();//handelng and sending via socket the events
private:
	SOCKET _server;//the 'clients' socket that connects to the server in the same port
};
#endif
```

connectClient.cpp

```
#pragma comment(lib, "d3d9.lib")
#include "ConnectClient.h"
using namespace std;
ConnectClient::ConnectClient(char* ip)//sets in the constructor the ip address given
{
	_ipAddress = ip;
}
int connectServerActivate(){
	return 0;
}
int ConnectClient::connectTo()
{
	int iResult;//sets the variable to store the errors from the socket functions
	WSADATA WSAData;
	SOCKADDR_IN addr;
	HBITMAP pic = NULL;
	WSAStartup(MAKEWORD(2, 0), &WSAData);
	_server = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);//initialize the socket that connects to the
	'server' side
	//initialize and set the data of the socket
	addr.sin_addr.s_addr = inet_addr(_ipAddress);
	addr.sin_family = AF_INET;
	addr.sin_port = htons(PORT);

	if (connect(_server, (SOCKADDR *)&addr, sizeof(addr)) == INVALID_SOCKET)//connect function through
	the specific port and ip
	{
		wprintf(L"connect function failed with error: %d\n", WSAGetLastError());
		system("PAUSE");
		return 1;}
	else{
		std::cout << "Connected to _server!" << std::endl;//checking is the function succeded
	}
	int horizontal = 0;
	int vertical = 0;
	//getting the coordinates of my screen
	horizontal = getScreenCoordinates()[0];
	vertical = getScreenCoordinates()[1];
	//it is not possible to send int via socket because the size of int is bigger then char, and that
	is why we need to convert int to char array
	char mouseCoord_to_send[8];
	mouseCoord_to_send[0] = horizontal & 0xff;
	mouseCoord_to_send[1] = (horizontal >> 8) & 0xff;
	mouseCoord_to_send[2] = (horizontal >> 16) & 0xff;
	mouseCoord_to_send[3] = (horizontal >> 24) & 0xff;
	mouseCoord_to_send[4] = vertical & 0xff;
	mouseCoord_to_send[5] = (vertical >> 8) & 0xff;
	mouseCoord_to_send[6] = (vertical >> 16) & 0xff;
	mouseCoord_to_send[7] = (vertical >> 24) & 0xff;
	iResult = send(_server, mouseCoord_to_send, (int)strlen(mouseCoord_to_send), 0);//sending the char
	array with the coordinates to the other computer
	if (iResult == SOCKET_ERROR) {
		cout<<"send failed: %d\n", WSAGetLastError();
		closesocket(_server);
		WSACleanup();
		return 1;}
```

```
sendingEventData();
	//coordinations(_server);
	printf("danielle mefageret");
	iResult = closesocket(_server);
	if (iResult == SOCKET_ERROR) {
		wprintf(L"closesocket function failed with error %d\n", WSAGetLastError());
		WSACleanup();
		return 1;
	}
	else
	{
		std::cout << "Client disconnected" << std::endl;
	}

	//WSACleanup();
	std::cout << "Socket closed." << std::endl << std::endl;
	system("PAUSE");

	return 0;
}
```

```
//This function is capturing the events in this computer and sending them via socket to the other in order
to eventually control it
int ConnectClient::sendingEventData()
{
	HANDLE hStdInput, hStdOutput, hEvent;
	INPUT_RECORD ir[128];//stores the event tipe and event itselfe
	DWORD nRead;//the number of input records read
	COORD xy;
	UINT i;
	hStdInput = GetStdHandle(STD_INPUT_HANDLE);//A handle to the console input buffer.
	hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);//A handle to the console output buffer.
	FlushConsoleInputBuffer(hStdInput);//Flushes the console input buffer. All input records currently in
	the input buffer are discarded.
	hEvent = CreateEvent(NULL, FALSE, FALSE, NULL); /*the function creates an auto-reset event object,
	the initial state of the event object is nonsignaled,  the event object is created without a name.
	the event gets a default security descriptor.*/
	HANDLE handles[2] = { hEvent, hStdInput };
	char bf[2] = { '1', '\0'};
	int iResult;
	char checkBuffer[2] = { '1', '\0' };//buffer that his value is 0 if the event is keyboard or 1 for
	mouse event
	while (WaitForMultipleObjects(2, handles, FALSE, INFINITE)){//Waits until one or all of the specified
	objects are in the signaled state or the time-out interval elapses
		ReadConsoleInput(hStdInput, ir, 128, &nRead);//Reads data from a console input buffer and
		removes it from the buffer.
		for (i = 0; i < nRead; i++)
		{
			switch (ir[i].EventType)//gets the event type
			{
			case KEY_EVENT:
				checkBuffer[0] = '0';
				iResult = send(_server, checkBuffer, 2, 0);//sends via socket that we are
				hendeling with key event
				if (iResult == SOCKET_ERROR) {
					cout << "send failed: " << WSAGetLastError();
					closesocket(_server);
					WSACleanup();
					return 1;
				}
				if (ir[i].Event.KeyEvent.wVirtualKeyCode == VK_ESCAPE)
					SetEvent(hEvent);
```

```
else
{
	xy.X = 0; xy.Y = 0;
	SetConsoleCursorPosition(hStdOutput, xy);//Sets the cursor position in the specified console screen
	buffer.
	bf[0] = ir[i].Event.KeyEvent.uChar.AsciiChar;

	iResult = send(_server, bf, 2, 0);//sending the key event
	if (iResult == SOCKET_ERROR) {
		cout << "send failed: "<< WSAGetLastError();
		closesocket(_server);
		WSACleanup();
		return 1;
	}
}
break;
case MOUSE_EVENT:
	checkBuffer[0] = '1';
	iResult = send(_server, checkBuffer, 2, 0);//sending via socket that we handle mouse event
	if (iResult == SOCKET_ERROR) {
		cout << "send failed: " << WSAGetLastError();
		closesocket(_server);
		WSACleanup();
		return 1;
	}
	xy.X = 0, xy.Y = 1;
	SetConsoleCursorPosition(hStdOutput, xy);//Sets the cursor position in the specified console screen
	buffer.
	char bufferX[2] = { '1', '\0' };
	char bufferY[2] = { '1', '\0' };
	char bufferPress[2] = { '1', '\0' };

	int x = ir[i].Event.MouseEvent.dwMousePosition.X;
	bufferX[0] = (char)x;
	iResult = send(_server, bufferX, 2, 0);//sends the X position of the mouse when (x,y) are the
	coordinates
	if (iResult == SOCKET_ERROR) {
		cout << "send failed: " << WSAGetLastError();
		closesocket(_server);
		WSACleanup();
		return 1;
	}

	int y = ir[i].Event.MouseEvent.dwMousePosition.Y;
	bufferY[0] = (char)y;//sends the Y position of the mouse when (x,y) are the coordinates
	iResult = send(_server, bufferY, 2, 0);
	if (iResult == SOCKET_ERROR) {
		cout << "send failed: " << WSAGetLastError();
		closesocket(_server);
		WSACleanup();
		return 1;
	}
	int press = (int)ir[i].Event.MouseEvent.dwButtonState & 0x07;
	bufferPress[0] = (char)press;
	iResult = send(_server, bufferPress, 2, 0);//sends 1 if left click, 2 if right click, 0 if none
	if (iResult == SOCKET_ERROR) {
		cout << "send failed: " << WSAGetLastError();
		closesocket(_server);
		WSACleanup();
		return 1;
	}
	break;}}};
	return 0;}
```

The ‘client’ side, which is the dominant computer has a different main from the ‘server’ side which is the controlled computer.

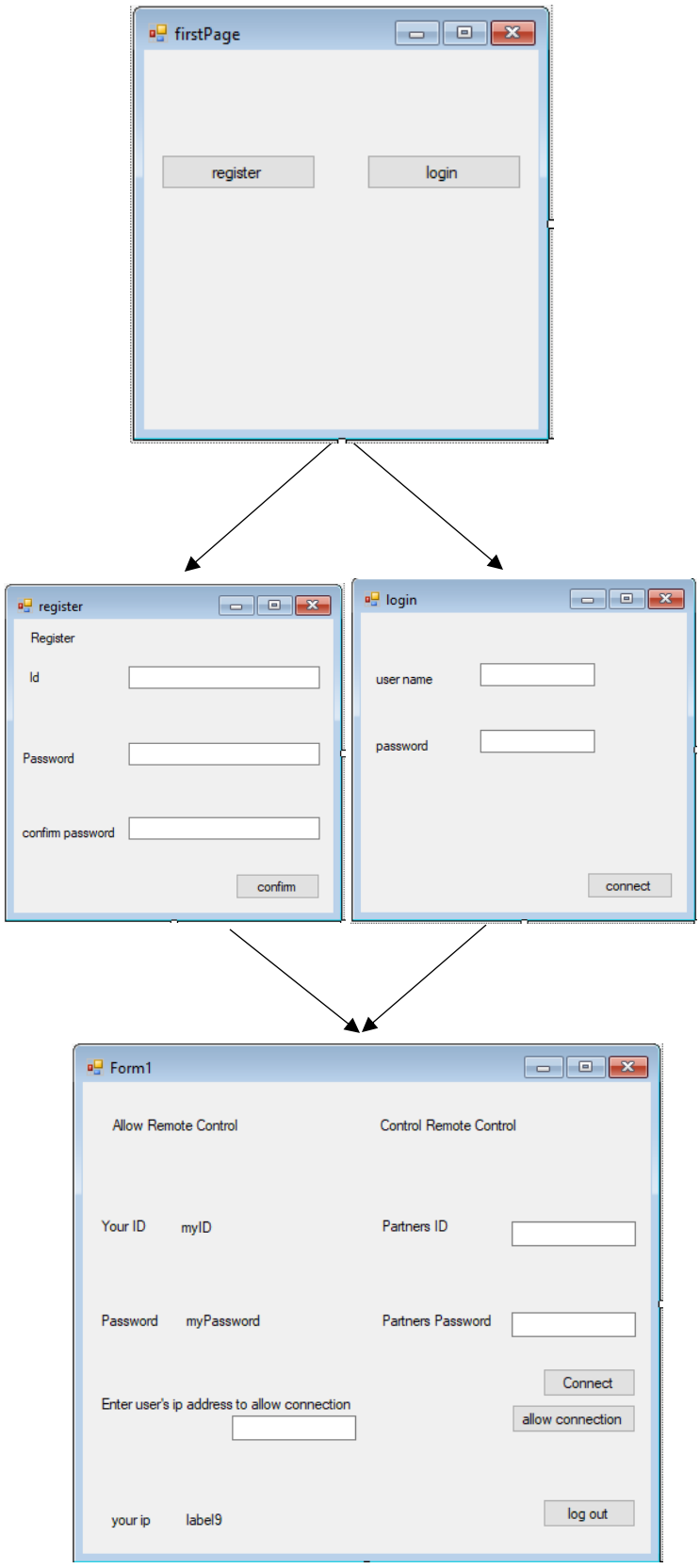
The server’s main.cpp

```
#pragma comment(lib, "Ws2_32.lib")
#include <stdio.h>
#include <iostream>
#include <Windows.h>
#include <winsock.h>
#include <thread>

#include "ConnectClient.h"
#include "ConnectServer.h"



using namespace std;

int main(char argc, char** argv)
{
	SOCKET *s = new SOCKET();
	ConnectServer *cc = new ConnectServer();
	cc->connectTo();

	system("PAUSE");
	return 0;
}
```

The client’s main.cpp

```
#pragma comment(lib, "Ws2_32.lib")
#include <stdio.h>
#include <iostream>
#include <Windows.h>
#include <winsock.h>
#include <thread>
#include "ConnectClient.h"
#include "ConnectServer.h"
using namespace std;

int main(char argc, char** argv)//The system is getting the ip address from the user interface as
starter variables in argv
{
	char* ip = argv[1];
	cout << ip;
	SOCKET *s = new SOCKET();
	ConnectClient *cc = new ConnectClient(ip);
	cc->connectTo();

	system("PAUSE");
	return 0;}
```

GUI


firstPage
register
login
register
Register
Id
Password
confirm password
confirm
login
user name
password
connect
Form1
Allow Remote Control
Control Remote Control
Your ID
myID
Partners ID
Password
myPassword
Partners Password
Connect
Enter user's ip address to allow connection
allow connection
your ip
label9
log out

The graphical interface, or GUI, is written in c# language as a windows application. The language is intended to be a simple, modern, general-purpose, object-oriented programming language. C# is intended to be suitable for writing applications for both hosted and embedded systems.

As the name states, the code is used for creating the UI which is the user interface- what the user sees while using the windows application, considering all the other parts are “behind the scenes”.

Now I will explain about every step in this part and the usage of each one of it.

Once entering the application, the first page that is shown is the option either to login or register to the system. As simple as it is, if the user entered the program for the first time, the registration option is the one correct for him. If the user already has a username and a password, the option he must choose is the login.

After choosing the correct option the window chosen is the one to open. If the option that was chosen is the register, the required data is the username, password and a confirmation of the password, just to check if the user remembers the password he chose. The first condition, After pressing the confirm button, is set in this part, and it is checking either the confirmation password matches the first password filled, and if the fields are not empty.

```
if (ID.Text != "" && Password.Text != "" && (Password.Text == confirmpsw.Text))
```

If the passwords do match, the next confirmation is in the hands of the server as explained in the server part of the book. In order to commit a connection between the GUI and the server, there is a need to send a post HTTP request to the server, with the data of the user entered. The data in the GUI antered by the user is stored in a dictionary:

```
Dictionary<string, string> data = new Dictionary<string, string>()
            {
                {"username", ID.Text },
                { "password", Password.Text},
                {"ip_address",ipAddress}
            };
```

The server is adding to the db the ip address of the user’s pc, that is sent from the GUI. The ip address found using a simple function:

```
public static string GetLocalIPAddress()
{
       var host = Dns.GetHostEntry(Dns.GetHostName());
       foreach (var ip in host.AddressList)
       {
              if (ip.AddressFamily == AddressFamily.InterNetwork)
              {
                     return ip.ToString();
              }
        }
        throw new Exception("No network adapters with an IPv4 address in the system!");
}
```

The GET http request is sent to the server with the data of the user and waits until a confirmation or a denial Is sent back.

```
HttpHandler handler = new HttpHandler("http://"+ SERVER_IP +"/register/");
confirm = handler.Post(data).Result;
```

If the confirm variable is “T”, the required user was successfully entered to the DB, else, the user wasn’t allowed to enter, if the username already exists in the db.

If the option that was chosen is the login, the GUI sends an http request as the connect button was pressed (if the fields are not empty), with the same data as in the register case. The ip is sent in the login because there is a possibility the user entered to his account from another network, and the IP is different.

If the server return “T”, the user is allowed to login, meaning there is a user with the username entered, and the password matches the username.

Following, the home page or the main page is open. In this part there are two options; connecting to a different computer, of allowing a different computer to control you.

Before specifying on the next part, we should discuss a feature that is presented in high level programming languages; threads or multithreads. Basically, multithreads is a feature that allows a couple of commands run at the same time.

Connecting to another computer:

In order to connect to another computer, one must enter the username and password of the computer he is willing to connect to. Once pressing the connect button, a HTTP request is sent to the server, with the details of the user you are aiming to control. If the server allows the connection, the ip of the other computer is sent as a response to the client, and the connection is set. The connection is set in two different aspects, the desktop sharing and the mouse and keyboard control. The two are happening at the same time, using multithreading (explained above).

Mouse and keyboard controlling: this part is mostly controlled from the c++ project, though we need to somehow open the project from the GUI. The client_server.exe project (the client side) is added in the bin-debug folder, and is being opened from cmd, that is also

opened from code.

```
System. Windows. Forms.MessageBox. Show("cmd");

var process = new Process ();
var startInfo = new ProcessStartInfo ();

startInfo.FileName = "client_server.exe";
startInfo.Arguments = received_ip;
startInfo.UseShellExecute = true;

process. StartInfo = startInfo;
process. Start();

process. WaitForExit();
```

desktop sharing: the desktop sharing is accruing similarly but differently. in this case also a command line is being opened, and a command is written to it from code, but this time, a project is not being opened, but an actual command. In this part we use an library named FFmpeg.

FFmpeg is a free open source software project, that is used to stream audio, video and other multimedia files and streams. Designed for command-line-based processing of video and audio files.

The operation of opening the command line is almost the same as opening the project, although in this case we are aiming to hide the command line.

```
Process cmd = new Process();//open a new process
cmd.StartInfo.FileName = "C:\\Users\\Danielle\\Desktop\\ffmpeg\\ffmpeg-20180319-
e5b4cd4-win64-static\\bin\\";//set open ffmpeg where the location in the cmp
cmd.StartInfo.CreateNoWindow = false;
cmd.StartInfo.UseShellExecute = true;//set the cmd be hidden
cmd.StartInfo.FileName += "ffplay.exe";

string command = "-f mpegts udp://";
string ip = received_ip;
string port = ":4000";

cmd.StartInfo.Arguments = command + ip + port;
```

Allowing connection:

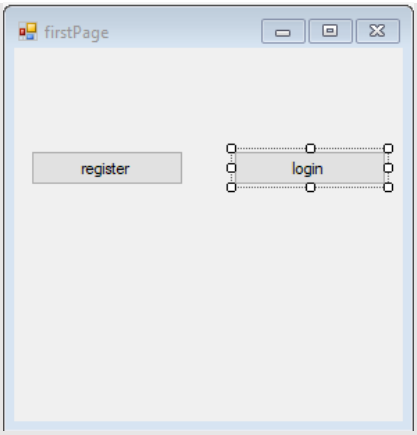
Once a user is willing to allow other user to control and connect him, he should press the allow connection button. Before pressing this button, he should enter the ip address of the other computer, for extra safety (The ip address is written on the homepage of every user). After entering the ip and pressing the button the connection is being allowed. The server_client.exe project is bring opened (in the same way as opened when connecting to other pc) and same library is used, but different command to share screen:`"-f gdigrab -i desktop -f mpegts udp:";`

Finally, logout option is also available, that once pressed the logout button, sending post HTTP request with the username, and the server is logging the username out.

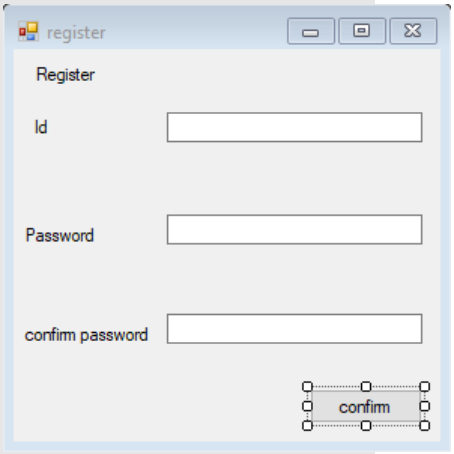
The code

firstPage.cs

```
using System;
using System.Windows. Forms;

namespace WindowsFormsApplication1
{
    public partial class firstPage: Form
    {
        public firstPage ()
        {
            InitializeComponent ();
        }

        private void register_Click (object sender, EventArgs e)
        {
            //open registration form
            register registerForm = new register ();
            registerForm.Show();
            this. Hide ();
        }

        private void login_Click (object sender, EventArgs e)
        {
            //open login form
            login log_in = new login ();
            log_in. Show ();
            this. Hide ();
        }
    }
```

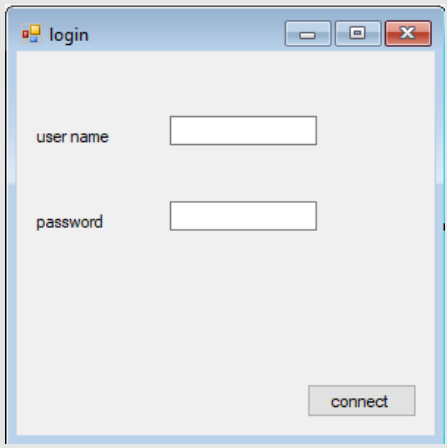


Register.cs



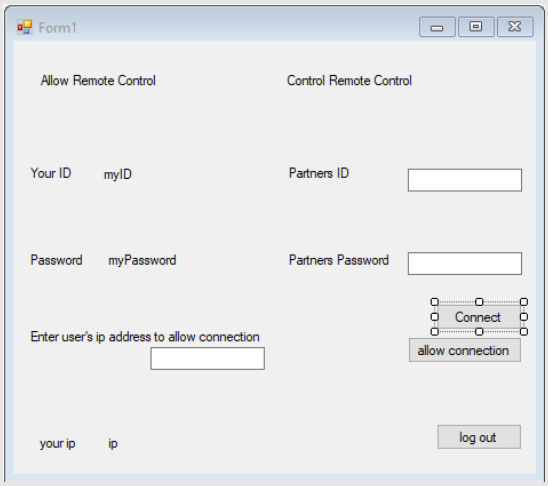

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

//using System.Web.Routing;
using Dns = System.Net.Dns;
using AddressFamily = System.Net.Sockets.AddressFamily;
namespace WindowsFormsApplication1
{
    public partial class register : Form
    {
        public string SERVER_IP = "localHost:3000";

        public register()
        {
            InitializeComponent();

        }
        private void confirm_button_Click(object sender, EventArgs e)
        {
            string ipAddress = GetLocalIPAddress();
            //check if the user connected to internet at all
            Dictionary<string, string> data = new Dictionary<string, string>()
            {
                {"username", ID.Text },
                { "password", Password.Text},
                {"ip_address",ipAddress}
            };
            string confirm = "";
            if (ID.Text != "" && Password.Text != "" && (Password.Text == confirmpsw.Text))
            {

                HttpHandler handler = new HttpHandler("http://"+ SERVER_IP +"/register/");
                confirm = handler.Post(data).Result;
                Console.WriteLine(confirm);


                //server confirmation check
                //RegisterRoutes(RouteTable.Routes);

                if(!confirm.Equals("F"))
                {
                    homePage hp = new homePage(ID.Text, Password.Text);
                    hp.Show();
                    this.Hide();
                }

            }
```

login.cs




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Net;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using Dns = System.Net.Dns;
using AddressFamily = System.Net.Sockets.AddressFamily;
namespace WindowsFormsApplication1
{
    public partial class login : Form
    {
        public string SERVER_IP = "localHost:3000";

        public login()
        {
            InitializeComponent();
        }
        private void connectBtm_Click(object sender, EventArgs e)
        {
            string ipAddress = GetLocalIPAddress();
            //check if the user connected to the internet at all
            Dictionary<string, string> data = new Dictionary<string, string>()
            {
                {"username", userNameBox.Text },
                { "password", passwordBox.Text},
                {"ip_address",ipAddress}
            };
            if(userNameBox.Text!="" && passwordBox.Text!="")
            {
                HttpHandler handler = new HttpHandler("http://"+ SERVER_IP +"/login/");
                string confirm = handler.Post(data).Result;
                //Console.WriteLine(taskResult);
                //if the username exist and the password match:
                if (!confirm.Equals("F"))
                {
                    homePage hp = new homePage(userNameBox.Text, passwordBox.Text);
                    hp.Show();
                    this.Hide();
                }
       }}
        public static string GetLocalIPAddress(){
            var host = Dns.GetHostEntry(Dns.GetHostName());
            foreach (var ip in host.AddressList)
            {
                if (ip.AddressFamily == AddressFamily.InterNetwork){
                    return ip.ToString();}
            }
            throw new Exception("No network adapters with an IPv4 address in the system!");
        }}
```

homepage.cs




```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Diagnostics;
using System.Net;
using System.Threading;
namespace WindowsFormsApplication1
{
    public partial class homePage : Form
    {
        public string SERVER_IP = "localHost:3000";
        static string id;//the username entered
        static string psw;//the password entered
        public homePage()
        {
            InitializeComponent();
            //initilizes the texts to display the data about the user, to the user
            myID.Text = id;
            myPsw.Text = psw;
        }
        public homePage(string new_id, string new_psw)
        {
            id = new_id;
            psw = new_psw;
            //initilizes the texts to display the data about the user, to the user
            InitializeComponent();
            myID.Text = id;
            myPsw.Text = psw;
            my_ip_address.Text = getMyIP();//sets to display the ip of the user on the screen

        }
        private void Form1_Load(object sender, EventArgs e)
        {
            //while not asked to connect by another user
            allowconnection.Enabled = true;
        }
       public string getMyIP()//function that gets the ip of the computer
        {
            var host = Dns.GetHostEntry(Dns.GetHostName());
            foreach (var ip in host.AddressList)
            {
                if (ip.AddressFamily == AddressFamily.InterNetwork)
                {
                    return ip.ToString();
                }
            }
            throw new Exception("No network adapters with an IPv4 address in the system!");
        }
```

```
private void connectBtm_Click(object sender, EventArgs e)//generates the ability to connect to
another pc
{
       Dictionary<string, string> data = new Dictionary<string, string>()//stores the data
       entered by the user
       {
              {"username", PartnerID.Text },
              { "password", PartnerPsw.Text}
       };
       HttpHandler handler = new HttpHandler("http://" + SERVER_IP + "/connect/");//the
       distanation of the request
       string received_ip = handler.Post(data).Result;//sending and receiving a response from
       the POST request
       if (received_ip != "F")//if the connction was confirmed
       {
              Thread cpp = new Thread(() => cpp_handler(received_ip, 1));//open the mouse and
              keyboard handeling project with the first thread
              cpp.Start();//start thread

              Thread ffmpeg = new Thread(() => ffmpeg_handler(received_ip, 1));//open the screen
              sharing in the second thread in order fpr them to run in the same time
              ffmpeg.Start();//start thread
        }


}

private void cpp_handler(string received_ip, int i)//handeling mouse and keyboard sharing
{
       var process = new Process();//opening a new process
       var startInfo = new ProcessStartInfo();
       if (i == 1)
       {
              startInfo.FileName = "client_server.exe";//setting the information of the process-
              the file that im willing to open
              startInfo.Arguments = received_ip;//get the argumants of the file
       }
       else
       {
              startInfo.FileName = "server_client.exe";
       }
       startInfo.UseShellExecute = true;//display the command line
       process.StartInfo = startInfo;
       process.Start();//start, open and run the command on cmd
       process.WaitForExit();
}
```

```
private void ffmpeg_handler(string received_ip, int check)//handeling the screen sharing
{
       Process cmd = new Process();//open a new process
       cmd.StartInfo.FileName = "C:\\Users\\Danielle\\Desktop\\ffmpeg\\ffmpeg-20180319-e5b4cd4-
       win64-static\\bin\\";//set open cmd
       cmd.StartInfo.CreateNoWindow = false;
       cmd.StartInfo.UseShellExecute = true;//set the cmd be hidden
       if (check == 1){
                cmd.StartInfo.FileName += "ffplay.exe";
                string command = "-f mpegts udp://";//server
                string ip = received_ip;
                string port = ":4000";
                cmd.StartInfo.Arguments = command + ip + port;
        }
        else{
                cmd.StartInfo.FileName += "ffmpeg.exe";
                string command = "-f gdigrab -i desktop -f mpegts udp:";//client
                string ip = IP.Text;
                string port = ":4000";
                cmd.StartInfo.Arguments = command + ip + port;
       }
       cmd.Start();
       cmd.WaitForExit();
        }
private void allowconnection_Click(object sender, EventArgs e)//allowing other computer to
connect yours
{
       if(IP.Text!=""){
              Dictionary<string, string> data = new Dictionary<string, string>()
              {
                    {"username", myID.Text }
              };
              HttpHandler handler = new HttpHandler("http://" + SERVER_IP +
              "/allowConnection/");//sending http request to allow the connection
              string getting = handler.Post(data).Result;
              Thread cpp = new Thread(() => cpp_handler("not important", 2));//open the ability
              to control the mouse and keyboard events
              /*the cpp_handler function is handeling the connection and the allowing conection
              cases. is case of a connection, the function is gtting the ip of the user
              he is connecting to, that is why the function id getting a string.
              if handeling allowness, no need of a string and that is why sending a not
              important string*/
              cpp.Start();
              Thread ffmpeg = new Thread(() => ffmpeg_handler(IP.Text, 0));//handeling scren
              sharing
              ffmpeg.Start();
            }
        }
        private void logout_Click(object sender, EventArgs e)//logout function{
            Dictionary<string, string> data = new Dictionary<string, string>(){
                {"username", myID.Text },
            };
            HttpHandler handler = new HttpHandler("http://" + SERVER_IP + "/logout/");
            string receive = handler.Post(data).Result;
            Console.Write(receive);
        }
```

process.cs- the main entry to the application

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace WindowsFormsApplication1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            handler = new ConsoleEventDelegate(ConsoleEventCallback);
            SetConsoleCtrlHandler(handler, true);
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new firstPage());
        }
        static bool ConsoleEventCallback(int eventType)
        {
            if (eventType == 2)
            {
                HttpHandler handler = new HttpHandler("http://localHost:3000/disconnect/");

                string taskResult = handler.Post(null).Result;
            }
            return false;
        }
        static ConsoleEventDelegate handler;   // Keeps it from getting garbage collected
        // Pinvoke
        private delegate bool ConsoleEventDelegate(int eventType);
        [DllImport("kernel32.dll", SetLastError = true)]
        private static extern bool SetConsoleCtrlHandler(ConsoleEventDelegate callback, bool
add);
    }
}
```

HttpHandler.cs

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Net.Http;
/// <summary>
/// Summary description for Class1
/// </summary>
namespace WindowsFormsApplication1
{
    public class HttpHandler
    {
        private String address = "";
        private static readonly HttpClient client = new HttpClient();

        public HttpHandler(String address)
        {
            this.address = address;
        }
        public void setAddress(String new_address){
            this.address = new_address;
        }

        public async Task<string> Post(Dictionary<string, string> data)
        {
            var content = new FormUrlEncodedContent(data);
            var response = await client.PostAsync(address, content).ConfigureAwait(false);
            var responseString = await response.Content.ReadAsStringAsync();
            return responseString;
        }
    }
}
```

Bibliography

Server - Node JS

https://www.w3schools.com/nodejs/nodejs_intro.asp

https://www.tutorialspoint.com/nodejs/index.htm

mouse and keyboard events – c++

http://www.cplusplus.com/doc/tutorial/

https://msdn.microsoft.com/en-us/library/windows/desktop/ms646310(v=vs.85).aspx

GUI – C#

https://stackoverflow.com/questions/3616010/start-command-windows-and-run-commands-inside

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/

concepts

https://www.networkworld.com/article/3239677/lan-wan/the-osi-model-explained-how-to-understand-and-remember-the-7-layer-network-model.html

Stack overflow questions

https://stackoverflow.com/questions/50524945/i-am-writing-a-simple-client-socket-application-but-after-the-connection-the-ser

https://stackoverflow.com/questions/50674707/im-trying-to-open-a-command-line-from-c-sharp-windows-application-in-order-to-ru