



**Department of Computer Science  
ENCS3320, COMPUTER NETWORKS  
Project#1: Socket Programming**

**G9 :**

**Amro Deek 1221642 , Section 2**

**Karmel Bader 1221473 , Section 2**

**Shaden Hamda 1220169 , Section 2**

**Date : 15/8/2024**

## Contents

Theory and Procedure Part.....	4
1. Introduction to Socket Programming.....	4
2. TCP and UDP Sockets .....	4
3. Commands in Networking .....	4
4. Basics of HTML and Web Server.....	4
<b>Task1:.....</b>	<b>5</b>
Q1: .....	5
Q2: .....	6
Q4: .....	11
<b>Task 2:.....</b>	<b>12</b>
Q1: .....	12
Q2: .....	14
<b>Task3:.....</b>	<b>16</b>
<b>Team Work.....</b>	<b>38</b>
Everyone's job:.....	38
<b>Issues and Limitations Part.....</b>	<b>39</b>
Solution.....	39
<b>References :</b> .....	<b>39</b>

Figure 1:Task1_Q2_partA.....	6
Figure 2:Task1_Q2_partB.....	6
Figure 3:Task1_Q2_partC.....	7
Figure 4:Task1_Q2_partD.....	7
Figure 5:Task1_Q2_partE.....	8
Figure 6:Task1_Q2_partF .....	8
Figure 7:Task1_Q3_AC .....	9
Figure 8:Task1_Q3_prefies.....	9
Figure 9:Task1_Q3_IP .....	10
Figure 10:Task1_Q3_peers .....	10
Figure 11:Task1_Q4.....	11
Figure 12:Task2_Q1_server code.....	12
Figure 13:Task2_Q1_client code .....	12
Figure 14:Task2_Q1_server side (run time).....	13
Figure 15: Task2_Q1_client side (run time).....	13
Figure 16:Task2_Q2_code with comments1 .....	14
Figure 17: Task2_Q2_code with comments2 .....	14
Figure 18: Task2_Q2_output.....	15
Figure 19: Task3_main_en.html run output1 .....	16
Figure 20: Task3_main_en.html run output2 .....	16
Figure 21: Task3_main_ar.html run output1 .....	17
Figure 22: Task3_main_ar.html run output2 .....	17
Figure 23: case1 request.....	18
Figure 24: case1 request on command line.....	18
Figure 25: case 2 request.....	19
Figure 26: case 2 request on command line.....	19
Figure 27: case 3 request.....	20
Figure 28: case 3 request on command line.....	20
Figure 29: case 4 request.....	21
Figure 30: case 4 request on command line.....	21
Figure 31: case 5 request.....	22
Figure 32: case 5 request on command line.....	22
Figure 33: case 6 request.....	23
Figure 34: case 6 request on command line.....	23
Figure 35: case 7 request.....	24
Figure 36: case 7 request on command line.....	24
Figure 37: case 8 request.....	25
Figure 38: case 8 request on command line.....	25
Figure 39: case 9 request.....	26
Figure 40: case 9 request on command line.....	26
Figure 41: mySiteSTDID.html running output.....	27
Figure 42: mySiteStDID.html running output with correct path .....	27
Figure 43: output of correct path .....	28
Figure 44: mySiteStDID.html running output with wrong path .....	28
Figure 45: output of wrong path.....	29
Figure 46: http request on command line .....	29
Figure 47: output of /so request.....	30
Figure 48: output of /itc request .....	30
Figure 49: output of testing the project on other computer.....	31
Figure 50: Task3.py code_1 .....	32
Figure 51: Task3.py code_2 .....	32
Figure 52: main_en.html code_1 .....	33
Figure 53: main_en.html code_2 .....	33
Figure 54: style.css code_1 .....	34
Figure 55: style.css code_2 .....	34
Figure 56: mySiteSTDID.html code_1.....	35
Figure 57: mySiteSTDID.html code_2.....	35
Figure 58: httpSerever.py code_1.....	36
Figure 59: httpServer code_2 .....	37

# Theory and Procedure Part

## 1. Introduction to Socket Programming

Socket programming is a method of enabling communication between two or more devices over a network. It involves creating endpoints (sockets) that allow data to be sent and received over the network. This project focuses on understanding and implementing basic concepts of socket programming using TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

## 2. TCP and UDP Sockets

- **TCP (Transmission Control Protocol):** TCP is a connection-oriented protocol that ensures reliable communication between devices. It guarantees that data sent from one socket will arrive at the other socket in the correct order and without loss. TCP is used for applications where data integrity is crucial (e.g., web browsing, email).
  - **TCP Socket Programming:** Involves creating a client-server architecture where the server waits for connection requests from clients. Once the connection is established, data is exchanged reliably between the two sockets.
- **UDP (User Datagram Protocol):** UDP is a connectionless protocol that allows data to be sent without establishing a connection between sender and receiver. It is faster than TCP but does not guarantee the delivery or order of data packets. It is used for applications where speed is more critical than reliability (e.g., streaming video, online gaming).

## 3. Commands in Networking

Socket programming requires a deep understanding of basic networking commands and protocols. These commands help in troubleshooting and testing network connectivity, resolving domain names, and diagnosing network-related issues. Some essential networking commands that might be used in this project are:

## 4. Basics of HTML and Web Server

The project also requires familiarity with basic HTML and web server concepts. This involves understanding how to serve HTML pages over the network using a web server, which listens for HTTP requests and responds with appropriate content, such as web pages, images, or files.

- **HTML:** The standard markup language for creating web pages. It structures the content that a web server sends to a client's web browser.**Web Server:** A server that processes HTTP requests and serves HTML content to clients (browsers). In this project, you would create a basic web server using sockets, which listens for incoming HTTP requests and responds with HTML pages

## **Task1:**

### **Q1:**

#### **Ping**

is like sending a quick message to another device on a network to check if it's available and how long it takes to respond. Imagine you're asking, and waiting for a back. If the device answers, it means it's online and reachable. If not, it might be offline or unreachable. It's a simple way to see if a website or server is up and running or if your computer has a connection to it.

#### **Tracert**

is like tracking the route your data takes across the internet. Every time you visit a website, your data hops between several devices (called routers) before it gets to its destination. Tracert shows you each "hop" along the way and how long it takes to get from one device to the next. This is helpful if something's slowing down your connection—you can pinpoint exactly where the slowdown is happening.

#### **Nslookup**

helps translate a website name (like example.com) into its numerical IP address, which is what computers actually use to find websites. It's like looking up a phone number for a contact. You can also use nslookup to dig into a website's DNS records (like the list of servers that handle its email). It's a handy tool for troubleshooting if a website isn't loading properly or you need to check its domain settings.

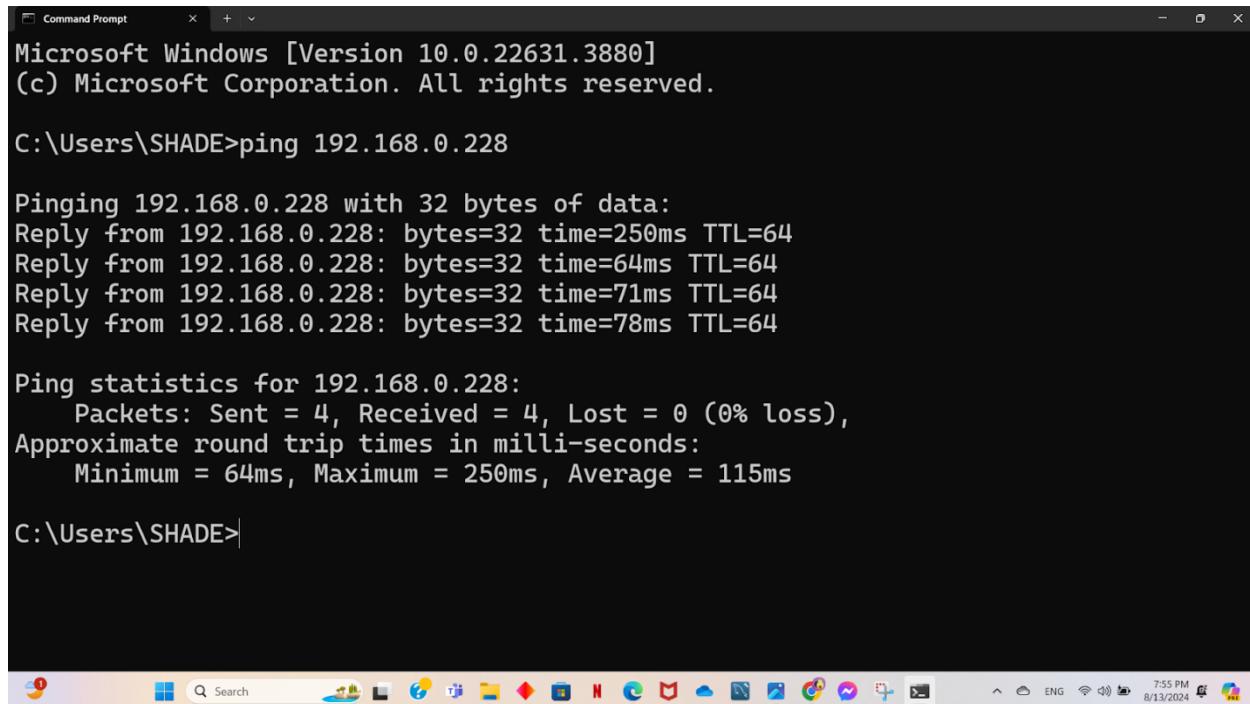
#### **Telnet**

lets remotely connect to another computer over a network, giving you a way to control it from afar, usually by typing commands. However, telnet isn't very secure because it doesn't encrypt the data it sends, which means someone could intercept sensitive information like passwords. These days, most people use a more secure option like SSH, but telnet can still be useful in specific situations for configuring devices.

## Q2:

\*Ping a device in the same network from laptop to smartphone

As the figure shows that the IP address 192.168.0.228, sent 4 packets. Each packet was received at different times, with an average delay of 115 ms. ‘There was no packet loss’.



```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>ping 192.168.0.228

Pinging 192.168.0.228 with 32 bytes of data:
Reply from 192.168.0.228: bytes=32 time=250ms TTL=64
Reply from 192.168.0.228: bytes=32 time=64ms TTL=64
Reply from 192.168.0.228: bytes=32 time=71ms TTL=64
Reply from 192.168.0.228: bytes=32 time=78ms TTL=64

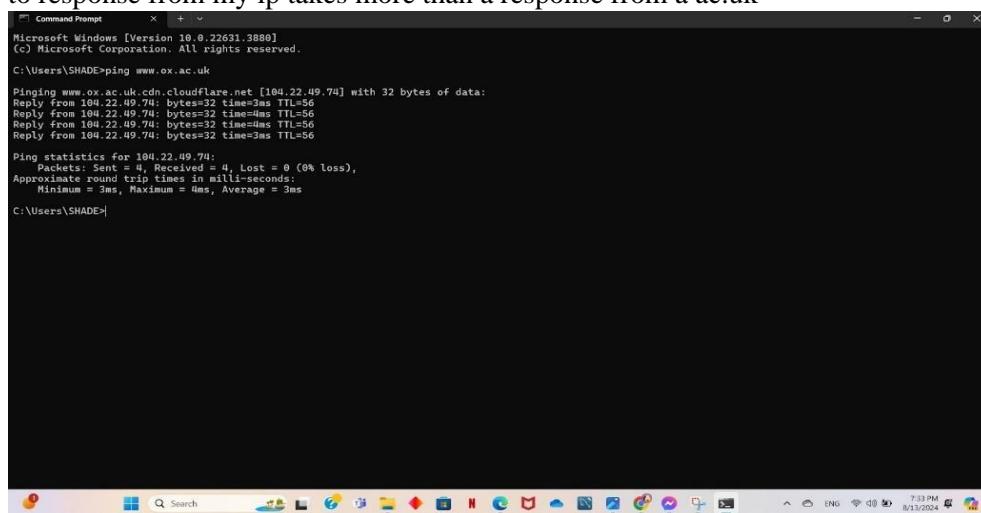
Ping statistics for 192.168.0.228:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 64ms, Maximum = 250ms, Average = 115ms

C:\Users\SHADE>
```

Figure 1:Task1\_Q2\_partA

\*Ping <http://www.ox.ac.uk>.

The figure shows that the IP response address is 104.22.49.74, and we sent 4 packets. Each packet was received at different times, with an average delay of 3ms. ‘There was no packet loss’.notice that the time to response from my ip takes more than a response from a ac.uk



```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>ping www.ox.ac.uk

Pinging www.ox.ac.uk.cdn.cloudflare.net [104.22.49.74] with 32 bytes of data:
Reply from 104.22.49.74: bytes=32 time=3ms TTL=56
Reply from 104.22.49.74: bytes=32 time=4ms TTL=56
Reply from 104.22.49.74: bytes=32 time=3ms TTL=56
Reply from 104.22.49.74: bytes=32 time=3ms TTL=56

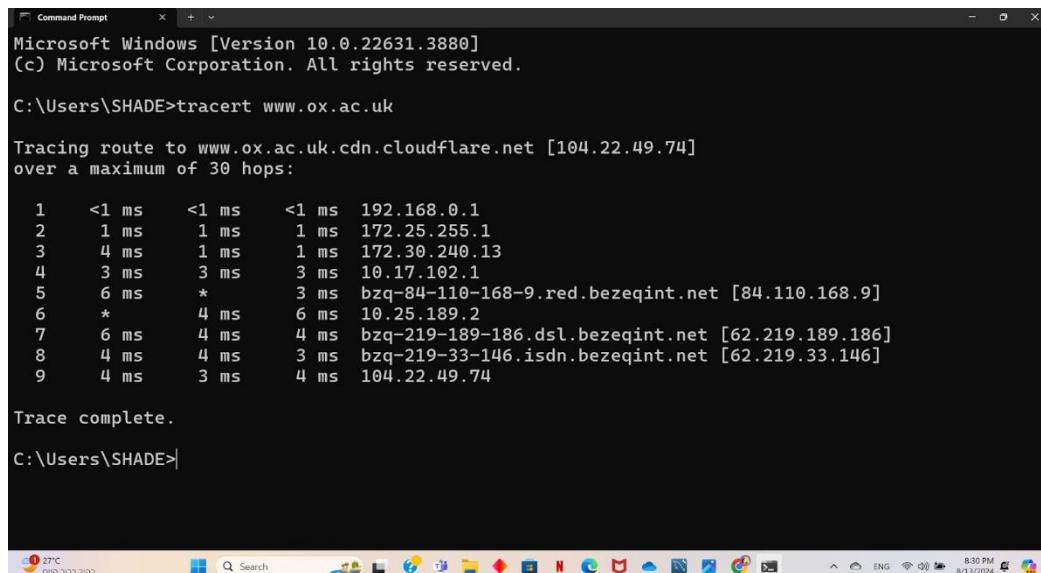
Ping statistics for 104.22.49.74:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 3ms, Maximum = 4ms, Average = 3ms

C:\Users\SHADE>
```

Figure 2:Task1\_Q2\_partB

\*tracert [www.ox.ac.uk](http://www.ox.ac.uk)

The tracert sent 3 messages per every hop(router) and the number of hops is 9 so the total is 27 , the (\*) means the packet didn't receive a response within the timeout period , and as we see there is no delays or dropped packets along the route, which suggests a good network path.



```
Command Prompt Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>tracert www.ox.ac.uk

Tracing route to www.ox.ac.uk.cdn.cloudflare.net [104.22.49.74]
over a maximum of 30 hops:

 1  <1 ms    <1 ms    <1 ms  192.168.0.1
 2  1 ms     1 ms     1 ms  172.25.255.1
 3  4 ms     1 ms     1 ms  172.30.240.13
 4  3 ms     3 ms     3 ms  10.17.102.1
 5  6 ms     *        3 ms  bzq-84-110-168-9.red.bezeqint.net [84.110.168.9]
 6  *        4 ms     6 ms  10.25.189.2
 7  6 ms     4 ms     4 ms  bzq-219-189-186.dsl.bezeqint.net [62.219.189.186]
 8  4 ms     4 ms     3 ms  bzq-219-33-146.isdn.bezeqint.net [62.219.33.146]
 9  4 ms     3 ms     4 ms  104.22.49.74

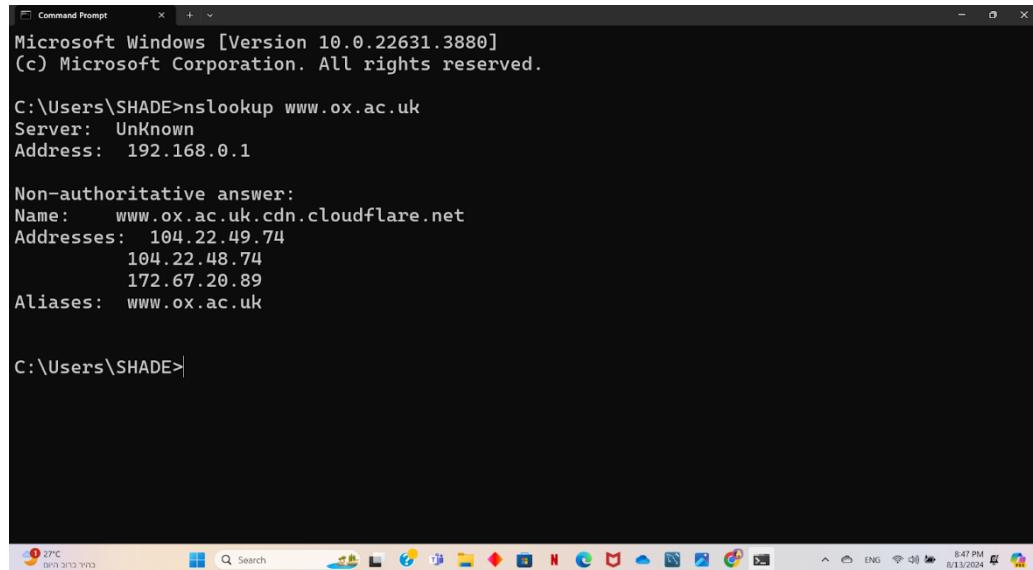
Trace complete.

C:\Users\SHADE>
```

Figure 3:Task1\_Q2\_partC

\*nslookup [www.ox.ac.uk](http://www.ox.ac.uk)

As the figure shows , it prints my router address , and prints the name ,aliases name , and addresses of the server which is the host that we sent a messages.



```
Command Prompt Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>nslookup www.ox.ac.uk
Server:  Unknown
Address:  192.168.0.1

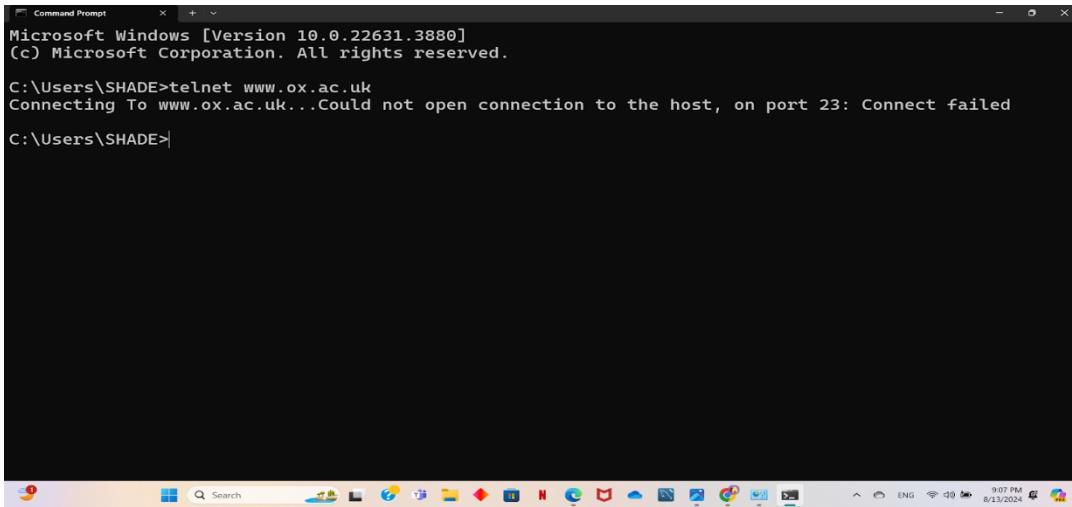
Non-authoritative answer:
Name:   www.ox.ac.uk.cdn.cloudflare.net
Addresses:  104.22.49.74
          104.22.48.74
          172.67.20.89
Aliases:  www.ox.ac.uk

C:\Users\SHADE>
```

Figure 4:Task1\_Q2\_partD

telnet [www.ox.ac.uk](http://www.ox.ac.uk)

When I run the command it give me that does not have this port open because Telnet is an outdated and insecure protocol.



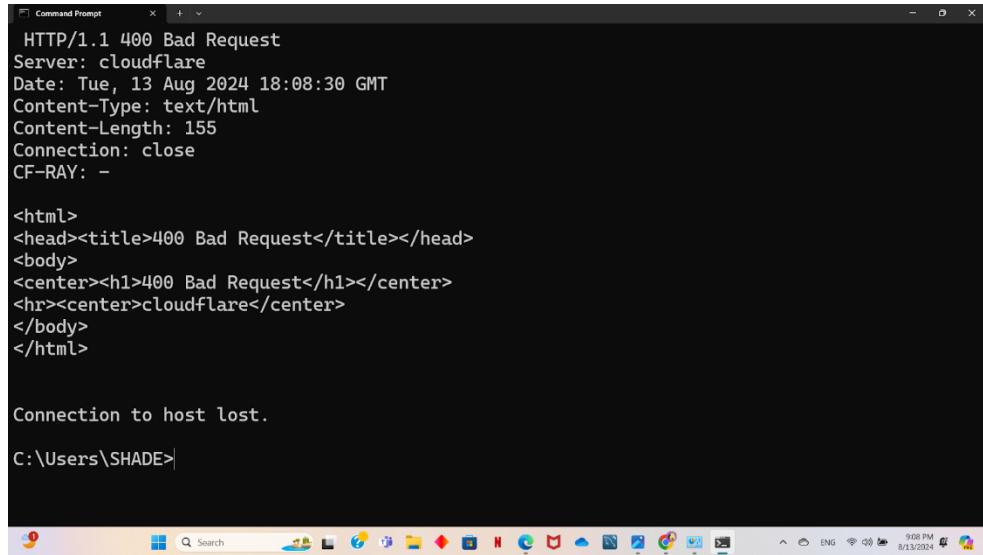
```
Command Prompt
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>telnet www.ox.ac.uk
Connecting To www.ox.ac.uk...Could not open connection to the host, on port 23: Connect failed
C:\Users\SHADE>
```

Figure 5:Task1\_Q2\_partE

Then I change the port to 80

That allow me to communicate with the server (give me bad request because i didn't give the server any specific request)



```
Command Prompt
HTTP/1.1 400 Bad Request
Server: cloudflare
Date: Tue, 13 Aug 2024 18:08:30 GMT
Content-Type: text/html
Content-Length: 155
Connection: close
CF-RAY: -

<html>
<head><title>400 Bad Request</title></head>
<body>
<center><h1>400 Bad Request</h1></center>
<hr><center>cloudflare</center>
</body>
</html>

Connection to host lost.

C:\Users\SHADE>
```

Figure 6:Task1\_Q2\_partF

## Q3:

(we tried to use [www.bgpview.io](http://www.bgpview.io) but it didn't work so we use bgp.he.net)

Take the IP address by using nslookup command then by using [Hurricane Electric BGP Toolkit \(he.net\)](https://bgp.he.net) and find the AS which is AS13335

104.22.49.74

Announced By		
Origin AS	Announcement	Description
AS13335	104.16.0.0/12	
AS13335	104.22.48.0/20	

Address has 2 hosts associated with it.

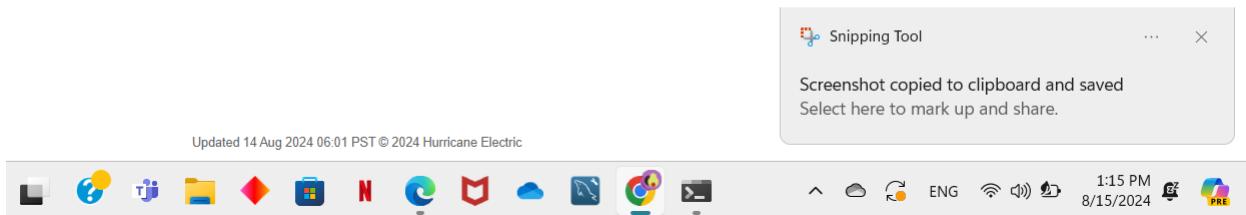


Figure 7:Task1\_Q3\_AC

### The prefixes:

INTERNET SERVICES

AS13335 Cloudflare, Inc.

Links: AS Info | Graph v4 | Graph v6 | Prefixes v4 | Prefixes v6 | Peers v4 | Peers v6 | Whois | IRR | IX | Traceroute

Company Website: <https://www.cloudflare.com>

Country of Origin: United States

Internet Exchanges: 323

Prefixes Originated (all): 3,906  
Prefixes Originated (v4): 2,014  
Prefixes Originated (v6): 1,892

Prefixes Announced (all): 7,021  
Prefixes Announced (v4): 5,129  
Prefixes Announced (v6): 1,892

RPKI Originated Valid (all): 3,705  
RPKI Originated Valid (v4): 1,821  
RPKI Originated Valid (v6): 1,884

RPKI Originated Invalid (all): 8  
RPKI Originated Invalid (v4): 3  
RPKI Originated Invalid (v6): 5

BGP Peers Observed (all): 1,514  
BGP Peers Observed (v4): 1,459  
BGP Peers Observed (v6): 711

Figure 8:Task1\_Q3\_prefies

## The IPs:

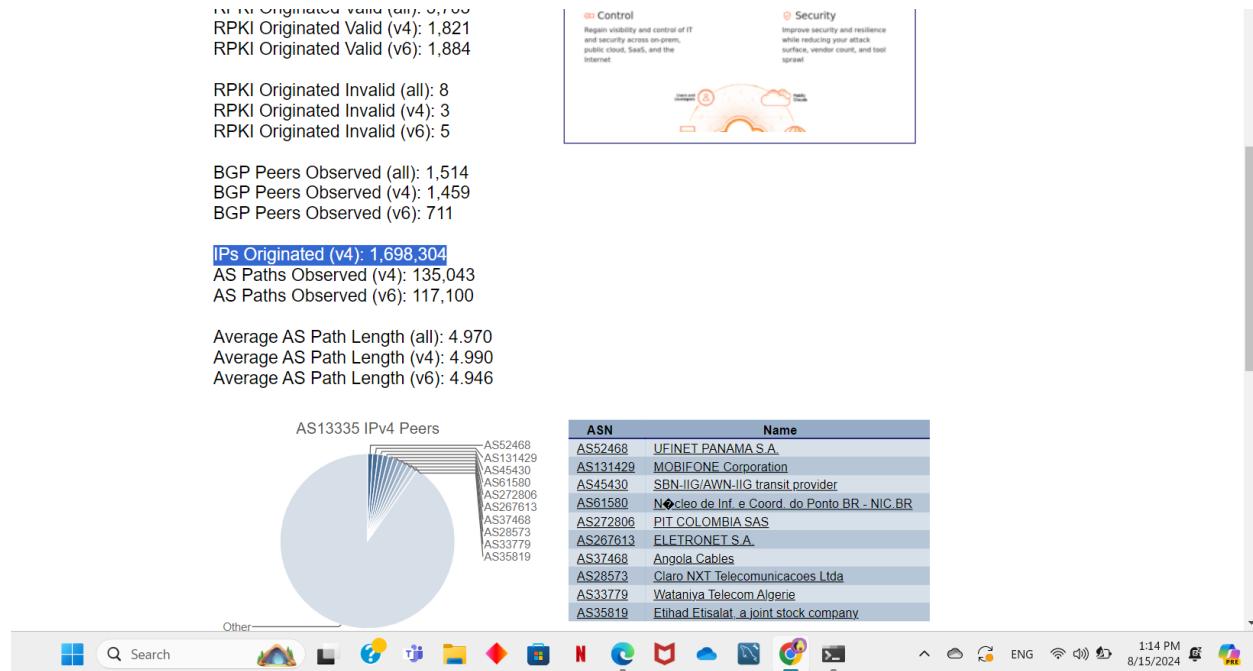


Figure 9:Task1\_Q3\_IP

## The peers:

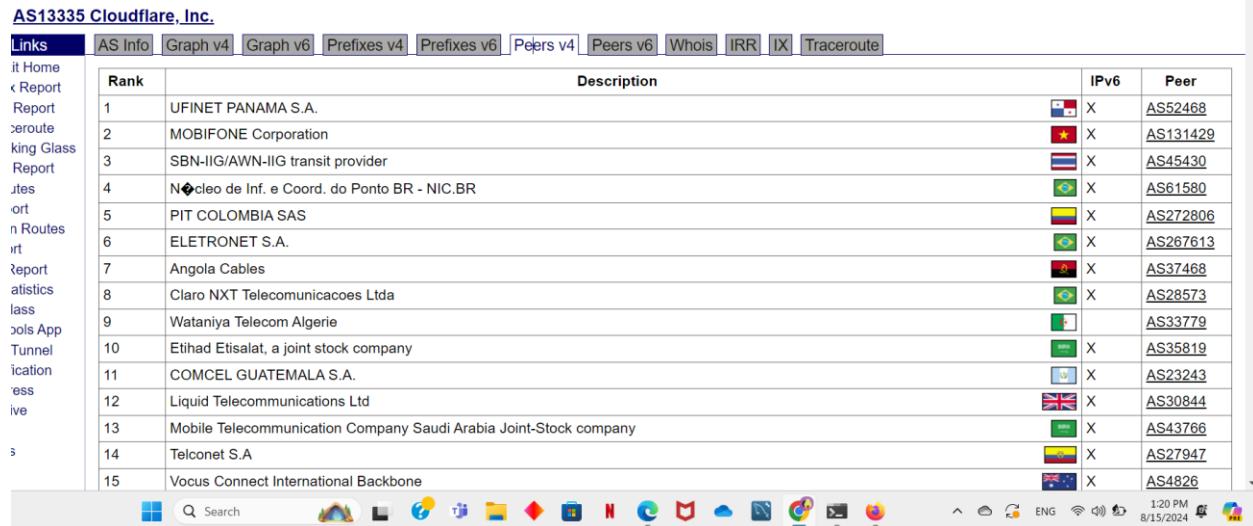


Figure 10:Task1\_Q3\_peers

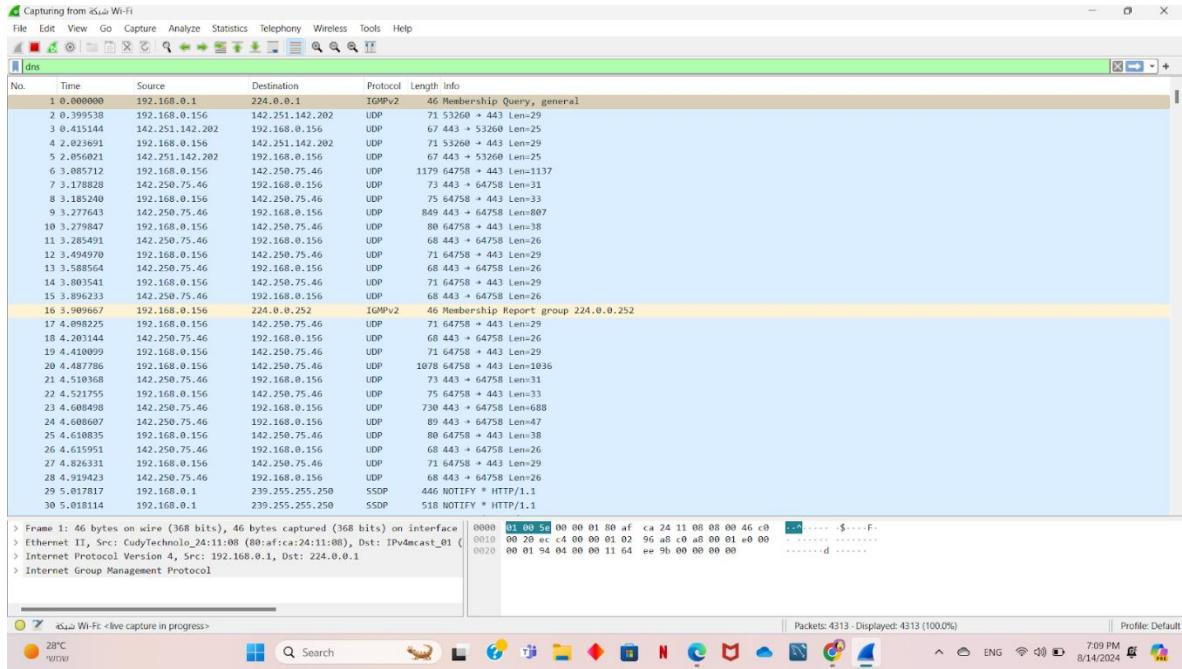
## The Tier 1 ISPs name:

Lumen Technologies (formerly known as Level 3)

## Q4:

\*Use wireshark (free tool and available online) to capture some DNS messages.

It shows a lot of packets that I received with details like the port and IP address to source and destination



*Figure 11:Task1\_Q4*

## Task 2:

## Q1:

Using socket programming, with TCP client server python applications , we open a connection between client and server so the client can send data to the server and replace all the vowels (aeiou/AEIOU) with '#' and return the string to the client and print it. We used a port # 1642 for this communication.

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a project named "NETWORK PRJ1" containing files: httpServer.py, mySiteSTDID.html, ph1.jpg, TestClient.py, and TestServer.py. The file "TestServer.py" is currently selected.
- Code Editor (Center):** Displays the Python code for "TestServer.py". The code uses sockets to accept connections and modify sentences based on vowels.
- Bottom Status Bar:** Shows the command prompt "PS C:\Users\fd\Desktop\Network Prj1> [ ]", file path "Ln 8, Col 1", and other status information like "Spaces: 4", "UTF-8", "CRLF", "Python 3.11.9 64-bit (Microsoft Store)", and "Result".

*Figure 12: Task2 Q1 server code*

The screenshot shows the Microsoft Visual Studio Code interface. The title bar reads "Network Prj1". The left sidebar has icons for Explorer, Search, Find, Open, Save, and Run. The Explorer view shows a project structure under "NETWORK PRJ1" containing "httpServer.py", "mySiteSTDID.html", "ph1.jpg", "TestClient.py" (selected), and "TestServer.py". The main editor area displays the following Python code:

```
from socket import *
serverName = gethostname() # 'servername'
serverPort = 1642
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())

modifiedSentence = clientSocket.recv(2048)
print('From Server:', modifiedSentence.decode())

clientSocket.close()
```

The bottom navigation bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The terminal tab shows the command "PS C:\Users\fd\Desktop\Network Prj1> [ ]". The status bar at the bottom provides information about the file (Ln 14, Col 21), encoding (UTF-8), and language (Python 3.11.9 64-bit (Microsoft Store)). It also includes a "Activate Windows" link.

*Figure 13:Task2\_Q1\_client code*

```

File Edit Selection View Go ... < > Network Prj1
Command Prompt - python TestServer.py
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

C:\Users\fd\Desktop>cd "Network Prj1"
C:\Users\fd\Desktop\Network Prj1>python TestServer.py
The server is ready to receive

```

The screenshot shows a Microsoft Windows terminal window titled "Network Prj1". The command "python TestServer.py" is being run. The output indicates that the server is ready to receive connections.

Figure 14: Task2\_Q1\_server side (run time)

**Client Side (run time):** after opening connection server becomes ready to receiver input from client and replace all vowels into # sign .

```

File Edit Selection View Go ... < > Network Prj1
EXPLORER ... TestClient.py mySiteSTDID.html httpServer.py TestServer.py
NETWORK PRJ1
H Command Prompt
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

P C:\Users\fd>cd Desktop
T C:\Users\fd\Desktop>cd "Network Prj1"
T C:\Users\fd\Desktop\Network Prj1>

C:\Users\fd\Desktop\Network Prj1>python TestClient.py
Input lowercase sentence:hello wold !!! i am anno al deek , student at bzu uni
From Server: #hll# wlrd !!! # i #m #mr# #l d##k , st##nt #t bz# #n#
C:\Users\fd\Desktop\Network Prj1>

```

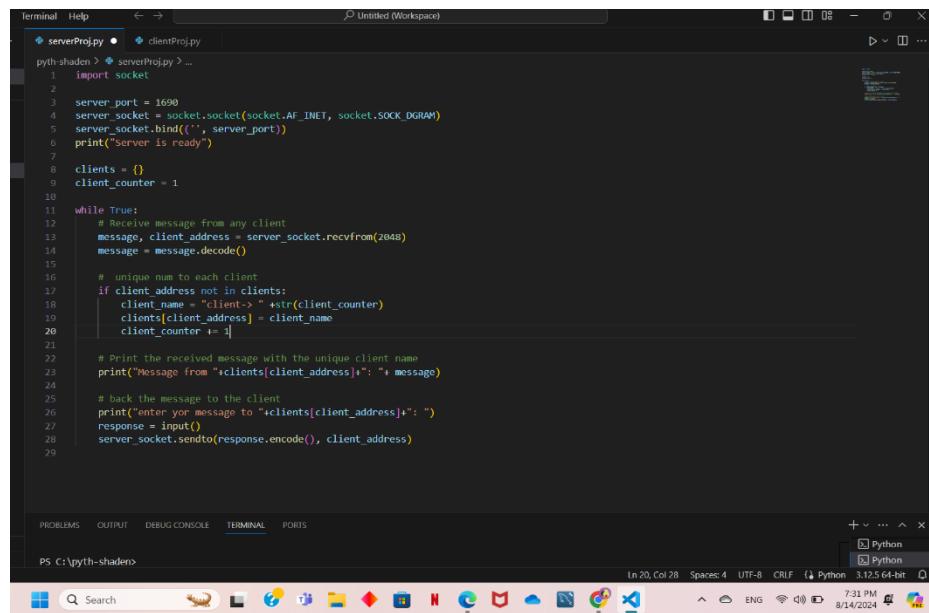
The screenshot shows a Microsoft Windows terminal window titled "Network Prj1". The command "python TestClient.py" is being run. The output shows a lowercase sentence being sent to the server, which then returns a modified version where vowels are replaced by # signs.

Figure 15: Task2\_Q1\_client side (run time)

## Q2:

\*note: my studID is 1220169 so my port should 0169 but the number cant start with 0 so I put it in the end to be 1690 (I told you in person)

This code allows the server to communicate with multiple clients over UDP, assigning each a unique identifier and print messages back. (each client has a unique name )



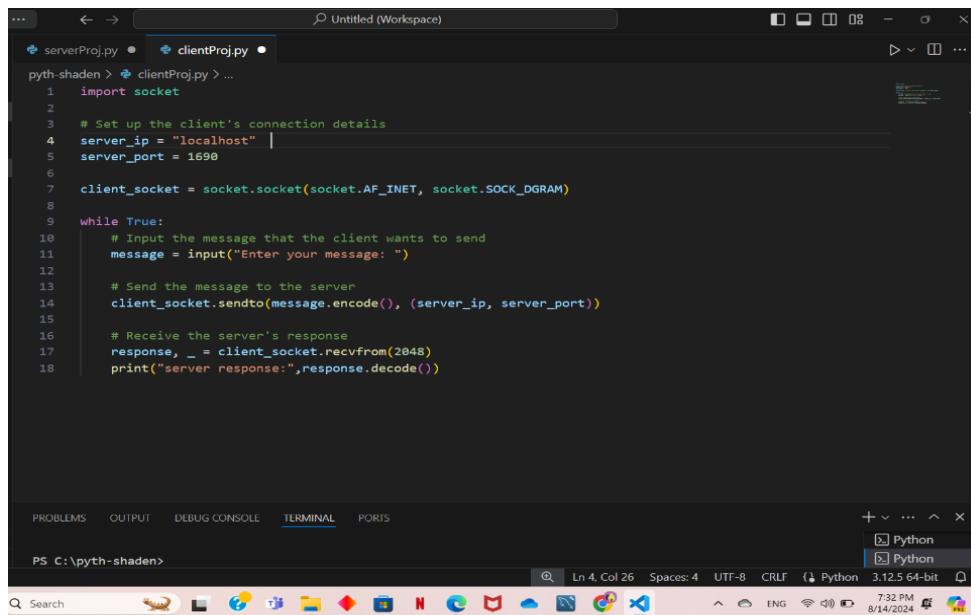
```
Terminal Help < -> Untitled (Workspace)
serverProj.py clientProj.py

pyth-shaden > serverProj.py > ...
1 import socket
2
3 server_port = 1690
4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5 server_socket.bind(('', server_port))
6 print("Server is ready")
7
8 clients = {}
9 client_counter = 1
10
11 while True:
12     # Receive message from any client
13     message, client_address = server_socket.recvfrom(2048)
14     message = message.decode()
15
16     # unique name to each client
17     if client_address not in clients:
18         client_name = "Client->" + str(client_counter)
19         clients[client_address] = client_name
20         client_counter += 1
21
22     # Print the received message with the unique client name
23     print("Message from " + clients[client_address] + ": " + message)
24
25     # back the message to the client
26     print("Enter your message to " + clients[client_address] + ": ")
27     response = input()
28     server_socket.sendto(response.encode(), client_address)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\pyth-shaden>
In 20, Col 28 Spaces: 4 UTT-8 CRLF Python 3.12.5 64-bit
7:31 PM 8/14/2024
```

Figure 16: Task2\_Q2\_code with comments1

This code enables the client to send messages to the server and receive responses



```
Terminal Help < -> Untitled (Workspace)
serverProj.py clientProj.py

pyth-shaden > clientProj.py > ...
1 import socket
2
3 # Set up the client's connection details
4 server_ip = "localhost" |
5 server_port = 1690
6
7 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8
9 while True:
10     # Input the message that the client wants to send
11     message = input("Enter your message: ")
12
13     # Send the message to the server
14     client_socket.sendto(message.encode(), (server_ip, server_port))
15
16     # Receive the server's response
17     response, _ = client_socket.recvfrom(2048)
18     print("server response:", response.decode())

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\pyth-shaden>
In 4, Col 26 Spaces: 4 UTT-8 CRLF Python 3.12.5 64-bit
7:32 PM 8/14/2024
```

Figure 17: Task2\_Q2\_code with comments2

There is the result after running both sides , the server(client) and two clients(as a example) , first run the server side to be ready to receive any messages , then run the client side and send a message to the server (client) and wait the answer

The screenshot shows three separate Windows Command Prompt windows running simultaneously. Each window has a title bar indicating it's a 'Command Prompt' window with a specific Python script name (e.g., 'python s', 'python c', 'python h'). The windows are arranged horizontally across the desktop.

**Left Window (Server):**

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>cd C:\
C:>cd pyth-shaden
C:\pyth-shaden>python serverProj.py
Server is ready
Message from client->1: hi
enter yor message to client->1
hii
Message from client->2: hi 2
enter yor message to client->2
hii 2
```

**Middle Window (Client 1):**

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>cd C:\
C:>cd pyth-shaden
C:\pyth-shaden>python clientProj.py
Enter your message: hi
server response: hii
Enter your message: |
```

**Right Window (Client 2):**

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SHADE>cd C:\
C:>cd pyth-shaden
C:\pyth-shaden>python clientProj.py
Enter your message: hi 2
server response: hii 2
Enter your message:
```

The desktop background features a scenic landscape with mountains and water. Icons for 'Desktop', 'Project 1', 'Oracle VM VirtualBox', and 'Epic Games Launcher' are visible in the top left corner. The taskbar at the bottom shows several pinned icons, including 'palestine.pn...', 'java assig...', 'Shaden - Chrome', and the 'Search' bar. The system tray on the right displays the date and time (8/14/2024, 7:54 PM), battery status, and network connectivity.

Figure 18: Task2\_Q2\_output

## Task3:

Here is the [main\\_en.html](#) file which the server should send in some cases:

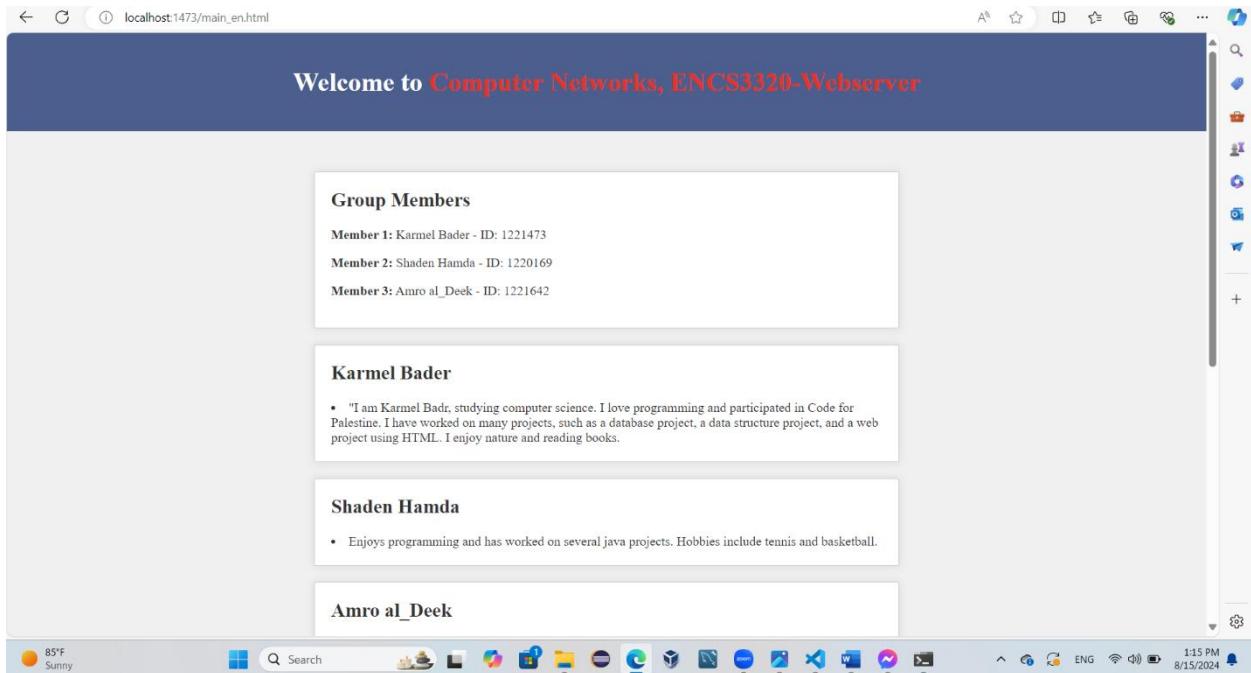


Figure 19: Task3\_main\_en.html run output1

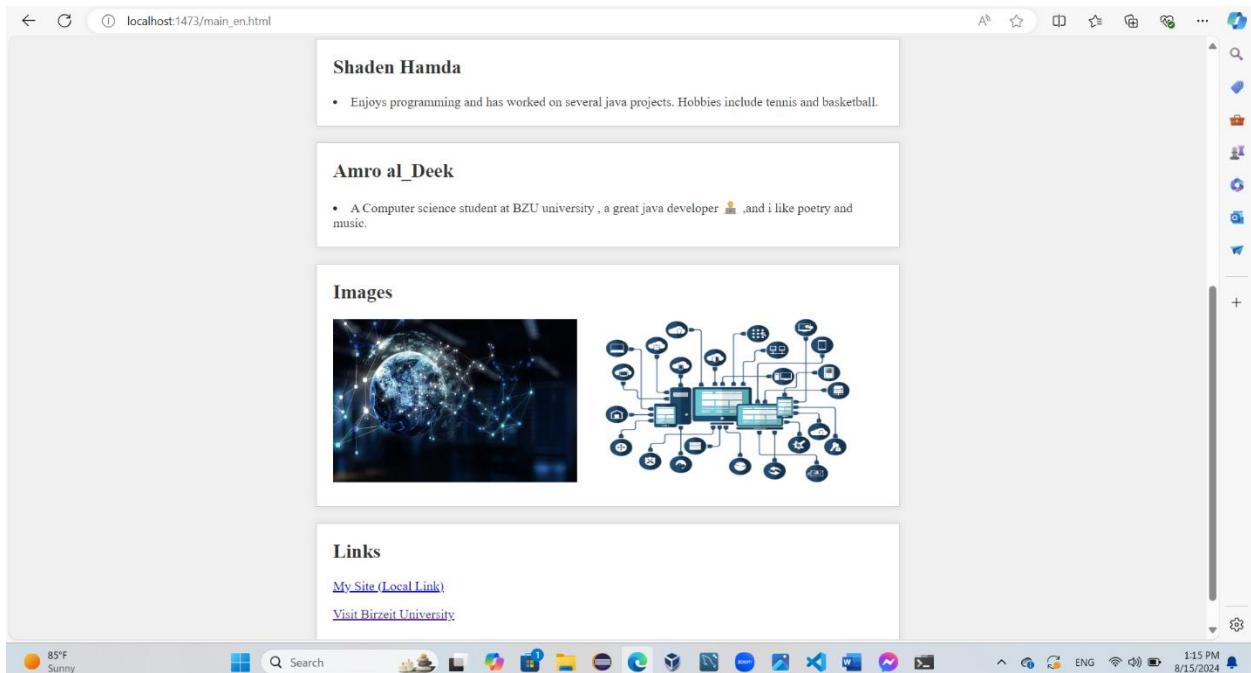


Figure 20: Task3\_main\_en.html run output2

## The main\_ar.html file:

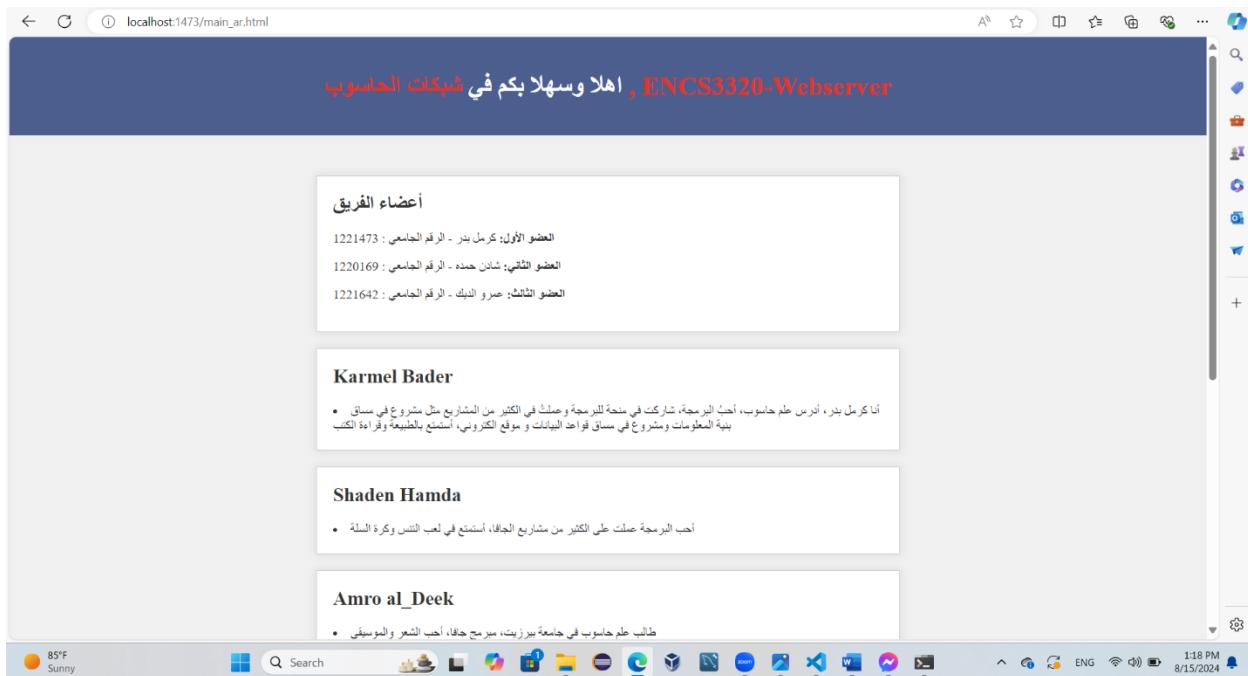


Figure 21: Task3\_main\_ar.html run output1

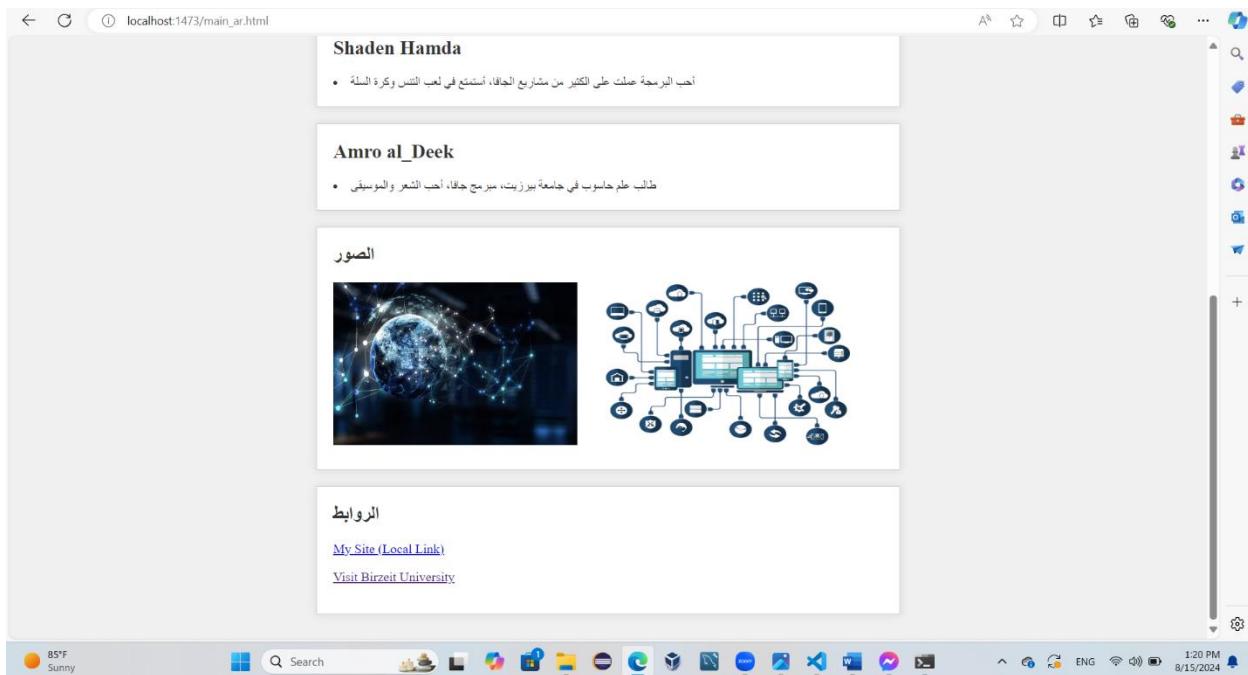


Figure 22: Task3\_main\_ar.html run output2

## Here is the cases of the request:

**Case1:** when the request [localhost:1473/](http://localhost:1473/) the server should send the main\_en.html file

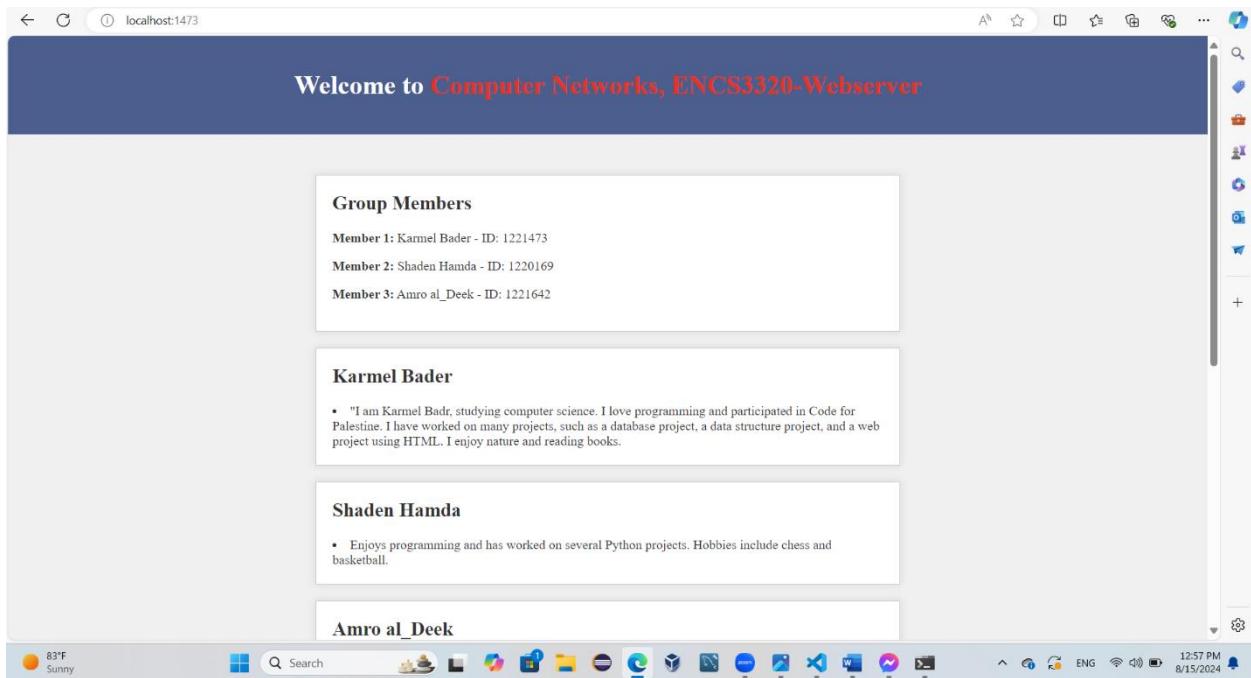


Figure 23: case1 request

The HTTP request printed on the command Line

A screenshot of a Windows Command Prompt window titled 'Command Prompt - python'. The prompt shows a log of network connections from IP 127.0.0.1 on port 61501. The log includes requests for files like 'style.css', 'network2.jpg', 'network1.png', and 'favicon.ico'. The taskbar at the bottom shows the date/time '12:57 PM 8/15/2024'.

Figure 24: case1 request on command line

**Case2:** when the request `localhost:1473/en` the server should send the `main_en.html` file

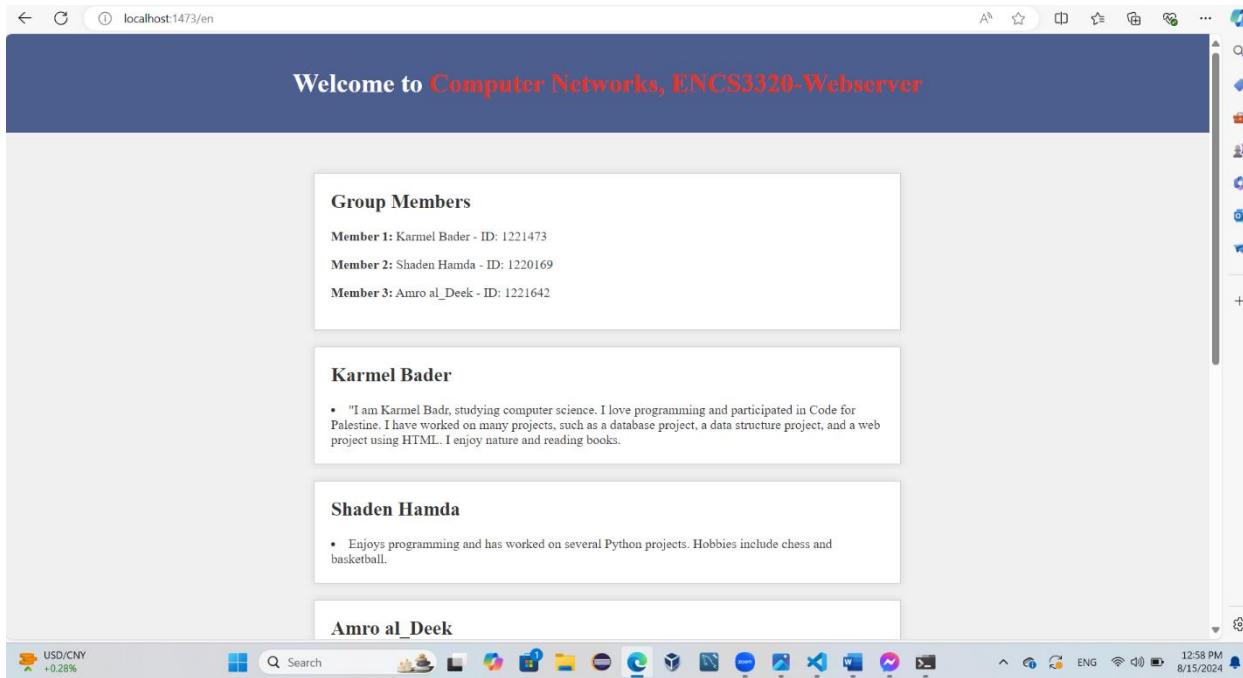


Figure 25: case 2 request

The HTTP request printed on the command Line

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\EASY LIFE>python OneDrive\Desktop/NetworkProj/Task3.py
The server is ready to receive
Got connection from IP: 127.0.0.1, Port: 61539
/en
Got connection from IP: 127.0.0.1, Port: 61542
/style.css
Got connection from IP: 127.0.0.1, Port: 61543
/network2.jpg
Got connection from IP: 127.0.0.1, Port: 61544
/network1.png
Got connection from IP: 127.0.0.1, Port: 61545
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61548
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61549
```

Figure 26: case 2 request on command line

**Case3:** when the request `localhost:1473/main_en.html` the server should send the `main_en.html` file

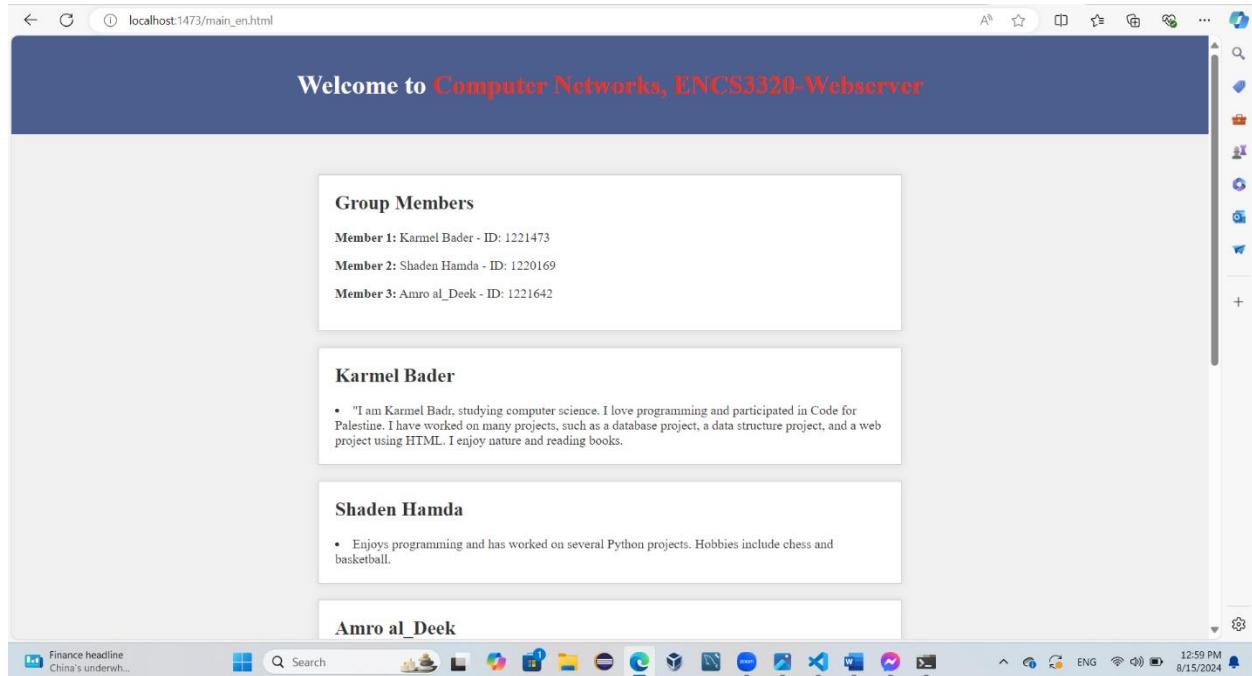


Figure 27: case 3 request

The HTTP request printed on the command Line

A screenshot of a Windows Command Prompt window titled "Command Prompt - python". The window shows a log of HTTP requests received by a server. The log includes the path requested and the IP and port of the client. The requests listed are: "The server is ready to receive", "Got connection from IP: 127.0.0.1, Port: 61576 /main\_en.html", "Got connection from IP: 127.0.0.1, Port: 61579 /style.css", "Got connection from IP: 127.0.0.1, Port: 61580 /network2.jpg", "Got connection from IP: 127.0.0.1, Port: 61581 /network1.png", "Got connection from IP: 127.0.0.1, Port: 61582 /favicon.ico", and "Got connection from IP: 127.0.0.1, Port: 61585 /favicon.ico". The command prompt window is set against a background of a desktop screen showing a taskbar with various pinned icons and the date/time as 8/15/2024, 12:59 PM.

Figure 28: case 3 request on command line

**Case4:** when the request `localhost:1473/index.html` the server should send the `main_en.html` file

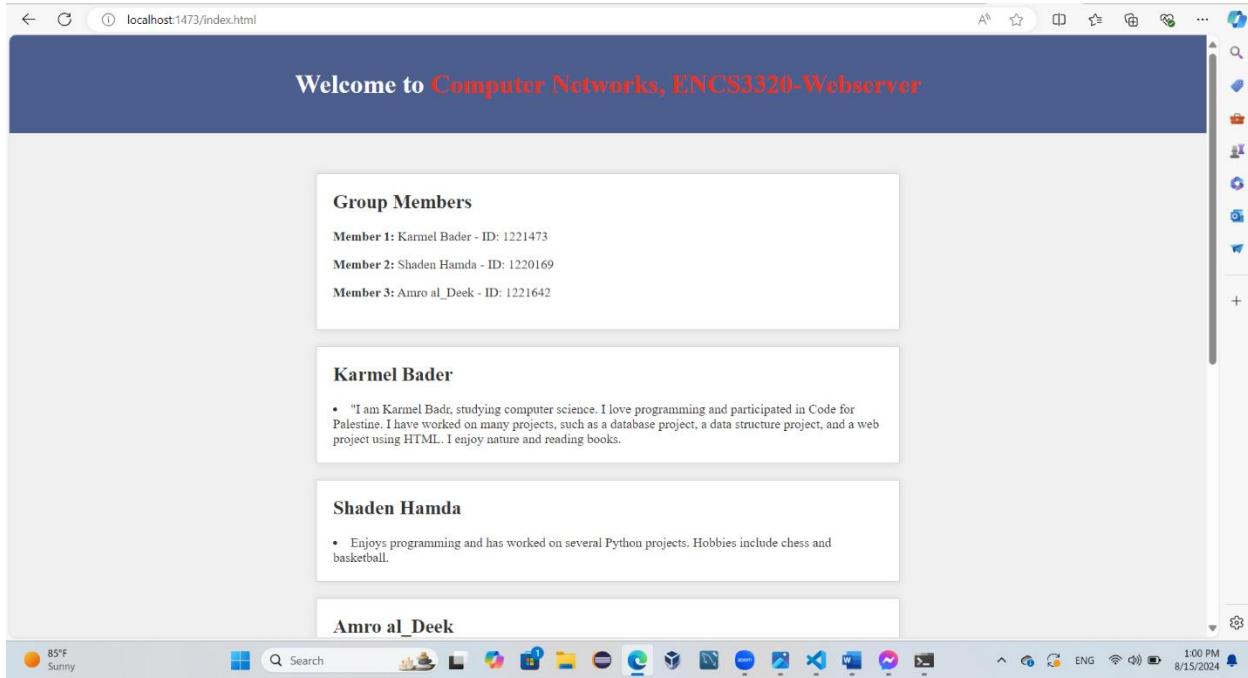


Figure 29: case 4 request

The HTTP request printed on the command Line

```
C:\Users\EASY LIFE>python OneDrive/Desktop/NetworkProj/Task3.py
Microsoft Windows [Version 10.0.22631.3888]
(c) Microsoft Corporation. All rights reserved.

C:\Users\EASY LIFE>python OneDrive/Desktop/NetworkProj/Task3.py
The server is ready to receive
Got connection from IP: 127.0.0.1, Port: 61612
/index.html
Got connection from IP: 127.0.0.1, Port: 61615
/style.css
Got connection from IP: 127.0.0.1, Port: 61616
/network2.jpg
Got connection from IP: 127.0.0.1, Port: 61618
/network1.png
Got connection from IP: 127.0.0.1, Port: 61620
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61621
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61622
```

Figure 30: case 4 request on command line

**Case5:** when the request `localhost:1473/main_ar.html` the server should send the `main_ar.html` file

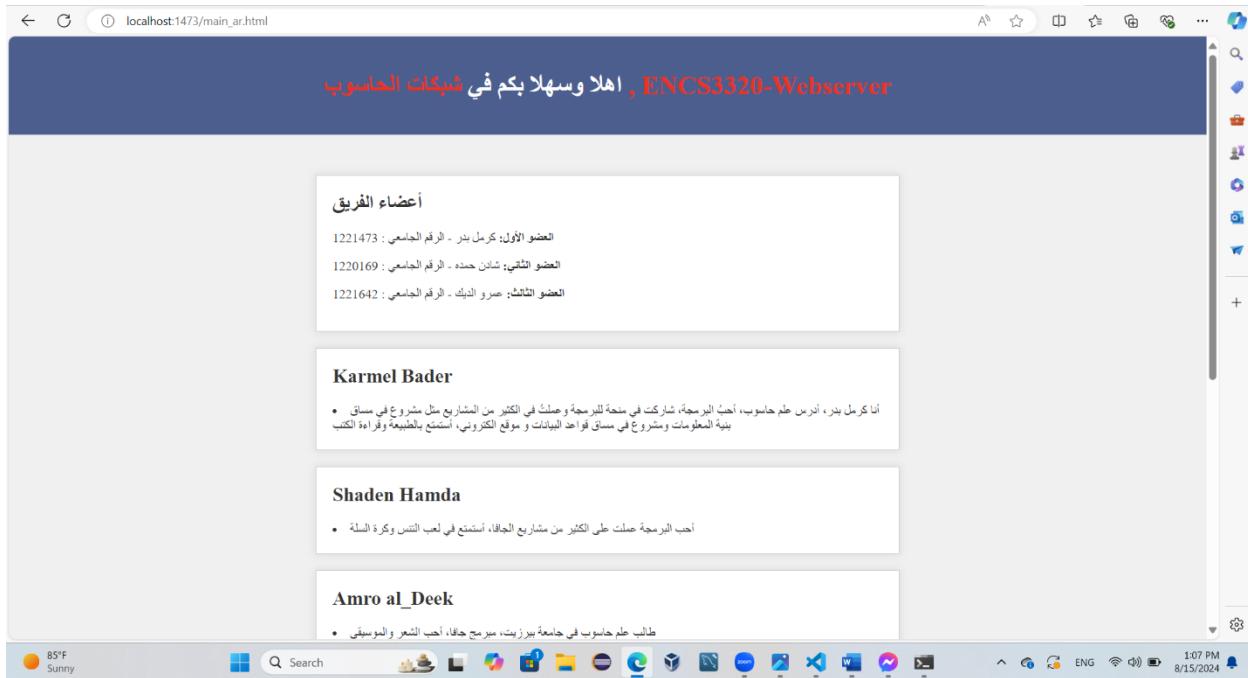


Figure 31: case 5 request

The HTTP request printed on the command Line

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\EASY LIFE>python OneDrive\Desktop\NetworkProj\Task3.py
The server is ready to receive
Got connection from IP: 127.0.0.1, Port: 61701
/main_ar.html
Got connection from IP: 127.0.0.1, Port: 61702
/style.css
Got connection from IP: 127.0.0.1, Port: 61704
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61705
```

Figure 32: case 5 request on command line

## Case6:

when the request `localhost:1473/network1.png` the server should send the `network1.png`

Note: `network1.png` is just an example any another image will work on it.

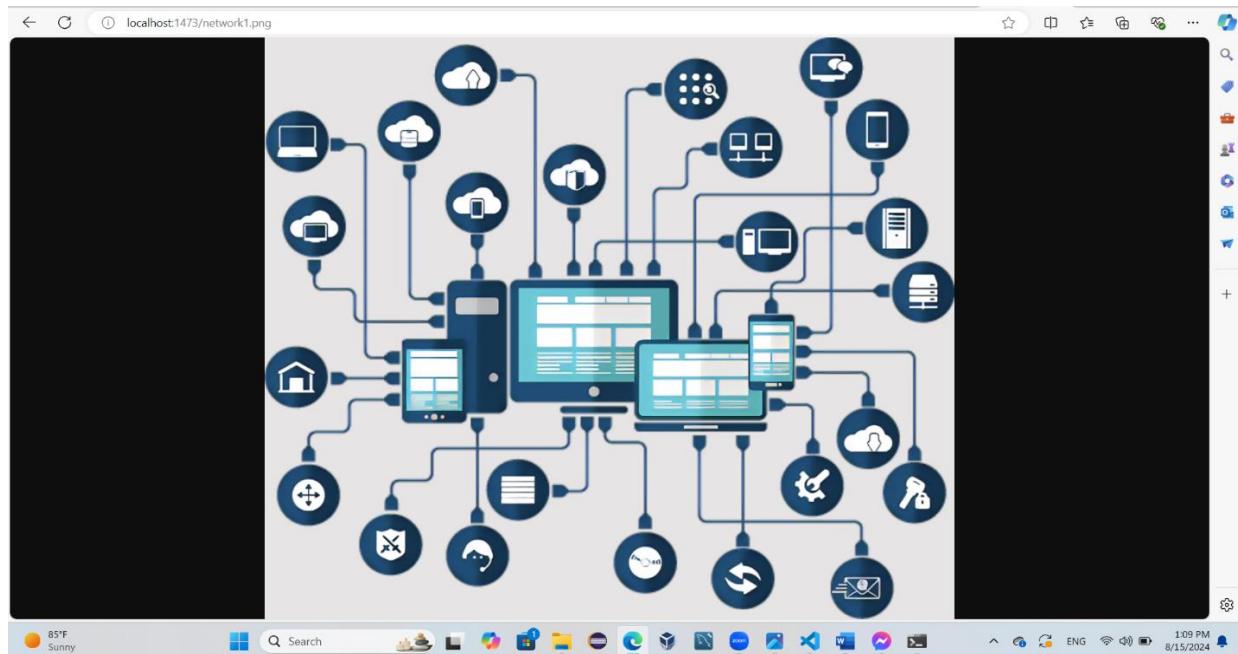


Figure 33: case 6 request

The HTTP request printed on the command Line

A screenshot of a Microsoft Windows Command Prompt window titled "Command Prompt - python". The window displays several lines of text representing network traffic logs. The logs show the server receiving connections from IP 127.0.0.1 on port 61735 for the file "/network1.png", port 61736 for "/favicon.ico", and port 61737 for "/favicon.ico". The background of the window shows a blurred view of the desktop environment, including the taskbar with various icons and the system tray.

Figure 34: case 6 request on command line

## Case7:

when the request `localhost:1473/network2.jpg` the server should send the `network2.jpg`

Note: `network2.jpg` is just an example any another image will work on it.

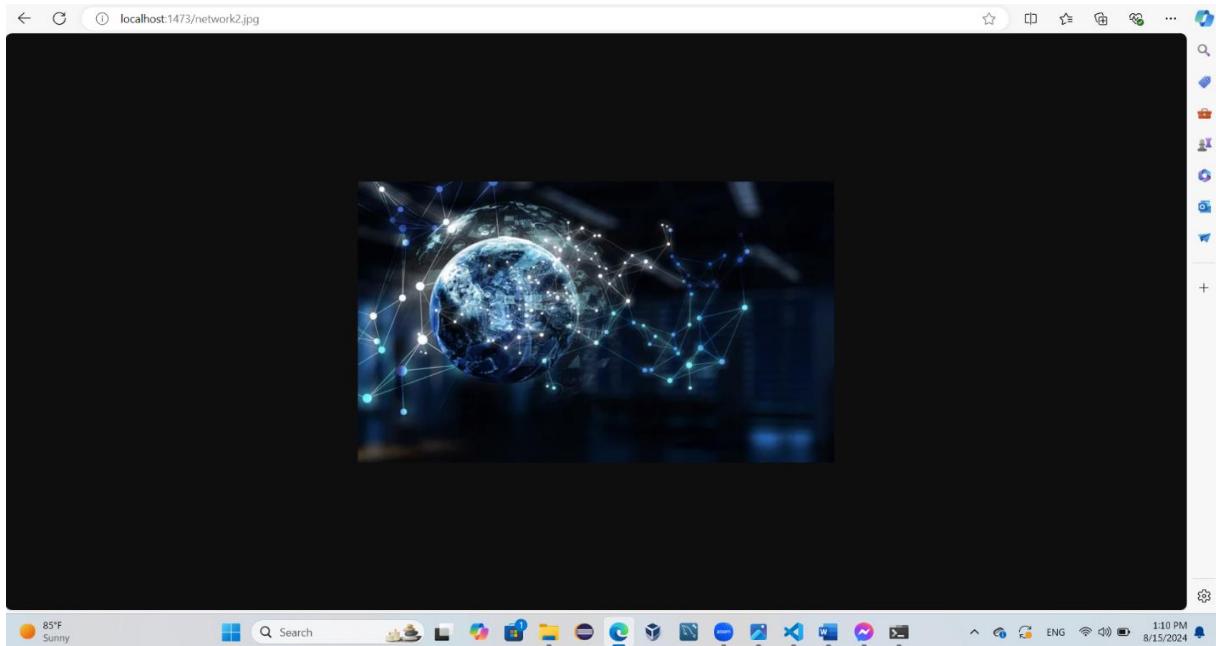


Figure 35: case 7 request

The HTTP request printed on the command Line

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\EASY LIFE>python OneDrive/Desktop/NetworkProj/Task3
.py
The server is ready to receive
Got connection from IP: 127.0.0.1, Port: 61763
/network2.jpg
Got connection from IP: 127.0.0.1, Port: 61764
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61765
/favicon.ico
```

Figure 36: case 7 request on command line

## Case8:

when the request [localhost:1473/style.css](http://localhost:1473/style.css) the server should send the [style.css](#)

Note: [style.css](#) is just an example any another file .css will work on it.

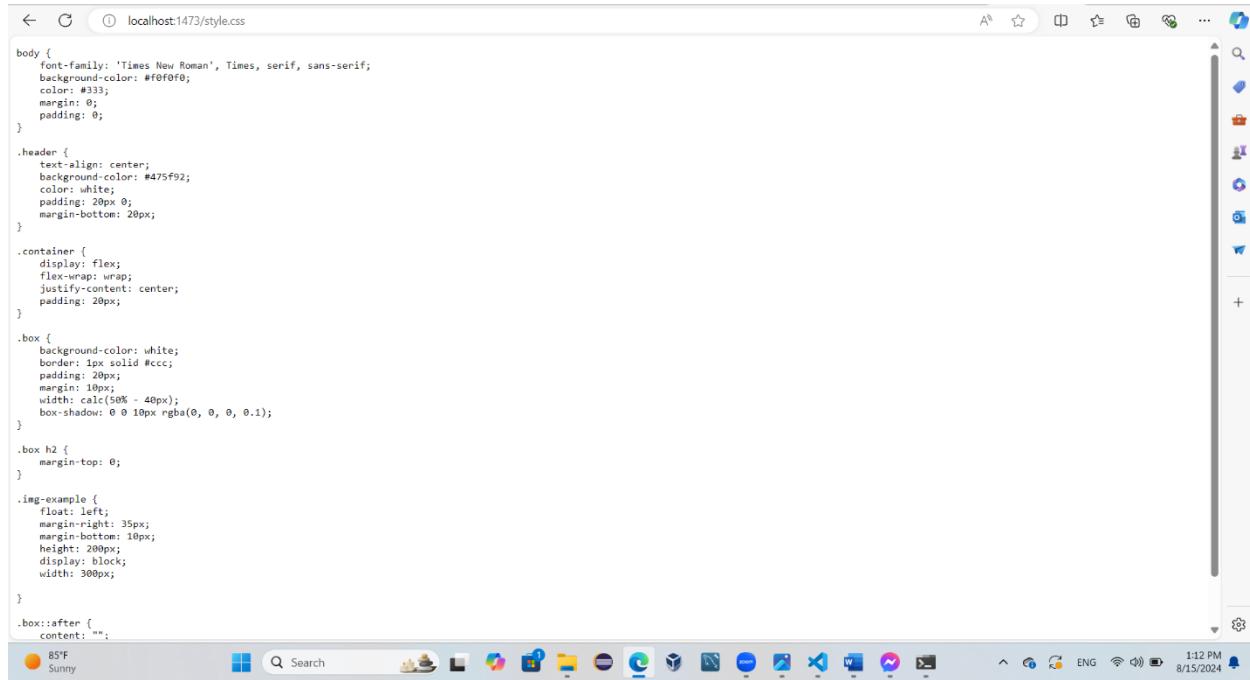


Figure 37: case 8 request

The HTTP request printed on the command Line

A screenshot of a Windows Command Prompt window titled 'Command Prompt - python'. The window displays the output of a Python script named 'Task3.py'. The output shows the server listening for connections and receiving requests for 'style.css', 'favicon.ico', and 'favicon.ico' again. The command prompt interface includes a title bar, a menu bar, and a taskbar at the bottom.

Figure 38: case 8 request on command line

## Case9:

when the request [localhost:1473/main\\_en.html](http://localhost:1473/main_en.html) the server should send the [main\\_en.html](#)

Note: main\_en.html is just an example any another file .html will work on it.

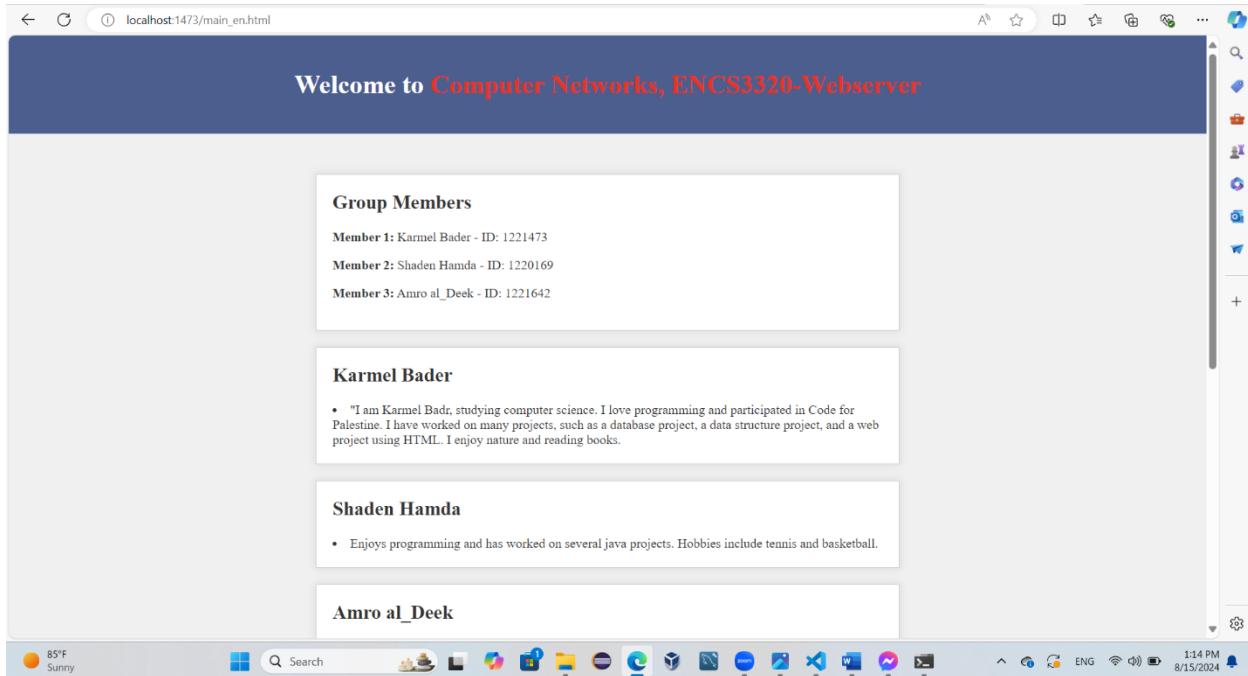


Figure 39: case 9 request

The HTTP request printed on the command Line

A screenshot of a Windows Command Prompt window titled 'Command Prompt - python C'. The window shows the following text:

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\EASY LIFE>python OneDrive/Desktop/NetworkProj/Task3.py
The server is ready to receive
Got connection from IP: 127.0.0.1, Port: 61808
/main_en.html
Got connection from IP: 127.0.0.1, Port: 61811
/style.css
Got connection from IP: 127.0.0.1, Port: 61812
/network2.jpg
Got connection from IP: 127.0.0.1, Port: 61813
/network1.png
Got connection from IP: 127.0.0.1, Port: 61814
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61817
/favicon.ico
Got connection from IP: 127.0.0.1, Port: 61818
```

The command prompt window has a dark theme and is located on a Windows desktop. The taskbar at the bottom shows various pinned icons and the system tray indicates it's 11:14 PM on 8/15/2024.

Figure 40: case 9 request on command line

**mySiteSTDID.html** file to get an image if the path is correct :

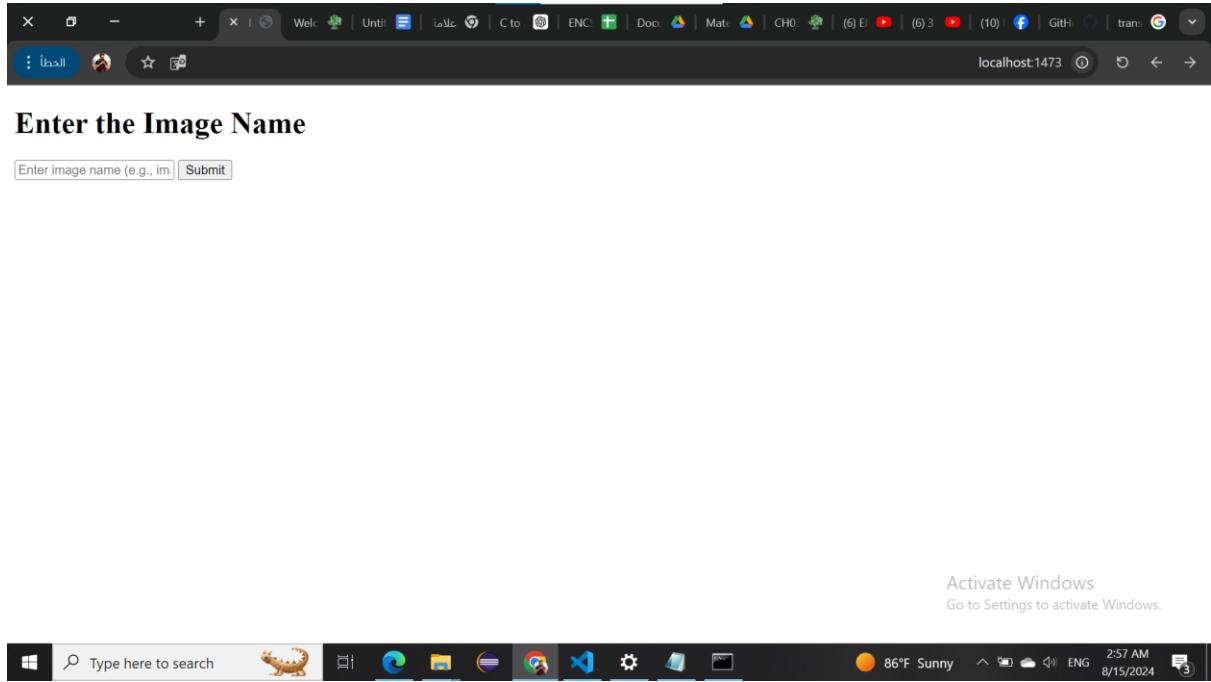


Figure 41: mySiteSTDID.html running output

Enter a correct path to show the image :

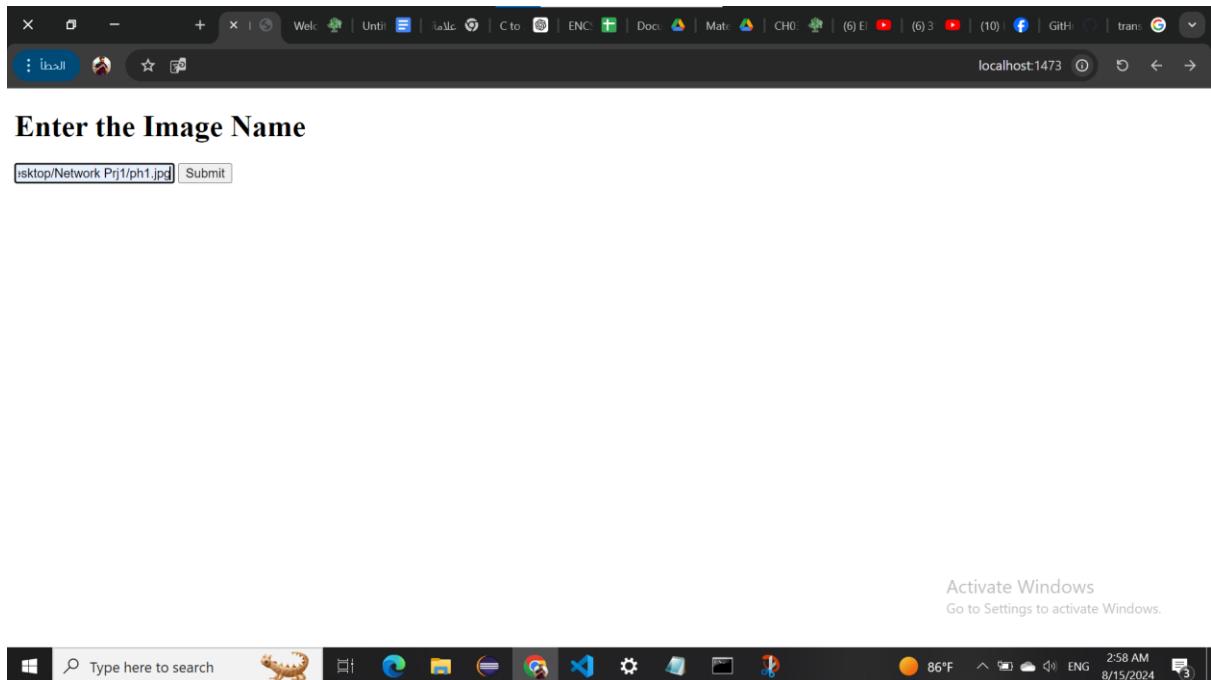


Figure 42: mySiteStDID.html running output with correct path

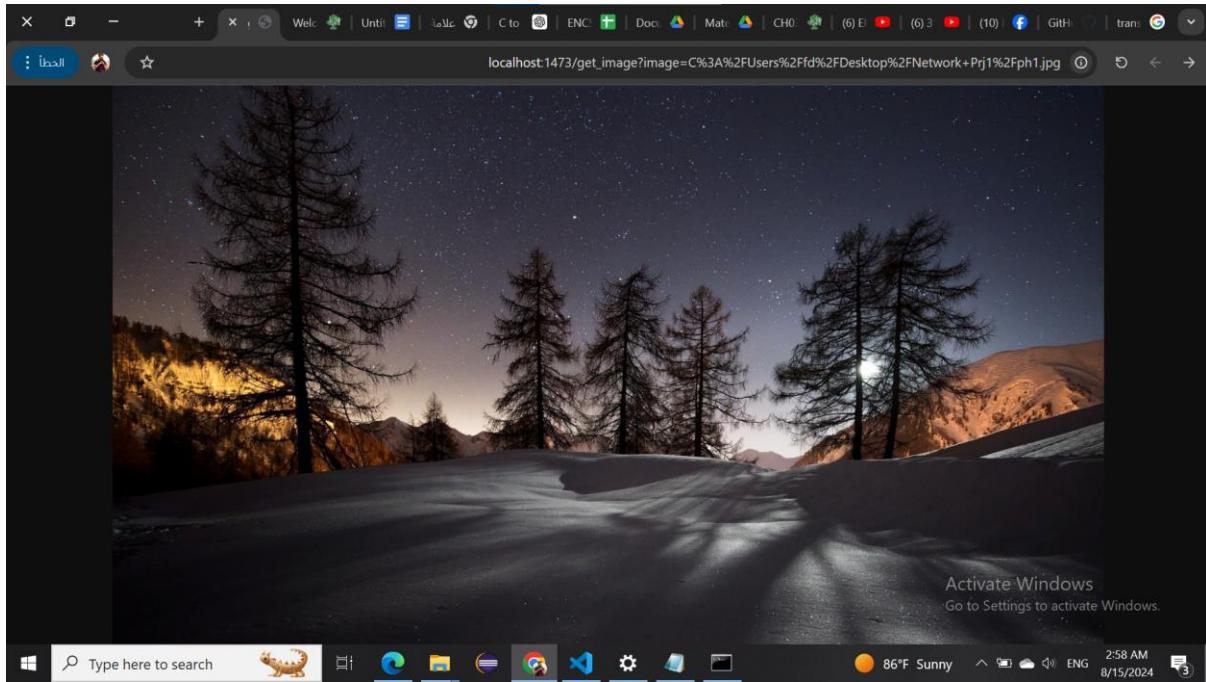


Figure 43: output of correct path

Enter a **wrong** path :

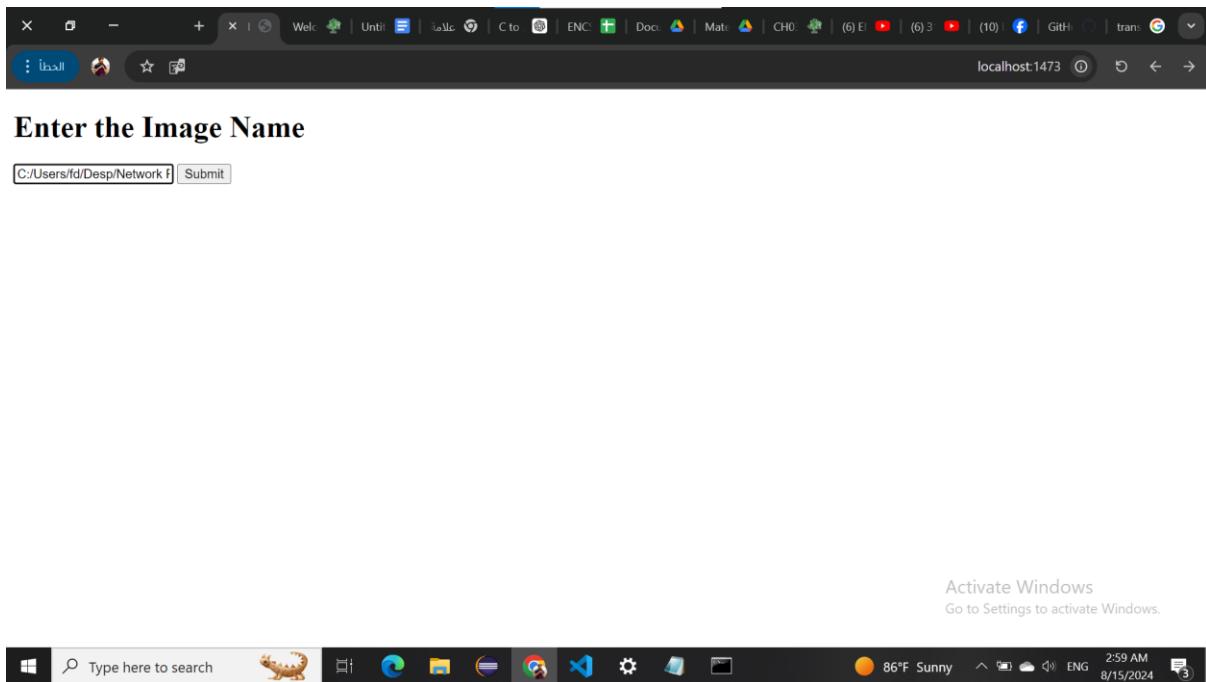
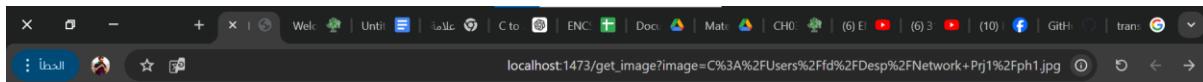


Figure 44: mySiteStDID.html running output with wrong path

Because the path is **wrong** or the image is **not exists** it shows a simple html file like :



## The file is not found

Amro Deek  
1221642

Client IP: 127.0.0.1  
Port: 65135



*Figure 45: output of wrong path*

the HTTP requests on the terminal window (command line window):

```
File Edit Selection View Go Run ... ← → Network Prj1
Command Prompt - python httpServer.py
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

C:\Users\fd>cd Desktop

C:\Users\fd\Desktop>cd "Network Prj1"

C:\Users\fd\Desktop\Network Prj1>python httpServer.py
Starting httpd server on port 1473
Request: GET / HTTP/1.1 from ('127.0.0.1', 65103)
127.0.0.1 - - [15/Aug/2024 02:57:07] "GET / HTTP/1.1" 200 -
Request: GET /get_image?image=C%3A%2FUsers%2Fd%2FDesktop%2FNetwork+Prj1%2Fph1.jpg HTTP/1.1 from ('127.0.0.1', 65124)
127.0.0.1 - - [15/Aug/2024 02:58:22] "GET /get_image?image=C%3A%2FUsers%2Fd%2FDesktop%2FNetwork+Prj1%2Fph1.jpg HTTP/1.1" 200 -
Request: GET /get_image?image=C%3A%2FUsers%2Fd%2FDesp%2FNetwork+Prj1%2Fph1.jpg HTTP/1.1 from ('127.0.0.1', 65135)
127.0.0.1 - - [15/Aug/2024 02:59:36] "GET /get_image?image=C%3A%2FUsers%2Fd%2FDesp%2FNetwork+Prj1%2Fph1.jpg HTTP/1.1" 404 -
```

*Figure 46: http request on command line*

If the request is /so then redirect to stackoverflow.com website :

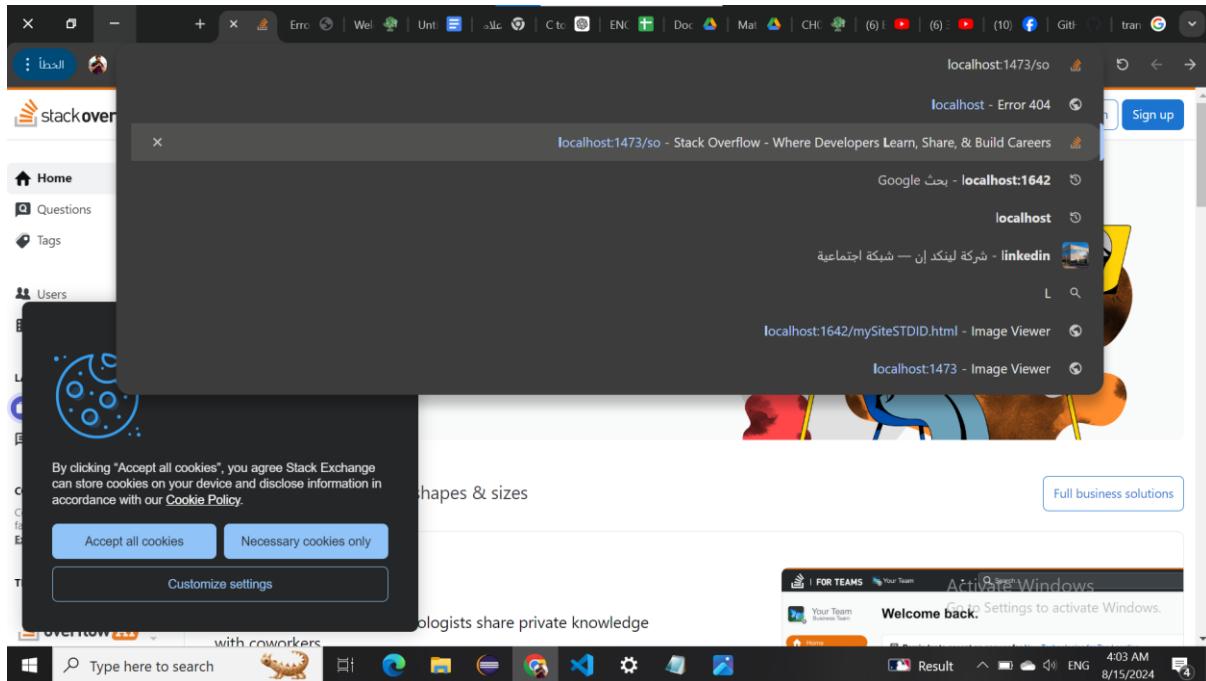


Figure 47: output of /so request

If the request is /itc then redirect to itc.birzeit.edu website:

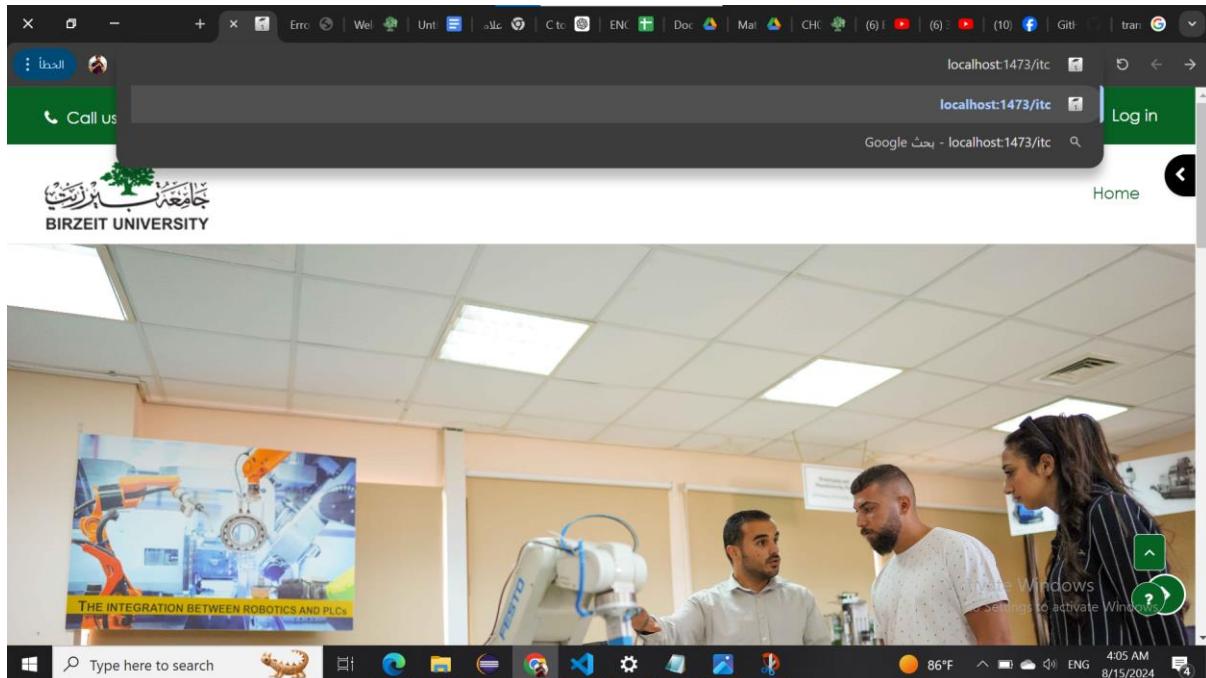


Figure 48: output of /itc request

## Test the project from a browser on a different computer

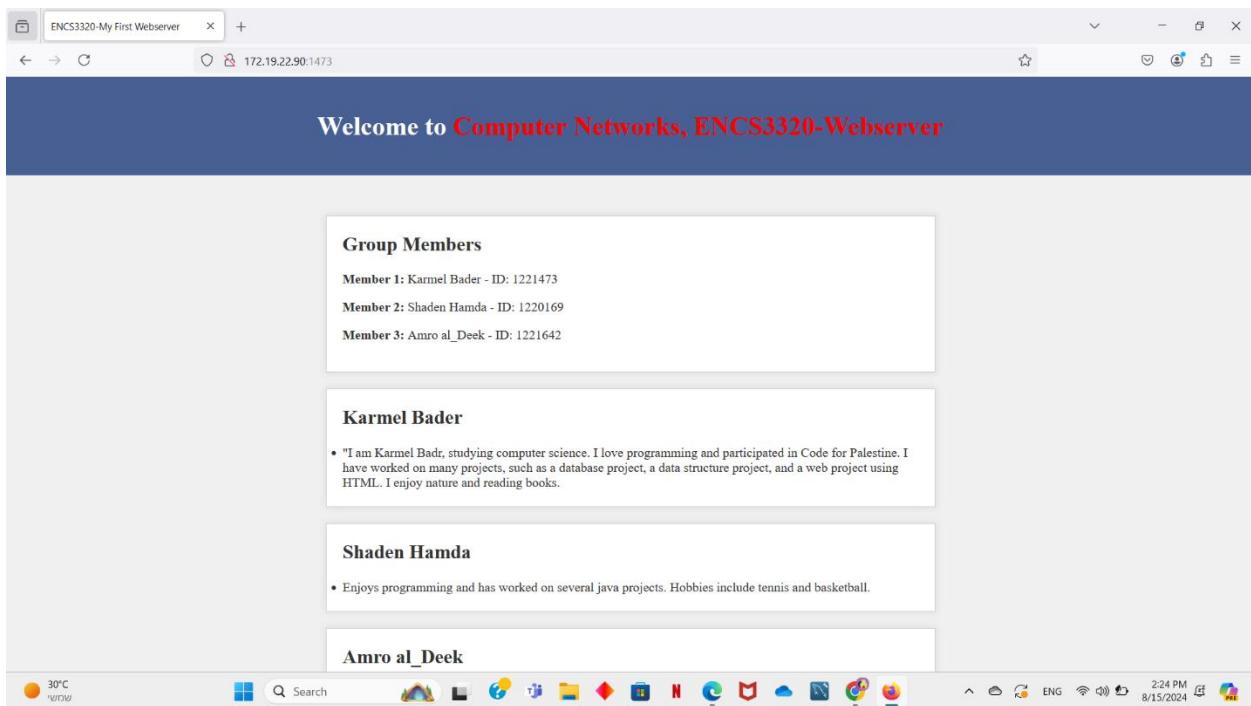
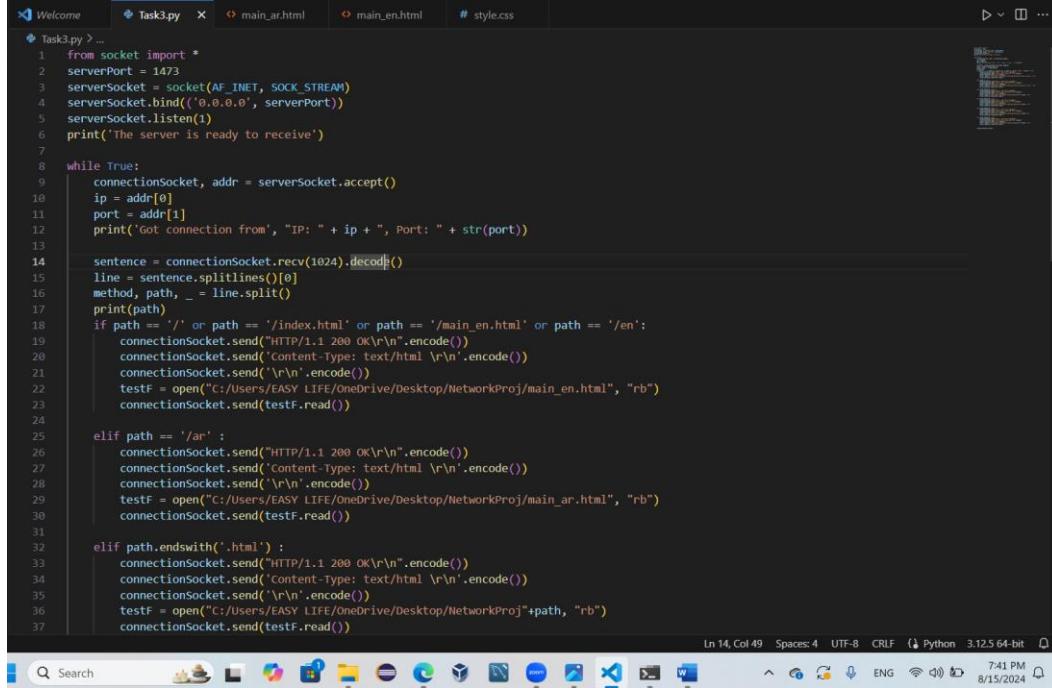


Figure 49: output of testing the project on other computer

**Task3 Code:** here is **Task3.py** code which handle the cases of the request, the requested path is checked by taking the first line of the request then splitting it and then comparing the different cases of the path to output the required one ( .html, .jpg ... etc).

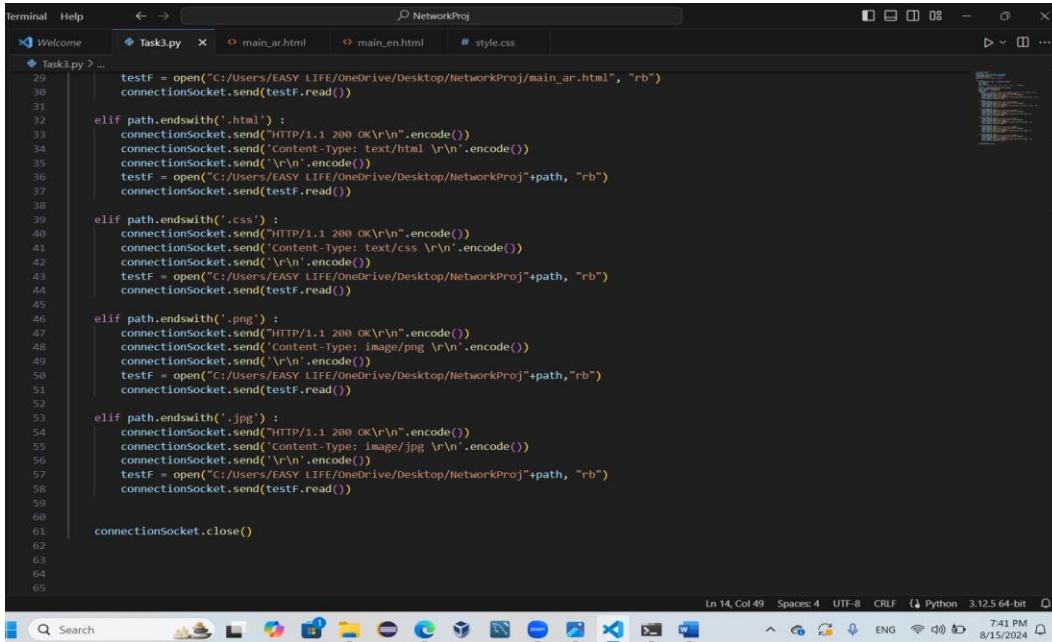


```

1 from socket import *
2 serverPort = 1473
3 serverSocket = socket(AF_INET, SOCK_STREAM)
4 serverSocket.bind(('', serverPort))
5 serverSocket.listen(1)
6 print('The server is ready to receive')
7
8 while True:
9     connectionSocket, addr = serverSocket.accept()
10    ip = addr[0]
11    port = addr[1]
12    print('Got connection from', 'IP: ' + ip + ', Port: ' + str(port))
13
14    sentence = connectionSocket.recv(1024).decode()
15    line = sentence.splitlines()[0]
16    method, path, _ = line.split()
17    print(path)
18    if path == '/' or path == '/index.html' or path == '/main_en.html' or path == '/en':
19        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
20        connectionSocket.send('Content-Type: text/html \r\n'.encode())
21        connectionSocket.send('\r\n'.encode())
22        testF = open("C:/Users/EASY LIFE/OneDrive/Desktop/NetworkProj/main_en.html", "rb")
23        connectionSocket.send(testF.read())
24
25    elif path == '/ar' :
26        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
27        connectionSocket.send('Content-Type: text/html \r\n'.encode())
28        connectionSocket.send('\r\n'.encode())
29        testF = open("C:/Users/EASY LIFE/OneDrive/Desktop/NetworkProj/main_ar.html", "rb")
30        connectionSocket.send(testF.read())
31
32    elif path.endswith('.html') :
33        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
34        connectionSocket.send('Content-Type: text/html \r\n'.encode())
35        connectionSocket.send('\r\n'.encode())
36        testF = open("C:/Users/EASY LIFE/OneDrive/Desktop/NetworkProj"+path, "rb")
37        connectionSocket.send(testF.read())
38
39    elif path.endswith('.css') :
40        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
41        connectionSocket.send('Content-Type: text/css \r\n'.encode())
42        connectionSocket.send('\r\n'.encode())
43        testF = open("C:/Users/EASY LIFE/OneDrive/Desktop/NetworkProj"+path, "rb")
44        connectionSocket.send(testF.read())
45
46    elif path.endswith('.png') :
47        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
48        connectionSocket.send('Content-Type: image/png \r\n'.encode())
49        connectionSocket.send('\r\n'.encode())
50        testF = open("C:/Users/EASY LIFE/OneDrive/Desktop/NetworkProj"+path, "rb")
51        connectionSocket.send(testF.read())
52
53    elif path.endswith('.jpg') :
54        connectionSocket.send("HTTP/1.1 200 OK\r\n".encode())
55        connectionSocket.send('Content-Type: image/jpg \r\n'.encode())
56        connectionSocket.send('\r\n'.encode())
57        testF = open("C:/Users/EASY LIFE/OneDrive/Desktop/NetworkProj"+path, "rb")
58        connectionSocket.send(testF.read())
59
60
61    connectionSocket.close()
62
63
64
65

```

Figure 50: Task3.py code\_1



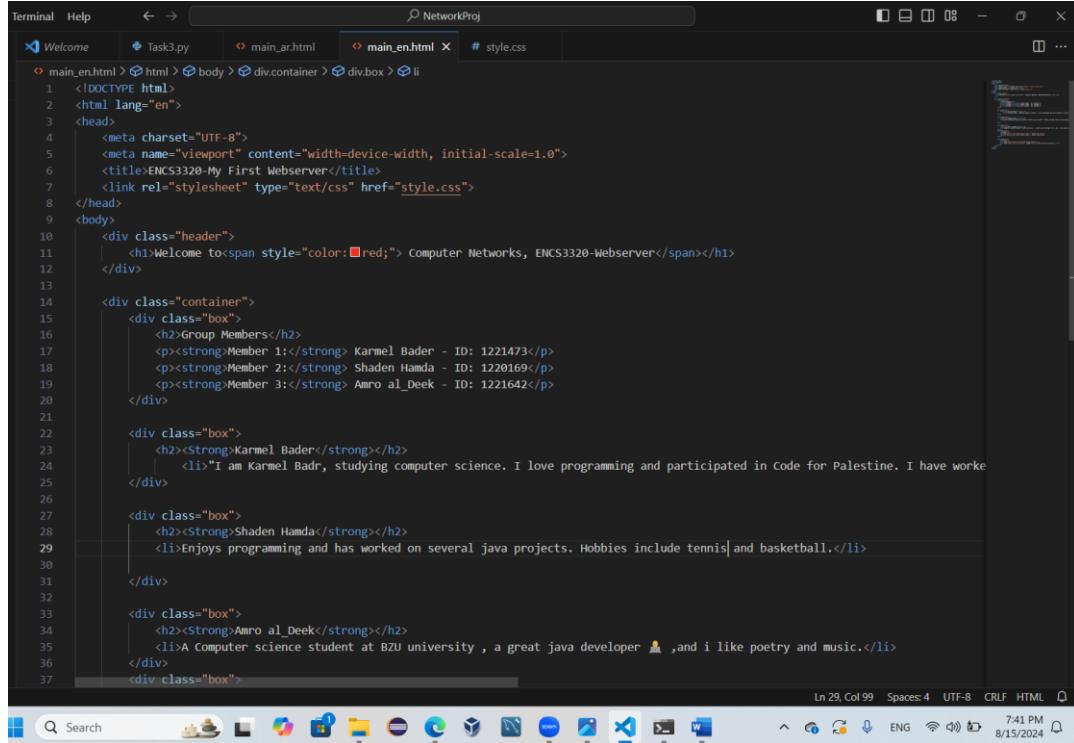
```

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

```

Figure 51: Task3.py code\_2

Here is the English version of the html page, which contains the group members information and two images also two links

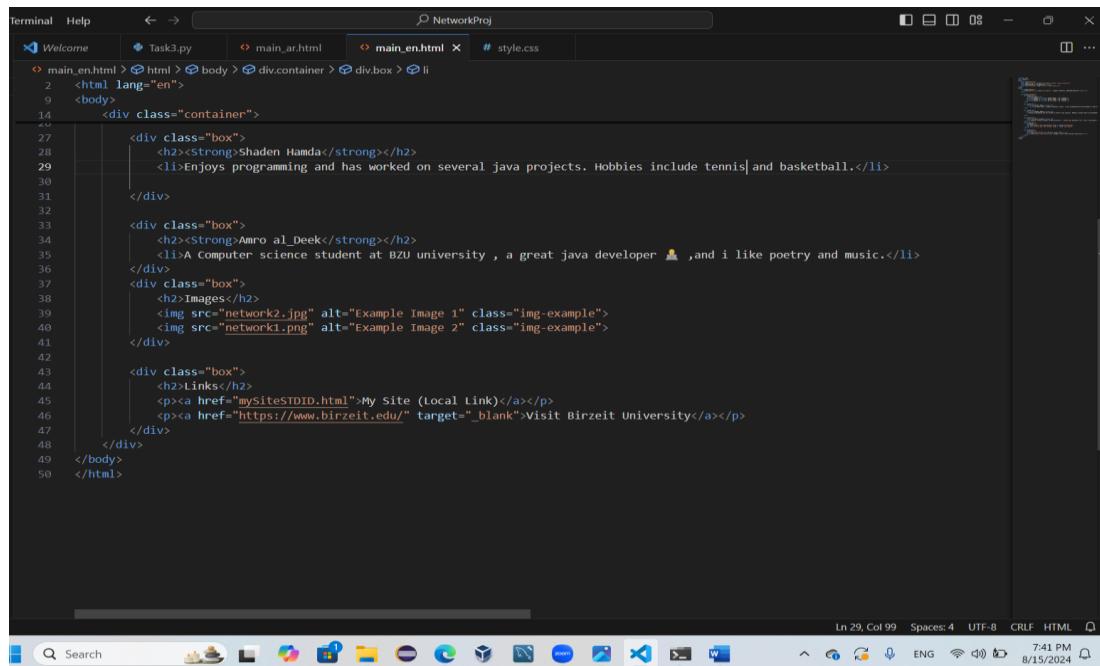


```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>ENCS3320-My First Webserver</title>
7   <link rel="stylesheet" type="text/css" href="style.css">
8 </head>
9 <body>
10  <div class="header">
11    <h1>Welcome to<span style="color:red;"> Computer Networks, ENCS3320-Webserver</span></h1>
12  </div>
13
14  <div class="container">
15    <div class="box">
16      <h2>Group Members</h2>
17      <p><strong>Member 1:</strong> Karmel Bader - ID: 1221473</p>
18      <p><strong>Member 2:</strong> Shaden Hamda - ID: 1220169</p>
19      <p><strong>Member 3:</strong> Amro al_Deek - ID: 1221642</p>
20    </div>
21
22    <div class="box">
23      <h2><strong>Karmel Bader</strong></h2>
24      <li>I am Karmel Badr, studying computer science. I love programming and participated in Code for Palestine. I have worke
25    </div>
26
27    <div class="box">
28      <h2><strong>Shaden Hamda</strong></h2>
29      <li>Enjoys programming and has worked on several java projects. Hobbies include tennis and basketball.</li>
30    </div>
31
32    <div class="box">
33      <h2><strong>Amro al_Deek</strong></h2>
34      <li>A Computer science student at BZU university , a great java developer 🚀 ,and i like poetry and music.</li>
35    </div>
36  </div>

```

Figure 52: main\_en.html code\_1



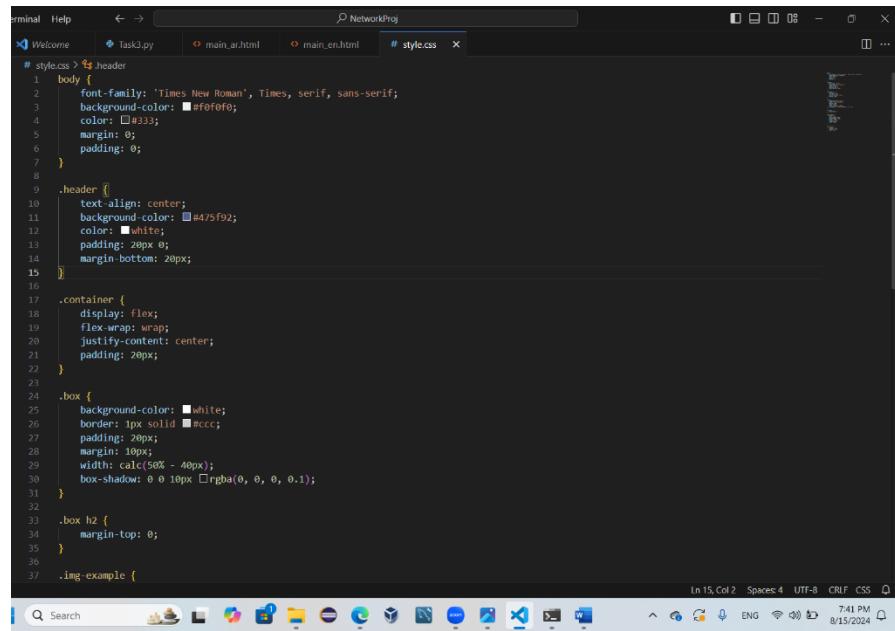
```

1 <!DOCTYPE html>
2 <html lang="en">
3 <body>
4   <div class="container">
5     <div class="box">
6       <h2><strong>Shaden Hamda</strong></h2>
7       <li>Enjoys programming and has worked on several java projects. Hobbies include tennis and basketball.</li>
8     </div>
9
10    <div class="box">
11      <h2><strong>Amro al_Deek</strong></h2>
12      <li>A Computer science student at BZU university , a great java developer 🚀 ,and i like poetry and music.</li>
13    </div>
14
15    <div class="box">
16      <h2>Images</h2>
17      
18      
19    </div>
20
21    <div class="box">
22      <h2>Links</h2>
23      <p><a href="mySiteSTIDID.html">My Site (Local Link)</a></p>
24      <p><a href="https://www.birzeit.edu/" target="_blank">Visit Birzeit University</a></p>
25    </div>
26
27  </div>
28 </body>
29 </html>

```

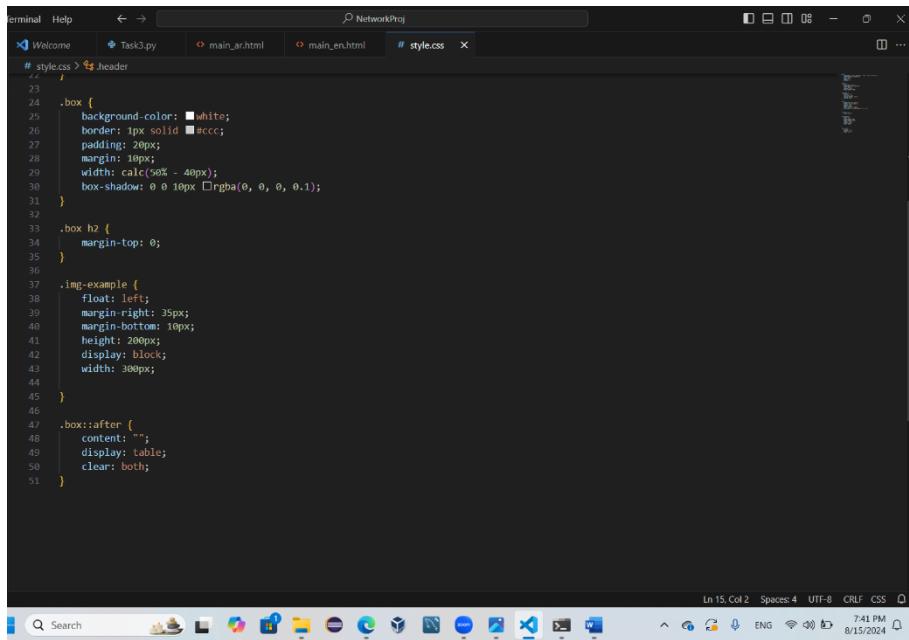
Figure 53: main\_en.html code\_2

Here is the **css** file which confirm the style of the html files ( both English and Arabic versions ).



```
# style.css > 4. header
1 body {
2     font-family: 'Times New Roman', Times, serif, sans-serif;
3     background-color: #f0f0f0;
4     color: #333;
5     margin: 0;
6     padding: 0;
7 }
8
9 .header {
10    text-align: center;
11    background-color: #475f92;
12    color: white;
13    padding: 20px 0;
14    margin-bottom: 20px;
15 }
16
17 .container {
18    display: flex;
19    flex-wrap: wrap;
20    justify-content: center;
21    padding: 20px;
22 }
23
24 .box {
25    background-color: white;
26    border: 1px solid #ccc;
27    padding: 20px;
28    margin: 10px;
29    width: calc(50% - 40px);
30    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
31 }
32
33 .box h2 {
34    margin-top: 0;
35 }
36
37 .img-example {
```

Figure 54: style.css code\_1

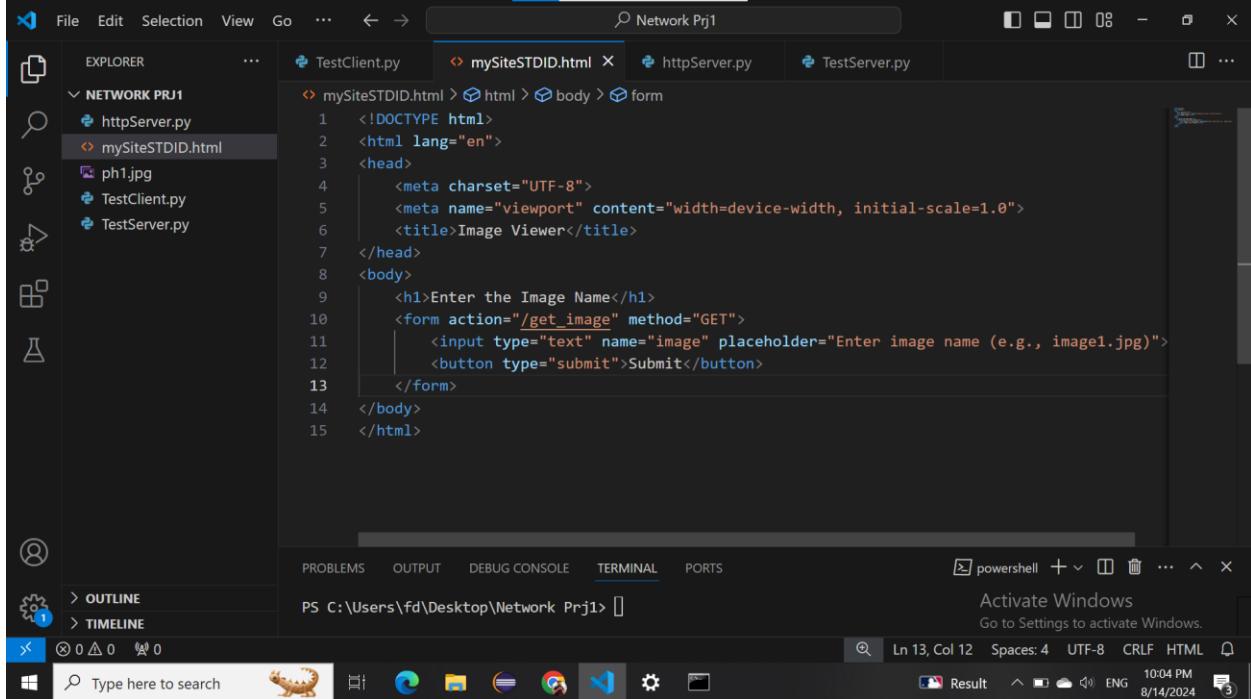


```
# style.css > 4. header
1
2
3 .box {
4    background-color: white;
5    border: 1px solid #ccc;
6    padding: 20px;
7    margin: 10px;
8    width: calc(50% - 40px);
9    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
10 }
11
12 .box h2 {
13    margin-top: 0;
14 }
15
16 .img-example {
17    float: left;
18    margin-right: 35px;
19    margin-bottom: 10px;
20    height: 200px;
21    display: block;
22    width: 300px;
23 }
24
25 .box::after {
26    content: "";
27    display: table;
28    clear: both;
29 }
```

Figure 55: style.css code\_2

**mySiteSTDID.html** code , we use the GET method (for sending data to server)

include user data in URL field of the HTTP GET request message.



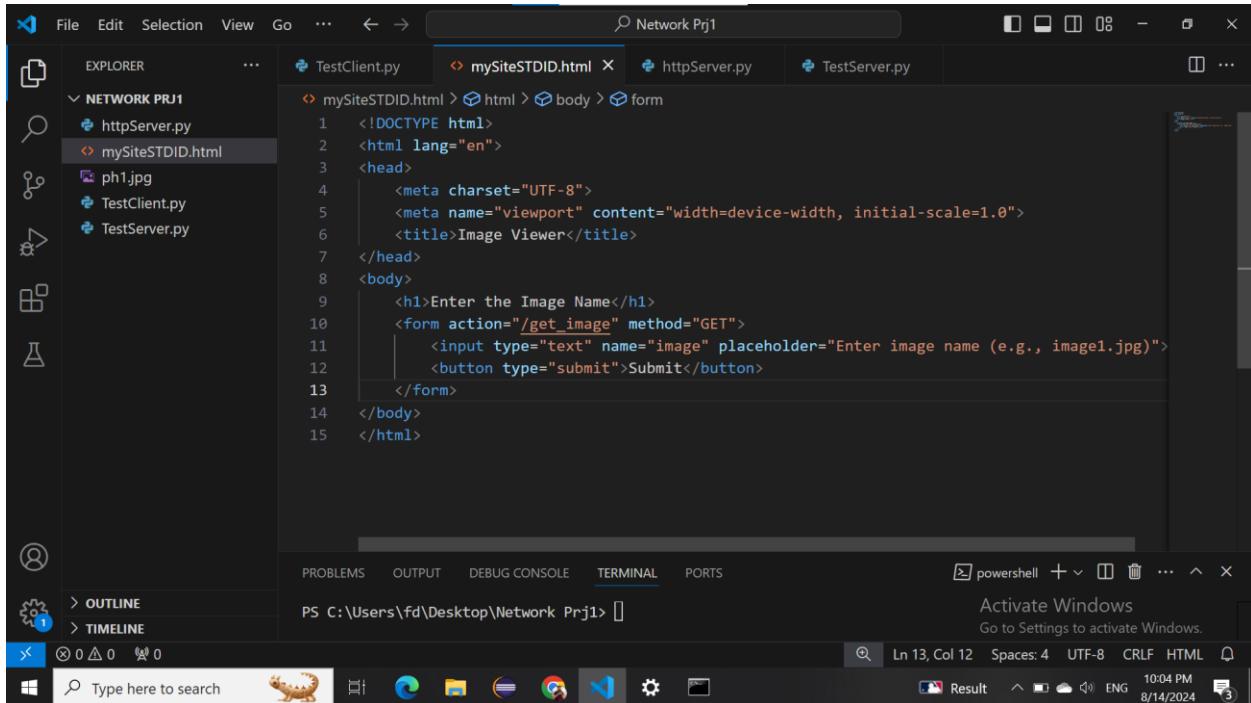
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the "NETWORK PRJ1" folder: httpServer.py, mySiteSTDID.html, ph1.jpg, TestClient.py, and TestServer.py.
- Code Editor:** Displays the content of mySiteSTDID.html. The code is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Viewer</title>
</head>
<body>
    <h1>Enter the Image Name</h1>
    <form action="/get_image" method="GET">
        <input type="text" name="image" placeholder="Enter image name (e.g., image1.jpg)">
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

- Terminal:** Shows the command PS C:\Users\fd\Desktop\Network Prj1>.
- Status Bar:** Shows the path PS C:\Users\fd\Desktop\Network Prj1>, line 13, column 12, spaces: 4, encoding: UTF-8, and date/time: 10:04 PM 8/14/2024.

Figure 56: mySiteSTDID.html code\_1



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the "NETWORK PRJ1" folder: httpServer.py, mySiteSTDID.html, ph1.jpg, TestClient.py, and TestServer.py.
- Code Editor:** Displays the content of mySiteSTDID.html. The code is identical to Figure 56:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Viewer</title>
</head>
<body>
    <h1>Enter the Image Name</h1>
    <form action="/get_image" method="GET">
        <input type="text" name="image" placeholder="Enter image name (e.g., image1.jpg)">
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

- Terminal:** Shows the command PS C:\Users\fd\Desktop\Network Prj1>.
- Status Bar:** Shows the path PS C:\Users\fd\Desktop\Network Prj1>, line 13, column 12, spaces: 4, encoding: UTF-8, and date/time: 10:04 PM 8/14/2024.

Figure 57: mySiteSTDID.html code\_2

and **httpServer.py** code , This Python script sets up a basic HTTP server that handles specific GET requests. It can redirect to predefined URLs, serve images from a directory based on user input, and display a custom HTML page if the requested image is not found. The server also logs each request and handles 404 errors with a user-defined response.we used port # 1473 (karmel's ID because my ID 1642 does not work , it seems to be reserved for something on my computer ).

Figure 58: httpServer.py code 1

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

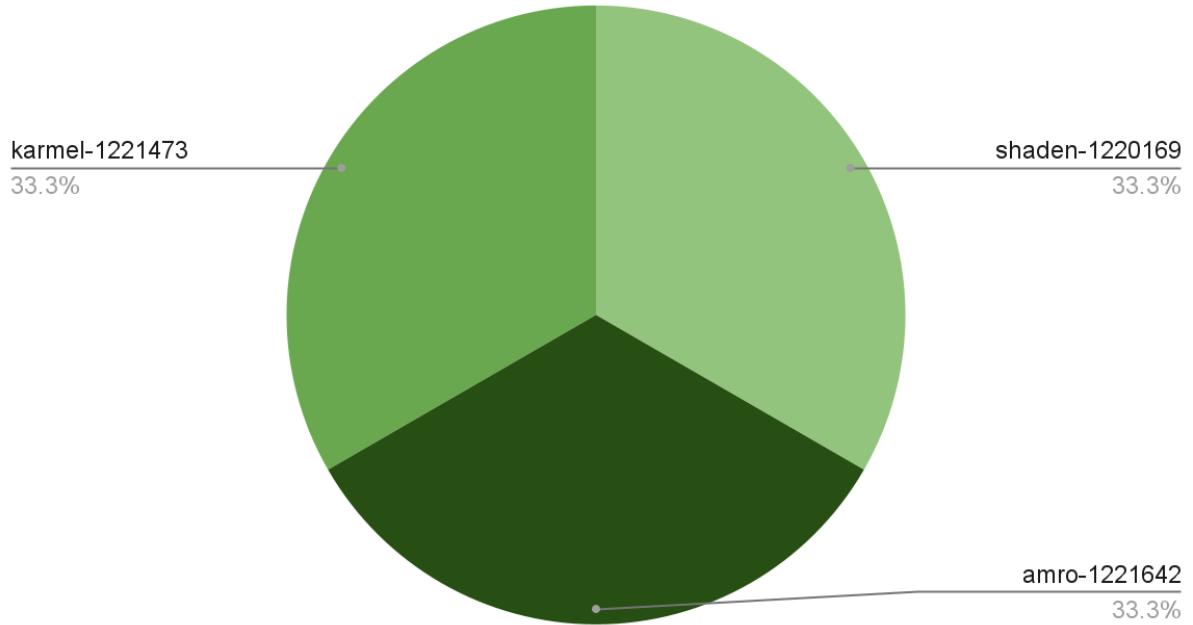
- File Explorer (Left):** Shows the project structure under "NETWORK PRJ1". Files include TestClient.py, mySiteSTDID.html, httpServer.py, ph1.jpg, TestClient.py, and TestServer.py.
- Code Editor (Center):** Displays the content of httpServer.py. The code defines a class MyServer that inherits from BaseHTTPRequestHandler. It includes methods for handling GET requests and serving files. It also defines a run function to start an HTTP server on port 1473.
- Bottom Status Bar:** Shows the current file is httpServer.py, line 28, column 13. Other status indicators include Python 3.11.9 64-bit (Microsoft Store), 10:16 PM, 79°F Sunny, ENG, and 8/15/2024.

```
File Edit Selection View Go ... ⏪ ⏩ Network Prj1
EXPLORER NETWORK PRJ1
httpServer.py
mySiteSTDID.html
ph1.jpg
TestClient.py
TestServer.py
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
<html>
<head><title>Error 404</title></head>
<body>
<h1 style="color: blue;">The file is not found</h1>
<p><b>Amro Deeks</b><br><b>1221642</b></p>
<p>Client IP: {self.client_address[0]}<br>Port: {self.client_address[1]}</p>
</body>
</html>
"""
self.wfile.write(error_page.encode())
def run(server_class=HTTPServer, handler_class=MyServer, port=1473):
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    print("Starting httpd server on port (port)")
    httpd.serve_forever()
if __name__ == "__main__":
    run(port=1473)
```

Figure 59: httpServer code\_2

## Team Work

Team work



### Everyone's job:

Shaden: Task 1(Q2,Q4),Task 2 (Q2)

Karmel: Task 1(Q1,Q3),Task 3 (to point 6)

Amro: Task 2(Q1) ,Task 3(from 7 to 11)

## **Issues and Limitations Part**

Web Page Not Functioning on Another Device We have encountered an issue where the web page is not functioning as expected on another device. While the page works correctly on the original device, it fails to operate or displays errors when accessed from a different one. This problem may be related to compatibility, browser settings, or device-specific configurations.

## **Solution**

the solution was to configure the server, making sure that the firewall is set up correctly, and determining whether we want to access the server locally or globally

## **References :**

[\*\*What are Traceroute, Ping, Telnet and Nslookup commands? - Hosting - Namecheap.com\*\*](#)

[\*\*Hurricane Electric BGP Toolkit \(he.net\)\*\*](#)