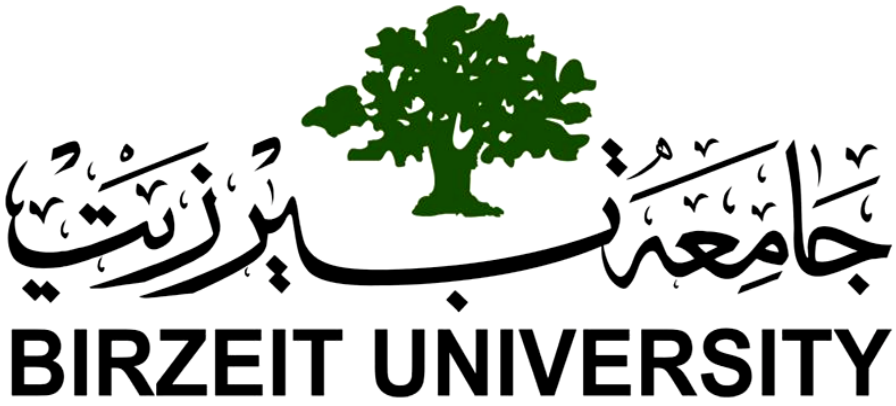


Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS3320-Computer Networks

Project#1

Socket Programming

Prepared by:
Hala Mohammaed 1210312
Diana Naseer 1210363
Rua Srour 1221727

Section:
Rua & Diana 1
Hala 2

Instructor:
Dr. Ibrahim Nemer

Date:
15/8/2024

Abstract

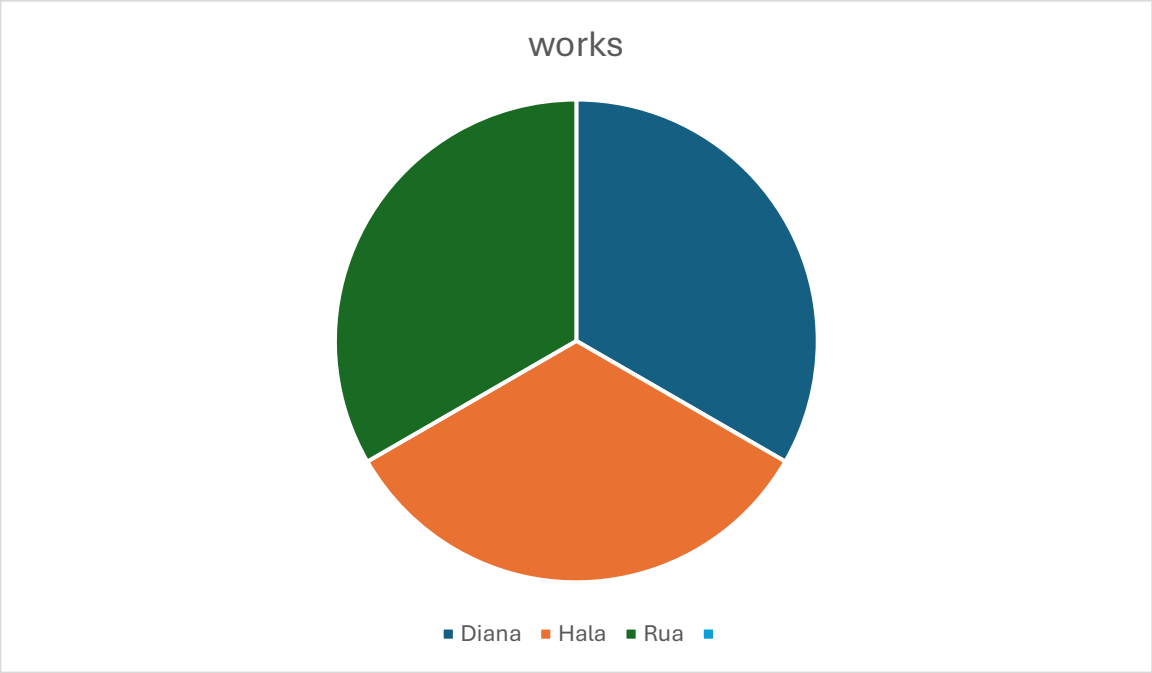
The aim of this project is to be familiar with some set of commands in terms of the networking course, with creating TCP and UDP sockets, basics of the HTML and web server and to work in teams. In order that the report will be divided into three parts, the first part is about Commands and Wireshark, the second one about socket Programming (TCP and UDP) Finally the third one about Web Server.First we will start with the command and Wireshark,the commands that we will take about them are ping, tracert, nslookup, and telnet.In addition we will use WireShrak to capture some messages.Second, implement socket programming for the purpose of having client-server communication that uses TCP and UDP. TCP ensures that data is transferred reliably but it does this by being connection-oriented even as faster connectionless communication is provided in UDP. In developing and testing simple network applications, this project shows the major distinctions and uses of these protocols.

Table of Contents

List of Figures:

Teamwork:

we divide the project into 3 parts, the first one for rua, the second one for Diana, and the last part for Hala, the project depends on the participation for our parts, and about this report each of member team take her part and work on it, then all of member work on report order.



Part one Commands and Wireshark:

Theory part 1:

“A command, in the context of technology and computing, is an instruction given by a user to a computer or software to perform a specific task. It can be a single word, a line of code, or a series of instructions that tell the computer what to do” [1]. In this project we will be specific about four commands which are ping, tracert, nslookup, and telnet. In a short way I will explain each one. First ping is a command that use for testing the availability of the device to reach an IP address for another device or a network by sending several packets to that IP or URL and wait for a response, the response will be either the whole packet with the Round-Trip Time for those packets to return to the device or there will not be a replay. The second one is tracert, it is a command that if it is followed by the IP either destination or domain name will return the information about each loop it done, the loops contain the hoops that has been discovered and the time for each loop. It is used when the device has a connection problem to a specific device or losing a packet to know in which hope the loss or failure happened. The third one is nslookup (name server lookup) it used to return the (IP) address or domain name system (DNS) record for a specific hostname. Finally, telnet commands are used to connect to a remote computer over a network and test if this port is open. It is also useful to determine the issue if you have trouble connecting via IMAO/POP or SMTP. In addition, at the end of this part we will use Wireshark. “Wireshark is a widely used, open-source network analyzer that can capture and display real-time details of network traffic. It is particularly useful for troubleshooting network issues, analyzing network protocols and ensuring network security. Networks must be monitored to ensure smooth operations and security” [2].

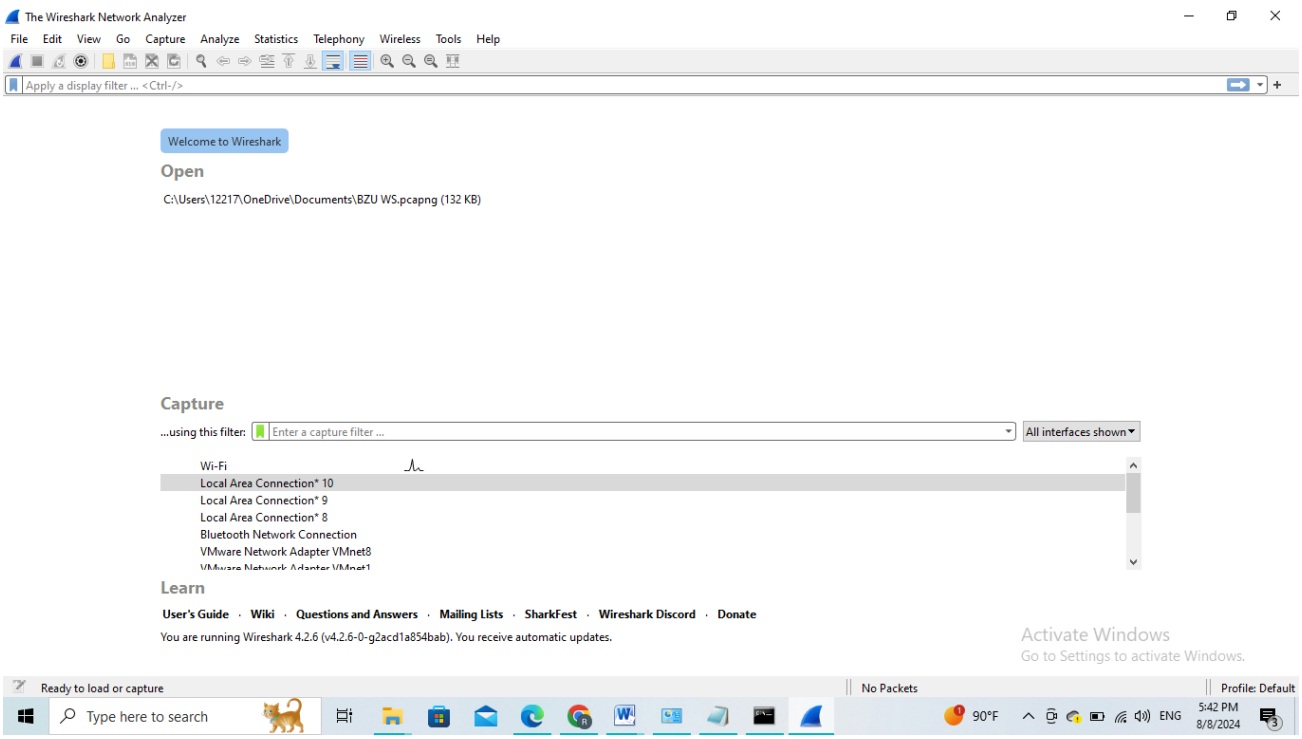


Figure 1:WireShark Interface. (Source: screenshot from my PC).

Figure 2:WireShark logo. (Source: quora).

Procedure part 1:

First, I made sure that my PC is connected to the internet and then I run the following commands as following:

ping a device in the same network, e.g. from a laptop to a smartphone.

1. I Clicked Start > Run.

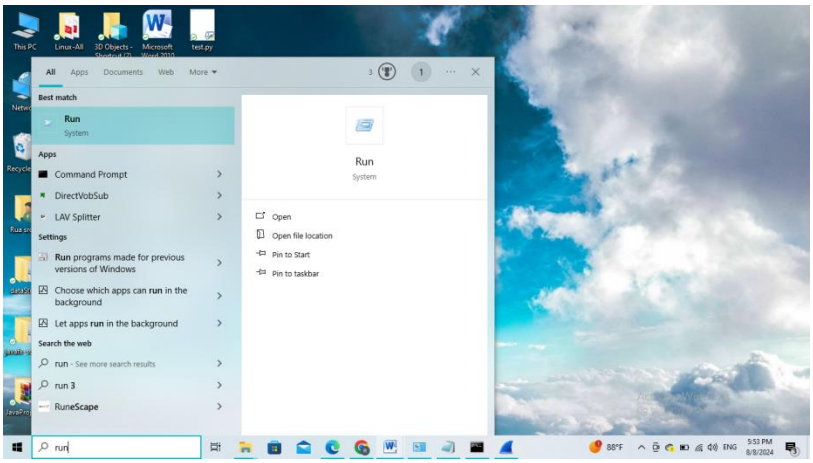


Figure 3step one to run a command. (Source: screenshot from my PC).

2.In the run box I entered "cmd" > OK.

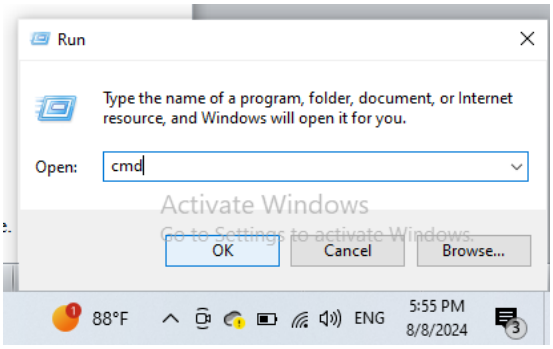


Figure 4:step tow to run a command. (Source: screenshot from my PC).

1. In the command prompt I entered "ping IP for the smart phone" without quotes > press ENTER.

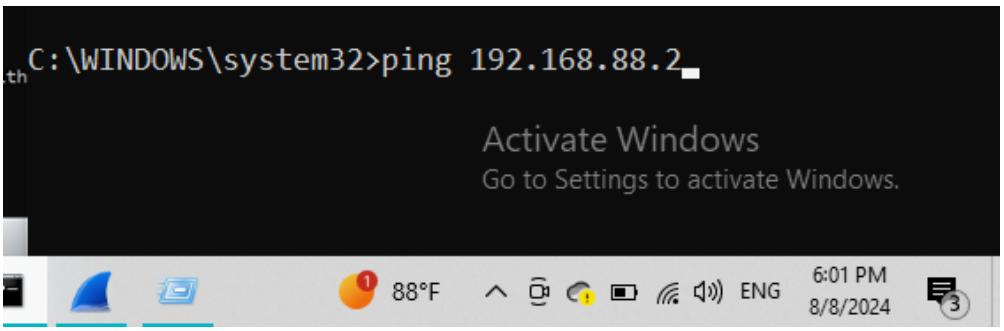


Figure 5:step three to run a command. (Source: screenshot from my PC).

2. Then a set of data that shows server internet speeds will be presented.

1.1.2.2 ping www.ox.ac.uk.

I repeated all the steps in 1.1.2.1 but instead **of entering** "ping IP for the smart phone" In the command prompt, I entered "ping www.ox.ac.uk" and the results will show server internet speeds

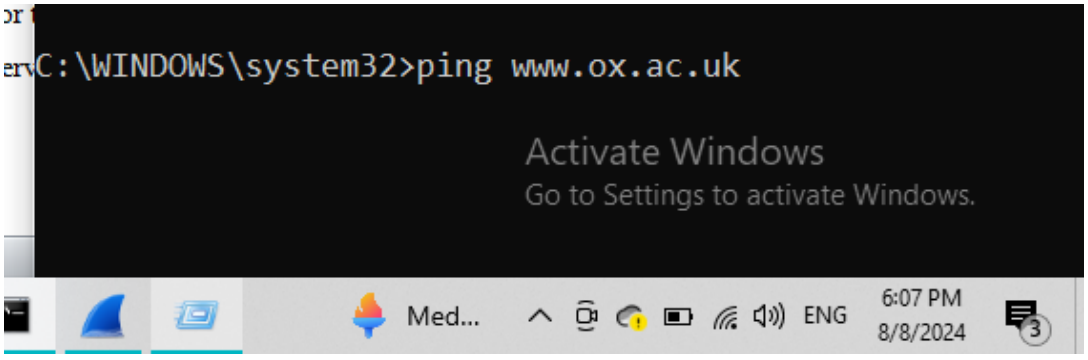


Figure 6:step three to run a command. (Source: screenshot from my PC).

1.1.2.3 tracert www.ox.ac.uk

1. I Clicked Start > Run.

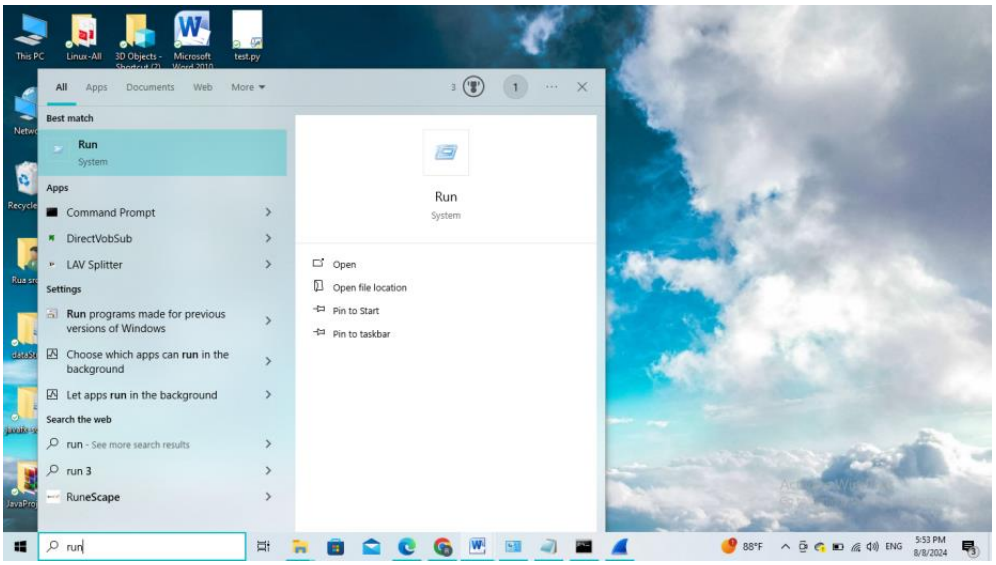


Figure 7 step one to run a command. (Source: screenshot from my PC).

2. In the run box I entered "cmd" > OK.

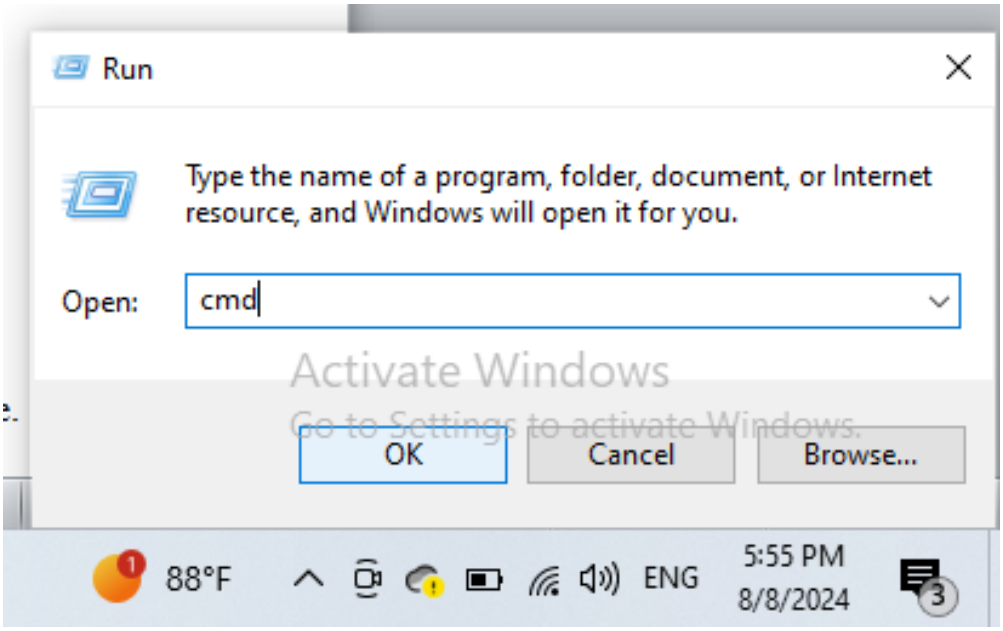


Figure 8step tow to run a command. (Source: screenshot from my PC).

3. In the command prompt I entered "tarcert Domain Name" without quotes > press ENTER.

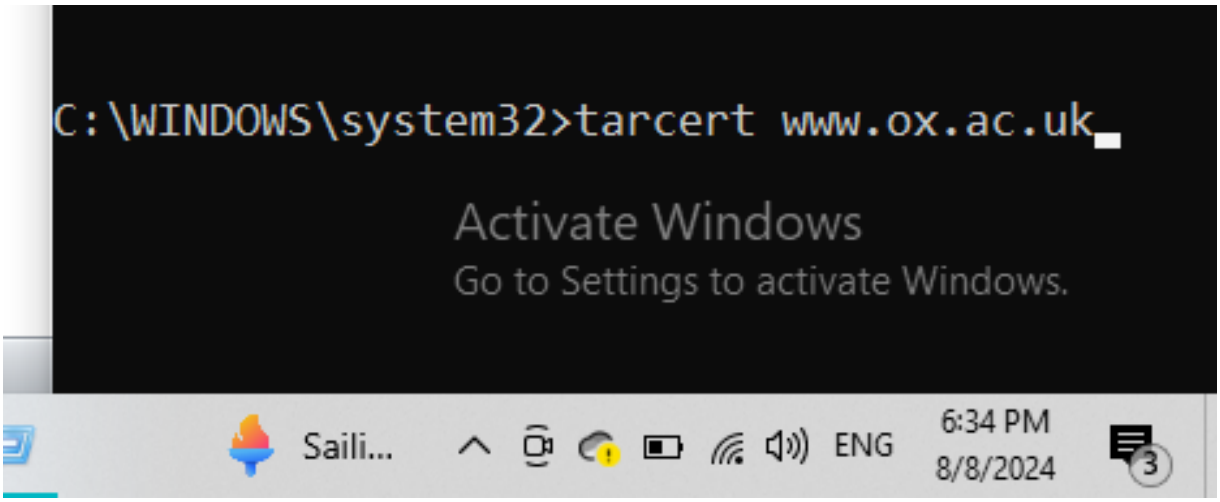


Figure 9step three to run a command. (Source: screenshot from my PC).

4. then details about the path that a packet takes to the specified destination will be shown.

1.1.2.4 nslookup www.ox.ac.uk.

.

1. I Clicked Start > Run.

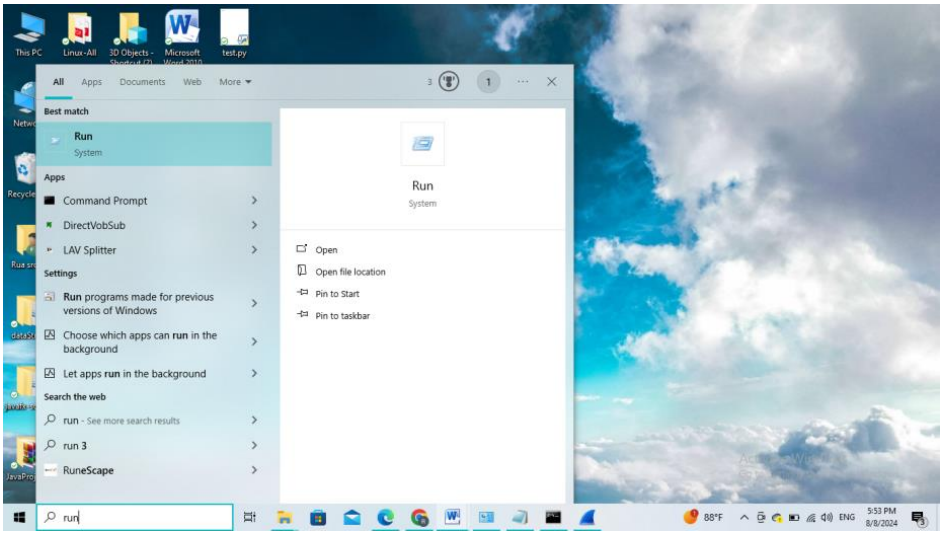


Figure 10 step one to run a command. (Source: screenshot from my PC).

1. In the run box I entered "cmd" > OK.

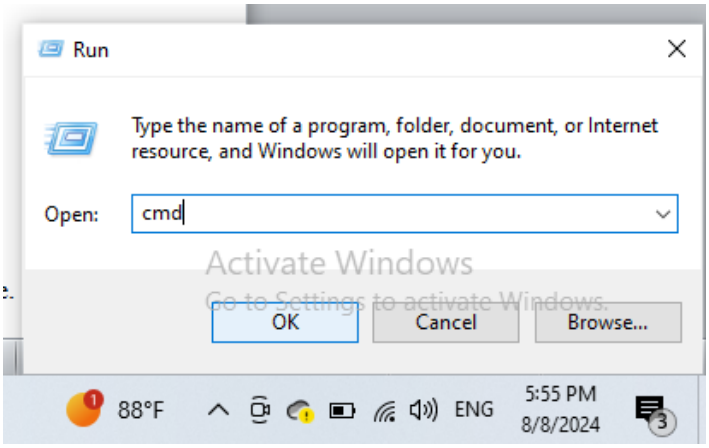


Figure 11step tow to run a command. (Source: screenshot from my PC).

2. In the command prompt I entered "nslookup domain name" without quotes > press ENTER.

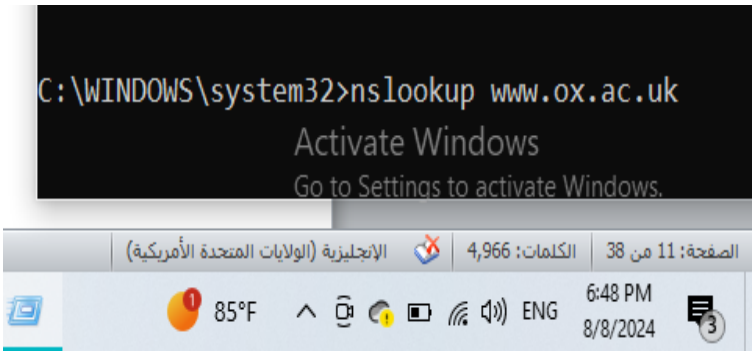


Figure 12 step three to run a command. (Source: screenshot from my PC).

3. then a set of retrieved detailed information about the specified domain will be shown.

1.1.2.5: telnet www.ox.ac.uk.

1. I Clicked Start > Run.

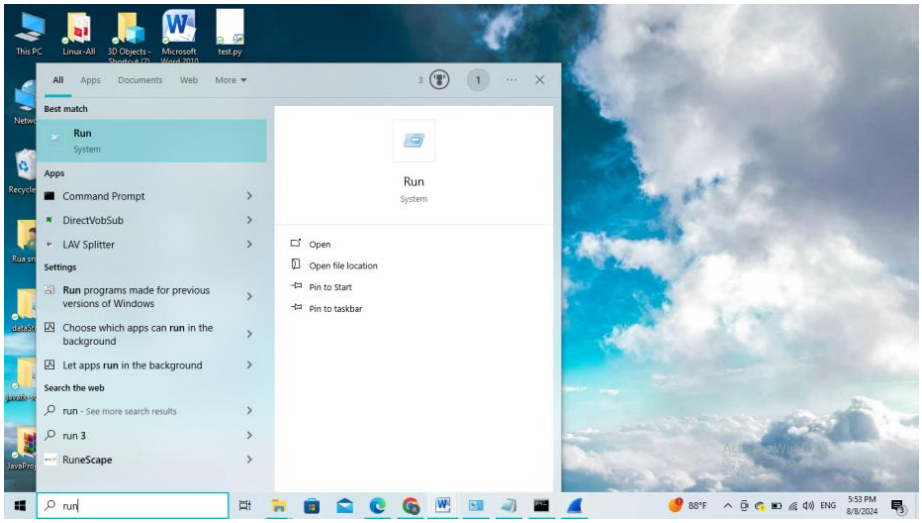


Figure 13 step one to run a command. (Source: screenshot from my PC).

2. In the run box I entered "cmd" > OK.

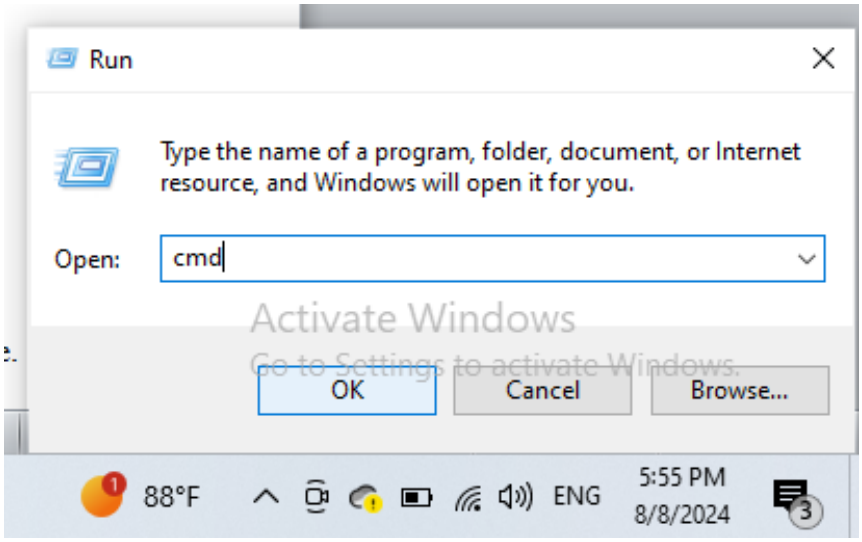


Figure 14step tow to run a command. (Source: screenshot from my PC).

3. In the command prompt I entered " telnet <target domain name> <target port>" without quotes > press ENTER.

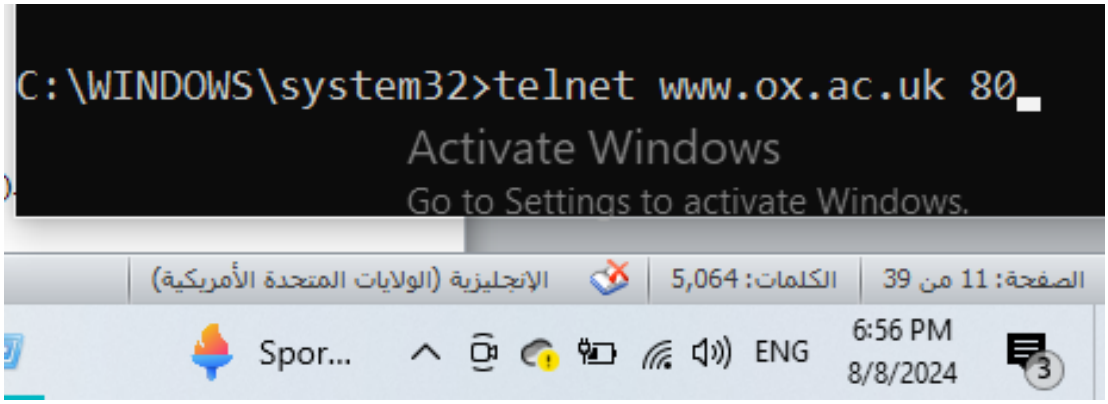


Figure 15step three to run a command. (Source: screenshot from my PC).

4. If the ping is successful, I should receive replies from the address that I am trying to ping.

1.1.2.6 Giving some details about autonomous system (AS) number, number of IPs, prefixes, peers,

name of Tier1-ISP of www.ox.ac.uk.

1- for getting autonomous system (AS) number, number of IPs and prefixes, will use bgpview.io [3]. The IP address is one of the IP's that I got from nslookup command.

Figure 16:BGPview interface. (Source: screenshot from my PC).

2- for the number peers, name of Tier1-ISP, I will use bgp. tools.

The AS name that we got from the previous website.

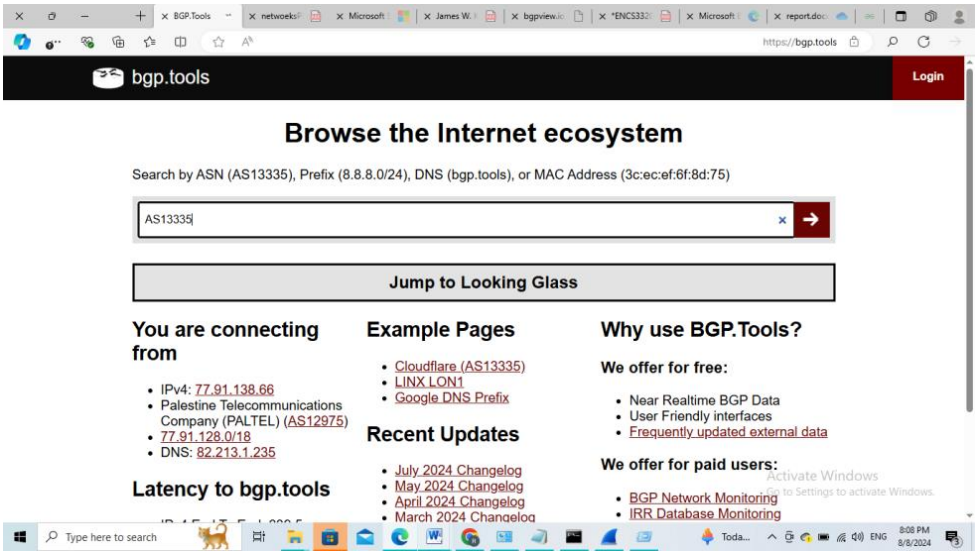


Figure 17:BGP.tools interface. (Source: screenshot from my PC).

1.1.2.7 Using wireshark to capture some DNS messages.

The DNS messages the I captured are from my home Wi-Fi traffic.

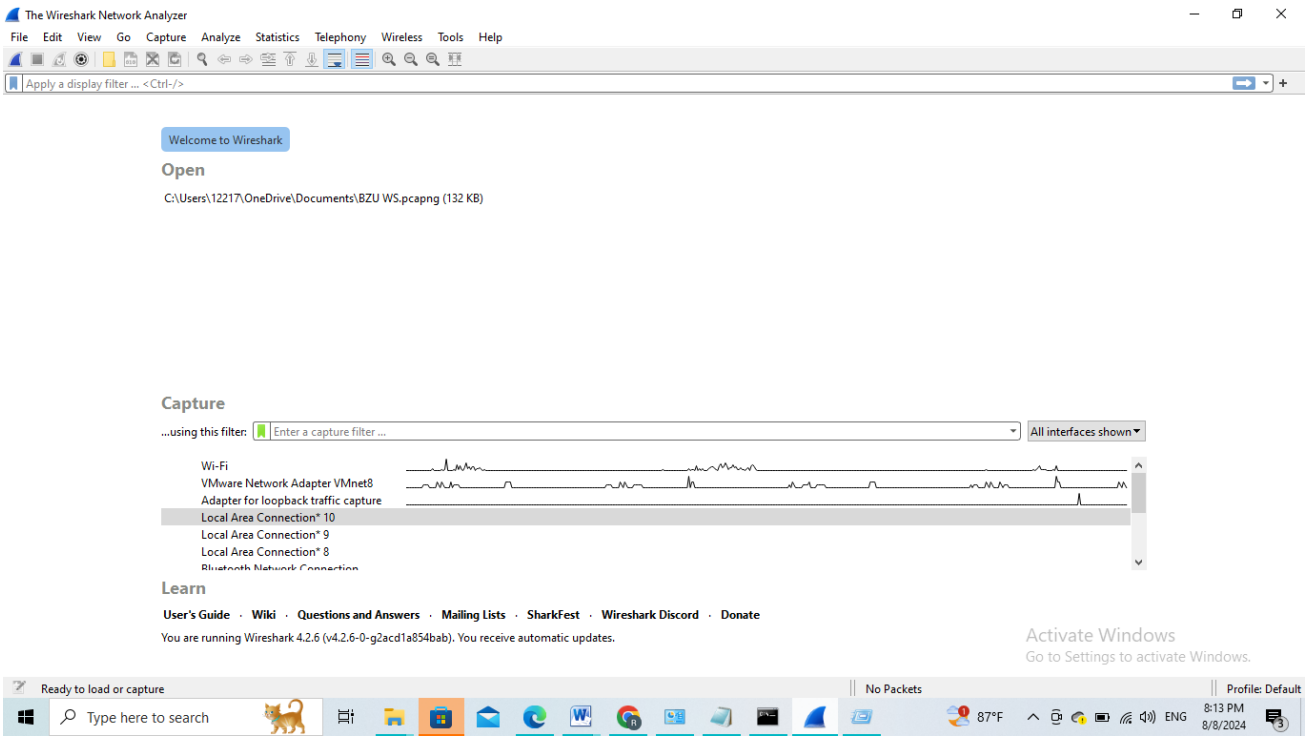


Figure 18:Wireshark interface. (Source: screenshot from my PC).

Results and Discussions Part 1:

1.1: Part one Commands and Wireshark:

1.2.1 the result of ping a smartphone with IP = 192.168.88.2 from a laptop at the same network:

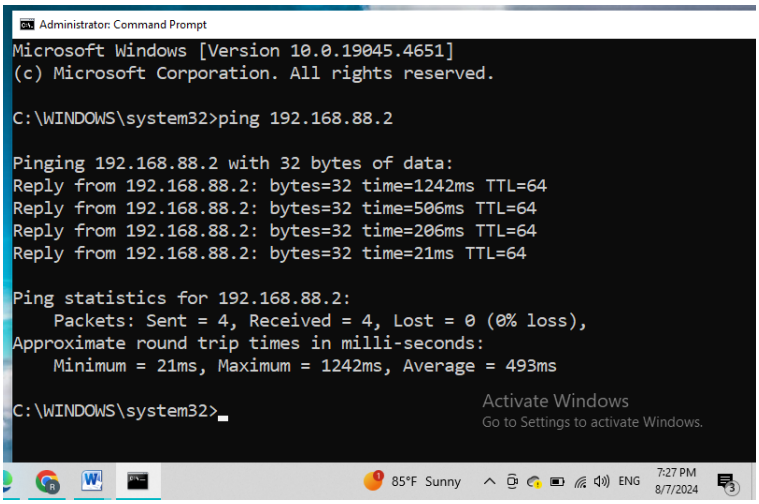


Figure 19:the replies from the address that I tried to ping. (Source: screenshot from my PC).

My Device ping the IP address of a smart phone that connect to the same network of my device
(IP =192.168.88.2) by sending four packet to this IP then wait for a replay, the replay as shown in Figure (20) has the information for each packet about the IP for the server which we want to ping, size of the packet and it is equal to 32 bytes, and what we look for it is the TTL (time to live) and it is equal to 64 msec this time mean that the time that the packet takes to trans to and from the specific IP to my device. Finally, since the lost is equal to zero and the four-packet come back that means the IP is reachable by my device by the network.

1.2.2 The result Of ping www.ox.ac.uk.

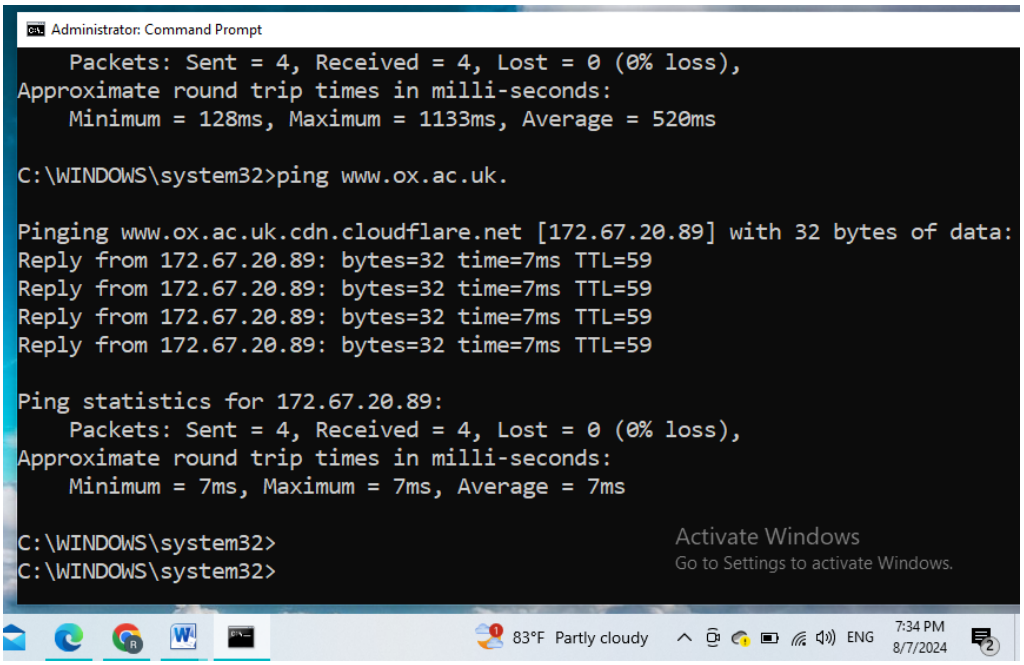


Figure 20: the replies from ping www.ox.ac.uk.. (Source: screenshot from my PC).

My Device ping the website of university of oxford by the domain name www.ox.ac.uk ,by sending four packets to this domain name IP, then it waits for a replay, the replay as shown in Figure 21 has the information for each packet about the IP for the server which we want to ping, size of the packet and it is equal to 32 bytes, and what we look for it is the TTL (time to live) and it is equal to 59 msec this time mean that the time that the packet takes to trans to and from the specific domain name to my device. Finally, since the loss is equal to zero and the four-packet comes back that means the domain name IP is reachable by my device by the network. . The location of the server from where I got the first response with IP 192.168.88.1 is in United States according to www.iplocation.net , as shown in Figure 22.

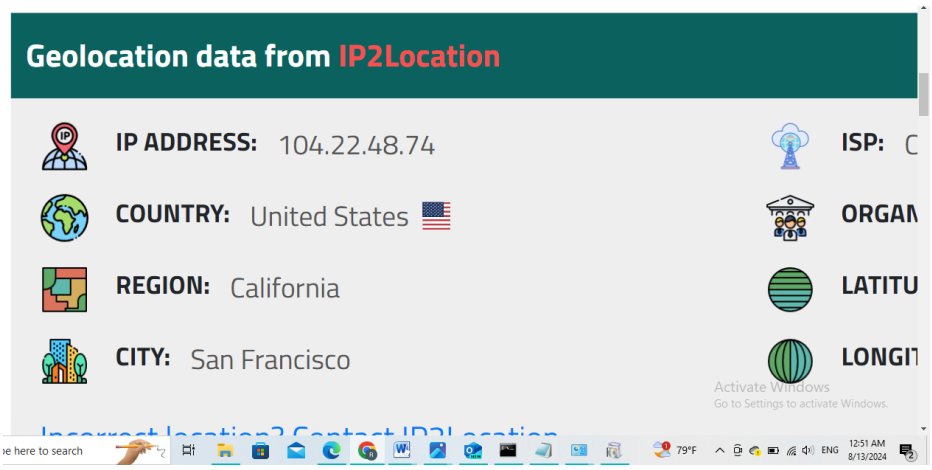


Figure 21:Location of the server with IP 104.22.48.74. (Source: IP location).

. The location of the server from where I got the first response with IP 192.168.88.1 is in Palestine according to www.iplocation.net , as shown in Figure.

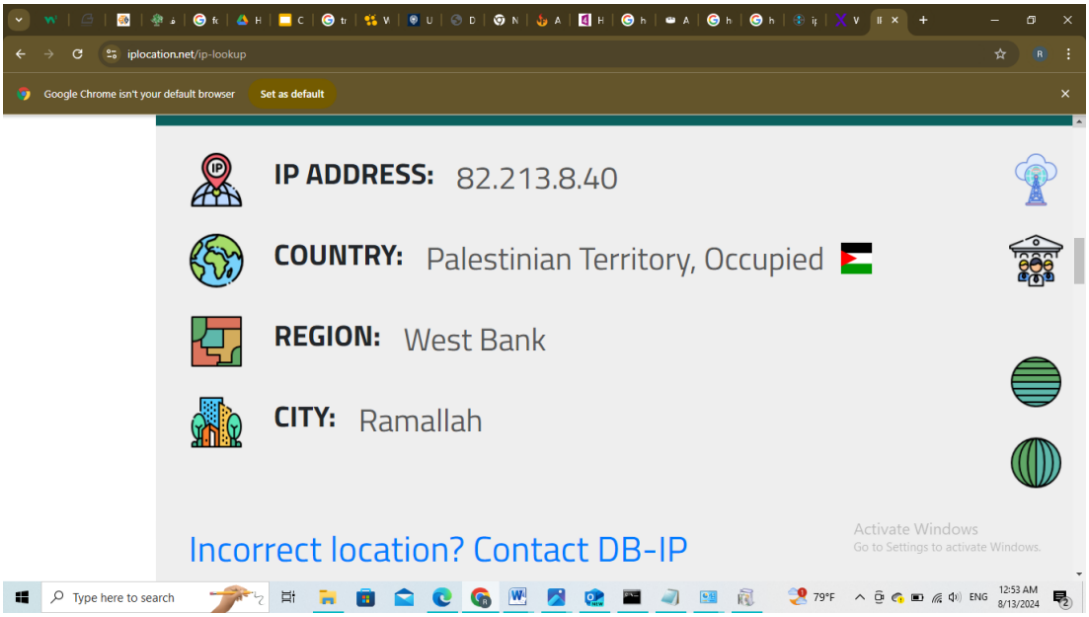


Figure 22:Location of the server with IP 104.22.48.74. (Source: IP location).

1.2.3 The result of tracert www.ox.ac.uk.

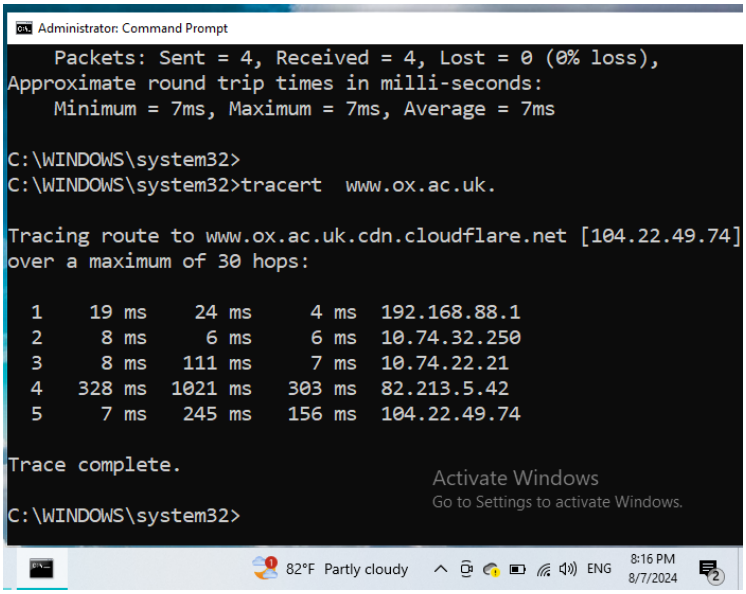


Figure 23:the replies from tracert www.ox.ac.uk.. (Source: screenshot from my PC).

The result of tracert command is similar to ping one, since it is work almost as the same way but treract provide more information about each loop .As shown in Figure 24, my device sent three packet for several hoops and in each loop it calculate the RTT time and provide me the IP of this hope. We can notice that the first packet of the fifth hop was reduced from 328ms to 7ms since the path that the packet takes in the fifth loop was congested in the fourth one. since the traffic in it reduced the packet want through it in the fifth loop since it is closer to my device.

Finally, the command provides a comment with "Trace complete "to be sure that the path to the www.ox.ac.uk server is reachable with those loops and hops and there is no error with any hope. Figure 25 explain the concept of this command.

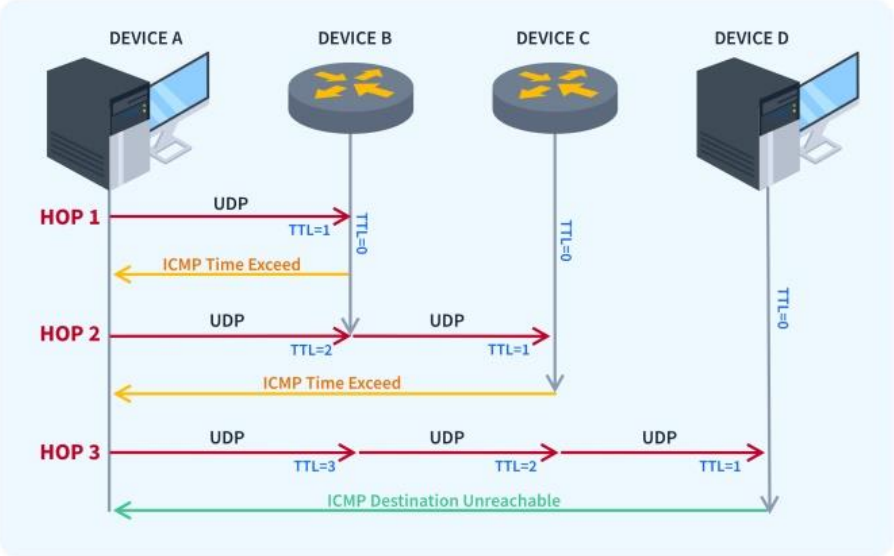


Figure 24:the concept of tracert command. (Source: cbtnuggets).

1.2.4 nslookup www.ox.ac.uk.

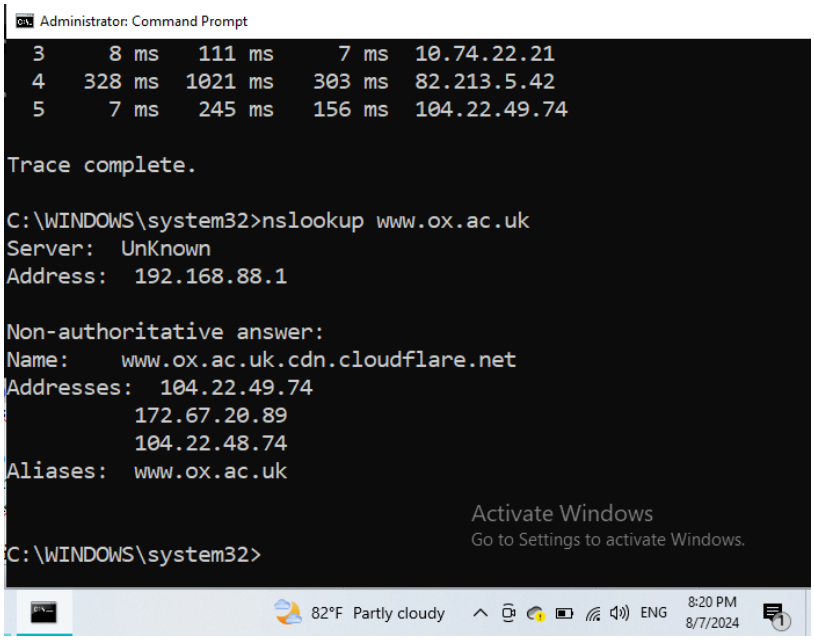


Figure 25:the reply from nslookup www.ox.ac.uk. (Source: screenshot from my PC).

As shown in Figure 26 nslookup command provide some information about **www.ox.ac.uk** Such as the name of I Don't know , the IP's of this **www.ox.ac.uk** (IP = 104.22.48.74, 172.67.20.89, 104.22.49.74) and finally the Aliases name www.ox.ac.uk. "An alias is a shortcut that identifies a group of hosts, networks, or interfaces" [7].

1.2.5

telnet www.ox.ac.uk.

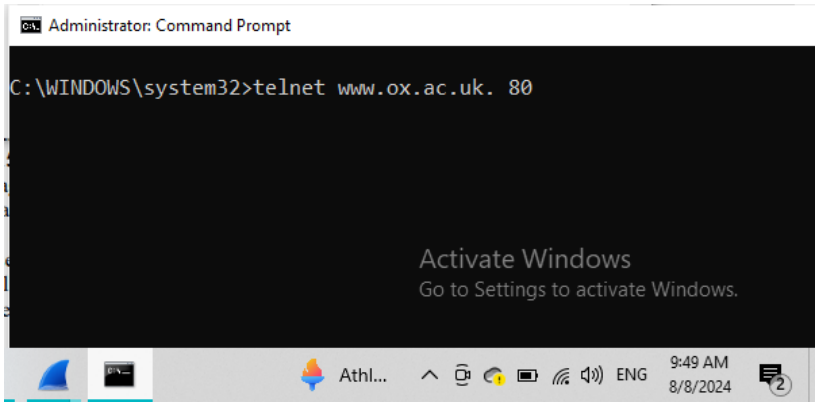


Figure 26:the replies from tracert www.ox.ac.uk.. (Source: screenshot from my PC).

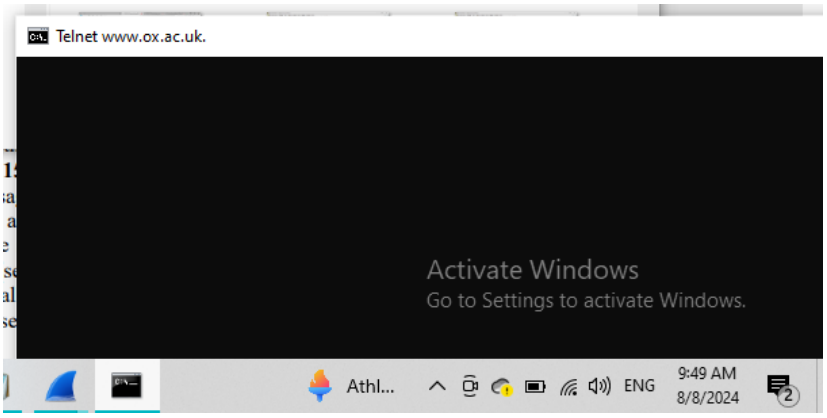


Figure 27:the replies from telnet www.ox.ac.uk. (Source: screenshot from my PC).

I used the Telnet command to connect to port 80 on a remote server of **www.ox.ac.uk** as shown in Figure 28. To verify if the path from my computer to the **www.ox.ac.uk** server is open over that port = 80.And the replay with blank screen means that the connection is done and success as shown in Figure 29.

1.2.6 some details about www.ox.ac.uk.

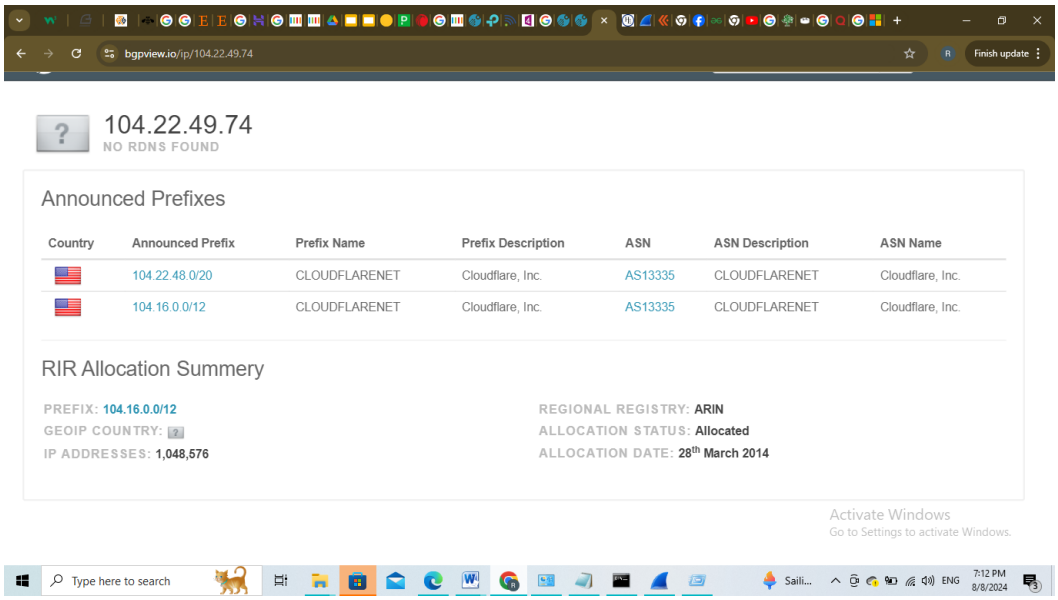


Figure 28:Information's from Bgp.view about www.ox.ac.uk server. (Source: Bgp.view).

As shown from Figure 29 from the Bgp.view the autonomous system (AS) number is AS13335 , the number of IPs is 1,048,576 , and the prefix number for the IP 104.16.0.0 is 12, for the IP 104.22.48.74is 20.

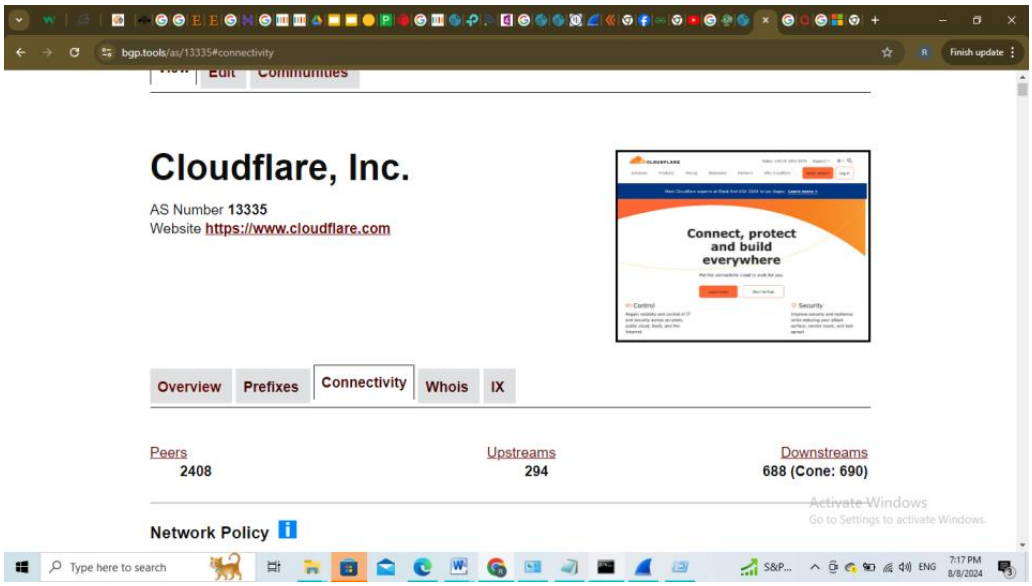


Figure 29::Information's from Bgp.tool about www.ox.ac.uk server. (Source: Bgp.tool).

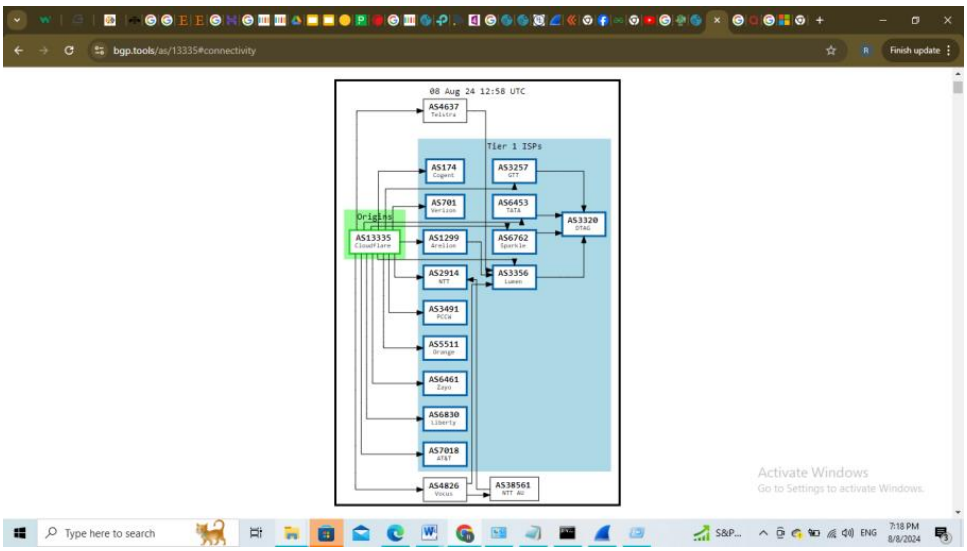


Figure 30:chart of www.ox.ac.uk links. (Source: Bgp.tool).

As shown from Figure 31 the number of peers is 2821, from Figure 32 the name of one of Tier1-ISP is AS3257.

- **Using Wireshark to capture some DNS messages.**

- **The result of capturing my home Wi-Fi traffic is as shown in Figure 33:**

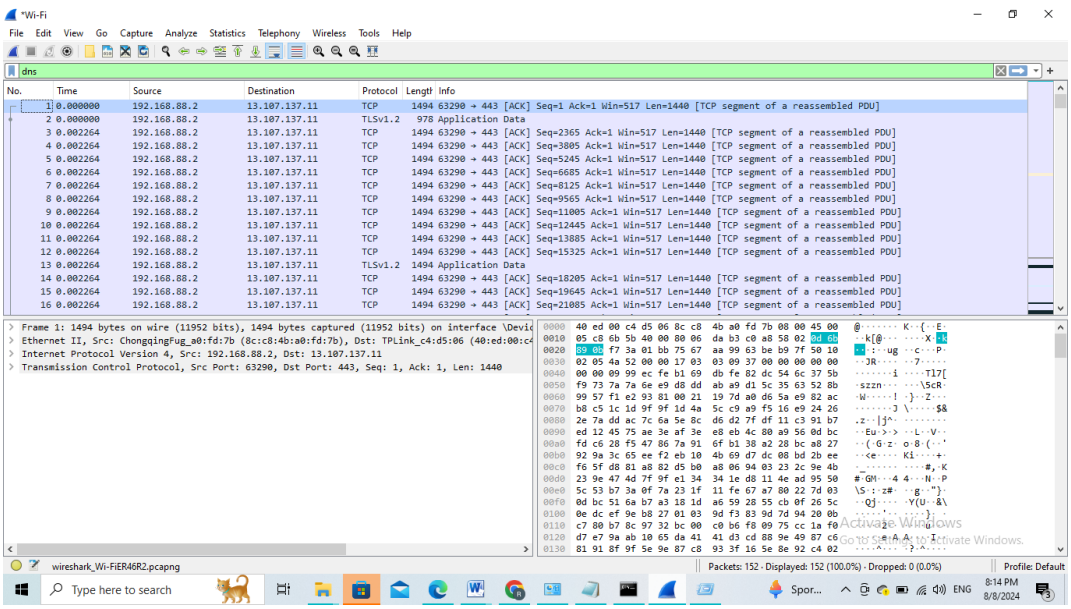


Figure 31:DNS massages from capturing the Wi-Fi traffic. (Source: wireshark).

1.3.1telnet is not recognized as internal or external command.

The first time I used talent command the replay was as shown in fig

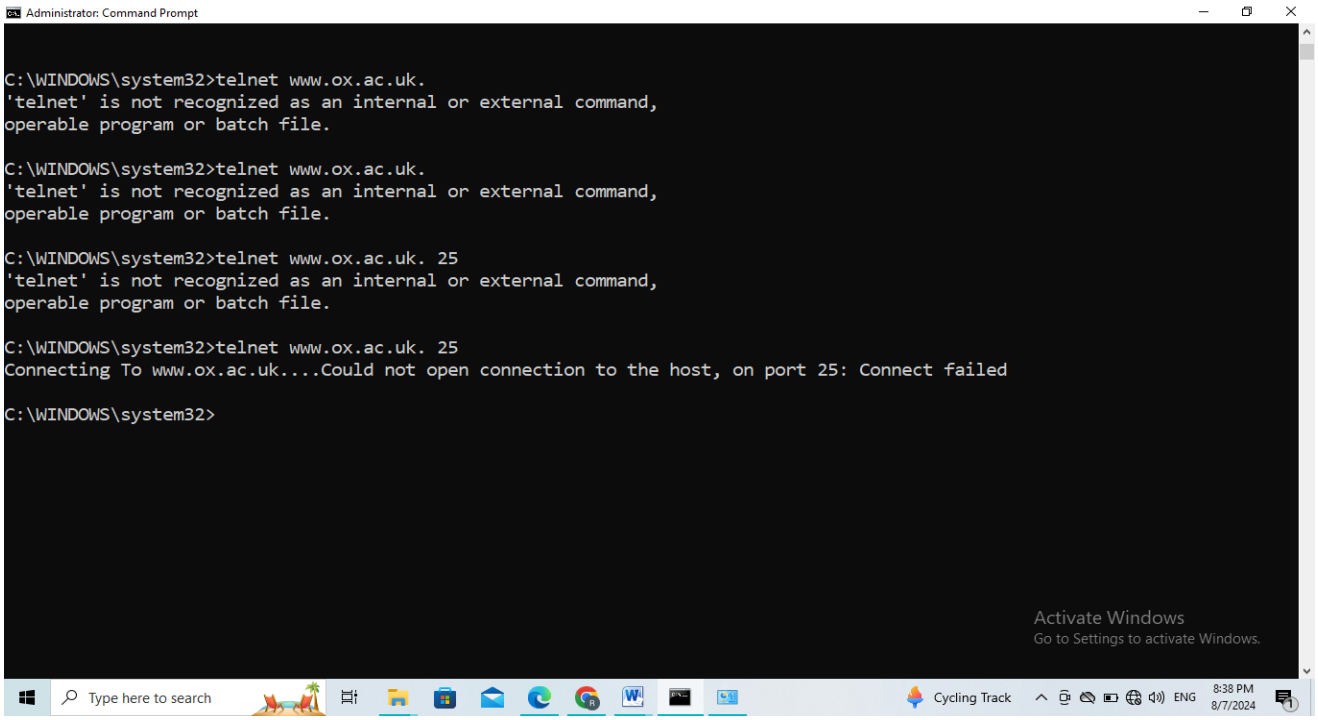


Figure 32:the replies from telnet www.ox.ac.uk. (Source: screenshot from my PC).

This reply means that the command telnet is not recognized as internal or external command.

And the solution was as following by the computer setting as following steps:

- I Opened the Control Panel.
- Click on Programs & Features.
- In left bar I selected “Turn Windows features on or off"
- Found "Telnet Client" and ticked it.
- Click "OK"

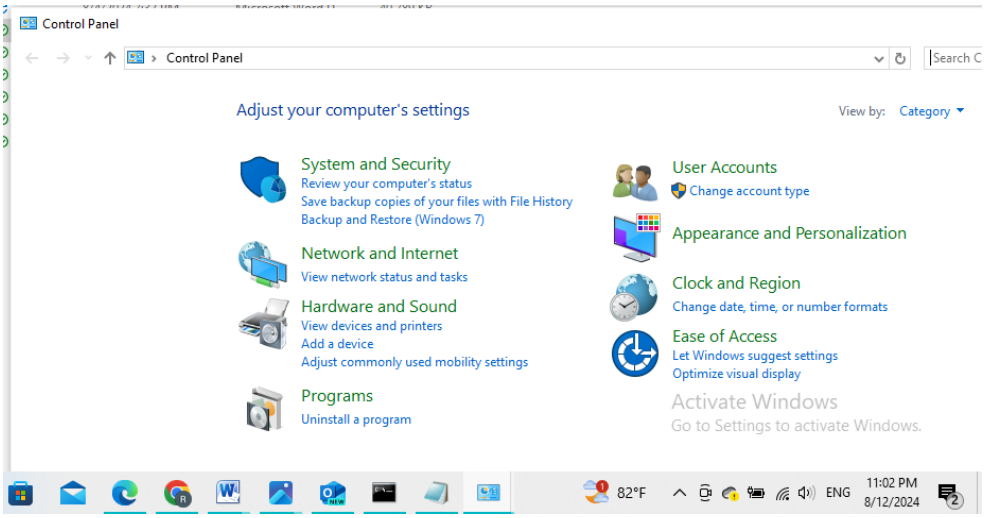


Figure 33:step one. (Source: screenshot from my PC).

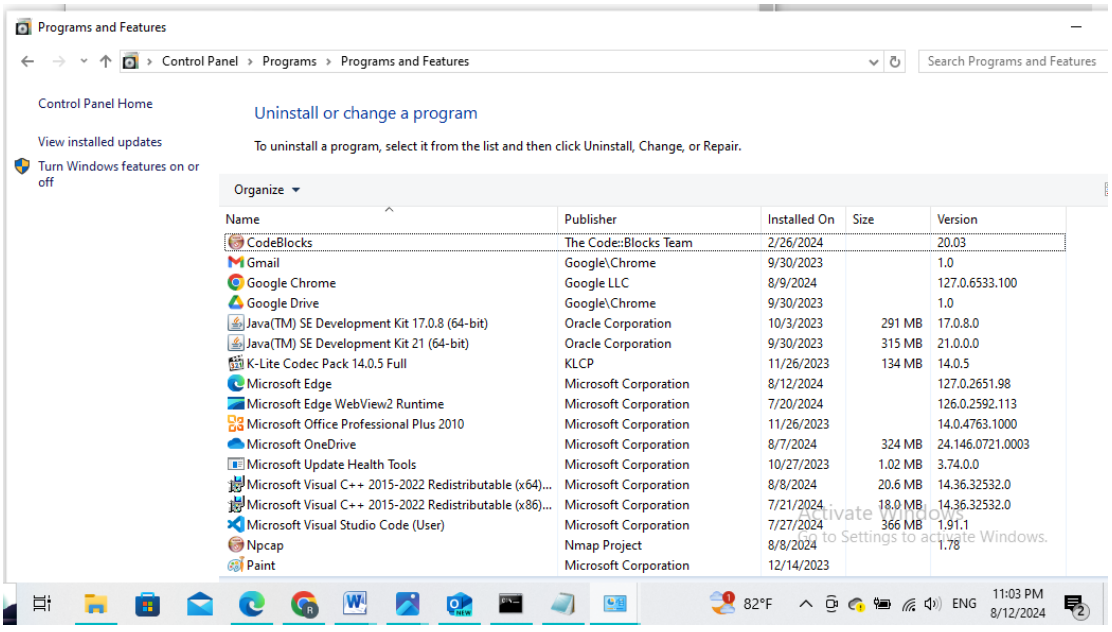


Figure 34:step tow. (Source: screenshot from my PC)

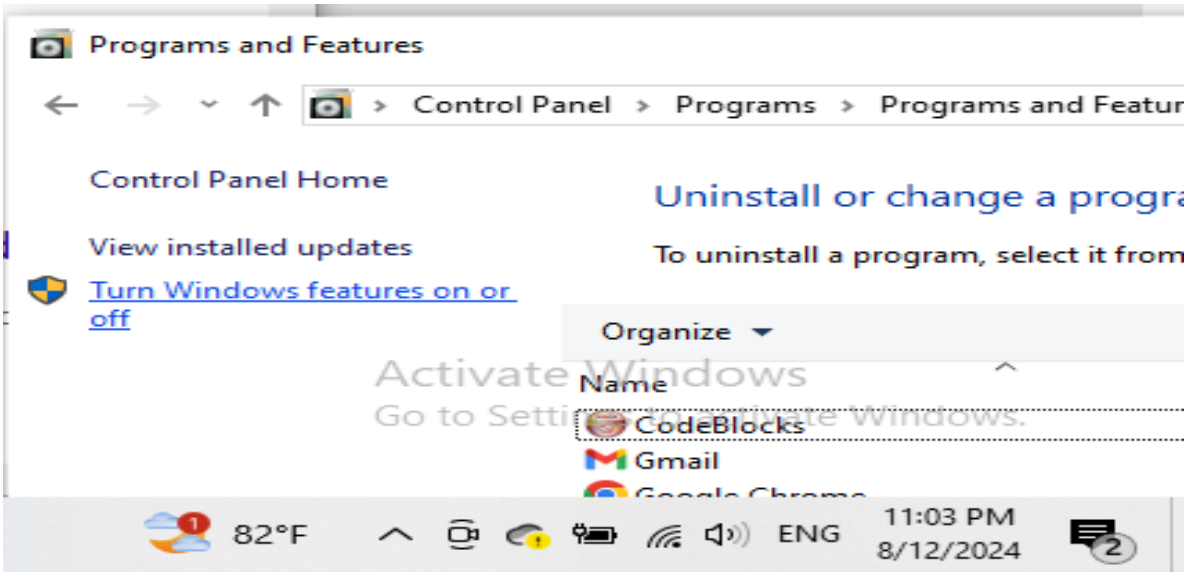


Figure 35:step three. (Source: screenshot from my PC).

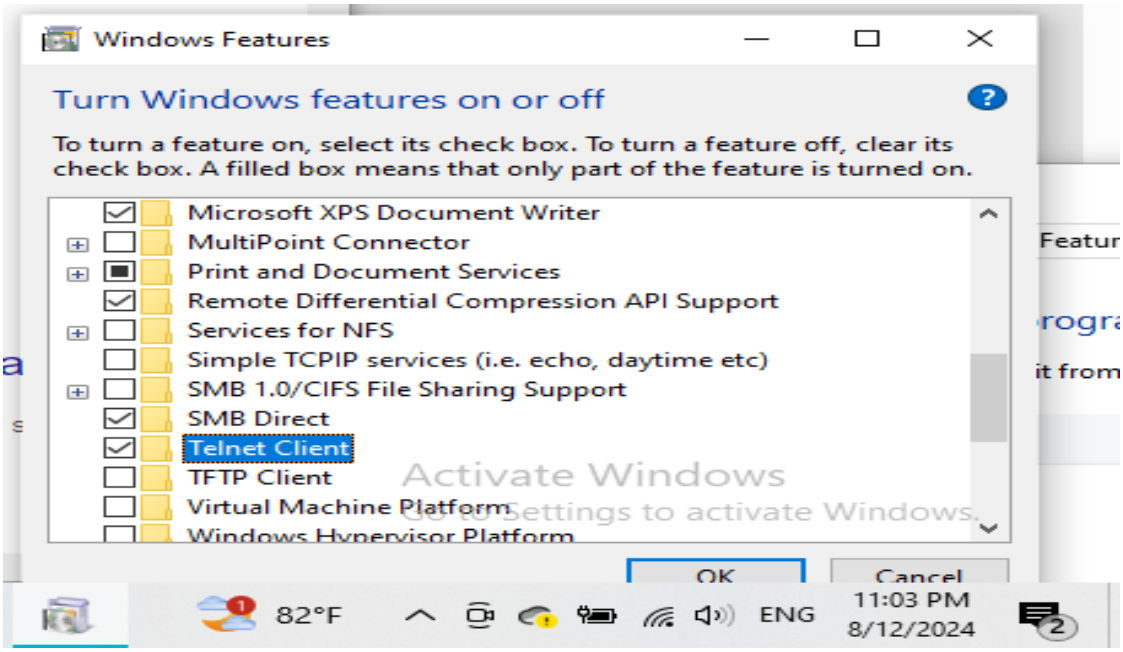


Figure 36:step four. (Source: screenshot from my PC).

Part 2: Socket Programming (TCP and UDP)

Theory:

Socket Programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.[12]

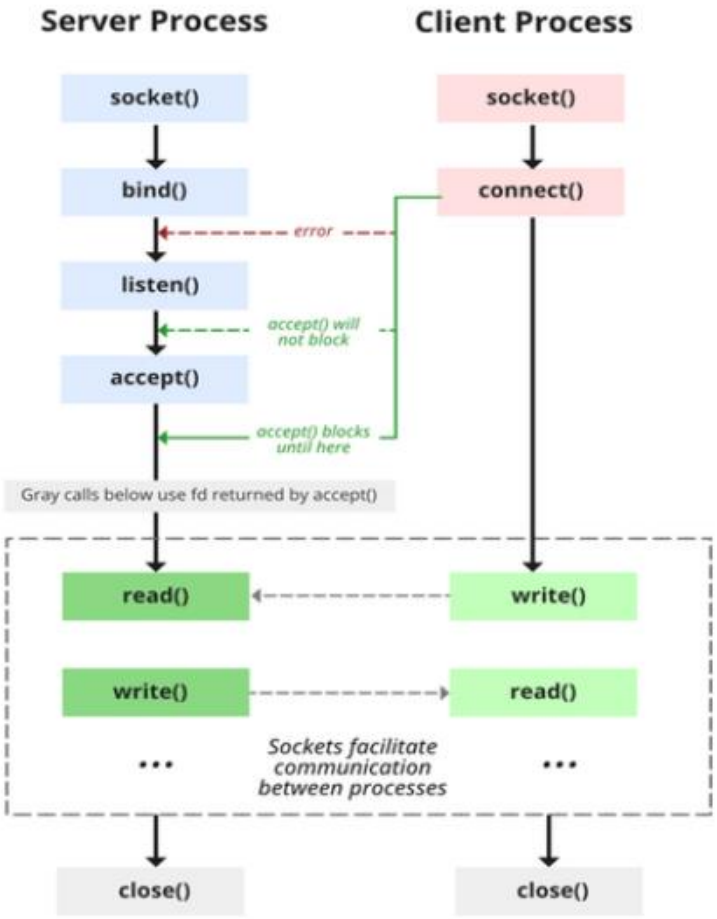


Figure 37:State Diagram for Server and Client Model [12]

Transmission Control Protocol (TCP): is a communications standard that enables application programs and computing devices to exchange messages over a network. It is designed to send packets across the internet and ensure the successful delivery of data and messages over networks.[13]

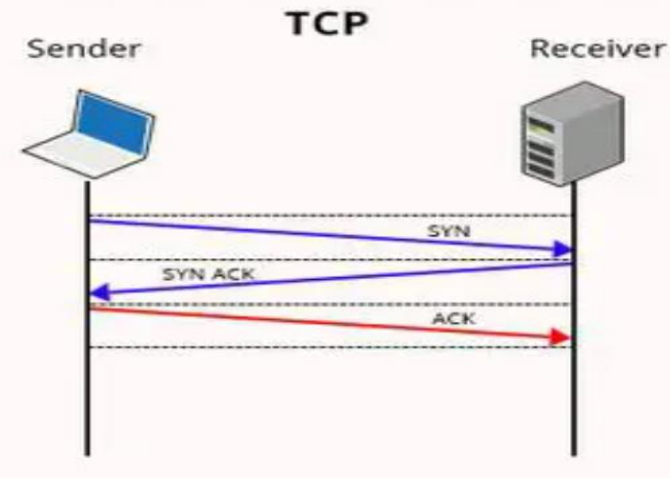


Figure 38:TCP connections.[14]

==========

User Datagram Protocol (UDP): is an alternative communications protocol to Transmission Control Protocol (TCP), used primarily for starting low-latency and loss-tolerating connections between applications and the internet. UDP is also known as a “stateless” protocol, meaning it doesn’t acknowledge that the packets being sent have been received. Due to UDP working this way, it is typically used for streaming services. You may hear some breakup in the audio or see some skips in the video, but UDP transmissions prevent the stream from stopping completely. [14]

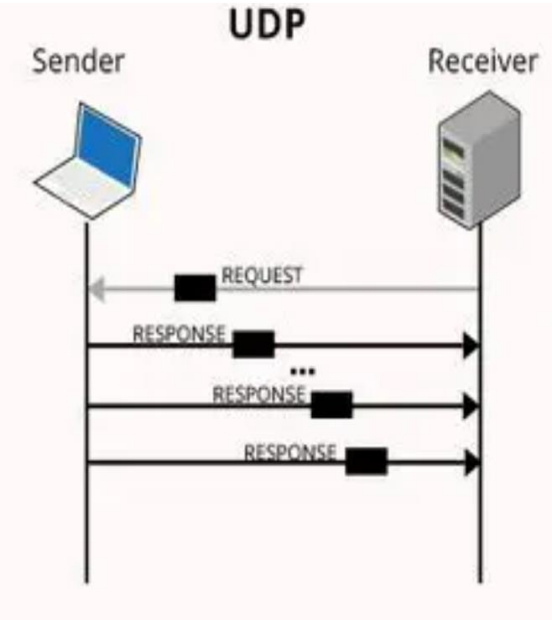


Figure 39:UDP connections.

procedure:

TCP Protocol:

- First in this part I write the TCP server code and another one for the client using python programming.
- And depending on our ID numbers: I talk (rua ID number:1221727) => 1727 represent the port number that I use for this part.
- As for as the IP address I talk it from my pc (that represented the server):

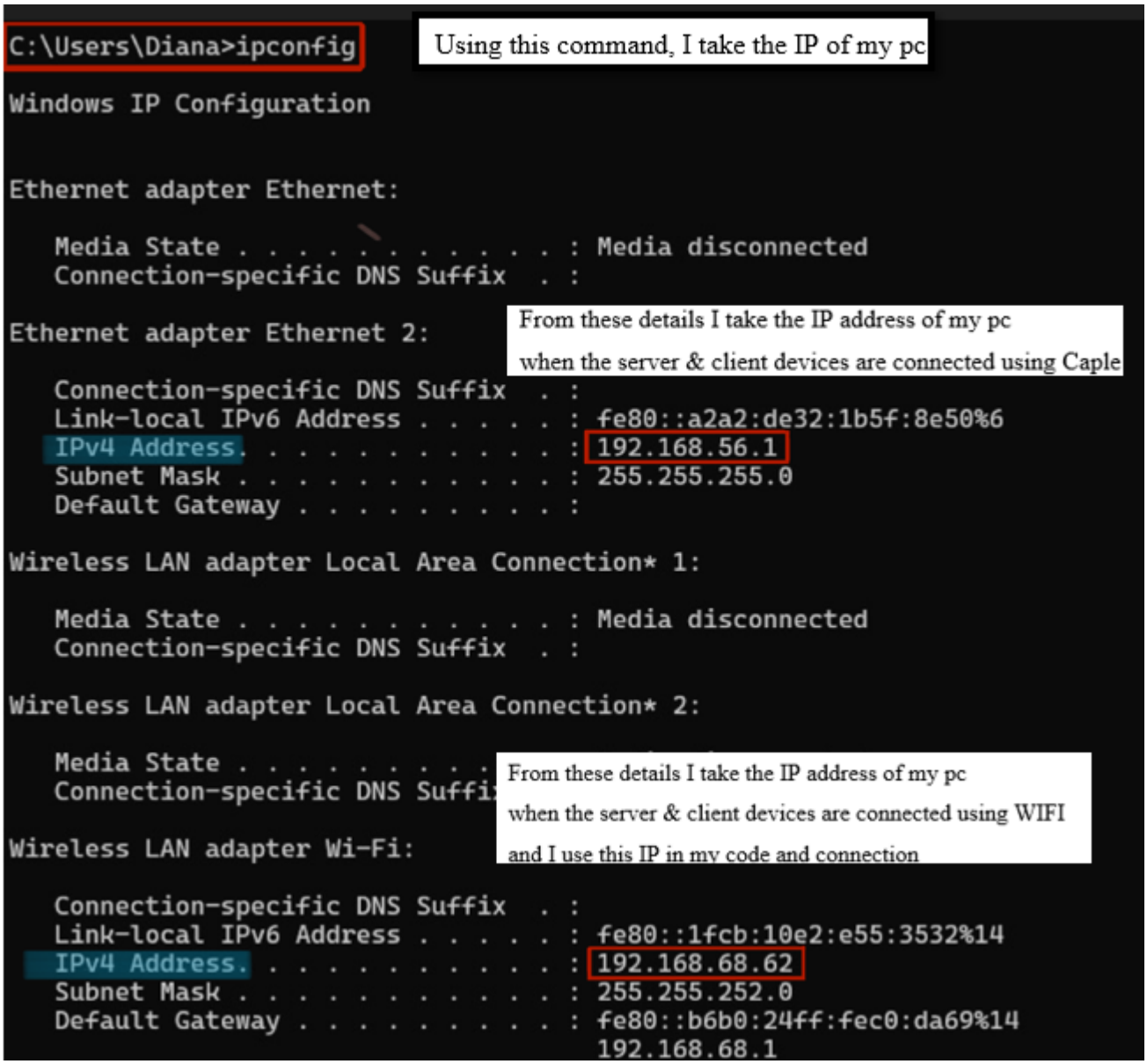


Figure 40:Selecte (Server) IP address from my PC.

in this part I implemented two python files for (server and client):

TCP Client Code:

In this part it will send input data from client to server, replace all the vowels (aeiou/AEIOU) with '#' in server section then it will return the string to client and print it.

Note:

I used a virtual machine (using Linux OS) as the client to send the message to the server (and the server is my PC (windows OS)).

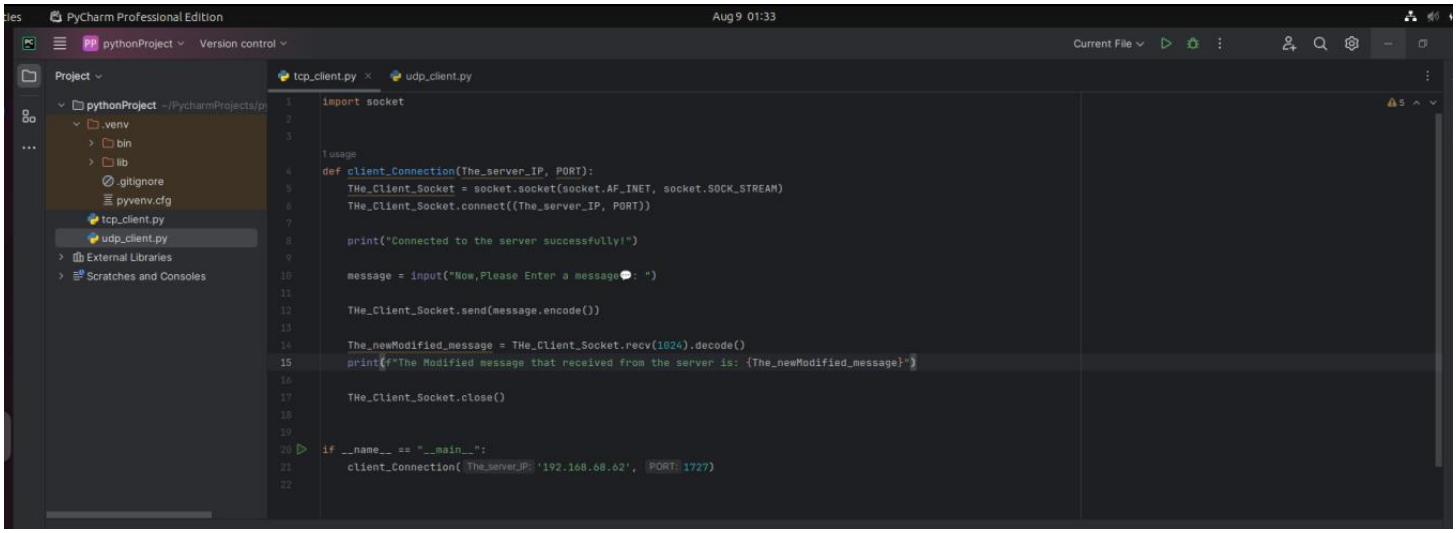


Figure 41:TCP_Client Code.

- Here I define function called (client connection) that accept server IP and the port as a parameters, then (*socket.AF_INET => the socket will use IPv4 and I take it from my pc I upload image previous that define this IP, #socket.SOCK_STREAM => specify that the socket is a TCP socket) then I connected this socket using these data*.
- *I add print messages to make the client clear about the state.*
- *Then the client sends a message to the server.*
- *Then using send () function that takes the message that the client write it then encode it.*
- It will receive the Client_Message from the server.

  *the 1024 is the buffer size in bytes then it will decode it
- *It should close the socket after finishing the process.*
- *Finally, in the main I invoke the function with passes choices The selected port and the IP*

TCP Server code:

In this part it will receive input data from client, and it will replace all the vowels (aeiou/AEIOU) with '#' then it will return the string to client and print it.

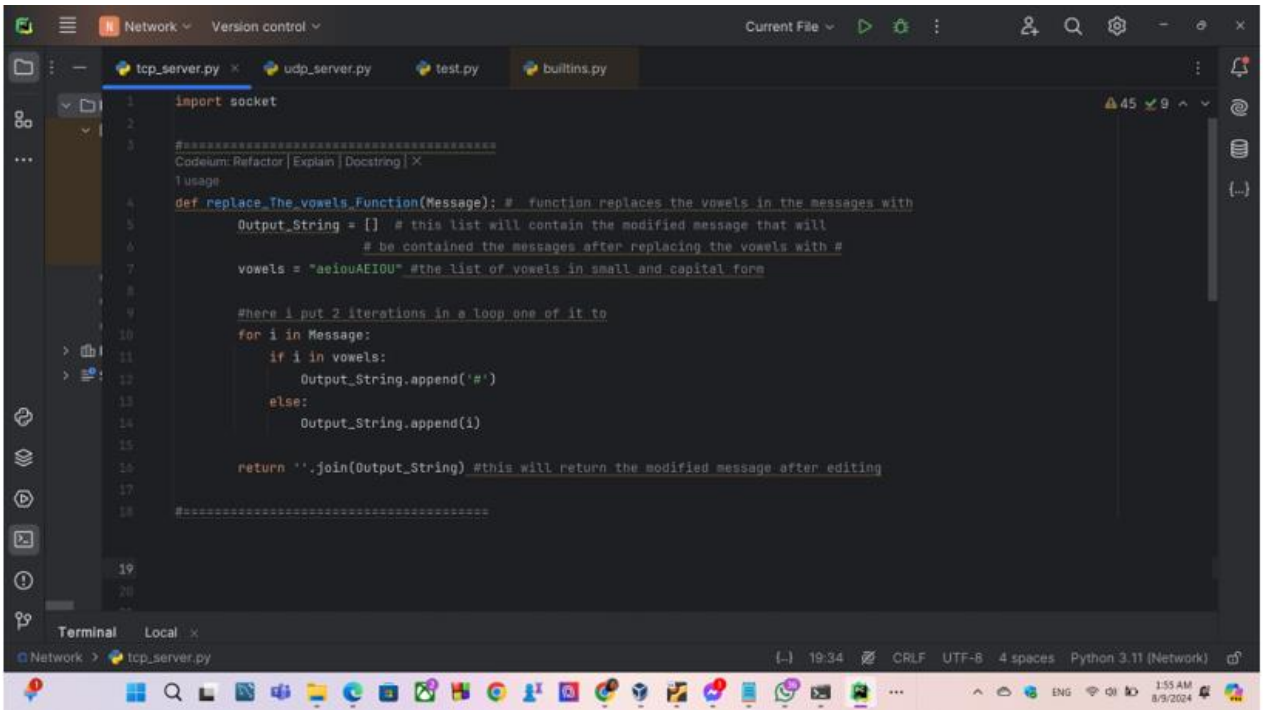


Figure 42: TCP_Server code part1.

- In this part, First I defined the function that called replace vowel's function, and take the message as a parameter, then I declared a list that called output string to store the message after changings on it (if it has vowels characters).
- Within the 2 iteration that iterate on the string (char by char) and search if it contains vowels characters (so it has it should replace it with '#' then add it to output list) if it not then just adds the char as it.
- Finally, joined this list and returned it when this function invoked.

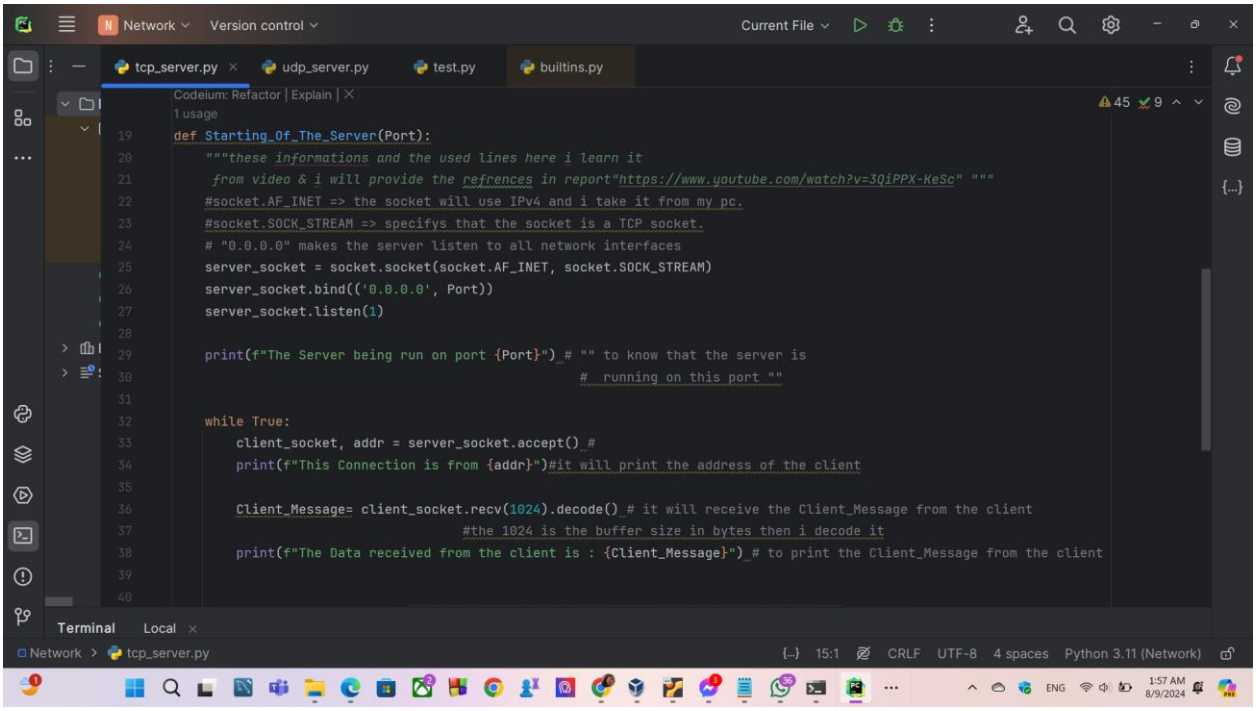


Figure 43:TCP_Server code part2.

These information I learned it from a specified videos in YouTube (I will provide the links in references part) + from chapter 2 slides.

First in staring the connection function I declared the socket and pass some parameters ," socket.AF_INET" => the socket will use IPv4 and I take it from my pc(192.186.68.62).

"socket.SOCK_STREAM" => specify that the socket is a TCP socket.

"0.0.0.0" makes the server listen to all network interfaces

- After this it will take the client socket and the address using accept () func.
- Finally, it will receive the Client_Message from the client

*the 1024 is the buffer size in bytes then it will decode it.

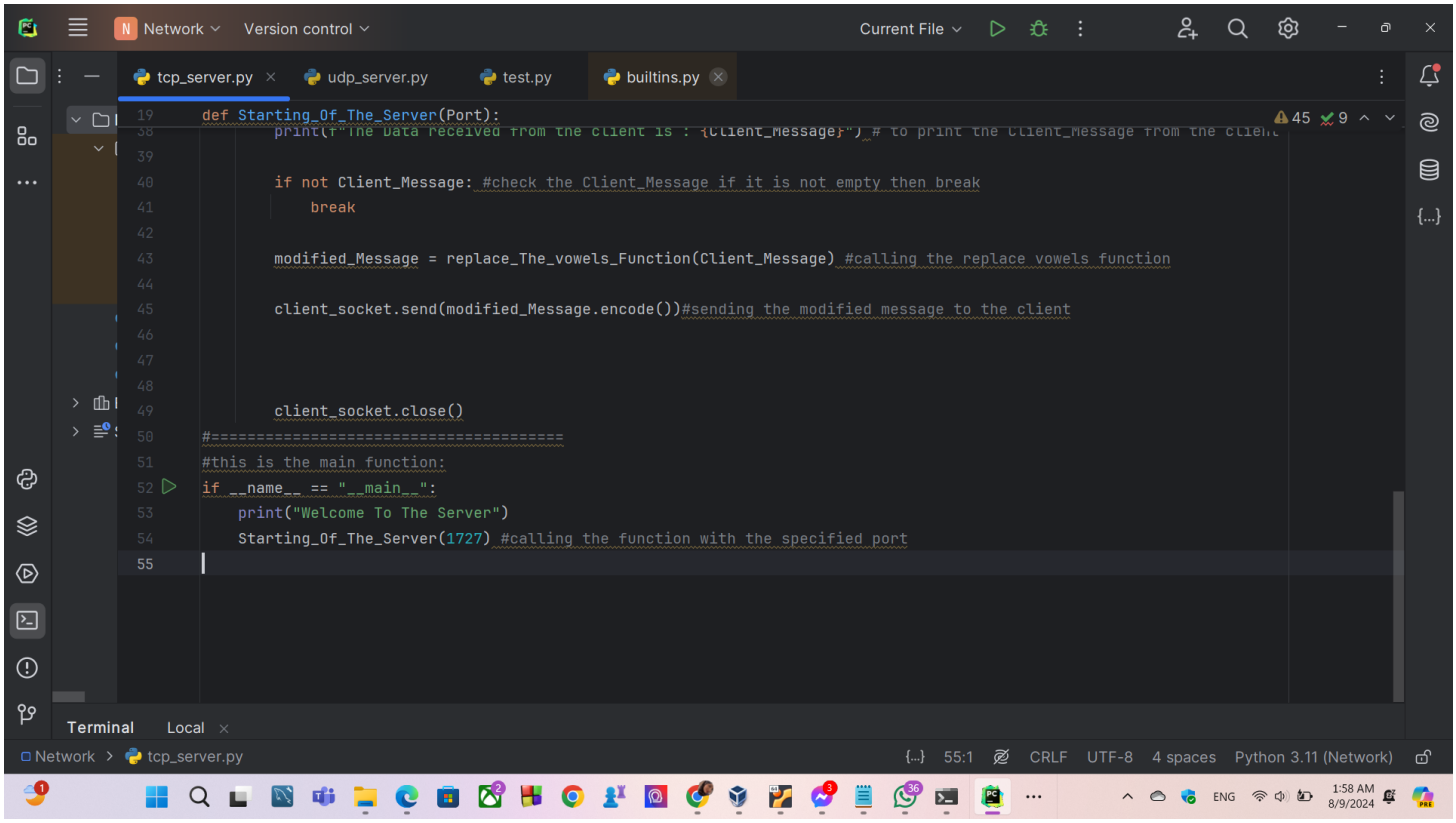
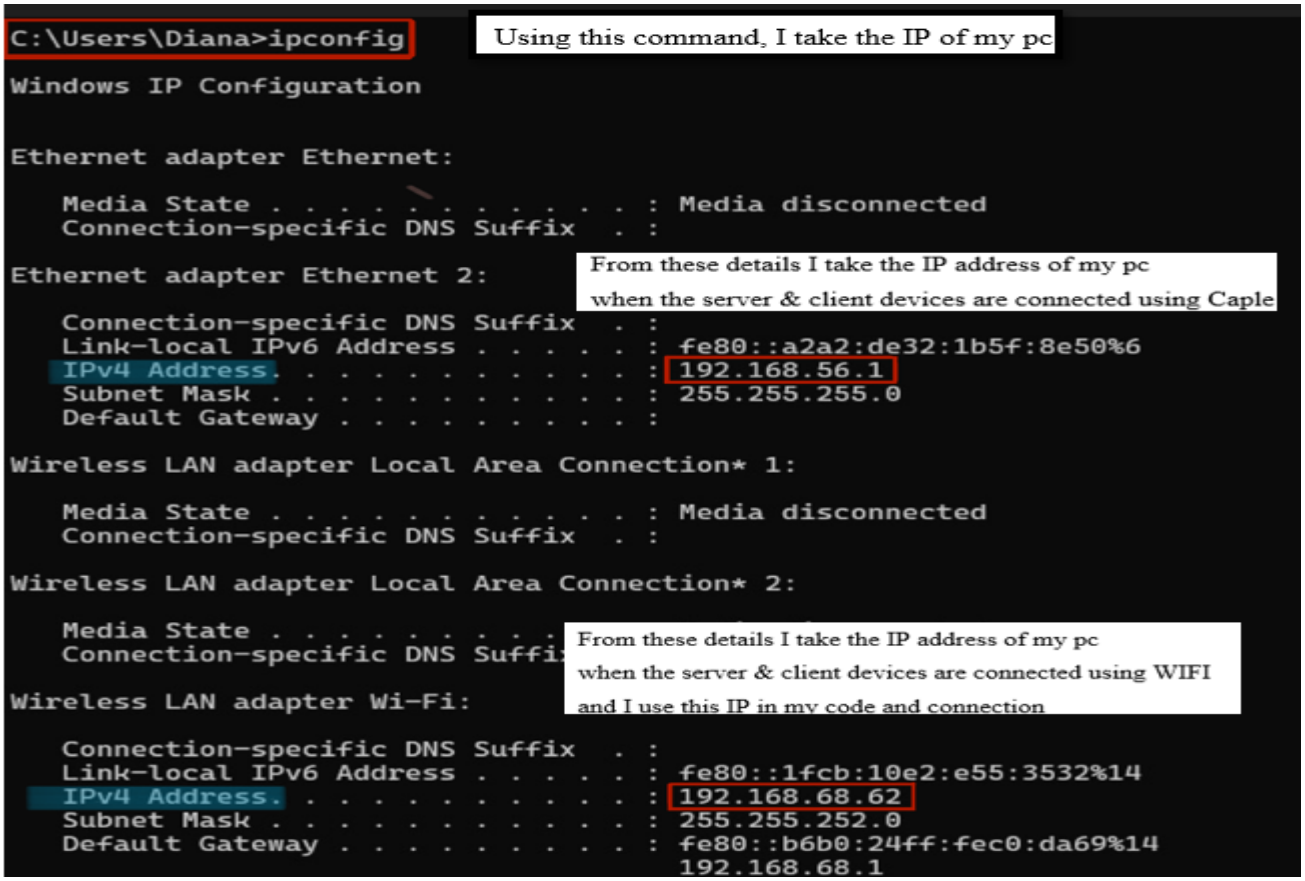


Figure 44:TCP_Server code part3.

==========

- ## UDP Protocol:

  - First in this part I write the UDP server code and another one for the client using python programming.
  - And depending on our ID numbers: I talk (rua ID number:1221727) => 1727 represent the port number that I use for this part.
  - As for as the IP address I talk it from my pc (that represented the server):



==========

in this part I implemented two python files for (server and client):

All peers (clients and server) can send and receive messages. In other words, a message sent by a peer will be received by another peer called server at this moment, others can also send to the same peer (server) after each other. The message should include peer (client) and its number as well as any message (e.g. “Hello”). The peer (server) lists the last received message from a peer (client) based on its communication with all peers.

- Note: I used a virtual machine (using Linux OS) as the client to send the message to the server (and the server is my PC (windows OS)).

UDP Server Code:

Here the server starts working in selected port then waits for the clients to start sending a message, then after the client sends a message, the server can replay this message, and can also show the list of the last communication of client messages.

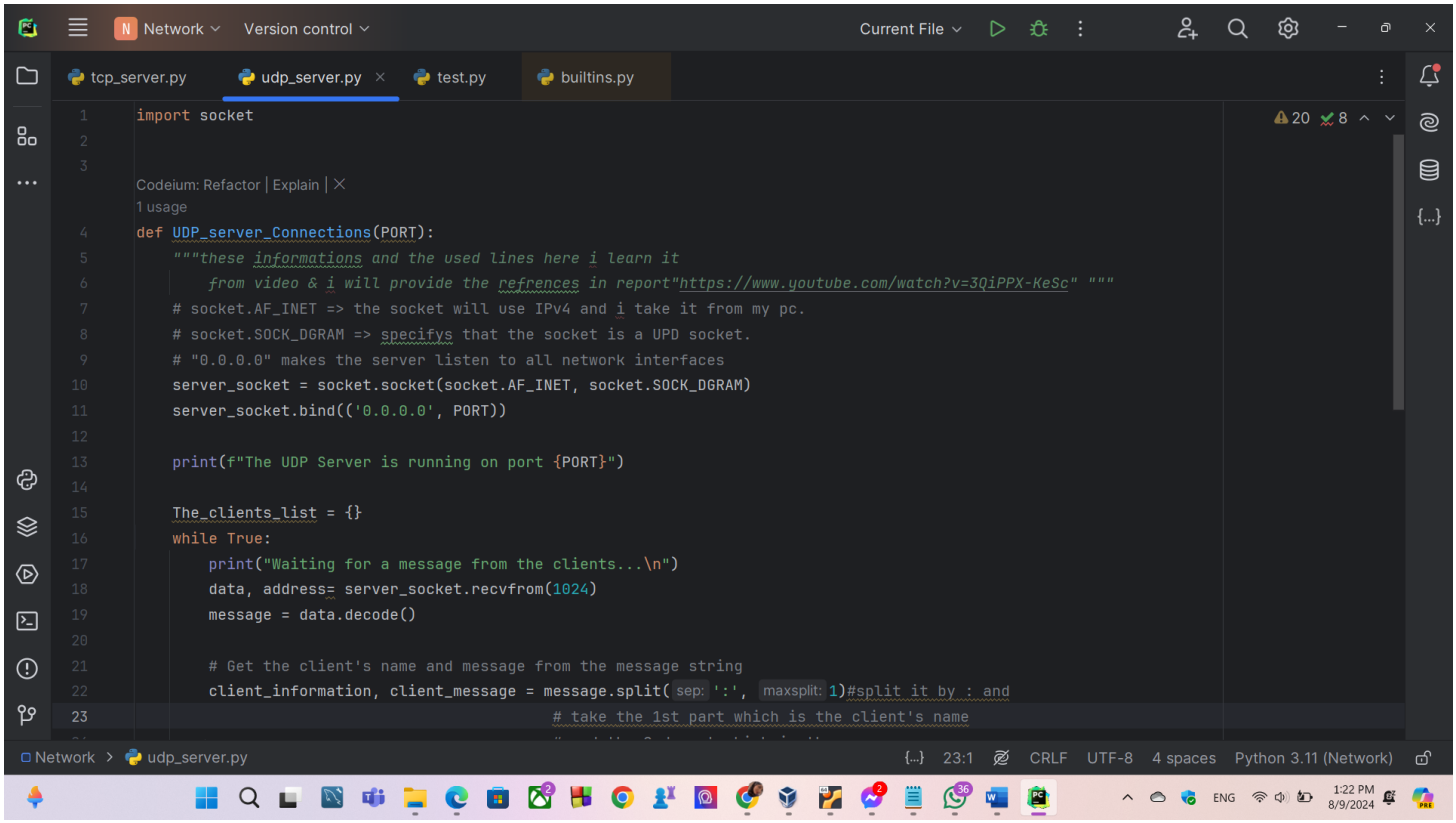


Figure 45:UDP server code part1.

These information I learned it from a specified videos in YouTube (I will provide the links in references part) and from slides.

First in staring the connection function I declared the socket and pass some parameters ," socket.AF_INET" => the socket will use IPv4 and I take it from my pc(192.186.68.62).

"socket.SOCK_DGRAM" => specify that the socket is a UDP socket.

"0.0.0.0" makes the server listen to all network interfaces

- After this it will take the client socket and the address using accept () func.
- Finally, it will receive the Client_Message from the client

*the 1024 is the buffer size in bytes then it will decode it.

Here it will list the last messages from all clients that contact this server func().

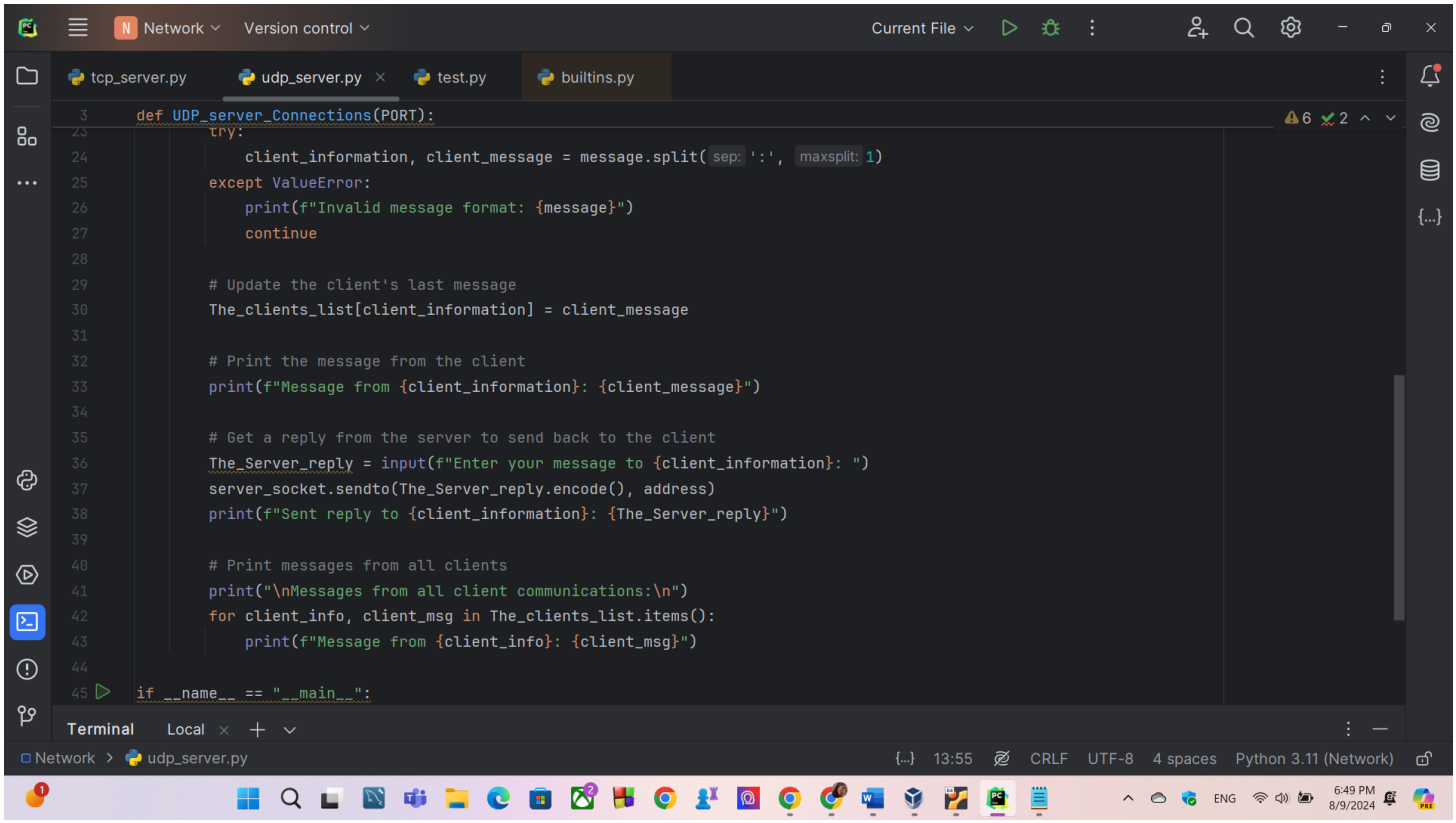


Figure 46:UDP server code part2.

==========

UDP Client code:

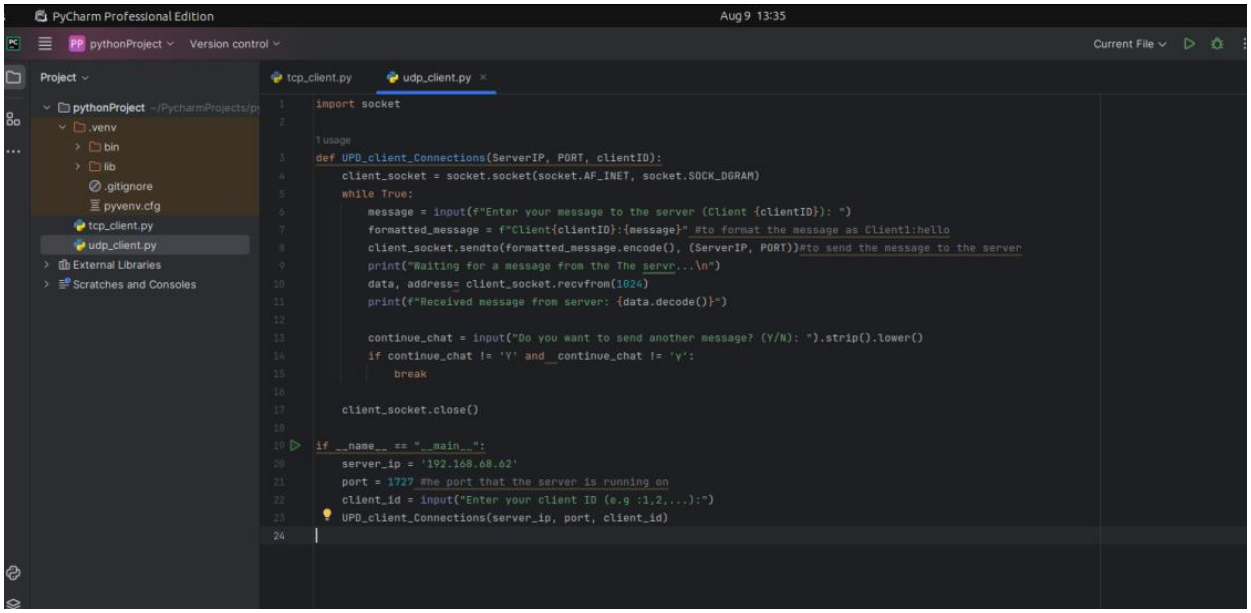


Figure 47:UDP client code.

- Here I define function called (udp client connection) that accept server IP and the port as a parameters, then (*socket.AF_INET => the socket will use IPv4 and I take it from my pc I upload image previous that define this IP, #socket.SOCK_DGRAM => specify that the socket is a UDP socket) then I connected this socket using these data*.
- *I add print messages to make the client clear about the state.*
- *Then the client sends a message to the server.*
- *Then using sendto () function that takes the message that the client write it then encode it.*
- It will receive the Client_Message from the server.

  *the 1024 is the buffer size in bytes then it will decode it.
- I added a print message to ask the client if he/she wants to continue this conversation.
- *It should close the socket after finishing the process.*
- *Finally, in the main I invoke the function with passes choices The selected port and the IP*

  ✓ Each client chooses a specific id to make the server clear about which client connected with it and can determine the last message from all clients that connected with the server.

Results and Discussions:

Screenshots of Part2_1:

Here when I run server code in terminal, the server starts working in our specified port and waits for the message from the client.

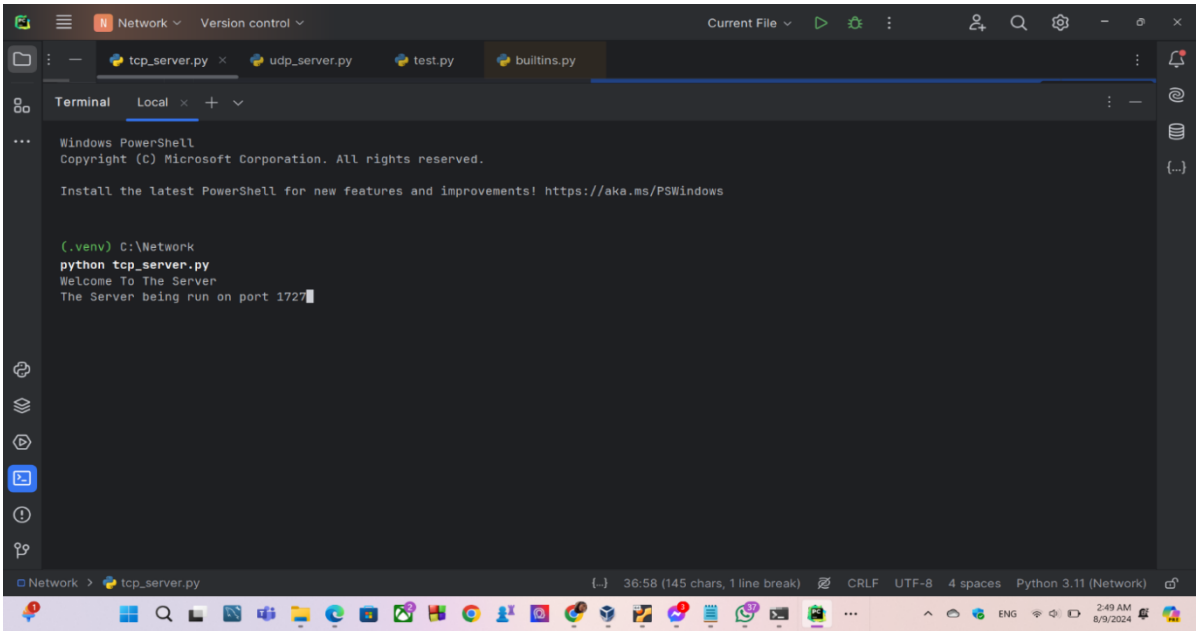


Figure 48:run the TCP server.

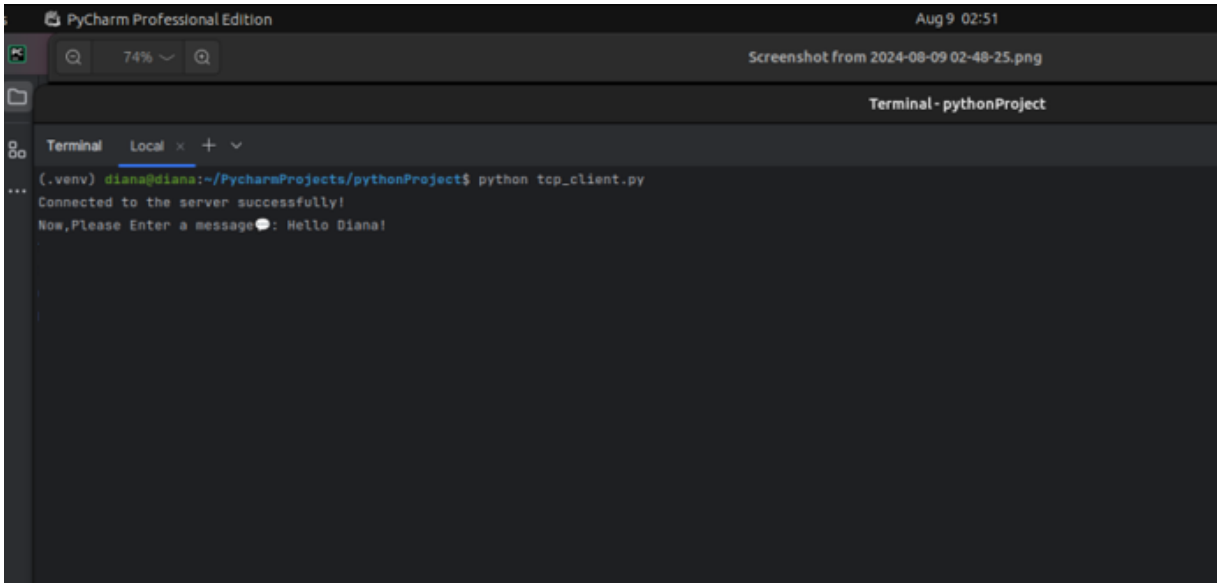


Figure 49: run the TCP client.

Here when I run client code in terminal, the client gives message to tell that its connected successfully with server then it will write the message and send it to the server.

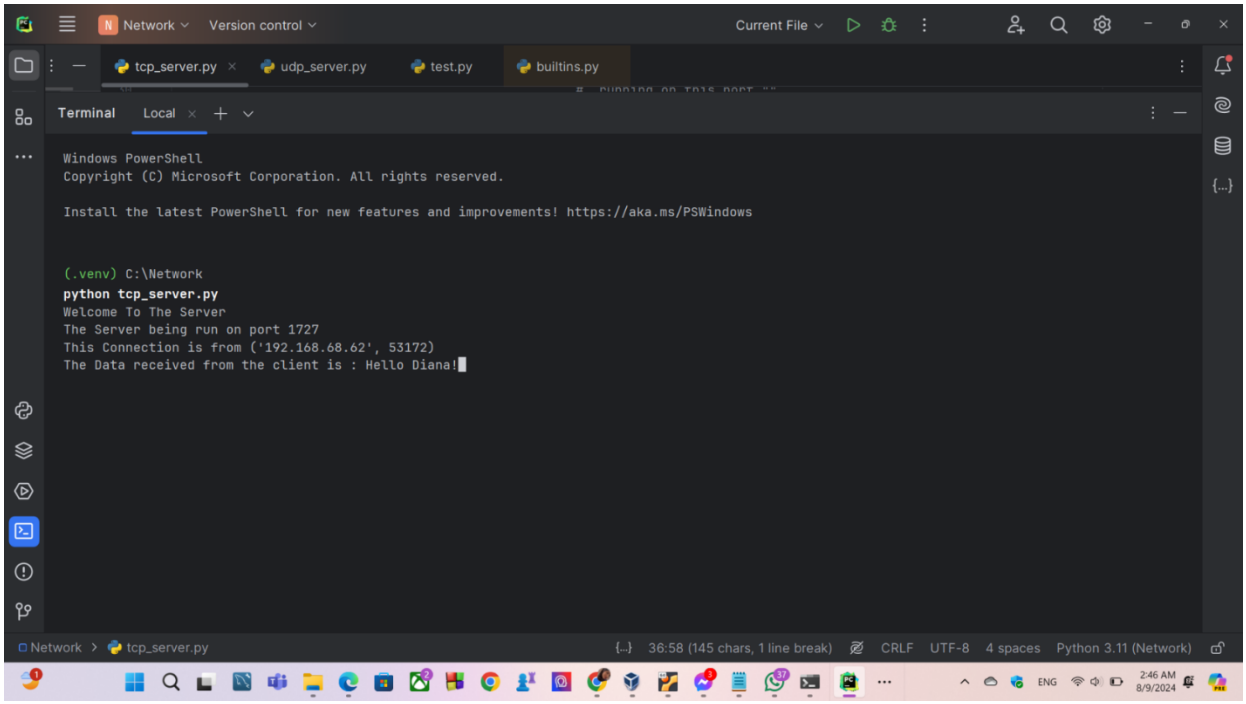


Figure 50:The response in server after sending message form the client.

Here it will get a message that gives us the connection work in which IP, and the address of the client that sent this message to the server. (Here the sent message it will enter the server code functions: which all vowels char will be replaced with '#' symbol and will send again to the client in this form.).

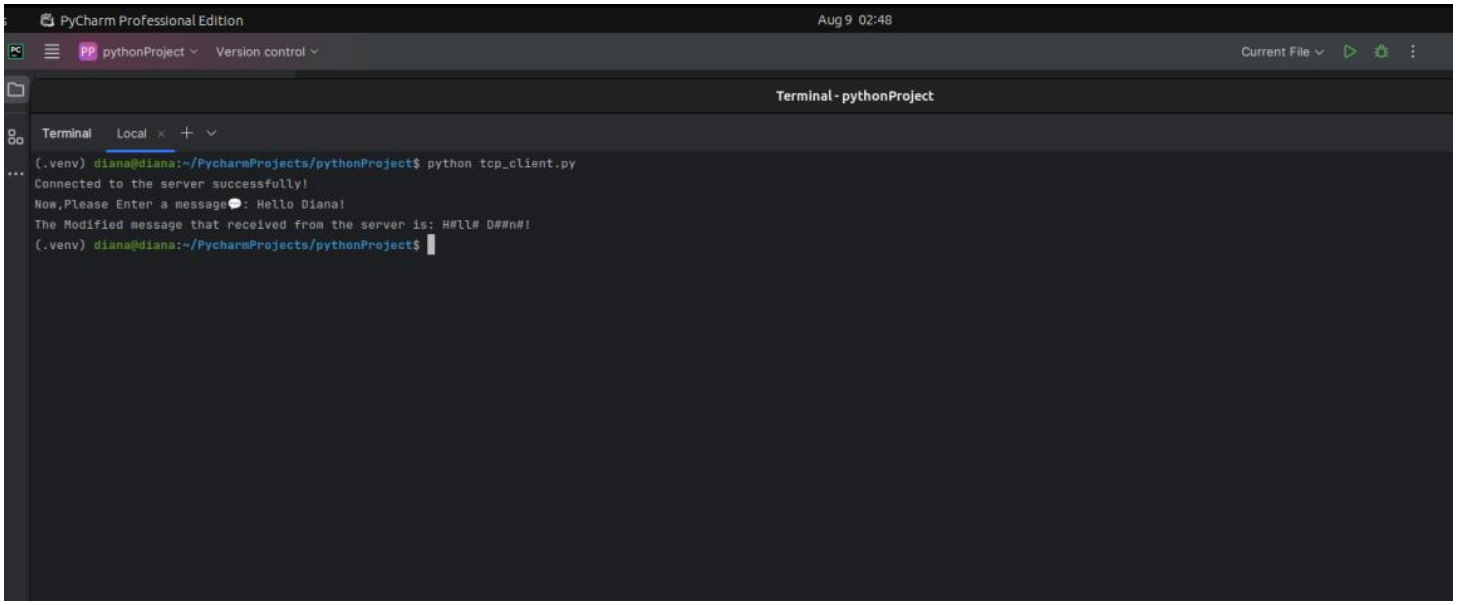


Figure 51:The state in client terminal after receiving the message form the server.

Here after receiving the modified message after changing (if it has vowels char), and it will appear in the terminal.

Screenshots of Part2_2:

Here when I run server code in terminal, the server starts working in our specified port and waits for the message from the client:

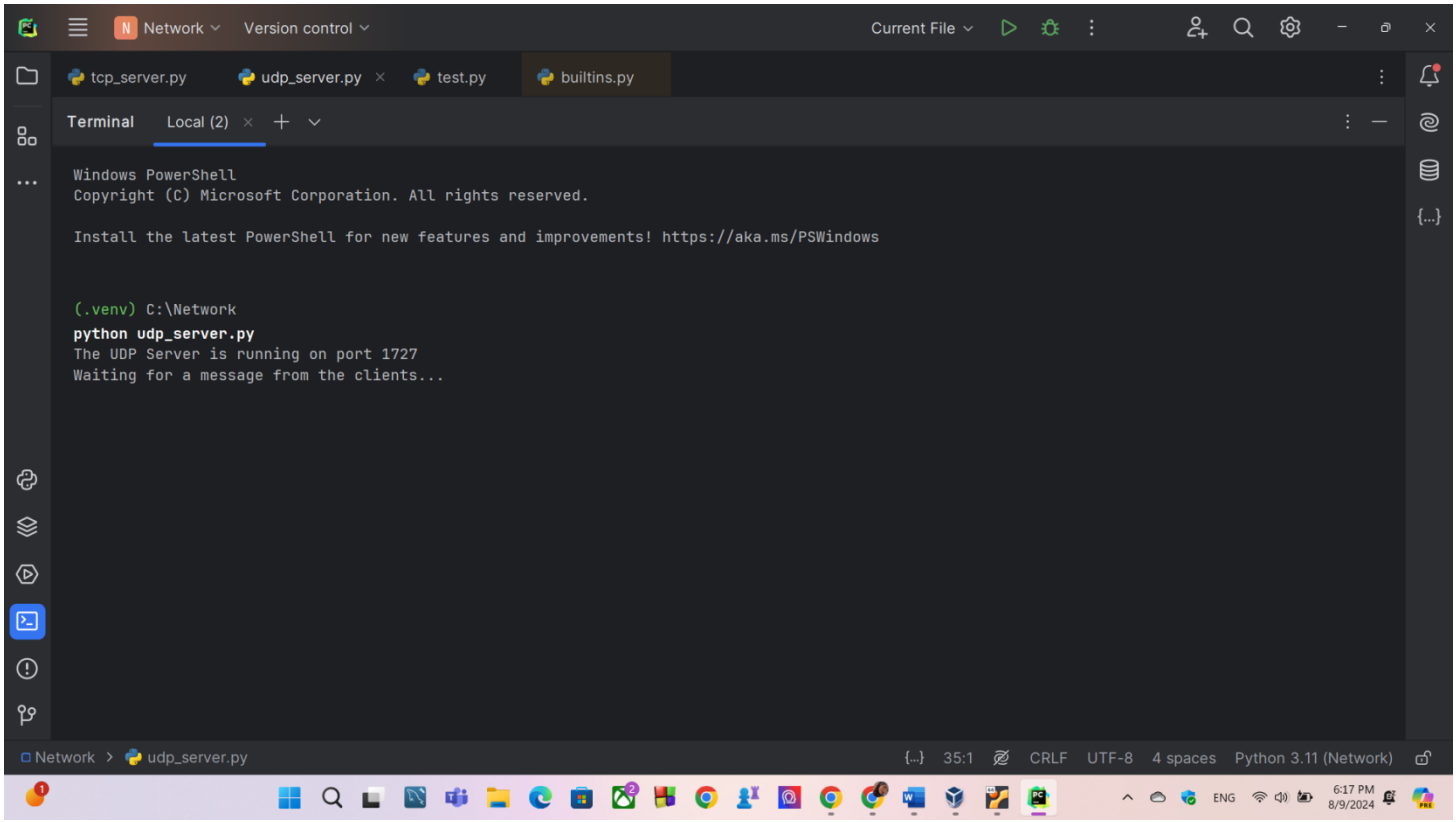


Figure 52:run the UDP server

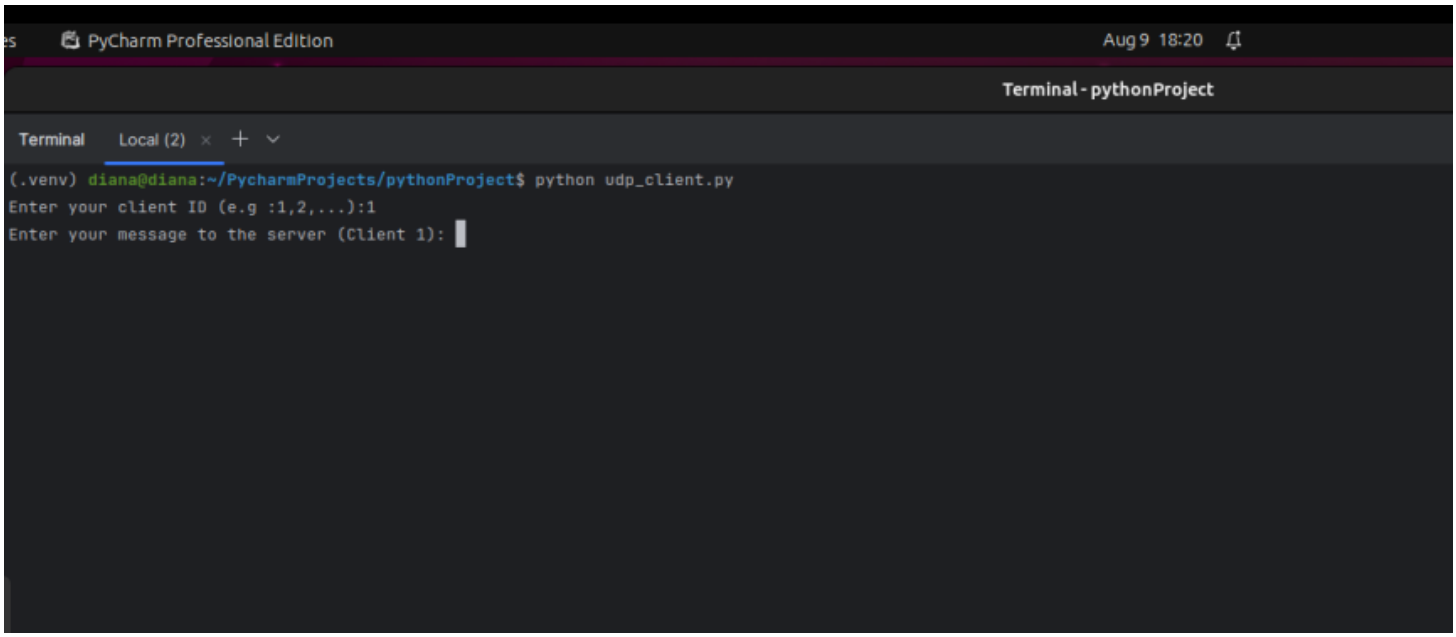


Figure 53:run the UDP client.

Here when I run client code in terminal, it should choose an ID then send to the server, then waits the server until send replay to this message.

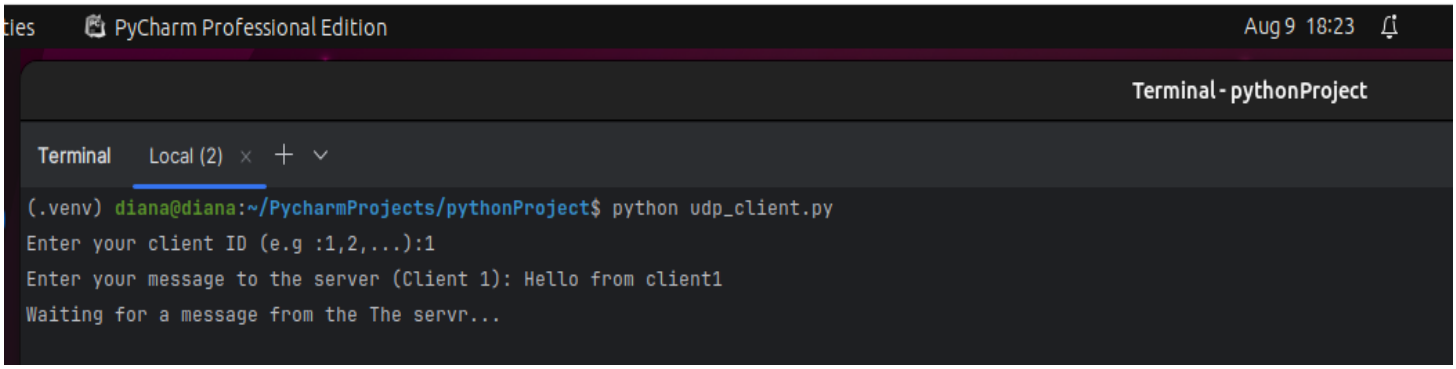


Figure 54:run the UPP client and write message to the server.

Here client 1 sends a message to the server and waits for the server to send the reply.

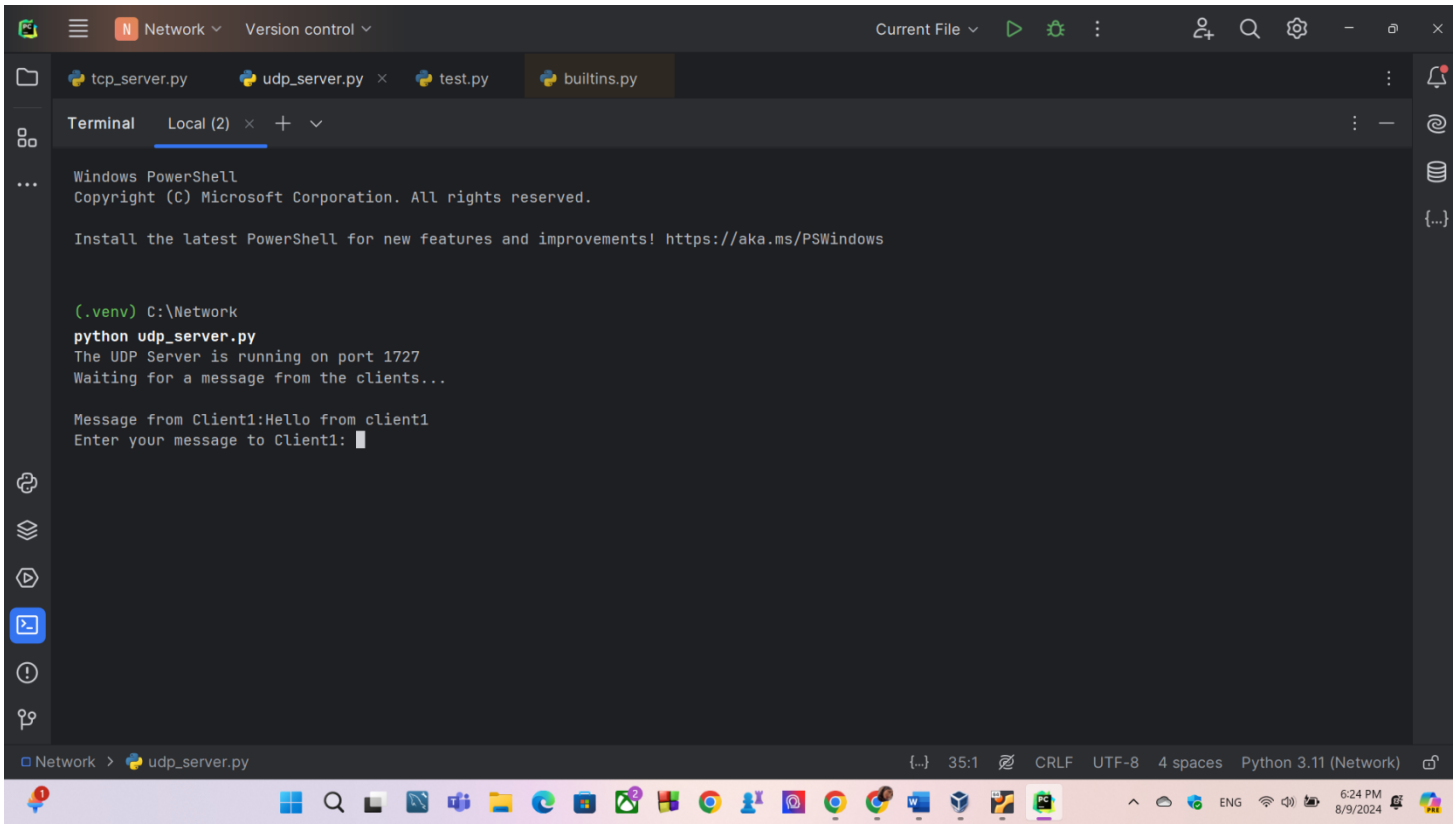


Figure 55:UDP server when have the message.

Here the server catches the message from client1 and now it should write reply to this client.

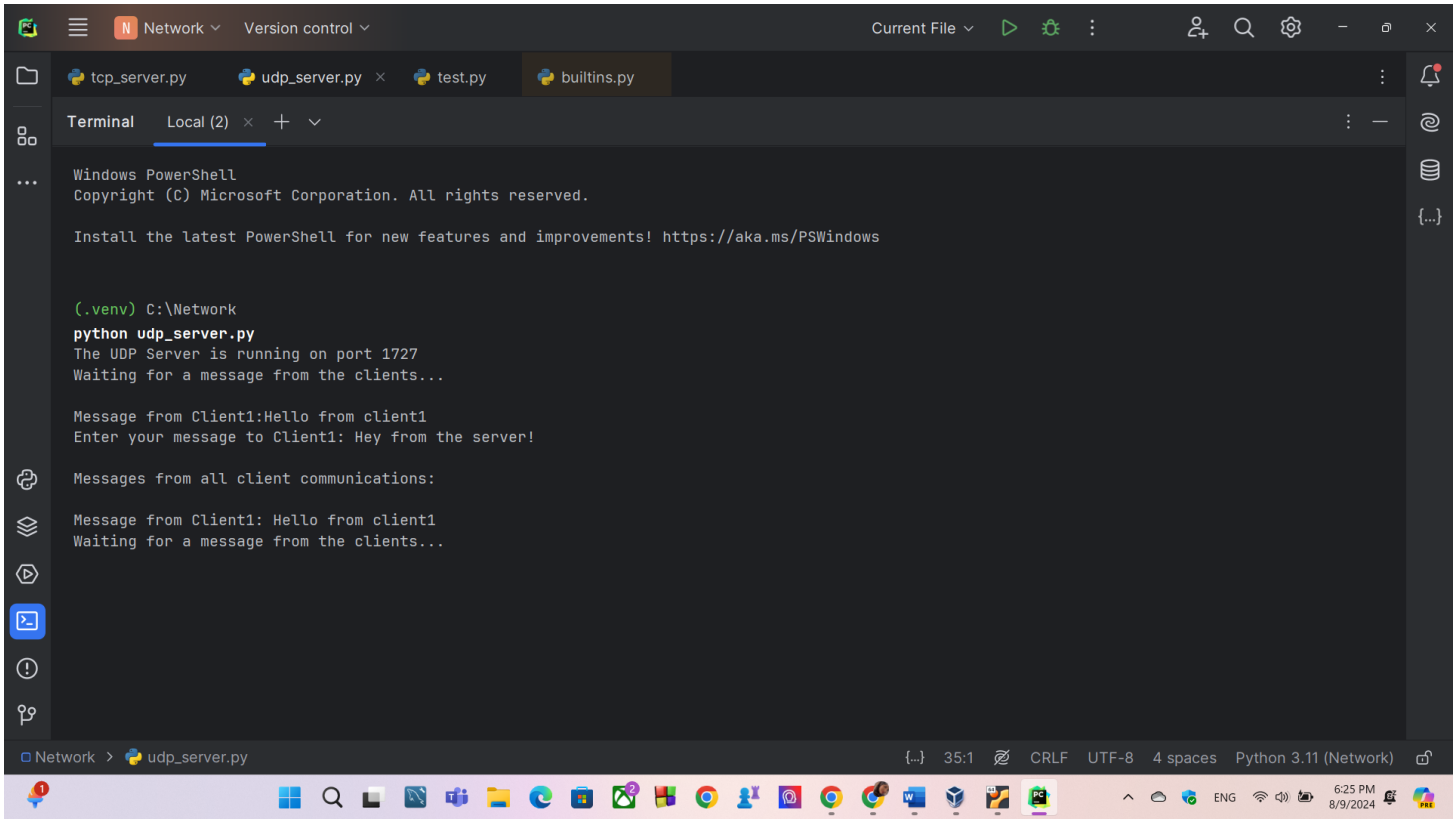


Figure 56:response from the server.

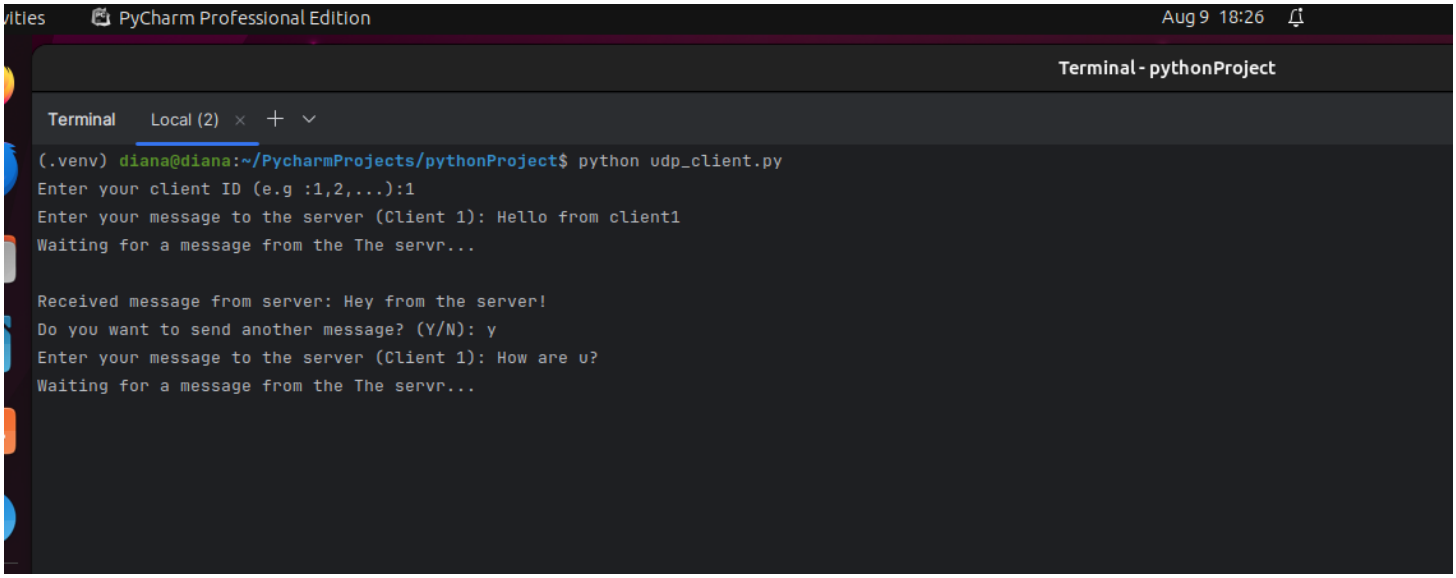


Figure 57:UDP client catch the response from server.

Here the client catches the message from server and (should answer the question if s/he need to continue this conversation if yes it will continue if not it will terminate this conv with the server and if run again and another client sent messages the server still in work and will save the last messages from all clients.).

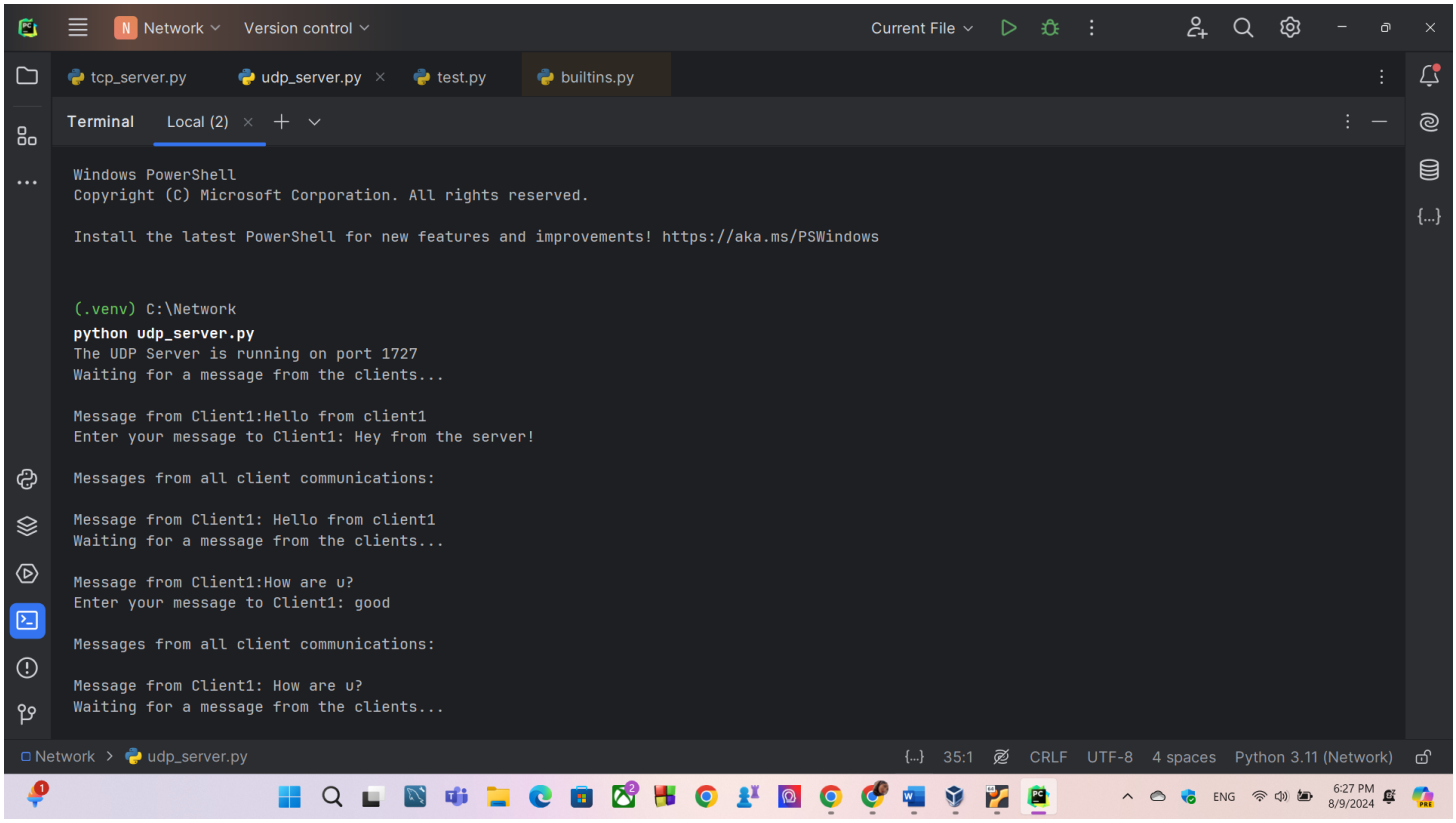


Figure 58:response again and the list of last messages from 1st client.

Here the client sends another message, and the server replies to it then the client can continue the chat or not.

If the client chooses not to continue the conversation, then another client can send to the same server but still if client1 and 2 are still in connection then can a lot of clients send to the same server.

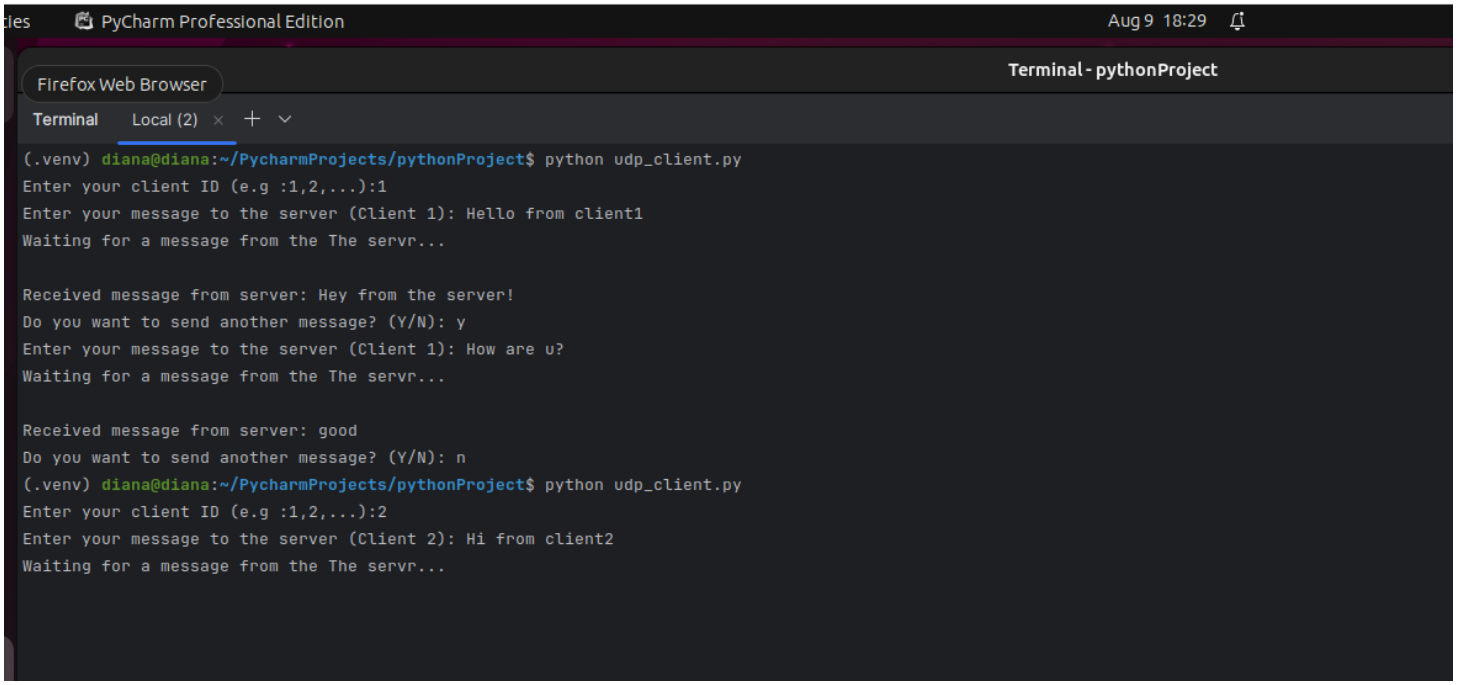


Figure 59:Another client sends to server.

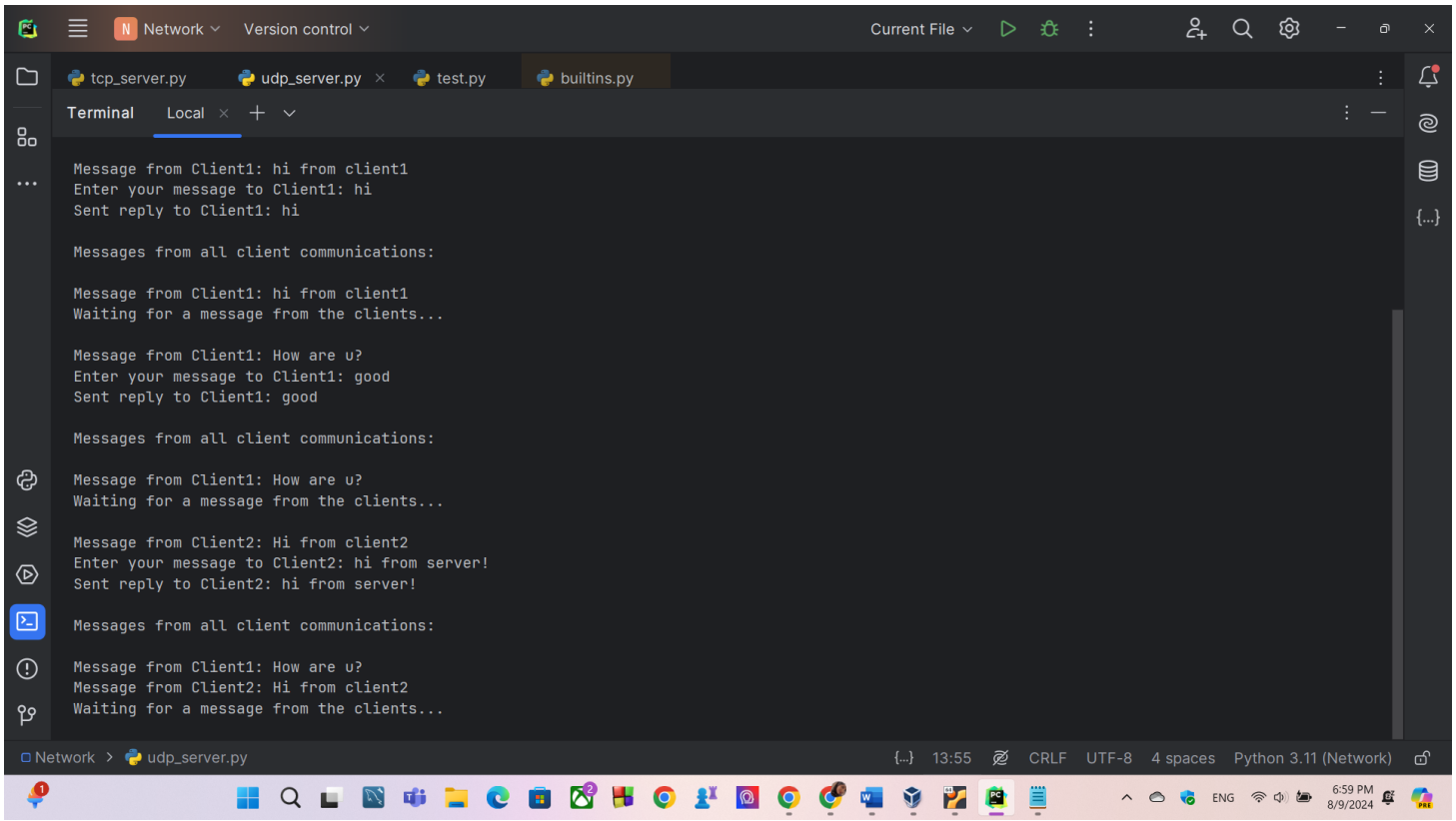


Figure 60:Response to client2 and list the communication.

Here the server receives message from another client2 then the server replies to it (with list of all last messages from all clients shows to the server.)

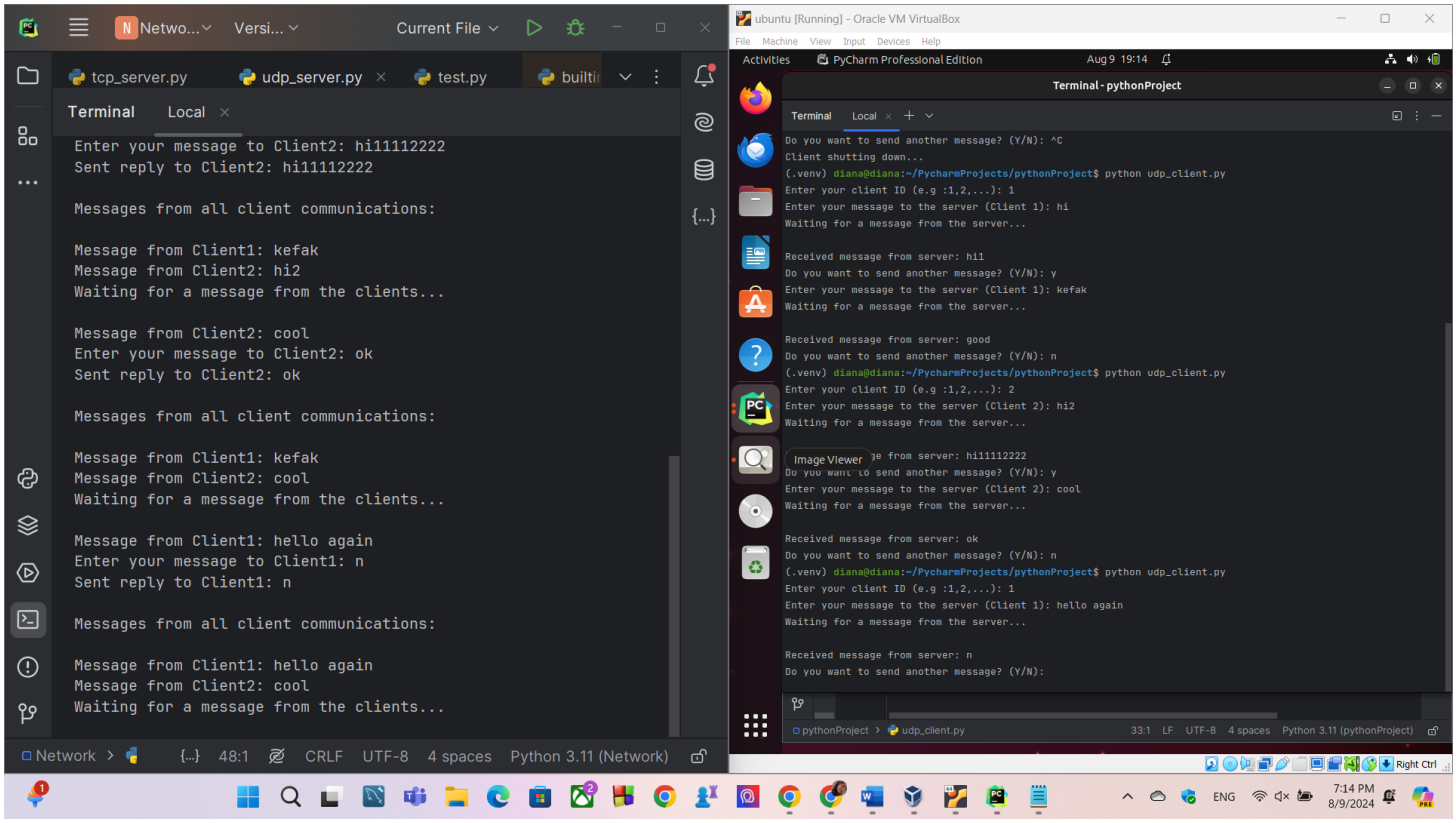


Figure 61:addition experiment to check validity of client.

Here I try to send from 2 clients, and I check if I can send message from client1 then from client2, then back to send from client1, look in the list of commendations update truly so the connection work will.

3-Part three Web Server:

3.1-Theory and procedure:

Web Servers

A web server consists of computer software and supporting hardware that may receive requests sent over HTTP or HTTPS, the secure variation of the network protocol designed for distributing web content. When a user agent—typically a web browser or web crawler—uses HTTP to request a web page or other resource, the server replies with either the resource's content or an error message. This begins communication. If set up properly, a web server can also receive, and store resources delivered by the user agent.[15]

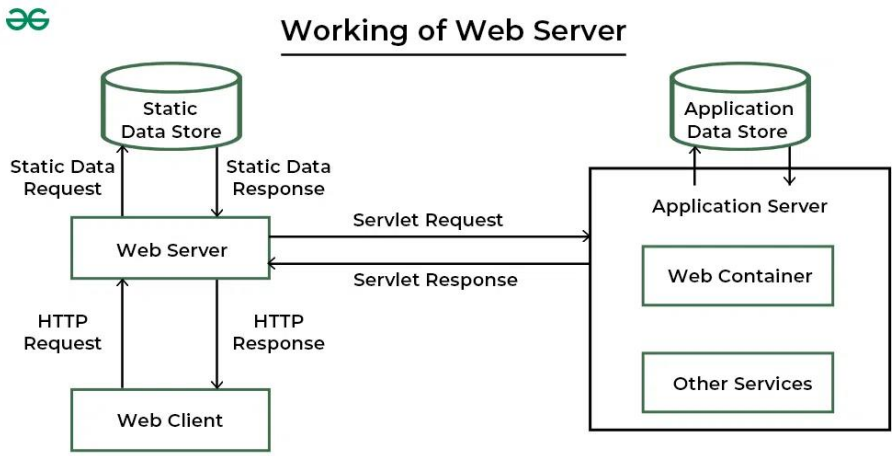


Figure 62::Web Server Architecture

Socket Programming

In order to do activities on socket objects (such as connect, receive, send, and listen), we must first instantiate them in our code. Sockets are just special objects that we build in our software and have unique techniques for handling network data and connections. The network stack, a unique component of the operating system kernel that is in charge of overseeing network operations, is what those methods actually call your operating system kernel.[16]

HTTP Protocol and Content Types

The resource's media type is indicated by the Content-Type header. The media type is a string that is sent with the file that describes its format. An image file, for instance, might have a media type of image/png, image/jpg, etc. [17]

In response, it provides the client with information about the kind of returned material. The kind of content that needs to load on the computer is discovered by the browser. When a browser receives a file's byte stream through the Content-type header, it performs a process known as MIME sniffing, in which it examines the stream it is receiving and loads the contents appropriately.[17]

Error Handling

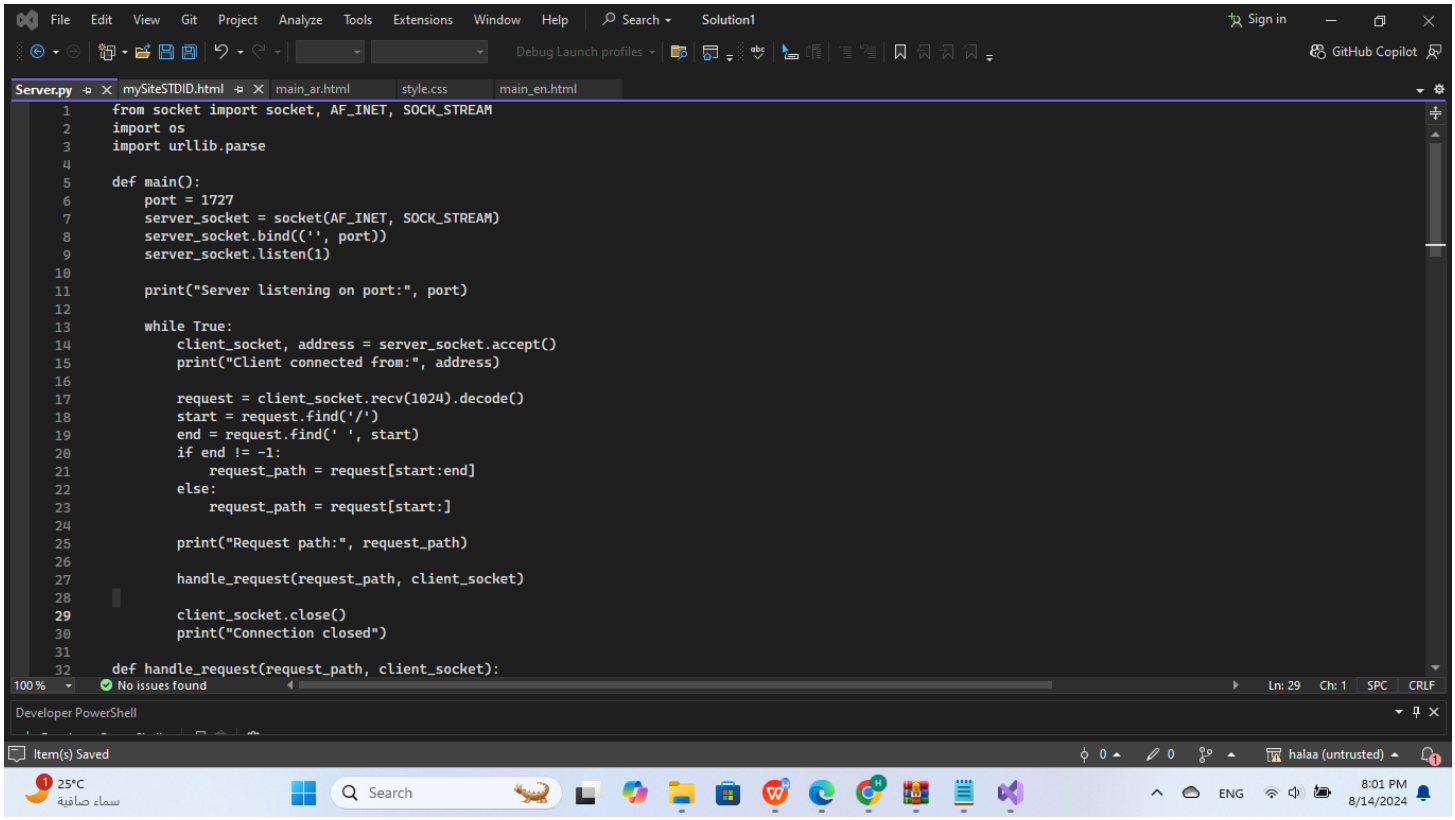
Programming and data processing both require error handling. It deals with how an application or system reacts to abnormalities or unforeseen circumstances while it is being used. Error handling in data processing might involve handling missing, corrupted, or inconsistent data.[18]

Redirection with Status Codes

The numerical codes that a client, like a web browser, receives from the server to indicate the request's status are known as HTTP status codes. Based on the code's initial digit, we can group them. 3xx is the category that includes the redirection status codes. The client should route their request to the new location as indicated by these codes, which signal that a requested resource has been relocated. Redirection status codes come in a variety of forms, including 302, 307, and 308. Every one of these has a distinct function and a set of guidelines. It's critical to comprehend these codes in order to preserve the website's integrity and offer a positive customer experience.[19]

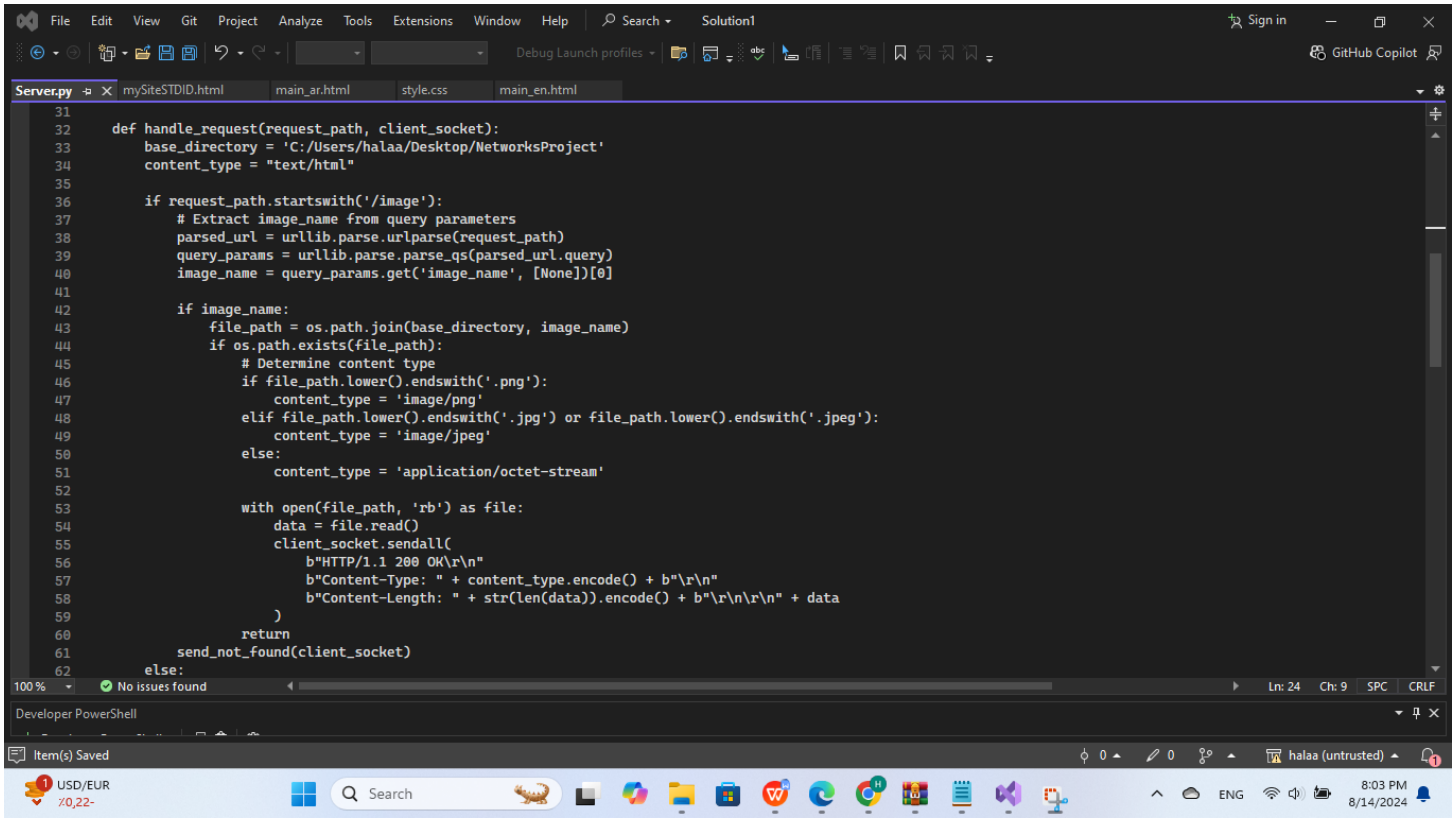
Localization Support:

The purpose of localization services is to modify goods, information, and services to meet the functional, linguistic, and cultural needs of a particular target market or audience. In order to make sure that the product or content resonates with the local audience, reflects their customs, preferences, and cultural subtleties, and functions well within their particular region, the localization process entails adjusting several parts beyond basic translation.[20]



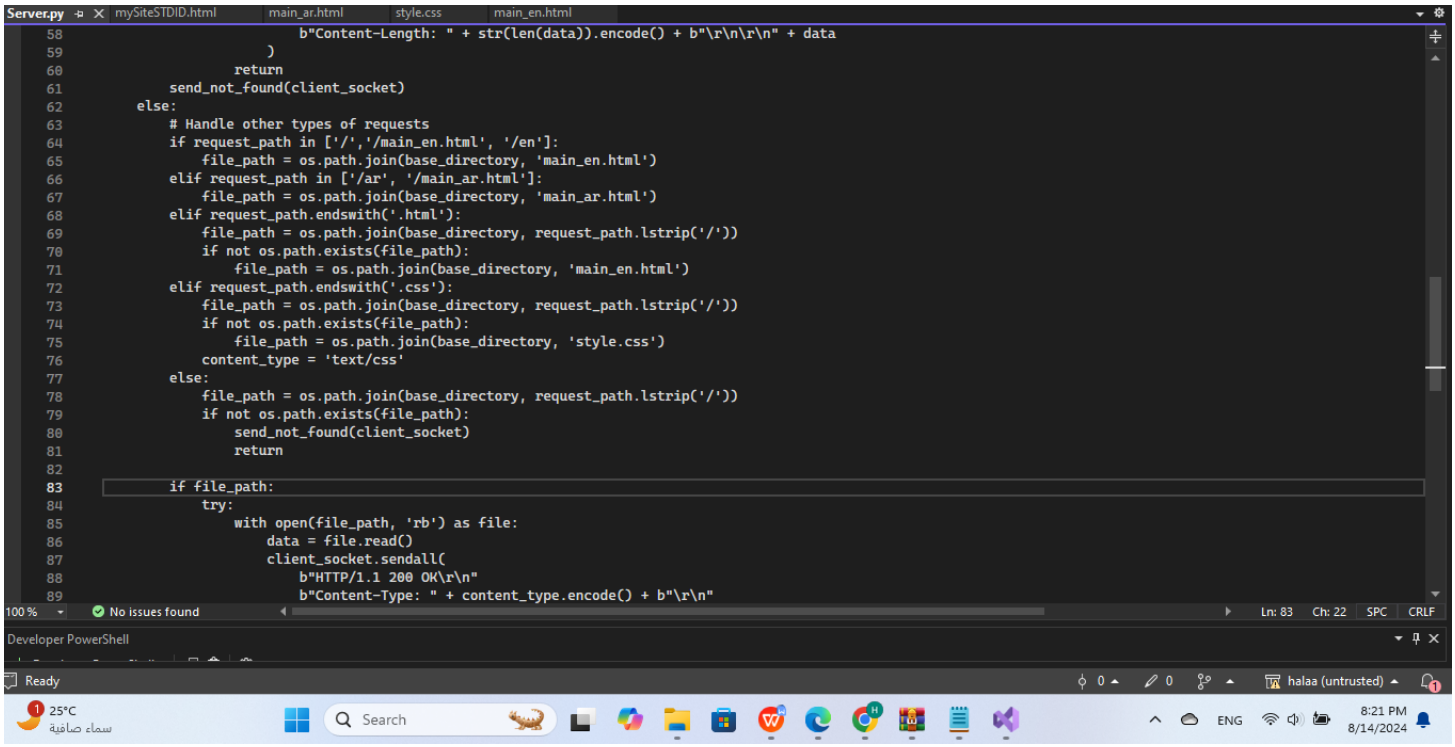
```python
from socket import socket, AF_INET, SOCK_STREAM
import os
import urllib.parse

def main():
    port = 1727
    server_socket = socket(AF_INET, SOCK_STREAM)
    server_socket.bind(('', port))
    server_socket.listen(1)

    print("Server listening on port:", port)

    while True:
        client_socket, address = server_socket.accept()
        print("Client connected from:", address)

        request = client_socket.recv(1024).decode()
        start = request.find('/')
        end = request.find(' ', start)
        if end != -1:
            request_path = request[start:end]
        else:
            request_path = request[start:]

        print("Request path:", request_path)

        handle_request(request_path, client_socket)

        client_socket.close()
        print("Connection closed")

def handle_request(request_path, client_socket):
```

Figure 63:Part1 of Server code



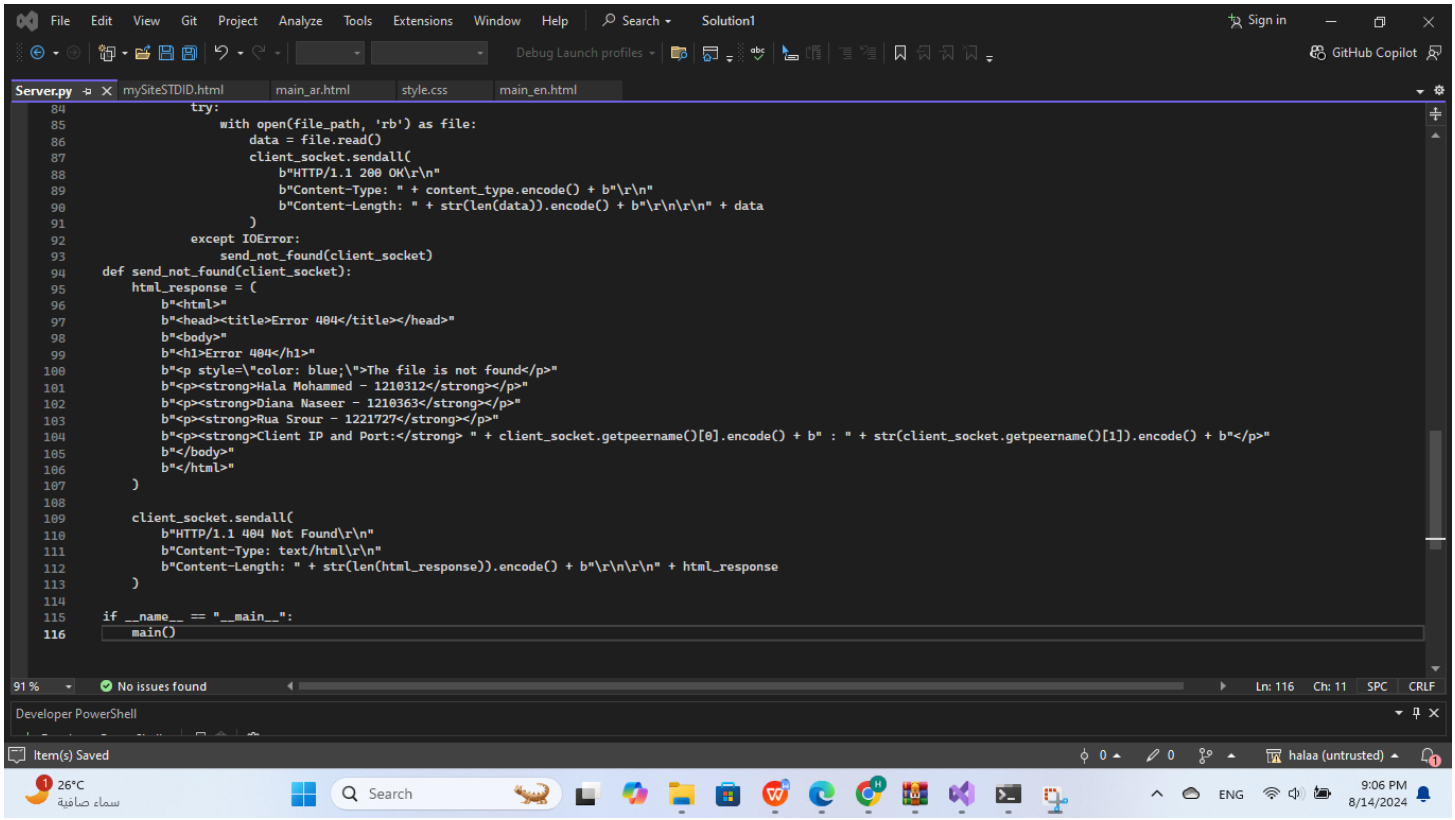
```python
def handle_request(request_path, client_socket):
    base_directory = 'C:/Users/halaa/Desktop/NetworksProject'
    content_type = "text/html"

    if request_path.startswith('/image'):
        # Extract image_name from query parameters
        parsed_url = urllib.parse.urlparse(request_path)
        query_params = urllib.parse.parse_qs(parsed_url.query)
        image_name = query_params.get('image_name', [None])[0]

        if image_name:
            file_path = os.path.join(base_directory, image_name)
            if os.path.exists(file_path):
                # Determine content type
                if file_path.lower().endswith('.png'):
                    content_type = 'image/png'
                elif file_path.lower().endswith('.jpg') or file_path.lower().endswith('.jpeg'):
                    content_type = 'image/jpeg'
                else:
                    content_type = 'application/octet-stream'

                with open(file_path, 'rb') as file:
                    data = file.read()
                    client_socket.sendall(
                        b"HTTP/1.1 200 OK\r\n"
                        b"Content-Type: " + content_type.encode() + b"\r\n"
                        b"Content-Length: " + str(len(data)).encode() + b"\r\n\r\n" + data
                    )
                return
        send_not_found(client_socket)
    else:
```

Figure 64:Part2 of Server code



```
                        b"Content-Length: " + str(len(data)).encode() + b"\r\n\r\n" + data
                    )
                return
            send_not_found(client_socket)
        else:
            # Handle other types of requests
            if request_path in ['/','/main_en.html', '/en']:
                file_path = os.path.join(base_directory, 'main_en.html')
            elif request_path in ['/ar', '/main_ar.html']:
                file_path = os.path.join(base_directory, 'main_ar.html')
            elif request_path.endswith('.html'):
                file_path = os.path.join(base_directory, request_path.lstrip('/'))
                if not os.path.exists(file_path):
                    file_path = os.path.join(base_directory, 'main_en.html')
            elif request_path.endswith('.css'):
                file_path = os.path.join(base_directory, request_path.lstrip('/'))
                if not os.path.exists(file_path):
                    file_path = os.path.join(base_directory, 'style.css')
                content_type = 'text/css'
            else:
                file_path = os.path.join(base_directory, request_path.lstrip('/'))
                if not os.path.exists(file_path):
                    send_not_found(client_socket)
                    return

            if file_path:
                try:
                    with open(file_path, 'rb') as file:
                        data = file.read()
                        client_socket.sendall(
                            b"HTTP/1.1 200 OK\r\n"
                            b"Content-Type: " + content_type.encode() + b"\r\n"
```

Figure 65:Part 3 of Server code



```
                try:
                    with open(file_path, 'rb') as file:
                        data = file.read()
                        client_socket.sendall(
                            b"HTTP/1.1 200 OK\r\n"
                            b"Content-Type: " + content_type.encode() + b"\r\n"
                            b"Content-Length: " + str(len(data)).encode() + b"\r\n\r\n" + data
                        )
                except IOError:
                    send_not_found(client_socket)
def send_not_found(client_socket):
    html_response = (
        b"<html>"
        b"<head><title>Error 404</title></head>"
        b"<body>"
        b"<h1>Error 404</h1>"
        b"<p style=\"color: blue;\">The file is not found</p>"
        b"<p><strong>Hala Mohammed - 1210312</strong></p>"
        b"<p><strong>Diana Naseer - 1210363</strong></p>"
        b"<p><strong>Rua Srour - 1221727</strong></p>"
        b"<p><strong>Client IP and Port:</strong> " + client_socket.getpeername()[0].encode() + b" : " + str(client_socket.getpeername()[1]).encode() + b"</p>"
        b"</body>"
        b"</html>"
    )

    client_socket.sendall(
        b"HTTP/1.1 404 Not Found\r\n"
        b"Content-Type: text/html\r\n"
        b"Content-Length: " + str(len(html_response)).encode() + b"\r\n\r\n" + html_response
    )

if __name__ == "__main__":
    main()
```

Figure 66:Part 5 of Server code

This is a Python program for a simple web server via socket programming to handle incoming HTTP requests. The program listens for client connections on port 1727. Immediately after a client connects, it reads the request, identifies the path of the requested resource, and delivers its content using the handle_request function. This function serves different file types such as HTML, CSS, and images from a given directory. If it is a request for an image, it will extract the image name in the URL and serve it if available. It serves up a custom 404 Not Found page in case it is a resource not found for smooth user experience.

```
    # Handle other types of requests
    if request_path in ['/','/main_en.html', '/en']:
        file_path = os.path.join(base_directory, 'main_en.html')
```

Figure 67:Handle Request

This part of the code checks for requests for one of the following : `/`, `/main_en.html`, or `/en`, for instance, `localhost:1727/` or `localhost:1727/en`. If one has one of these paths, then it sends in response one and the same file - `main_en.html`, with `Content-Type` in the headers.

First, we run the server code. It appears on the terminal that it is listening to the port.

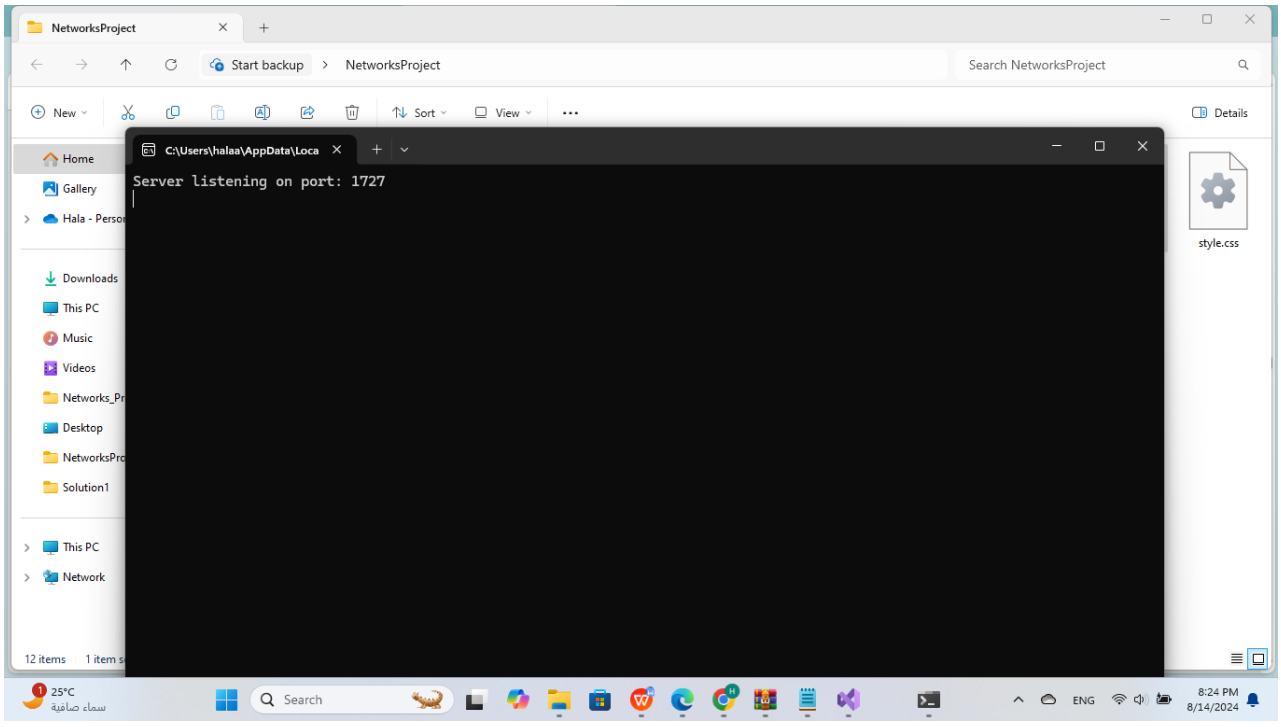


Figure 68:run the server

Second, we go to the browser and search for http://localhost:1727/main_en.html and the following appears:

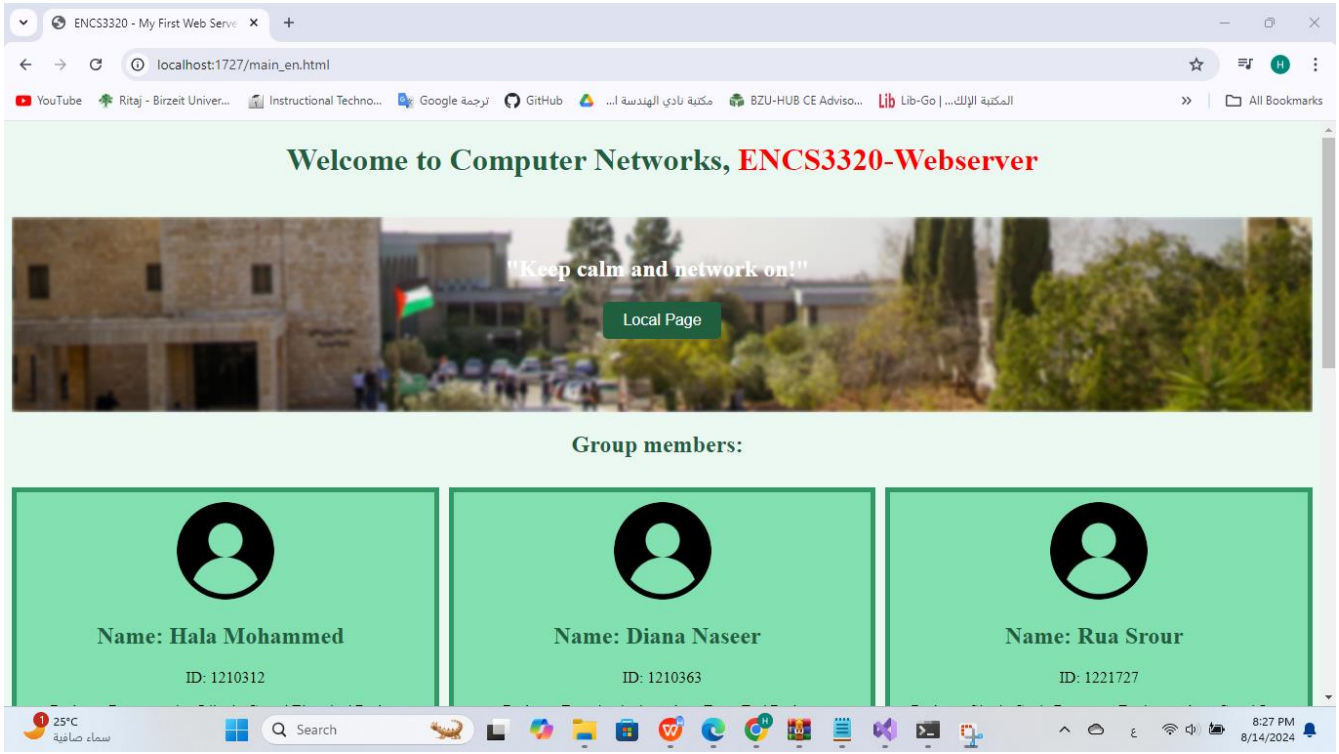


Figure 69:main_en Page Part1

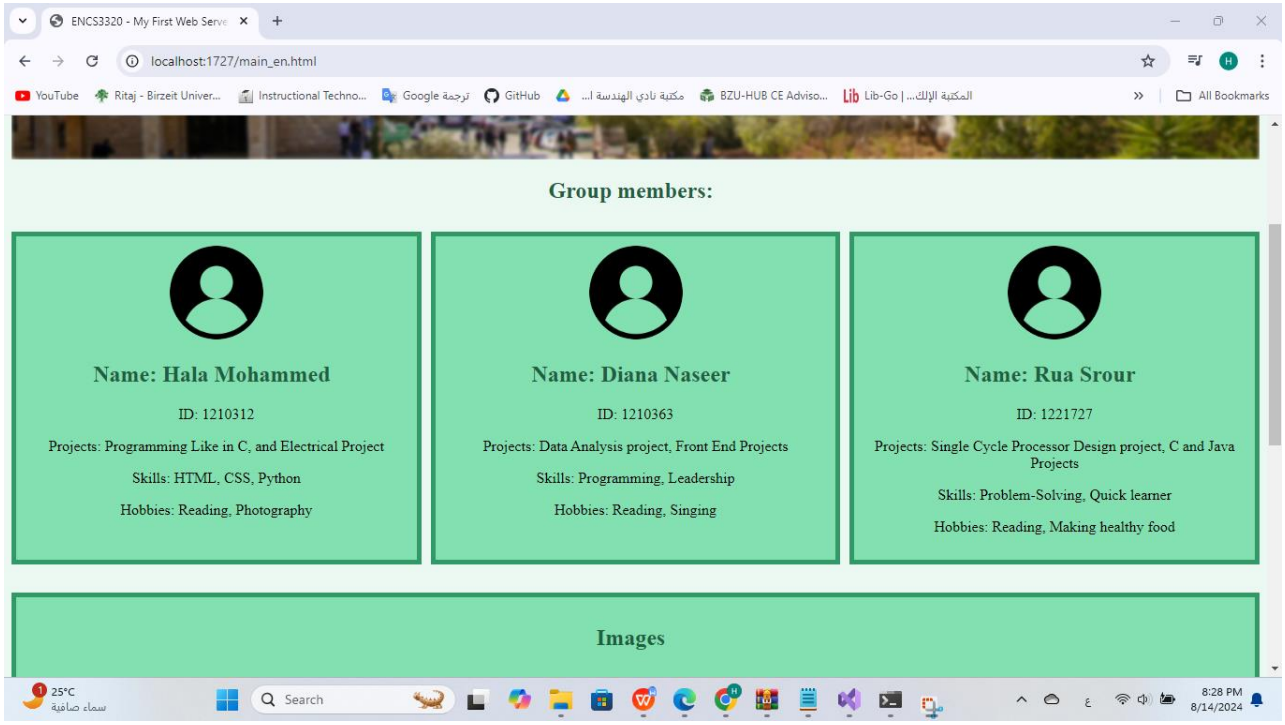


Figure 70:main_en Page Part2

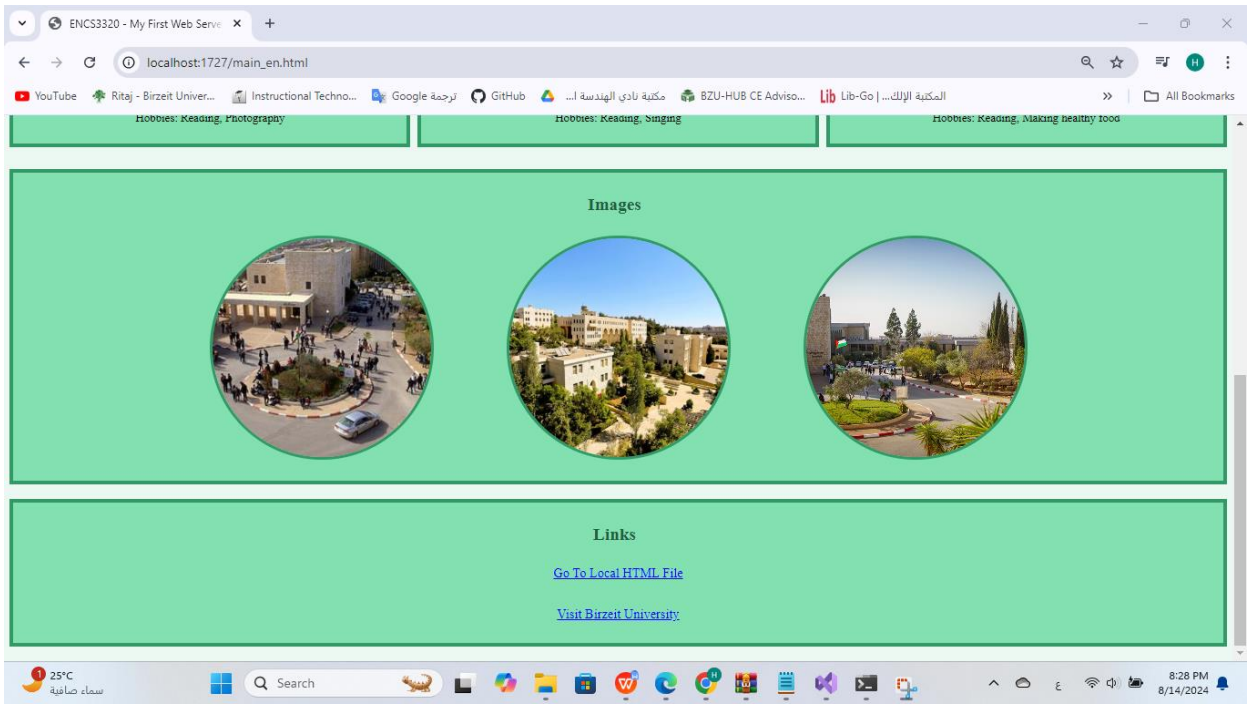


Figure 71:main_en Page Part3

This button takes us to the local page.

Figure 72:Local Page Button

This is the local page.

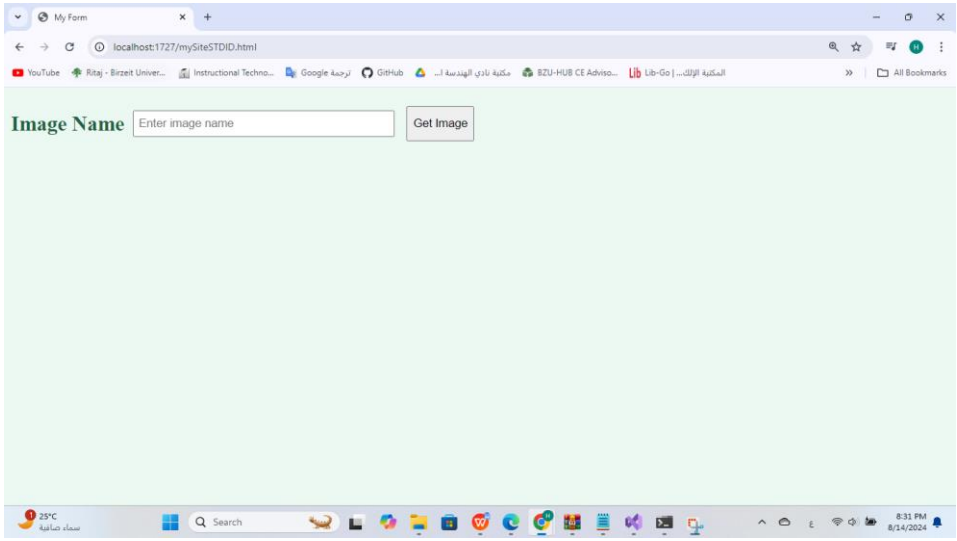


Figure 73:Local Page

Here we find two links, the first takes us to the local page like the previous button.

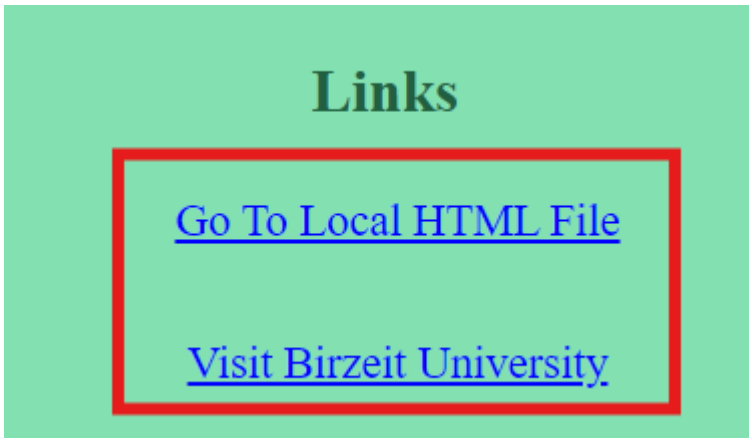


Figure 74:Links

The second link takes us to the next page.

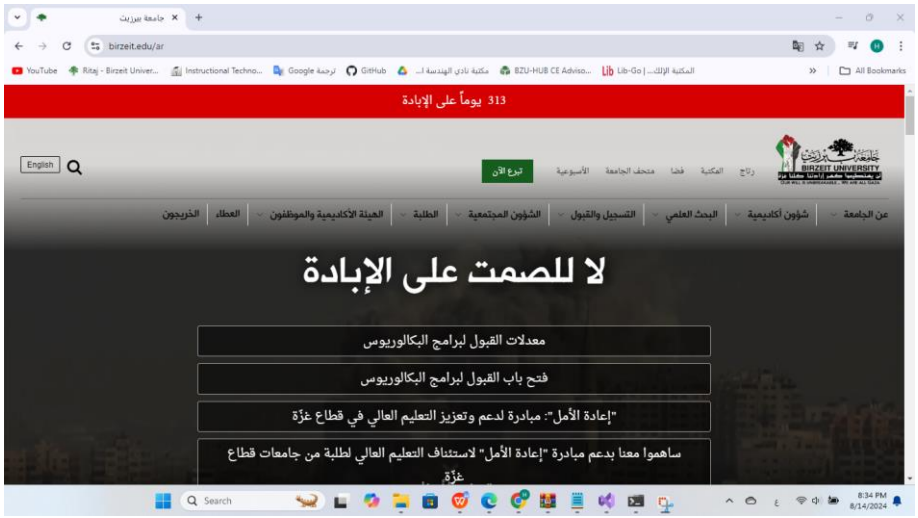


Figure 75:BZU Page

Also, after running the server, we can request the page in Arabic http://localhost:1727/main_ar.html and it will appear as the English page with all its details, with only changing the language.

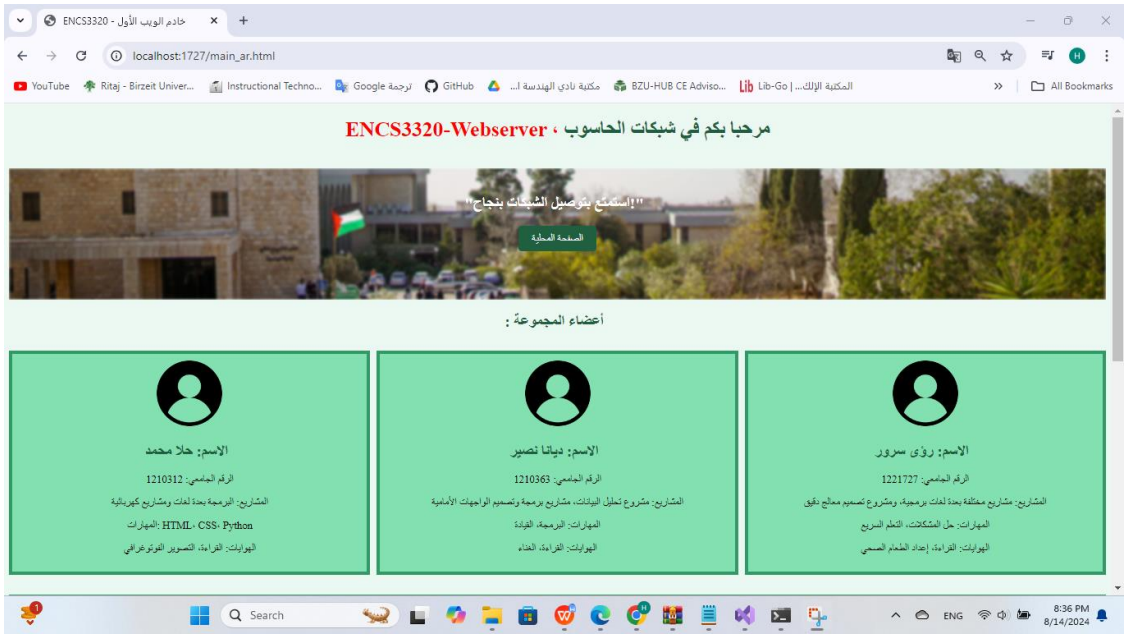


Figure 76:main_ar Page Part1

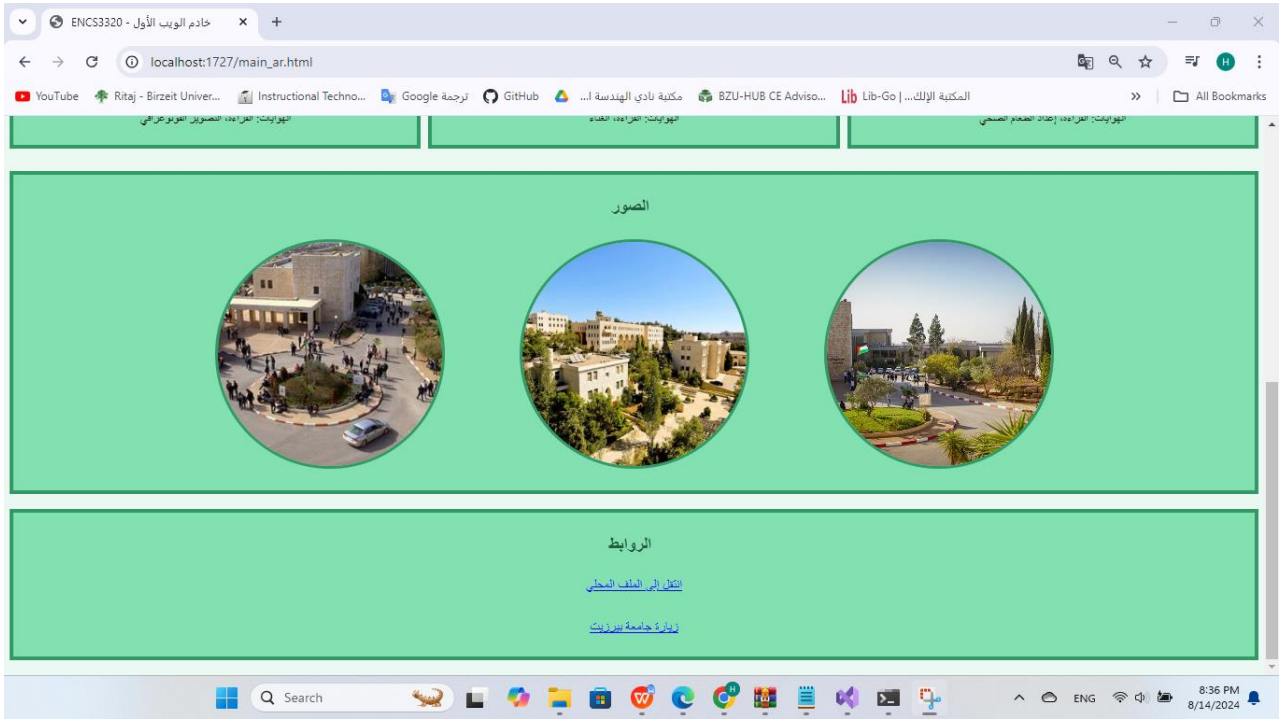


Figure 77:main_ar Page Part2

The previous two pages and the local page were written using HTML and CSS was used to arrange the page layout.

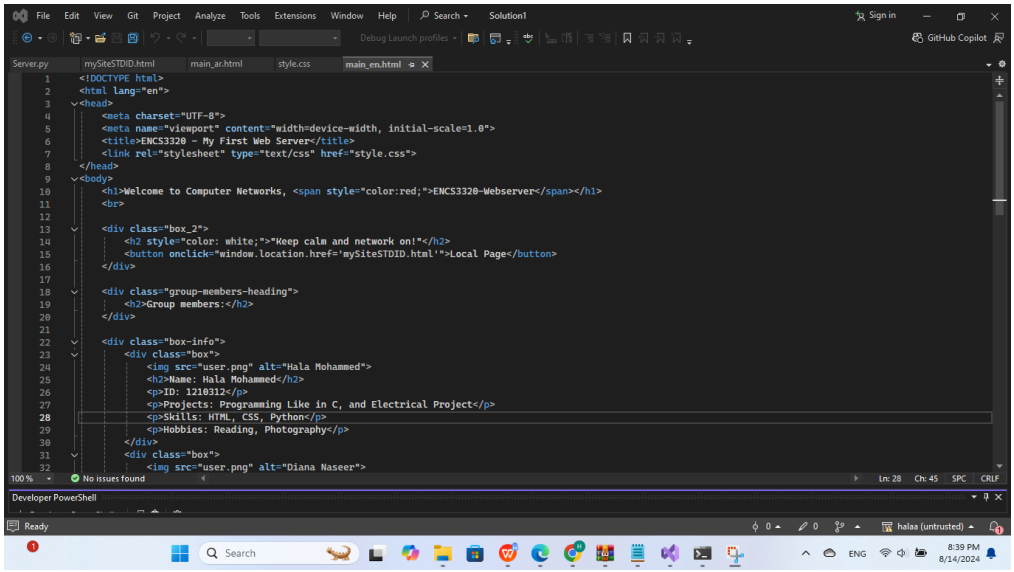


Figure 78:main_en Code Part1

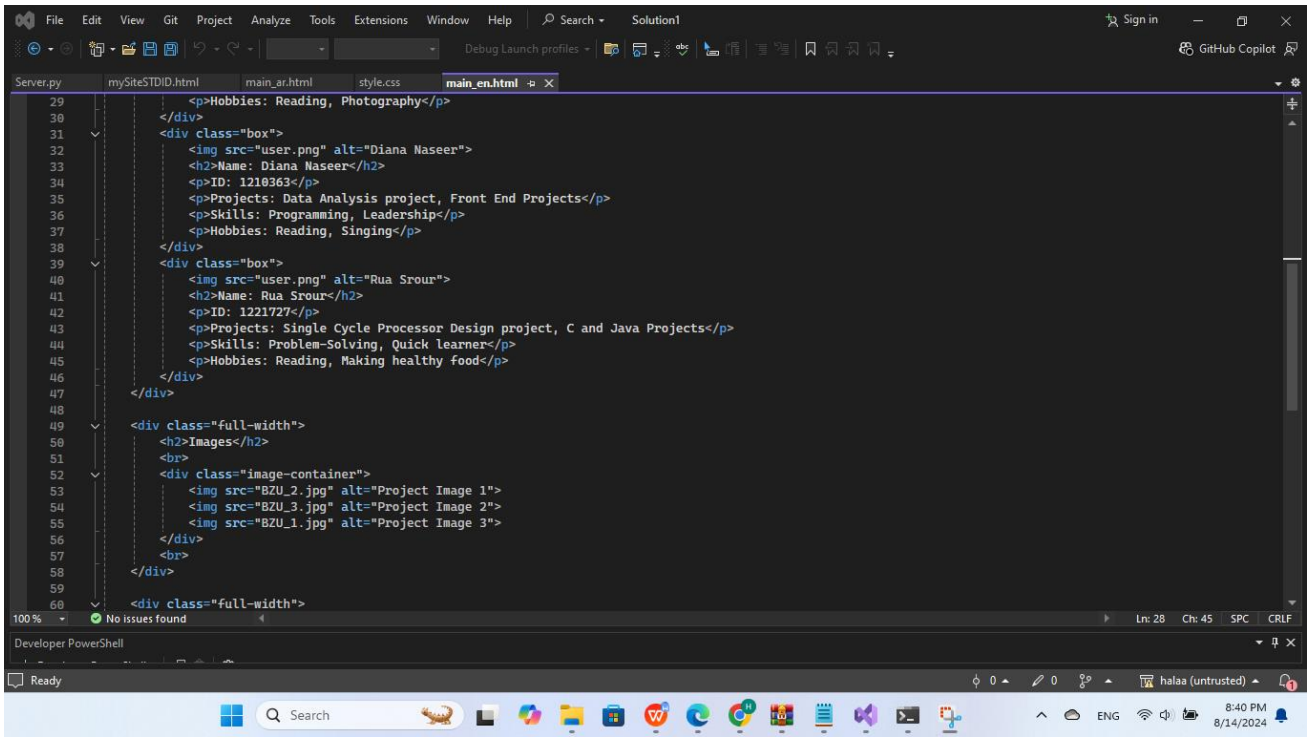


Figure 79:main_en Code Part2

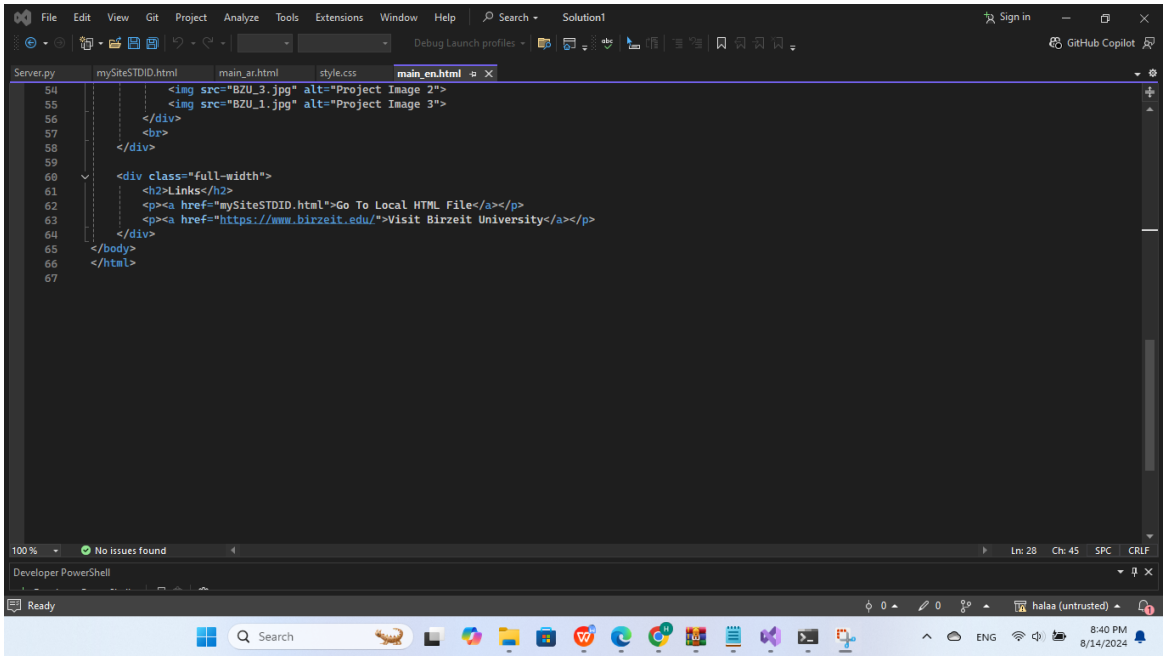


Figure 80:main_en Code Part3

The HTML file is describe a web page for Computer Networks project . The first description is the title of the page, which is "ENCS3320 - My First Web Server". The main content in the body includes a heading for welcoming the user to the course, displaying the course name in red, and a quotation related to motivation with a button. Upon clicking the latter, the user will be able to access some HTML file available locally `mySiteSTDID.html`. Also It contains a section to our group members , which shows photos, names, IDs, projects, skills, and hobbies; also, it contains a part that represents some pictures of Birzeit University and another part that contains the local HTML file and the website of Birzeit University.

The `<meta charset="UTF-8">' tag will let the server know that the webpage is in UTF-8 encoding, supporting a wide range of characters and symbols from most languages. This helps text display correctly across different browsers and devices. The class `<meta name="viewport" content="width=device-width, initial-scale=1.0">` sets the value of width to the width of the devices and sets the initial zoom level to 1.0. This attribute does, more or less, what one might expect from its name: it changes the mobile devices' scaling of a webpage. Thus, this will help develop a responsive page to offer an excellent user experience since the page is going to be viewable and well-formatted on all screen sizes.

The Arabic version of the Main page is written exactly like the English version with the language changed.

The Local Page HTML code

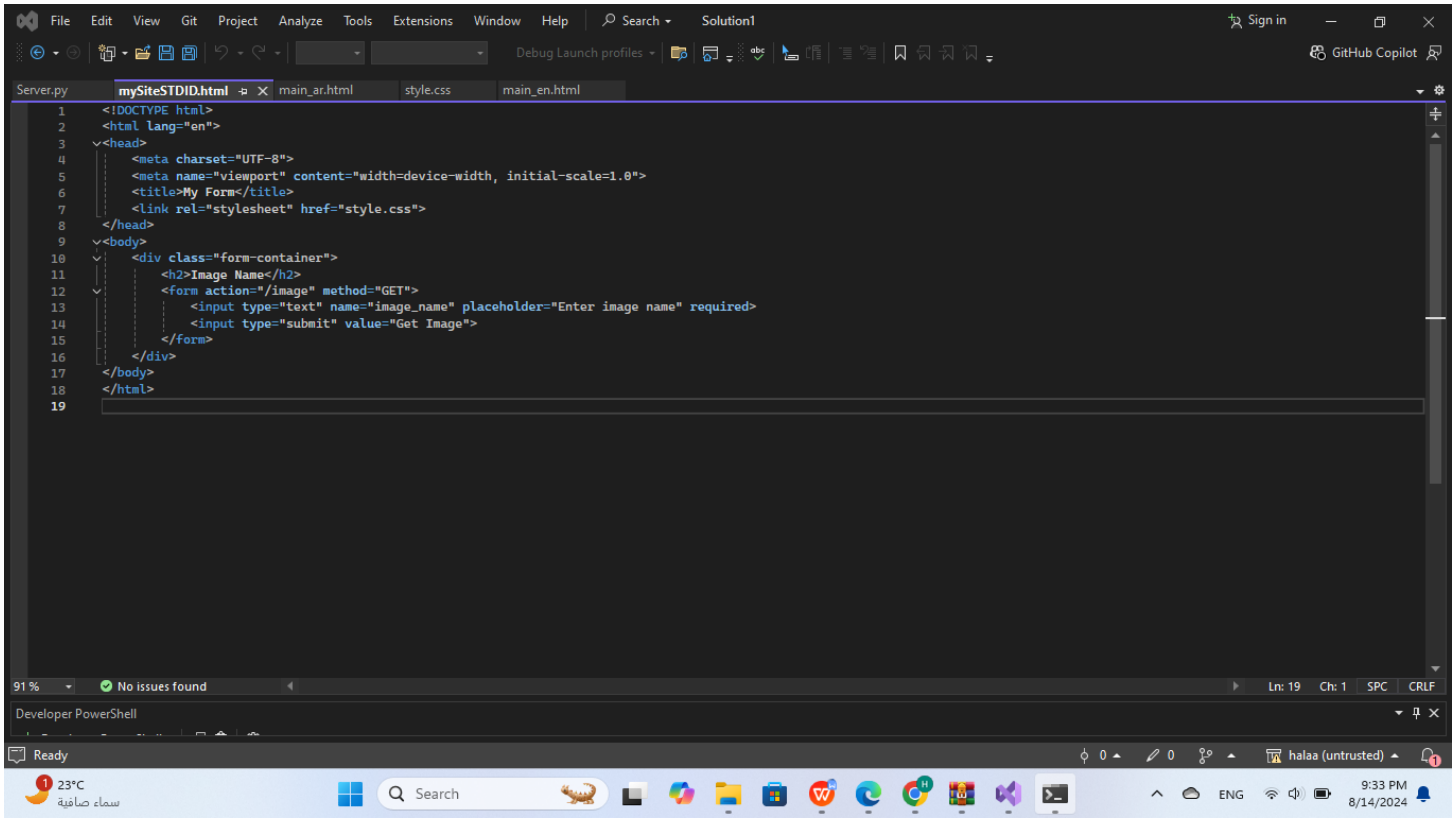


Figure 81:The Local Page HTML code

On the page http://localhost:1727/mySiteSTDID.html the purpose is for the user to search for an image with several different extensions and if it is not there or an error occurs, an error message will certainly be displayed.

If I search for image that is exist (jpg)

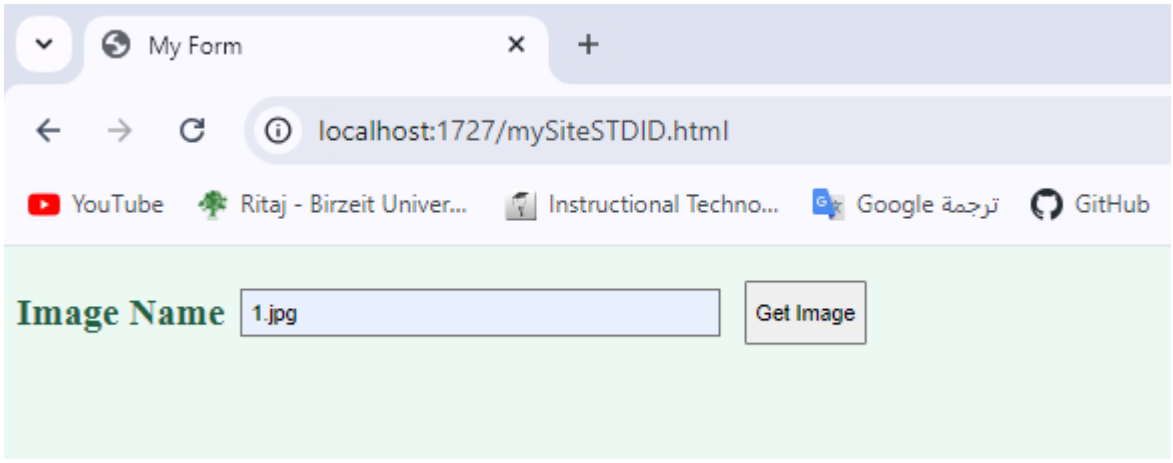


Figure 82:search for image that is exist (jpg)

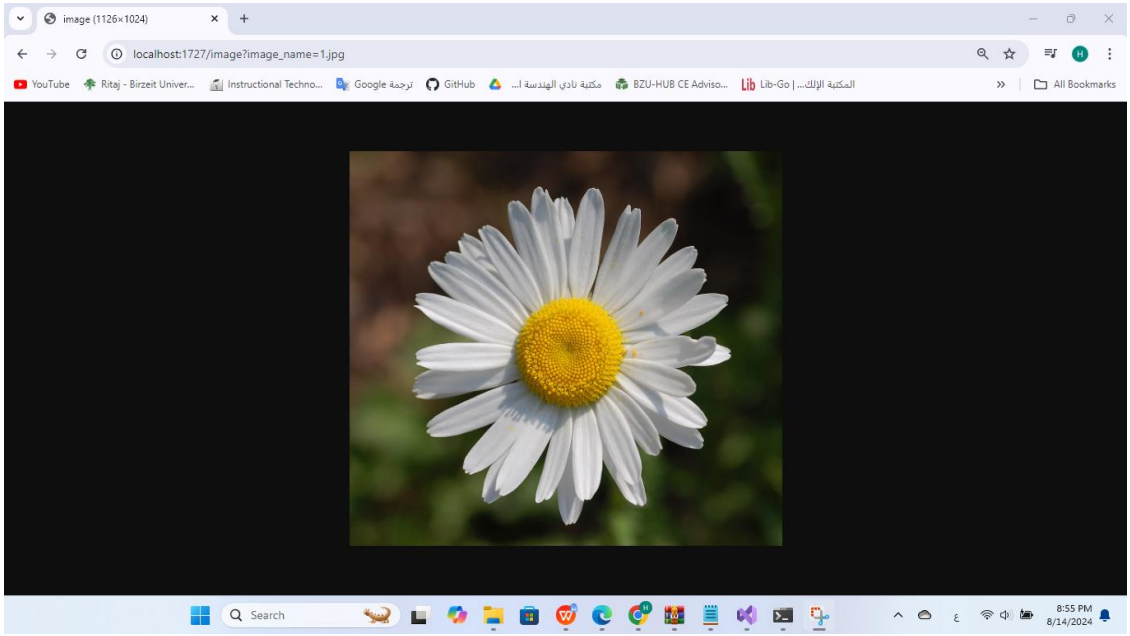


Figure 83:Result of search for image that is exist (jpg)

If I search for image that is exist (png)

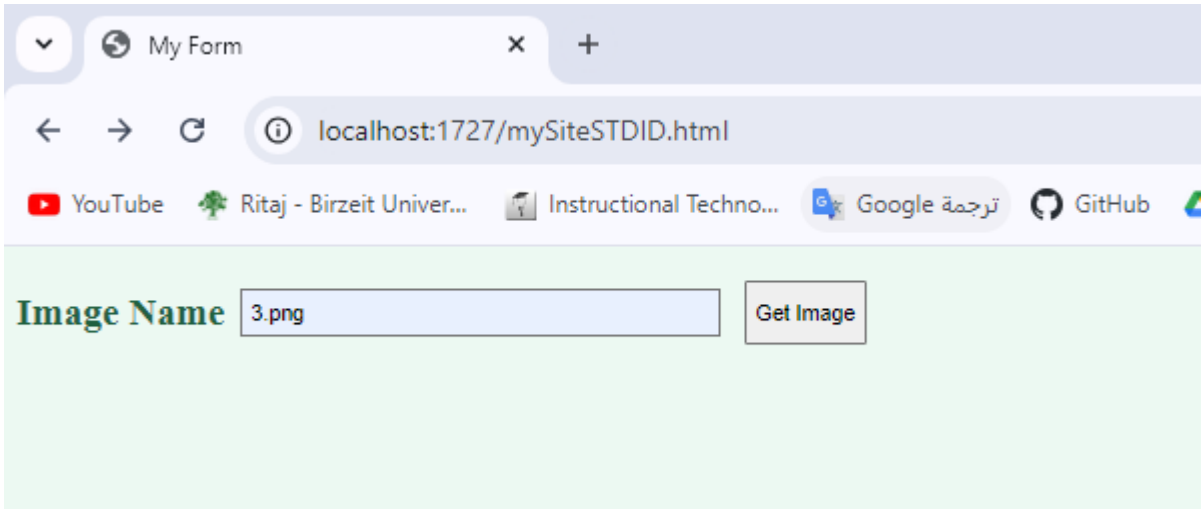


Figure 84:search for image that is exist (png)

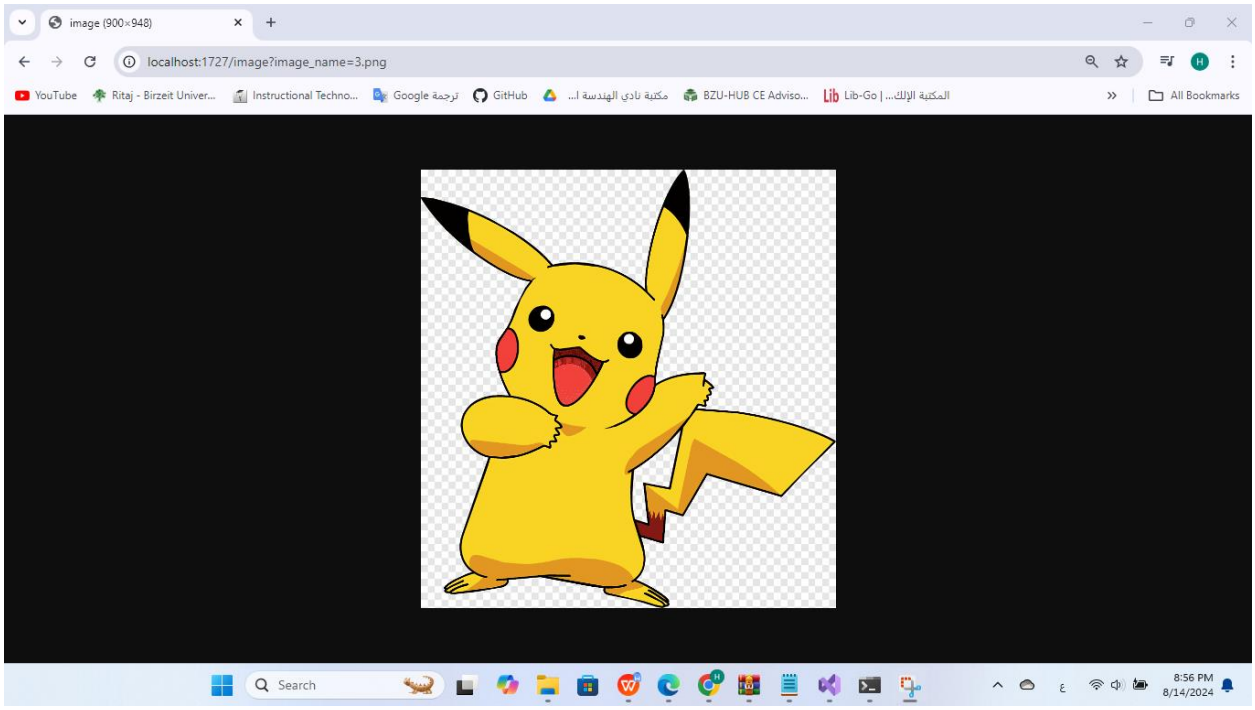


Figure 85:Result of search for image that is exist (png)

If I search for image that is not exist

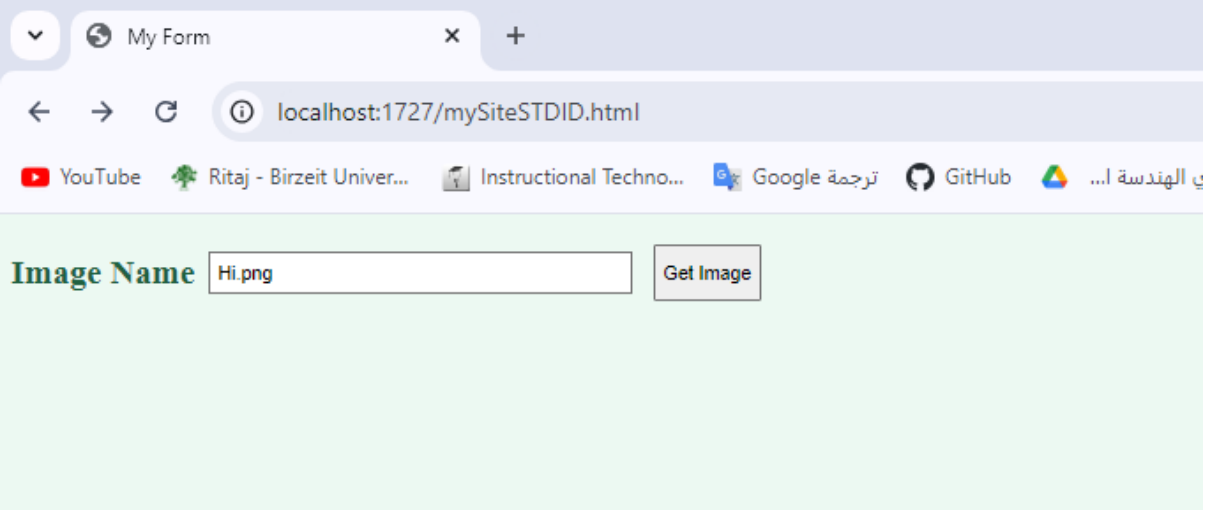


Figure 86:search for image that is not exist

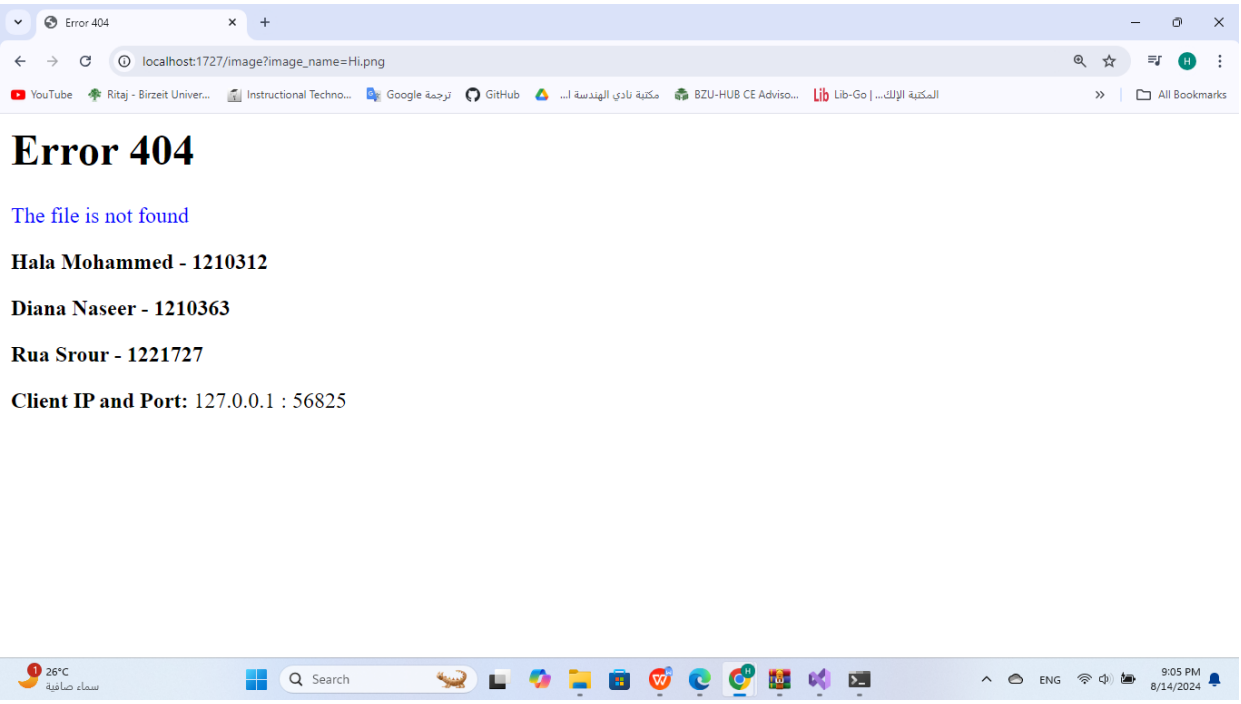


Figure 87:Result of search for image that is not exist

3.2- Results and Discussions:

When I ask for http://localhost:1727/main_en.html

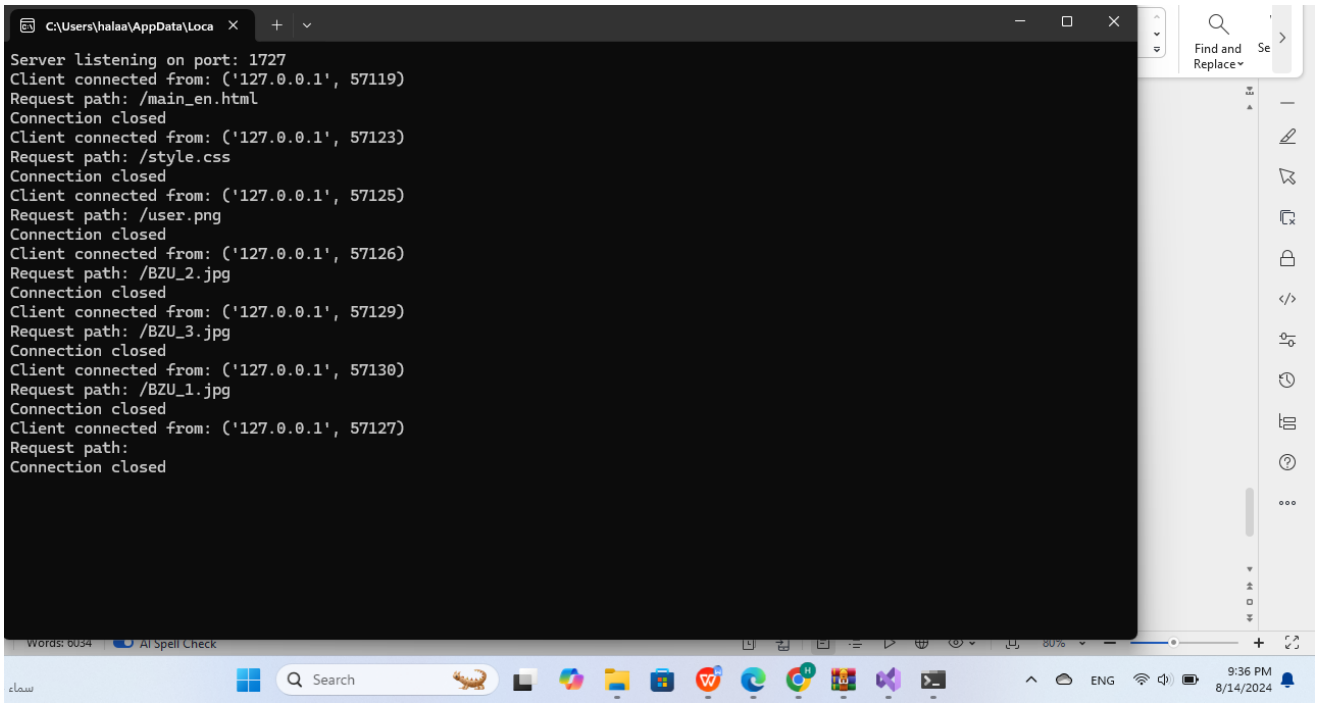


Figure 88:Server Result for http://localhost:1727/main_en.html

These log entries show that some client is connecting from 127.0.0.1 and sending various requests for resources on the server. The IP address and the port number of the client are maintained for each of the connections, and it places the requested particular resource and serves it back, such as main_en.html, style.css and various image files, including user.png and others. Here, the server looks for the appropriate file in its directory to each request and sends it back to the client. The "Connection closed" notification indicates that a server closes a connection after serving each request.For http://localhost:1727/main_ar.html is the same

When I ask for http://localhost:1727/mySiteSTDID.html OR https://www.birzeit.edu/ar

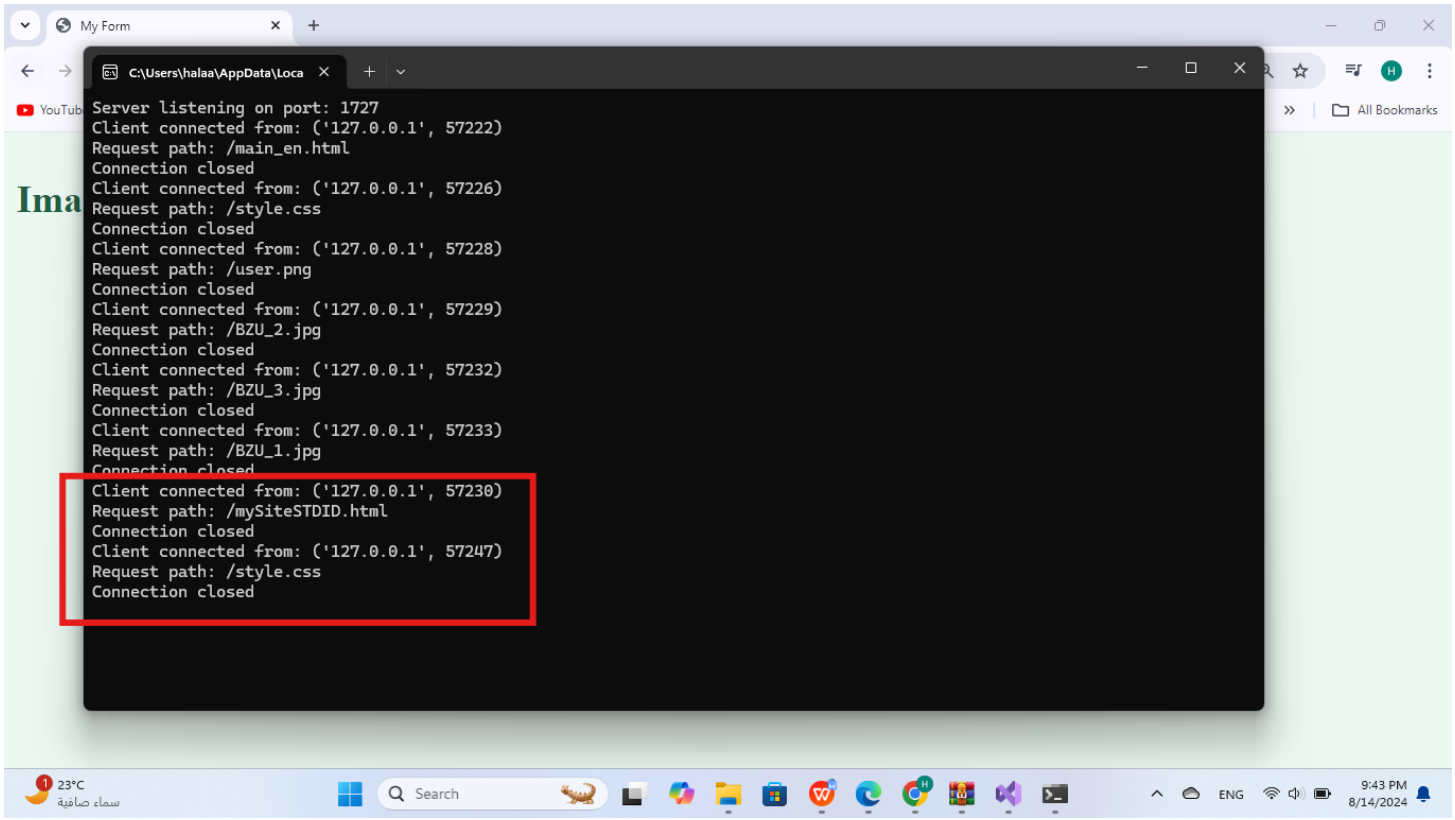


Figure 89:Web Server Result2

When I search for exist image

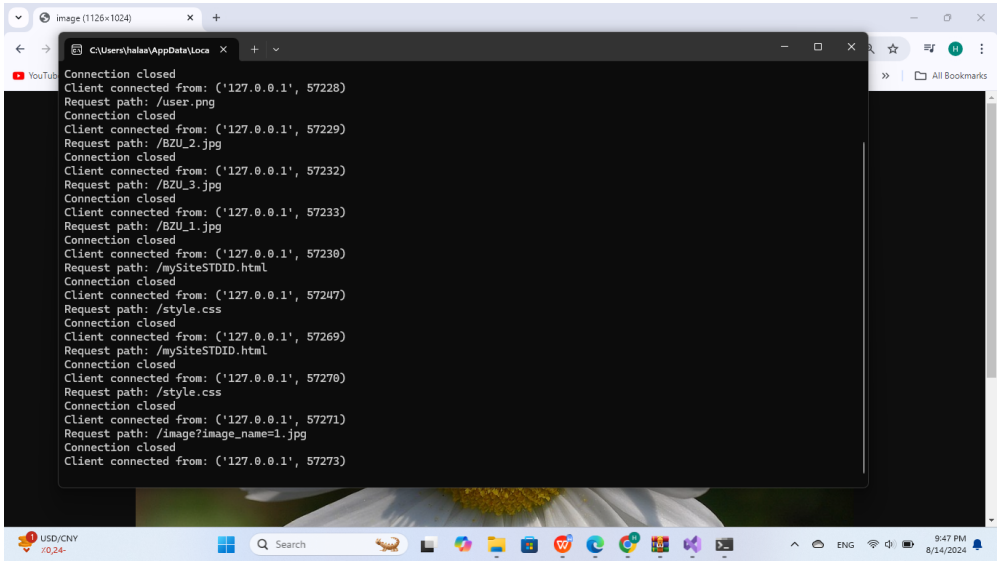


Figure 90:Web Server Result 3

When I search for not exist image

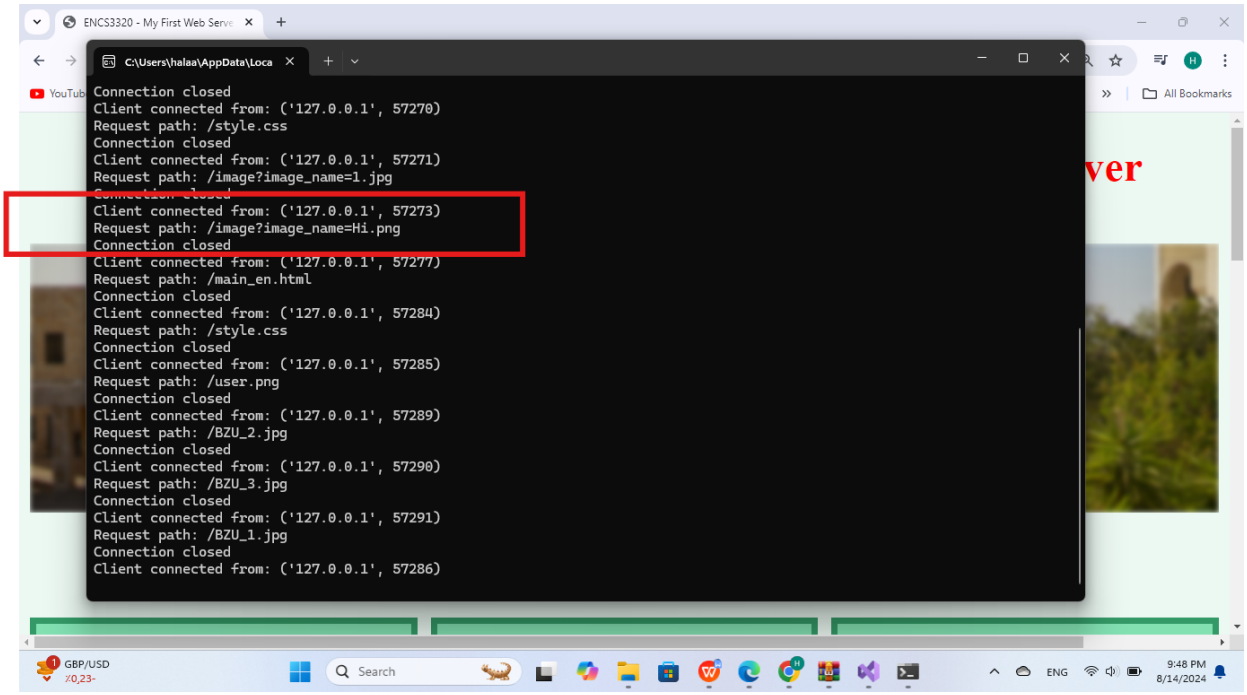


Figure 91:Web Server Result 4

This log shows that a connection from the client at 127.0.0.1 is made to the server, meaning that these are local requests. It is supposed to fetch back and serve an image file with the name `1.jpg' upon receiving a request for `/image? image_name=1.jpg' from the initial connection. Following this, it responds to the request and closes the connection. The second connection shows a similar request for `/image? image_name=Hi.png', asking the server to return the image file `Hi.png. Once this request is processed, the connection is also closed.

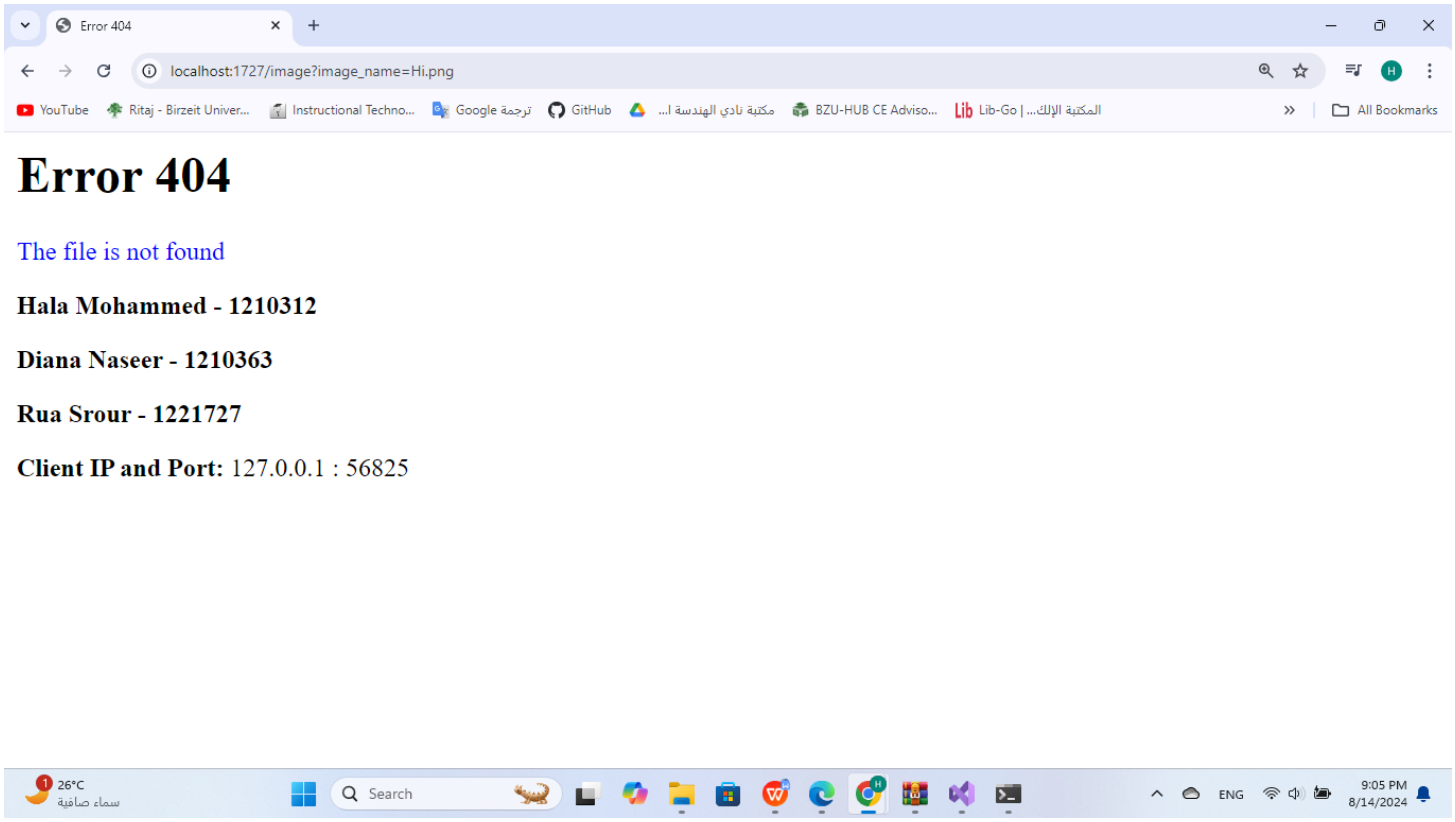


Figure 92:Error Page

The error page displaying "Client IP and Port: 127.0.0.1 : 57317" gives details about the source of the request. It is evident that it originated from the local machine by noticing the IP address `127.0.0.1`. That is, the request came from the same computer on which the server is running. The port number `57317` gets dynamically allocated to the clients' machine for this particular connection.

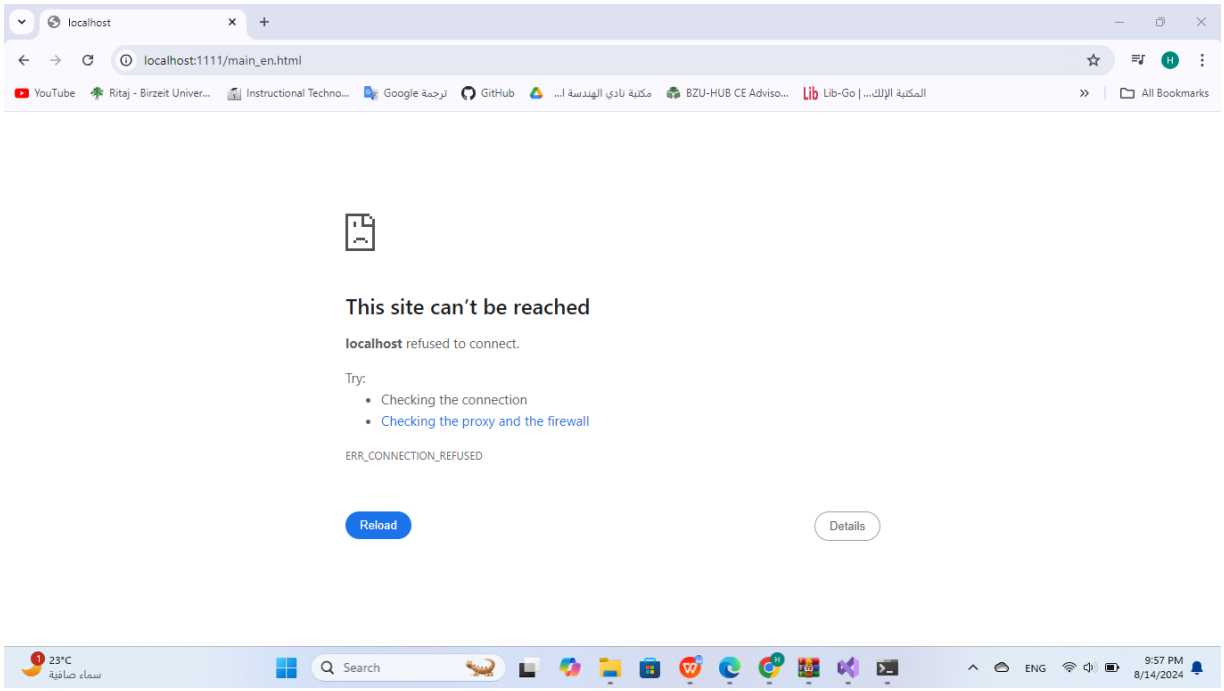


Figure 93:Try a new Port

Now, in the case of setting the server port, one should remember that the declared port number within the server code is very critical for proper communication. For example, in our case server code was set to listen in a port 1727, but here we are trying to access using port 1111 similar to http://localhost:1111/main_en.html. The connection failed. This will not work because a server does not have an instruction to listen on port 1111; it, instead, is awaiting requests on port 1727.

Test web server on another device

First we need the IPv4 for the first device,we can get it by searching for ipconfig on the terminal.

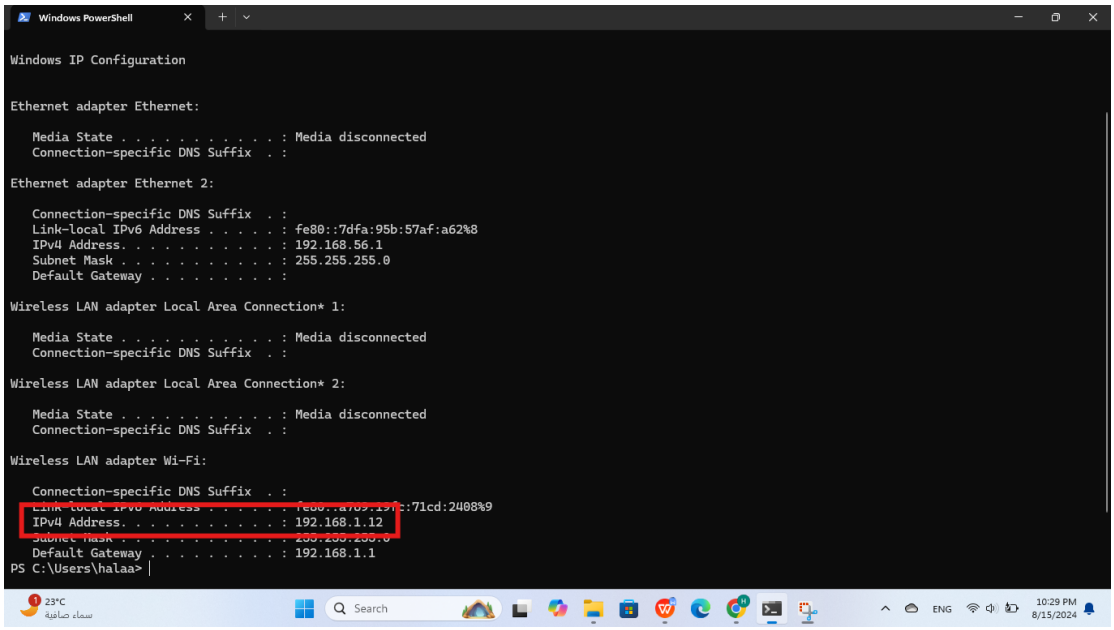


Figure 94:Test web server on another device Part1

Then we run the server code

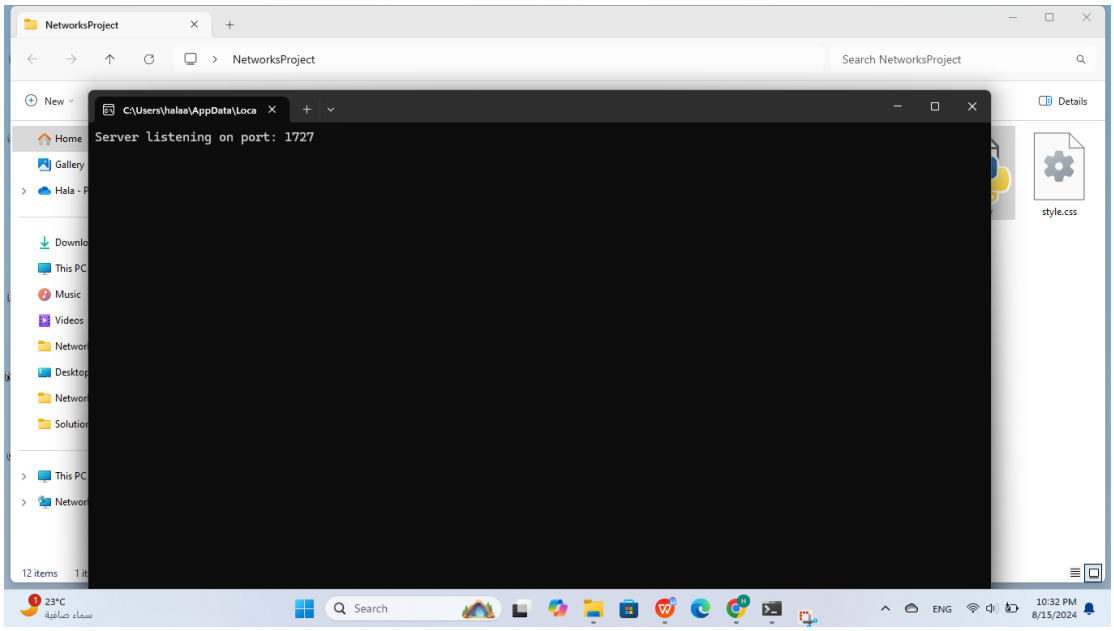


Figure 95:Test web server on another device Part2

Then we search for the page on the other device using the port number and the IPv4 number, with making sure that the two devices are connected to the same Internet network.

http://192.168.1.12:1727/main_en.html

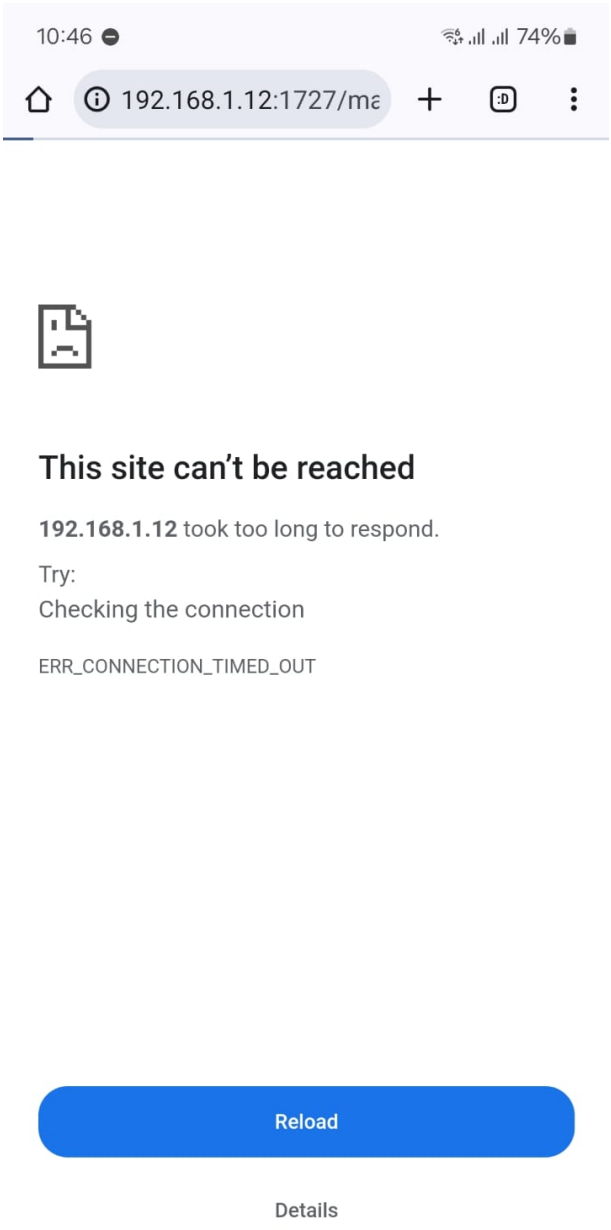


Figure 96:Test web server on another device Part3

The page did not load and was tested on several other devices with the same result .

I searched for a solution to this problem and make sure that the firewall on my computer is not blocking incoming connections on port 1727

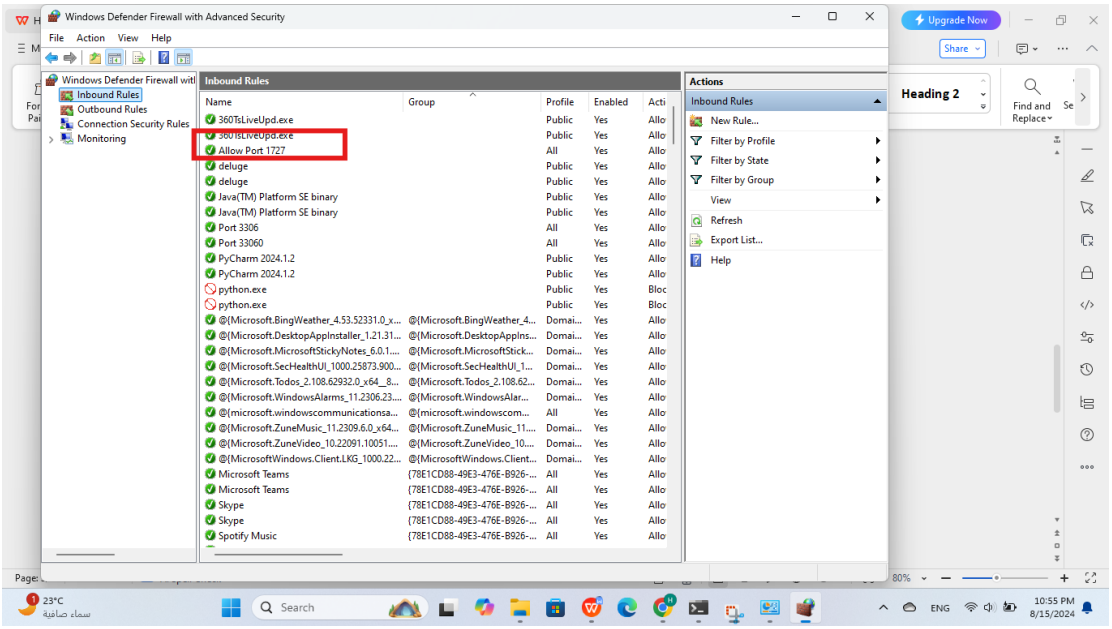


Figure 97::Test web server on another device Part4

I tried more than once and it didn't work):

References:

[1]: lenovo.

https://www.lenovo.com/in/en/glossary/command/?orgRef=https%253A%252F%252Fwww.google.com%252F

[Accessed on 4 August 2024 at 10:01 PM].

==========

[2]: tech target. [online image]

[Accessed on 8 August 2024 at 5:37PM].

https://www.techtarget.com/whatis/definition/Wireshark#:~:text=Wireshark%20is%20a%20widely%20used,ensure%20smooth%20operations%20and%20security.

==========

[3]: Quora [online image]

[Accessed on 8 August 2024 at 5:46PM].

https://www.quora.com/Why-is-Wireshark-useful

==========

[4]: BGP view.

[Accessed on 8 August 2024 at 7:10 PM].

https://bgpview.io/

==========

[5]: BGP tools. [online image]

[Accessed on 8 August 2024 at 8:05 PM].

https://bgp.tools/

==========

[6] cbtnuggets. [online image]

[Accessed on 8 August at 8:45 PM].

https://www.cbtnuggets.com/blog/technology/networking/traceroute-command-in-linux

==========

[7]: watch guard. [online image]

[Accessed on 8 August at 10:24 PM].

https://www.watchguard.com/help/docs/fireware/12/en-us/Content/en-US/policies/aliases_about_c.html#:~:text=An%20alias%20is%20a%20shortcut,can%20also%20create%20new%20aliases.

==========

[8]: IP location. [online image]

[Accessed on 13 August at 1:16 AM].

https://www.iplocation.net/ip-lookup#google_vignette

==========

[9]: Socket programming: https://www.youtube.com/watch?v=3QiPPX-KeSc&t=1133s

[Accessed on 10 August at 1:12 PM].

==========

[10]: Previous Video for socket preprograming (for Dr. Ibrahim Nimer):

https://drive.google.com/drive/u/0/folders/1JvfyNheYXJHFmg1mpP_3-ePPmLIx2478

[Accessed on 11 August at 9:15 PM].

==========

[11]: another video for socket programing that it helps me in my code:

https://www.youtube.com/watch?v=bwTAVGg_kVs

[Accessed on 8 August at 8:15 AM].

==========

[12]: Socket programing Definition:

https://www.geeksforgeeks.org/socket-programming-cc/

[Accessed on 13 August at 2:15 AM].

==========

[13]: TCP definition:

https://www.fortinet.com/resources/cyberglossary/tcp-ip

[Accessed on 13 August at 3:21 PM].

==========

[14]: UDP definition:

https://cyberhoot.com/cybrary/transmission-control-protocol-tcp/

[Accessed on 13 August at 3:21 PM].

==========

[15] :Web Server

https://en.wikipedia.org/wiki/Web_server

[Accessed on 14 August at7:18 PM].

==========

[16]: Socket Programming

https://www.datacamp.com/tutorial/a-complete-guide-to-socket-programming-in-python

[Accessed on 14 August at7:37PM].

==========

[17]: HTTP Protocol and Content Types

https://www.geeksforgeeks.org/http-headers-content-type/

[Accessed on 14 August at7:47PM].

==========

[18] : Error Handling

https://www.dremio.com/wiki/error-handling/

[Accessed on 14 August at7:52PM].

==========

[19] : Redirection with Status Codes

https://www.baeldung.com/cs/redirection-status-codes

[Accessed on 14 August at8:02PM].

==========

[20] : **Localization Support**:

https://poeditor.com/blog/localization-services/

[Accessed on 14 August at8:14PM].