



BIRZEIT UNIVERSITY

**Department of Electrical and Computer Engineering**

**ENCS3320 - Computer Networks**

**Summer Semester 2023/2024**

**Project#1: Socket Programming**

**Prepared by:**

Mouath Masalmah - 1220179

Zaid Mousa - 1221833

Bahaa Bani Shamsaa - 1220252

**Instructor:** Dr.Ibrahim Nemer

**Section:** SECTION\_1

# Table of Contents

|   |    |
|---|----|
| <b>Theory:</b> .....  | 5  |
| <b>Task 1: Commands &amp; Wireshark</b> .....   | 5  |
| 1. Ping: .....  | 5  |
| 2. Tracert: .....   | 5  |
| 3. Nslookup: .....  | 5  |
| 4. Telnet: .....  | 5  |
| <b>Task 2: Socket Programming (TCP and UDP)</b> .....   | 5  |
| 11. TCP Client-Server Application: .....  | 5  |
| 12. UDP Client-Server Application: .....  | 5  |
| <b>Task 3: Web Server</b> .....   | 6  |
| 15. Web Server Application: .....   | 6  |
| <b>Procedure:</b> .....   | 7  |
| <b>Task 1: Commands &amp; Wireshark</b> .....   | 7  |
| 5. Ping a device in the same network: .....   | 7  |
| 6. Ping www.ox.ac.uk: .....   | 7  |
| 7. Tracert www.ox.ac.uk: .....  | 7  |
| 8. Nslookup www.ox.ac.uk:.....  | 7  |
| 9. Telnet www.ox.ac.uk: .....   | 7  |
| 10. Wireshark DNS Capture: .....  | 7  |
| <b>Task 2: Socket Programming (TCP and UDP)</b> .....   | 7  |
| 13. TCP Client-Server: .....  | 7  |
| 14. UDP Client-Server: .....  | 8  |
| <b>Task 3: Web Server</b> .....   | 8  |
| 16. Web Server Implementation: .....  | 8  |
| 17. HTML and CSS: .....   | 8  |
| 18. Testing and Documentation: .....  | 8  |
| <b>Result&amp;Discussion</b> .....  | 9  |
| <b>Task1: Commands &amp; Wireshark</b> .....  | 9  |
| Part 1: In your words, what are ping, tracert, nslookup, and telnet? .....  | 9  |
| Part 2: Make sure that your computer is connected to the internet and then run the following commands .....                           | 10 |
| Part 3: Give some details about autonomous system (AS) number, number of IPs, prefixes, peers, name of Tier1-ISP of www.ox.ac.uk..... | 16 |
| Task 2:.....  | 21 |
| Part 1:.....  | 21 |
| Part 2:.....  | 27 |
| Task 3:.....  | 33 |
| 1-.....   | 33 |
| 2.....  | 50 |
| 3.....  | 51 |

|                                      |           |
|--------------------------------------|-----------|
| 4.....                               | 52        |
| 5.....                               | 53        |
| 6.....                               | 54        |
| 7 and 8:.....                        | 55        |
| 9.....                               | 60        |
| 10.....                              | 62        |
| 11.....                              | 63        |
| <b>Problems and challenges:.....</b> | <b>64</b> |
| Work Done:.....                      | 66        |
| References.....                      | 67        |

# Theory:

## Task 1: Commands & Wireshark

### 1. Ping:

- Ping is a network diagnostic tool used to test the reachability of a host on an IP network. It works by sending ICMP Echo Request packets to the target host and waiting for an Echo Reply. This helps in measuring the round-trip time and identifying network issues.

### 2. Tracert:

- Tracert (or traceroute) is a utility that traces the path that a packet takes from the source to the destination across multiple routers. It helps in identifying the route, delays, and any points of failure along the path.

### 3. Nslookup:

- Nslookup is used to query the Domain Name System (DNS) to obtain domain name or IP address mappings. It's a helpful tool for troubleshooting DNS-related issues and understanding the DNS structure.

### 4. Telnet:

- Telnet is a protocol that allows you to connect to another machine on the network, typically for managing servers or network devices. It's an older protocol and lacks security, making it less commonly used today.

## Task 2: Socket Programming (TCP and UDP)

### 11. TCP Client-Server Application:

- TCP is a connection-oriented protocol that ensures reliable data transmission between client and server. In this task, the client sends a string to the server, which processes it (replacing vowels with #) and sends it back.

### 12. UDP Client-Server Application:

- UDP is a connectionless protocol that allows fast data transmission with no guarantee of delivery. The server listens on a specific port, and clients send messages to it. The server keeps track of the last message received from each client.

## **Task 3: Web Server**

### **15. Web Server Application:**

- A web server listens for HTTP requests from clients (browsers) and serves them content based on the request. The server needs to handle various content types (HTML, CSS, images) and redirect or return error pages as necessary.

# **Procedure:**

## **Task 1: Commands & Wireshark**

5. **Ping a device in the same network:**
  - Connect your laptop and smartphone to the same Wi-Fi network.
  - Open the command prompt on your laptop.
  - Type ping <smartphone IP> to test connectivity.
6. **Ping www.ox.ac.uk:**
  - In the command prompt, type ping www.ox.ac.uk.
  - Note the response times and analyze the location based on the IP.
7. **Tracert www.ox.ac.uk:**
  - Type tracert www.ox.ac.uk in the command prompt.
  - Observe the hops and response times along the route to Oxford University.
8. **Nslookup www.ox.ac.uk:**
  - Type nslookup www.ox.ac.uk in the command prompt.
  - Record the IP address returned and verify its ownership.
9. **Telnet www.ox.ac.uk:**
  - Type telnet www.ox.ac.uk in the command prompt.
  - Attempt to establish a connection and note the result.
10. **Wireshark DNS Capture:**
  - Open Wireshark and start capturing packets.
  - Perform some DNS queries (e.g., visit a website).
  - Stop the capture and analyze the DNS messages.
  - Take screenshots showing the time and date.

## **Task 2: Socket Programming (TCP and UDP)**

13. **TCP Client-Server:**
  - Write a server program that listens on a port.
  - Write a client program that connects to the server and sends a string.
  - The server replaces vowels with # and sends the modified string back.
  - The client prints the received string.
14. **UDP Client-Server:**
  - Write a UDP server that listens on a port.
  - Write UDP clients that send messages to the server.

- The server logs the messages and the clients print their received messages.
- Test the communication between multiple clients and the server.

## Task 3: Web Server

### 16. Web Server Implementation:

- Write a server program that listens on a specific port.
- Handle different URL requests by serving the appropriate HTML or CSS files.
- Implement logic to return custom error pages for invalid requests.
- Include images, links, and redirects as specified.
- Test the server by making various requests from a browser (e.g., <http://localhost:1515/en>).

### 17. HTML and CSS:

- Create HTML files (main\_en.html, main\_ar.html, etc.) that include information about the team, images, and links.
- Style the pages using CSS, ensuring the layout is visually appealing.
- Store all files in the appropriate directories for the server to access.

### 18. Testing and Documentation:

- Run the server and access it through a browser.
- Document the theory, procedure, and results with explanations and screenshots.

# Result&Discussion

## Task1: Commands & Wireshark

### Part 1: In your words, what are ping, tracert, nslookup, and telnet?

**Ping:** is a network tool that tests the possibility of sending messages between devices by transmitting small data packets. It also measures the time taken for these packets to be sent and received, helping to identify any network issues that need to be resolved.

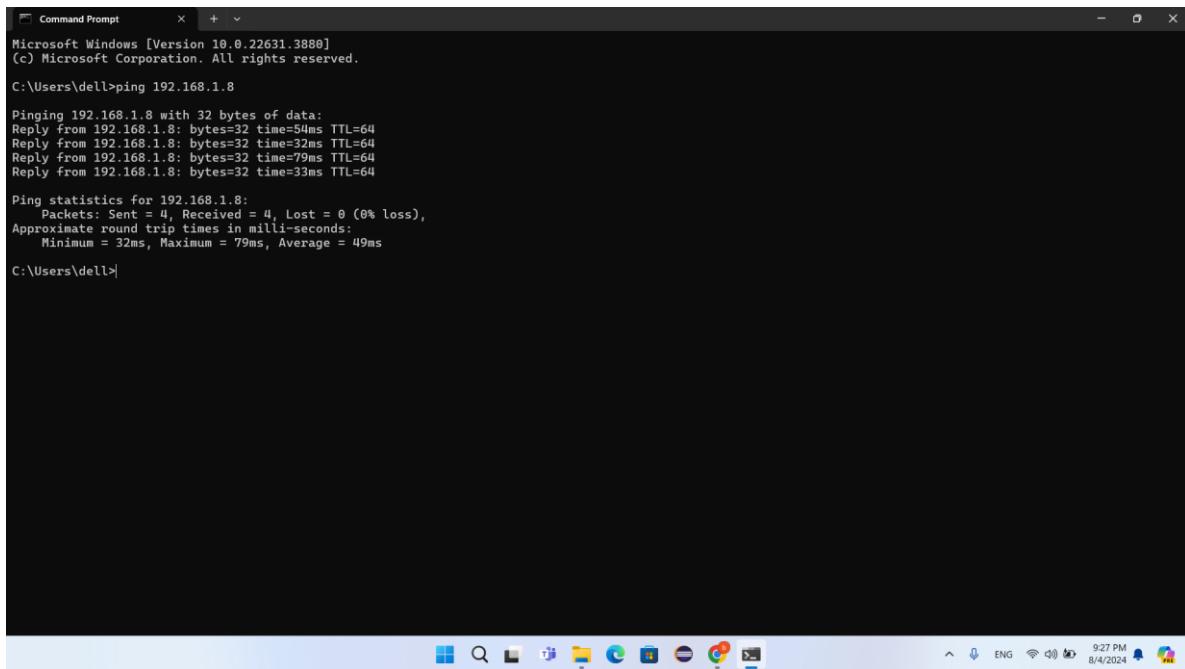
**Tracert:** It is a network tool that displays the path that data takes to reach a destination by showing all the routers it passes through. It is useful for seeing the path and checking for any network delays along the way.

**Nslookup:** It is a command line tool used in network administration to find the domain name or IP address associated with a website. It helps diagnose DNS problems by examining and retrieving DNS records.

**Telnet:** It is a protocol that allows users to connect to remote computers over the Internet or a local network, allowing them to interact with the remote device as if they were directly in front of it, issuing commands and controlling the system through a text interface.

**Part 2:** Make sure that your computer is connected to the internet and then run the following commands

**Point A:** ping a device in the same network, e.g. from a laptop to a smartphone.



```
Command Prompt
Microsoft Windows [Version 10.0.22631.3888]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dell>ping 192.168.1.8

Pinging 192.168.1.8 with 32 bytes of data:
Reply from 192.168.1.8: bytes=32 time=54ms TTL=64
Reply from 192.168.1.8: bytes=32 time=32ms TTL=64
Reply from 192.168.1.8: bytes=32 time=79ms TTL=64
Reply from 192.168.1.8: bytes=32 time=33ms TTL=64

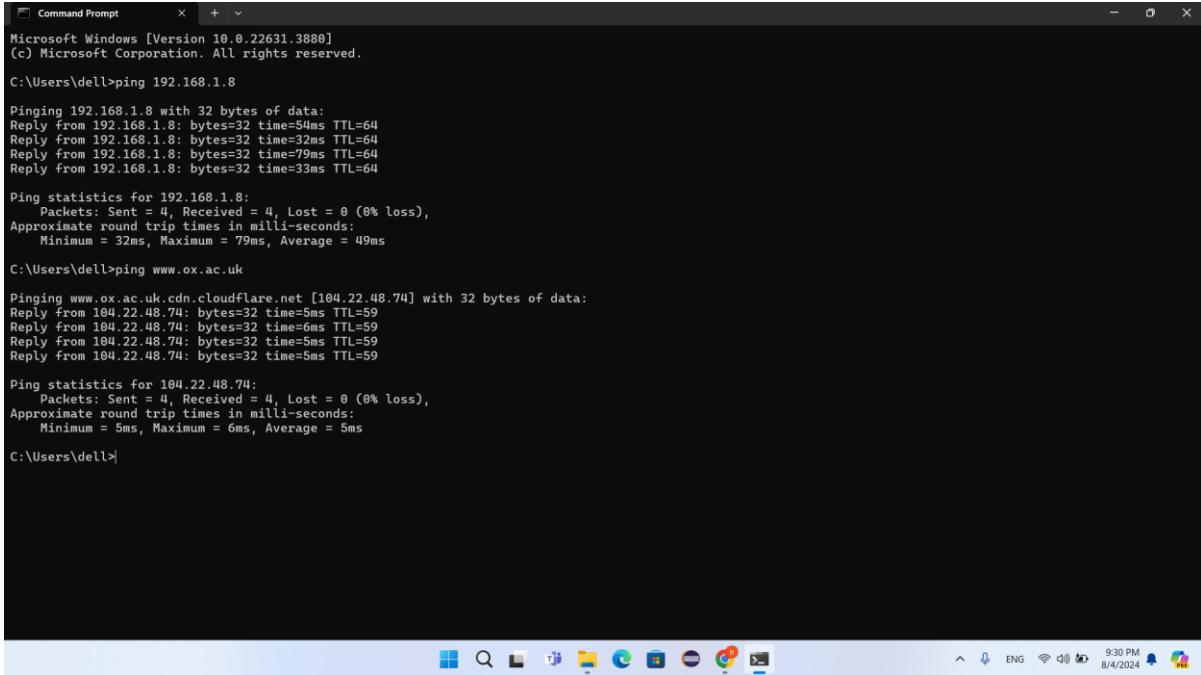
Ping statistics for 192.168.1.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 32ms, Maximum = 79ms, Average = 49ms

C:\Users\dell>
```

Figure 1: pinging my network

Here when we ping a device in the same network, my laptop sends small data packets to your smartphone to check if it's online. If the smartphone is connected, it replies, and we get the response time, showing that the device is reachable.

## Point B: ping [www.ox.ac.uk](http://www.ox.ac.uk).



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

C:\Users\dell>ping 192.168.1.8

Pinging 192.168.1.8 with 32 bytes of data:
Reply from 192.168.1.8: bytes=32 time=54ms TTL=64
Reply from 192.168.1.8: bytes=32 time=32ms TTL=64
Reply from 192.168.1.8: bytes=32 time=79ms TTL=64
Reply from 192.168.1.8: bytes=32 time=33ms TTL=64

Ping statistics for 192.168.1.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 32ms, Maximum = 79ms, Average = 49ms

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

Pinging www.ox.ac.uk.cdn.cloudflare.net [104.22.48.74] with 32 bytes of data:
Reply from 104.22.48.74: bytes=32 time=5ms TTL=59
Reply from 104.22.48.74: bytes=32 time=6ms TTL=59
Reply from 104.22.48.74: bytes=32 time=5ms TTL=59
Reply from 104.22.48.74: bytes=32 time=5ms TTL=59

Ping statistics for 104.22.48.74:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 5ms, Maximum = 6ms, Average = 5ms

C:\Users\dell>
```

Figure 2: pinging www.ox.ac.uk

When we ping `www.ox.ac.uk`, my computer sends data packets to the Oxford University website's server. The server replies if it's online, and you see the response time. This helps to check if the website is reachable and how quickly it responds.

**Point C:** From the ping results, specify the location of the server from where you got the response? Explain your answer briefly.

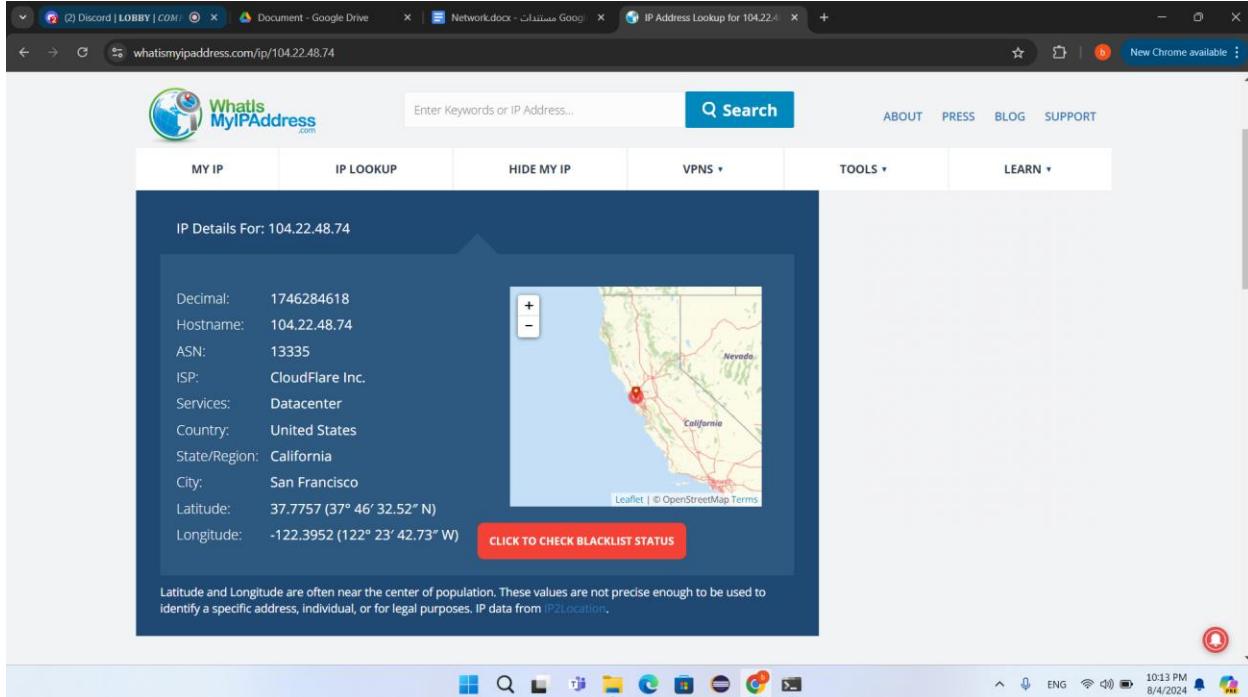
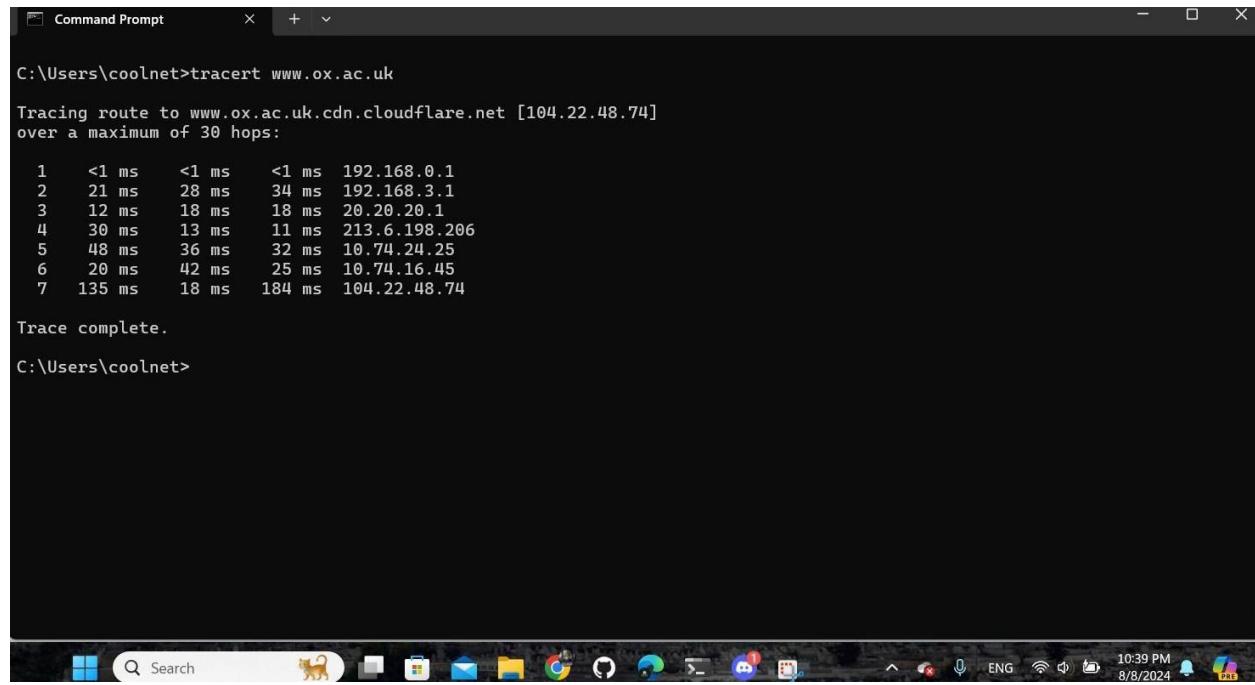


Figure 3:location server response

From the ping results alone, we typically cannot directly determine the exact physical location of the server. The ping command shows me the IP address of the server and the round-trip time (RTT), but it doesn't provide geographic information.

However, we can use the IP address obtained from the ping results and perform an IP geolocation lookup to estimate the server's location. This method isn't perfectly accurate, but it gives a general idea of where the server might be located.

## Point D: tracert [www.ox.ac.uk](http://www.ox.ac.uk).



```
Command Prompt

C:\Users\coolnet>tracert www.ox.ac.uk
Tracing route to www.ox.ac.uk.cdn.cloudflare.net [104.22.48.74]
over a maximum of 30 hops:
1 <1 ms <1 ms 192.168.0.1
2 21 ms 28 ms 34 ms 192.168.3.1
3 12 ms 18 ms 18 ms 20.20.20.1
4 30 ms 13 ms 11 ms 213.6.198.206
5 48 ms 36 ms 32 ms 10.74.24.25
6 20 ms 42 ms 25 ms 10.74.16.45
7 135 ms 18 ms 184 ms 104.22.48.74

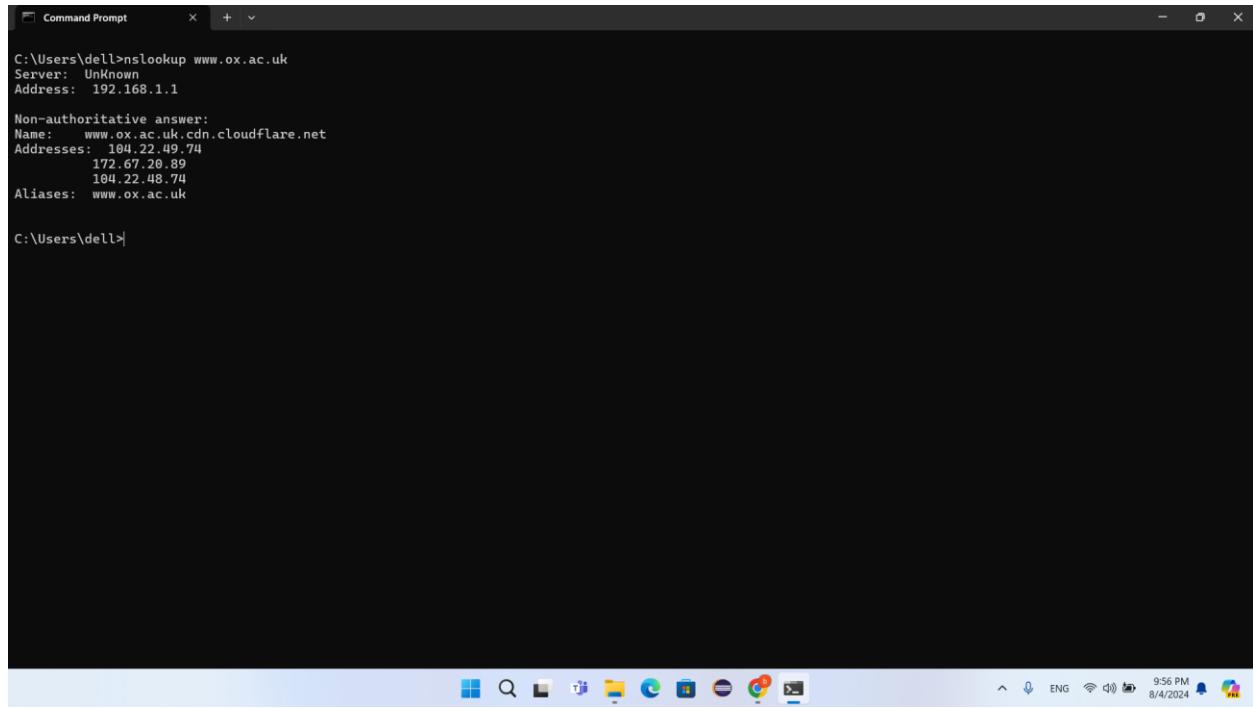
Trace complete.

C:\Users\coolnet>
```

Figure 4:tracert [www.ox.ac.uk](http://www.ox.ac.uk)

That shows me the path my data takes from my device to Oxford University's server, listing each router (hop) along the way and how long it takes to reach each one.

## Point E: nslookup *www.ox.ac.uk*.



```
C:\Users\dell>nslookup www.ox.ac.uk
Server: Unknown
Address: 192.168.1.1

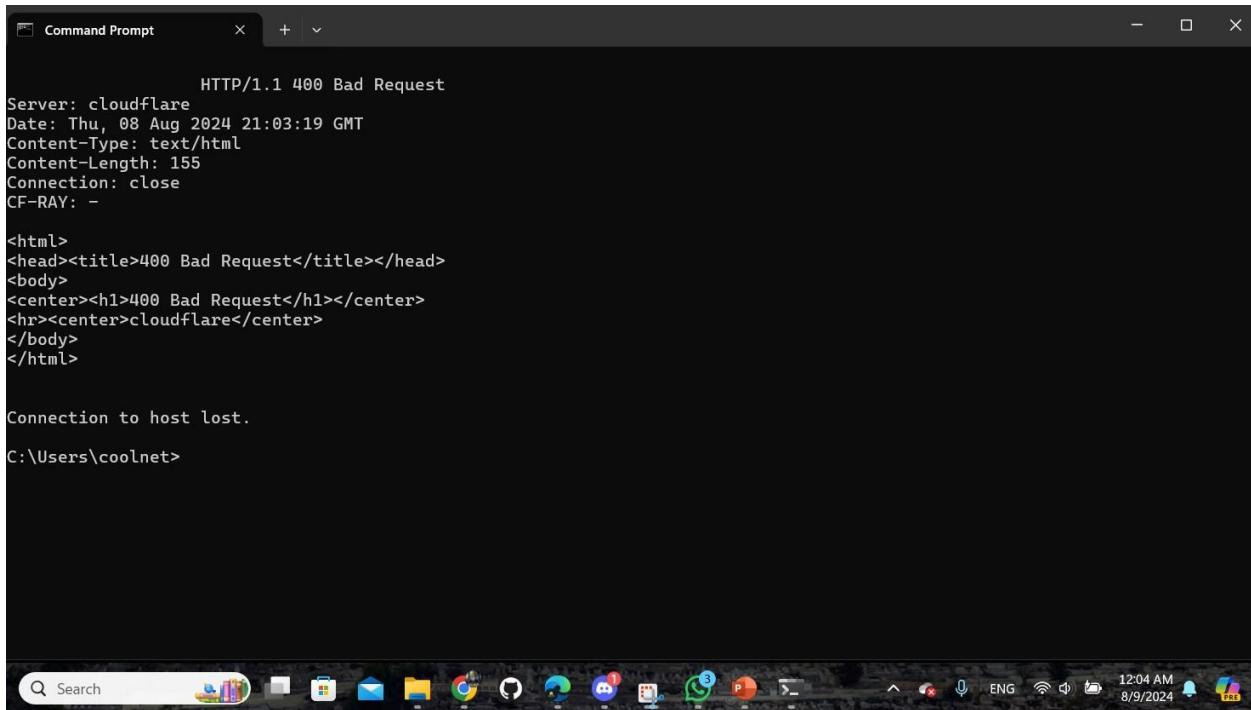
Non-authoritative answer:
Name: www.ox.ac.uk.cdn.cloudflare.net
Addresses: 104.22.49.74
          172.67.20.89
          104.22.48.74
Aliases: www.ox.ac.uk

C:\Users\dell>
```

Figure 5:nslookup *www.ox.ac.uk*

It retrieves the IP address of the Oxford University website by querying a DNS server.

## The point F: telnet [www.ox.ac.uk](http://www.ox.ac.uk).



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window displays the following text:

```
HTTP/1.1 400 Bad Request
Server: cloudflare
Date: Thu, 08 Aug 2024 21:03:19 GMT
Content-Type: text/html
Content-Length: 155
Connection: close
CF-RAY: -1000000000000000000

<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\coolnet>
```

The taskbar at the bottom shows various icons for apps like File Explorer, Google Chrome, and Microsoft Edge. The system tray indicates the date and time as 8/9/2024, 12:04 AM.

Figure 6: telnet [www.ox.ac.uk](http://www.ox.ac.uk)

The "Bad Request" error happens because Telnet sends an incomplete or improperly formatted request that the server can't understand. The server is reachable, but the request isn't correct.

**Part 3:** Give some details about autonomous system (AS) number, number of IPs, prefixes, peers, name of Tier1-ISP of [www.ox.ac.uk](http://www.ox.ac.uk).

The screenshot shows a web interface for querying an IP address. At the top, there is a banner with the text "Unlock the Intelligence Handbook: Your Guide to Cyber Threat Intelligence" and a "Get it for Free" button. Below the banner, the IP address "104.22.48.74" is displayed, with a question mark icon and the text "NO RDNS FOUND".

**Announced Prefixes**

| Country | Announced Prefix | Prefix Name   | Prefix Description | ASN     | ASN Description | ASN Name         |
|---------|------------------|---------------|--------------------|---------|-----------------|------------------|
| US      | 104.22.48.0/20   | CLOUDFLARENET | Cloudflare, Inc.   | AS13335 | CLOUDFLARENET   | Cloudflare, Inc. |
| US      | 104.16.0.0/12    | CLOUDFLARENET | Cloudflare, Inc.   | AS13335 | CLOUDFLARENET   | Cloudflare, Inc. |

**RIR Allocation Summary**

PREFIX: 104.16.0.0/12  
GEOIP COUNTRY: US  
IP ADDRESSES: 1,048,576

REGIONAL REGISTRY: ARIN  
ALLOCATION STATUS: Allocated  
ALLOCATION DATE: 28<sup>th</sup> March 2014

Figure 7: Details about tier1 ISP for [www.ox.ac.uk](http://www.ox.ac.uk)

Here we use this website to find all details we need about "[www.ox.ac.uk](http://www.ox.ac.uk)" that we need.

## Part 4: Wireshark

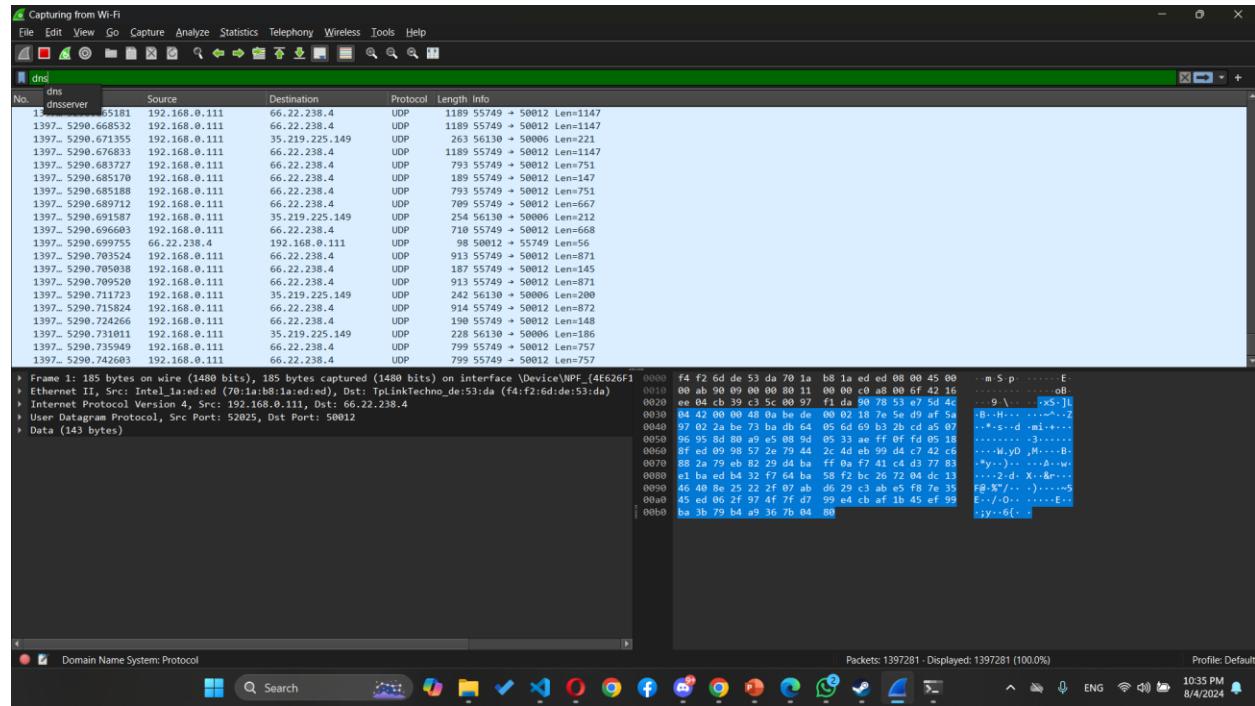


Figure 8: dns on wire shark

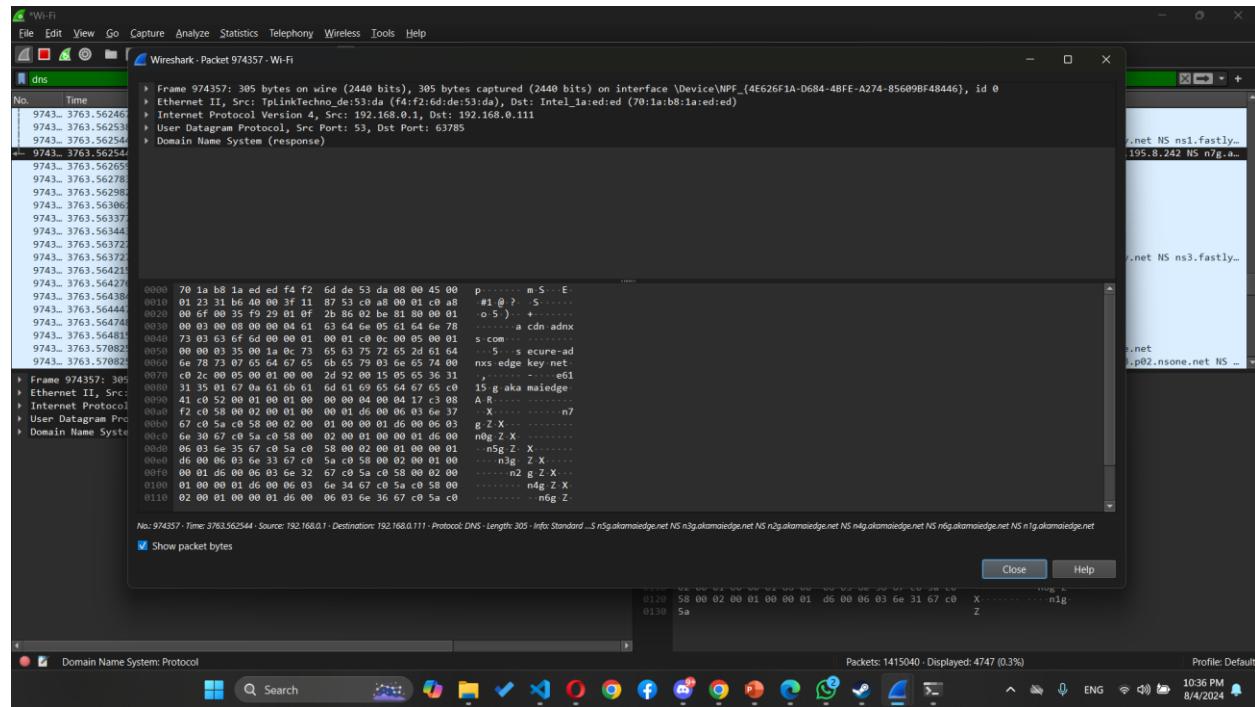
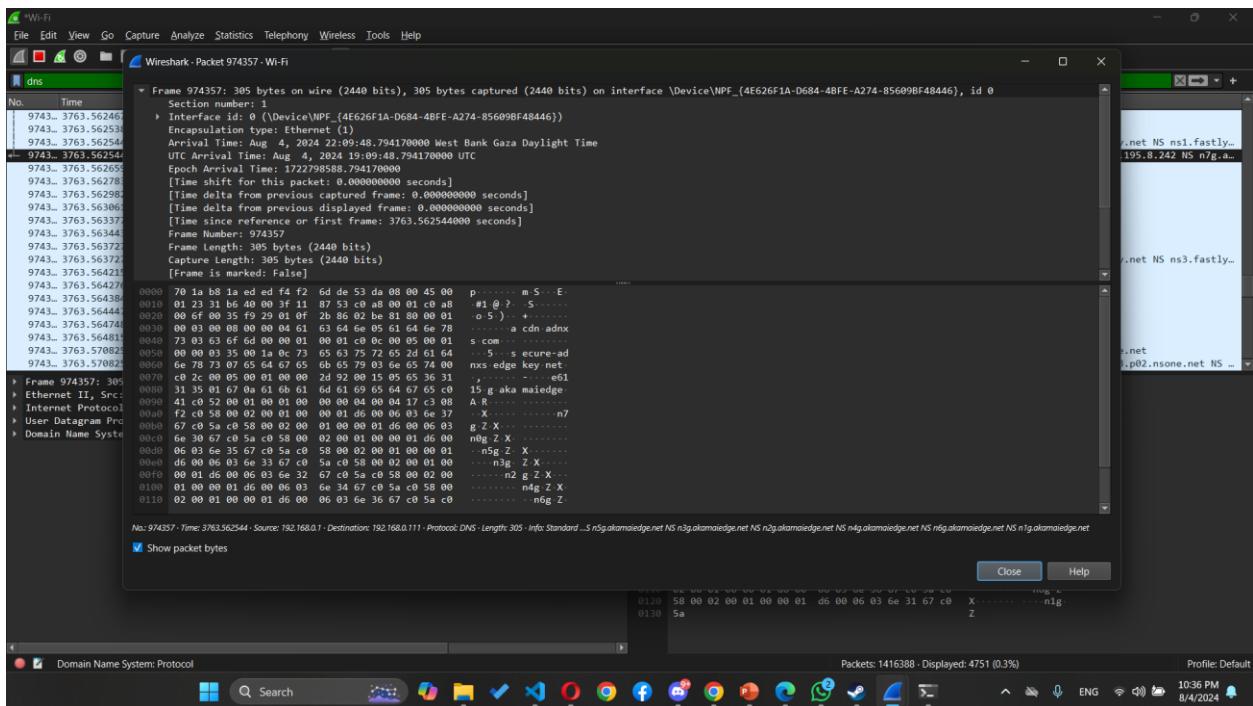
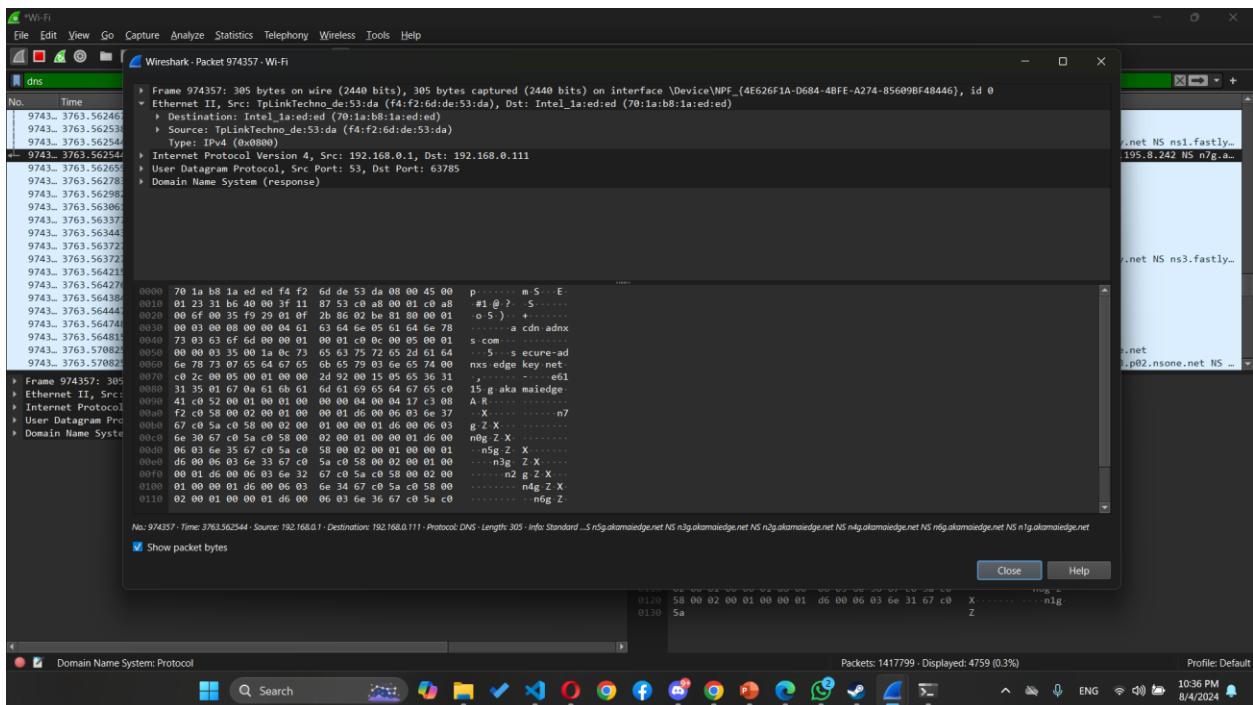


Figure 9: details for one of the packets



*Figure 10: frame details on the packet*



*Figure 11: ethernet II details*

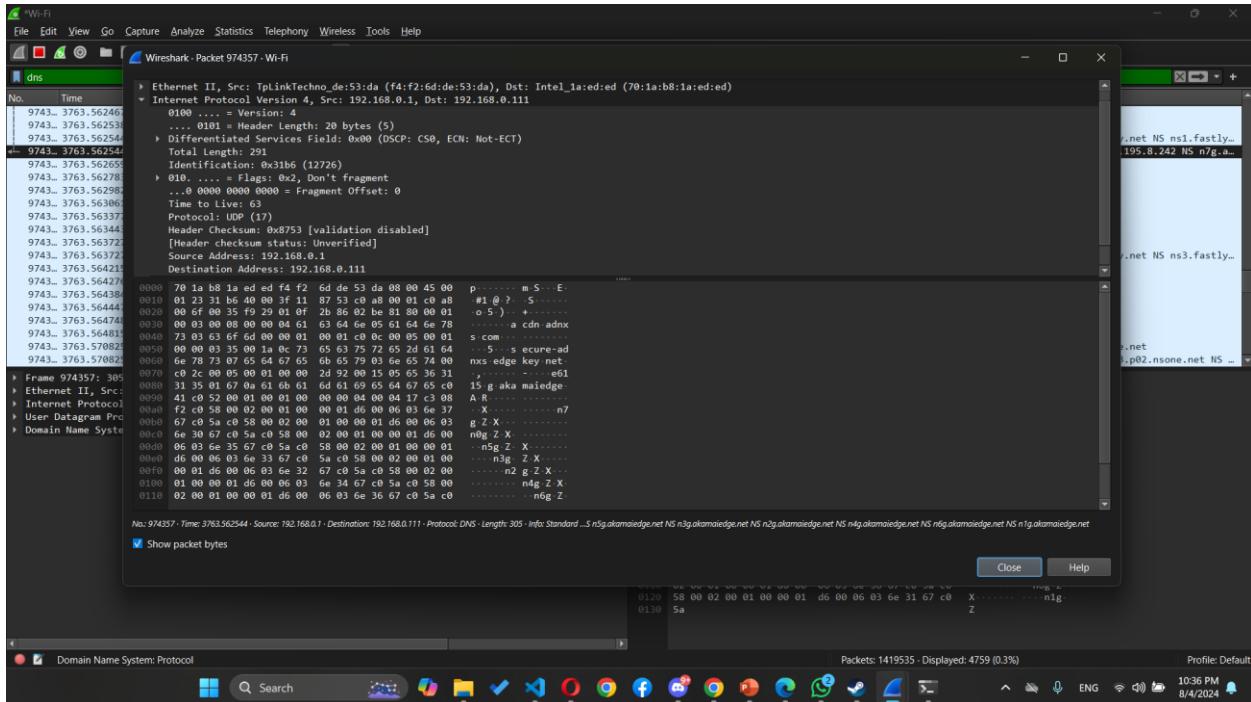


Figure 12: details for the source and destination for it

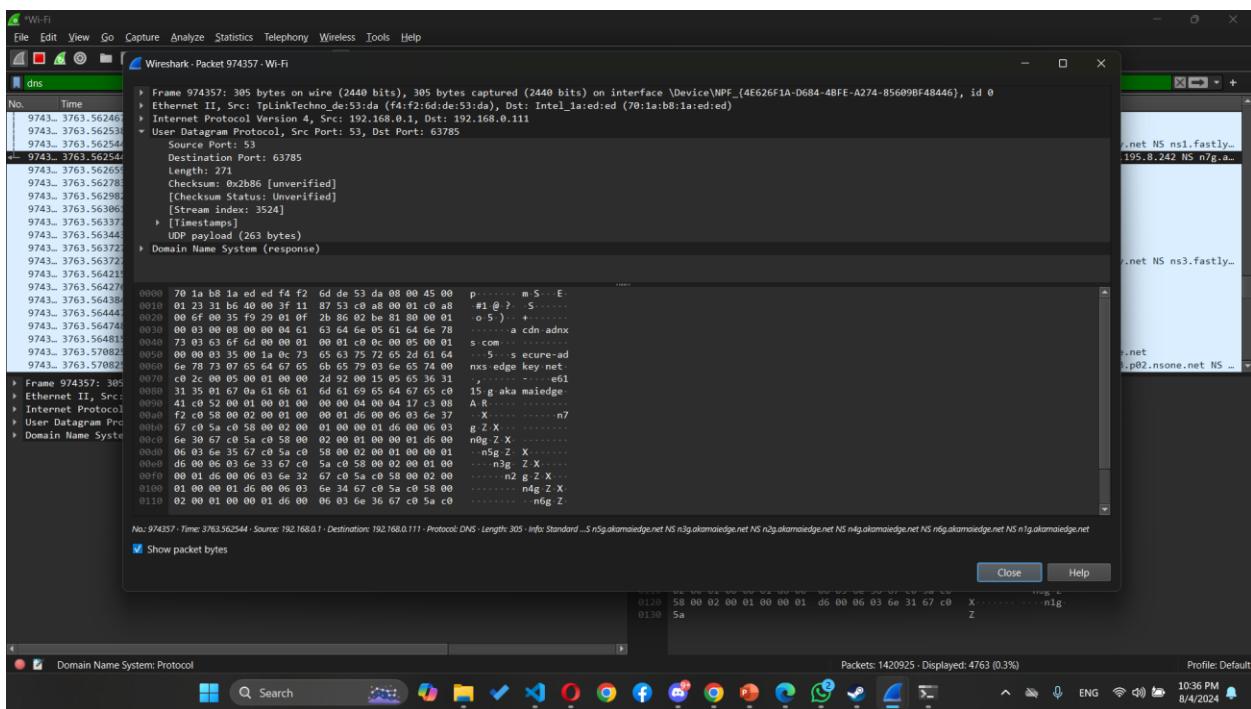


Figure 13: user datagram protocol

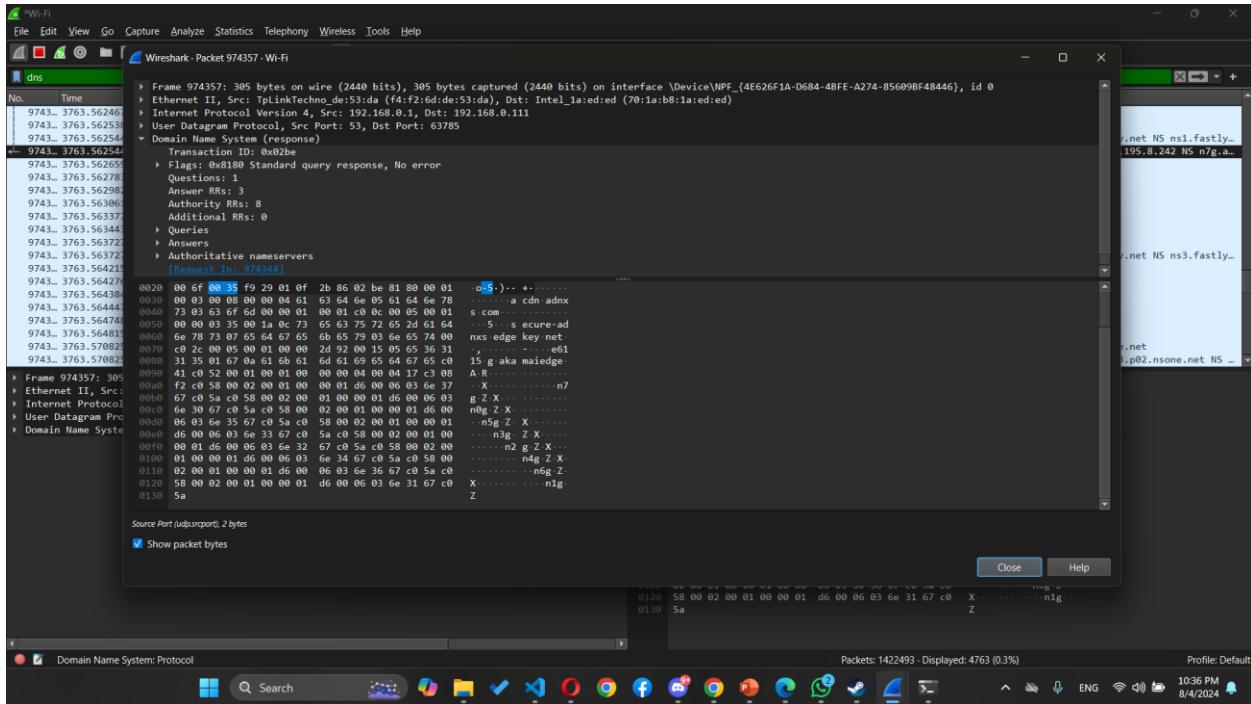


Figure 14: domain name system

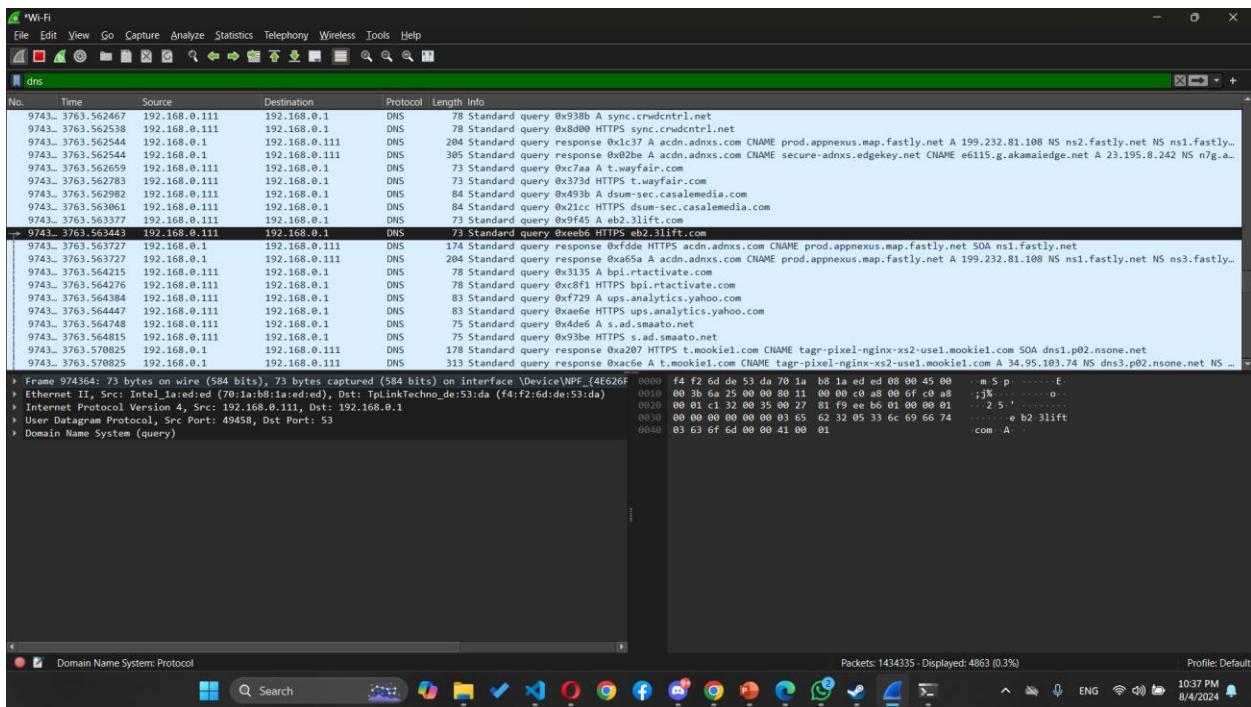


Figure 15: The packet that we show the details about it

- Capturing DNS Messages:
  1. Start Wireshark: Open Wireshark, choose your network interface, and begin capturing packets.
  2. Filter DNS Traffic: Type `dns` in the display filter to show only DNS packets.
- Examining DNS Queries:
- Query Packets: Displays DNS queries where the client requests the IP address for a domain name from the DNS server.
- Analyzing DNS Responses:
- Response Packets: Shows DNS responses containing the IP address corresponding to the requested domain.
- Detailed Packet Analysis:
- Packet Details: Click on a DNS packet to see details such as:
  - Transaction ID: Matches queries to responses.
  - Flags: Indicates if the packet is a query or response and other info.
  - Questions: Domain name requested.
  - Answers: IP address provided by the server.
- Interpreting Data:
  - Hex Data: The raw packet data is shown in hexadecimal at the bottom, with Wireshark translating it for easier understanding.
- Common DNS Operations:
  - Standard Query: Client request for a domain's IP address.
  - Response: DNS server's reply with the resolved IP address.

## Task 2:

### Part 1:

#### - The Server Code:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar has icons for file operations like Open, Save, Find, and Copy/Paste. The main editor area displays a Python script named 'server.py' with the following code:

```
C:\> Users\cooNet> Desktop> network project> task2> 1> server.py > ...
1  from socket import *
2
3  serverPort = 1183
4  serverIp = 'localhost'
5
6  vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
7
8  serverSocket = socket(AF_INET, SOCK_STREAM)
9  serverSocket.bind((serverIp, serverPort))
10 serverSocket.listen(1)
11 print("Server is on and waiting for a connection...")
12
13 while True:
14     connectionClient, clientAddress = serverSocket.accept()
15     print(f"Connected to {clientAddress}")
16
17     connectionClient.send("Enter your message: ".encode())
18     clientMessage = connectionClient.recv(1209).decode()
19     print('Message from client:', clientMessage)
20
21     newMessage = ''.join(['#' if char in vowels else char for char in clientMessage])
22
23     connectionClient.send("This is the new message: ".encode())
24     connectionClient.send(newMessage.encode())
25
26     connectionClient.close()
27     print("Connection closed.")
28
29 break
```

The bottom status bar shows the command prompt PS C:\Users\cooNet\Desktop\network project\task2\1>, the file name server.py, and the line number 29. It also displays various system icons and the date/time 8/13/2024.

Figure 16: server code

The server code simply opens a connection and waits for only one client. After that, the server receives a message from the client, converts the upper and lower case vowels to #, then sends it back to the client and stops the connection and print message about this.

**- The Client Code:**

The screenshot shows the Visual Studio Code interface. The left sidebar has icons for files, folders, and other project-related items. The main editor area contains the following Python code:

```
client1.py
from socket import *
serverIp = 'localhost'
serverPort = 1183
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverIp, serverPort))
receivedData = clientSocket.recv(1200).decode()
print(receivedData)
sentData = input(' ')
clientSocket.send(sentData.encode())
print(clientSocket.recv(1200).decode())
print(clientSocket.recv(1200).decode())
clientSocket.close()
```

The terminal tab at the bottom shows the command run: PS C:\Users\coolnet\Desktop\network project\task2> & 'c:\Users\coolnet\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\coolnet\.vscode\extensions\ms-python.python.debug\2024.10.0-win32-x64\bundled\libs\debugpy\adapter'...'\debugpy\launcher' "51375" ... 'c:\Users\coolnet\Desktop\network project\task2\1\client1.py'. It also displays the message "Enter your message:" followed by a cursor. The status bar at the bottom right shows the file is 3.9.13 64-bit, the date is 8/13/2024, and the time is 11:30 PM.

Figure 17: client code

The client code simply sends a connection request to the server. After the server accepts the connection, the client sends a message to the server, which the server returns in the new format, prints it, and then disconnects from the server.

- The First Server Output Message:

The screenshot shows the Visual Studio Code interface. The top bar has tabs for 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The status bar at the bottom shows 'Ln 3, Col 18' and 'Python 3.9.13 64-bit'. The main area has two tabs open: 'client.py' and 'server.py'. The 'server.py' tab contains the following Python code:

```
C:\> Users > coolnet > Desktop > network project > task2 > 1 > server.py > ...
1   from socket import *
2
3   serverPort = 1183
4   serverIp = 'localhost'
5
6   vowels = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']
7
8   serverSocket = socket(AF_INET, SOCK_STREAM)
9   serverSocket.bind((serverIp, serverPort))
10  serverSocket.listen(1)
11  print("Server is on and waiting for a connection...")
12
13  while True:
14      connectionClient, clientAddress = serverSocket.accept()
15      print(f"Connected to {clientAddress}")
16
17      connectionClient.send("Enter your message: ".encode())
18      clientMessage = connectionClient.recv(1200).decode()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

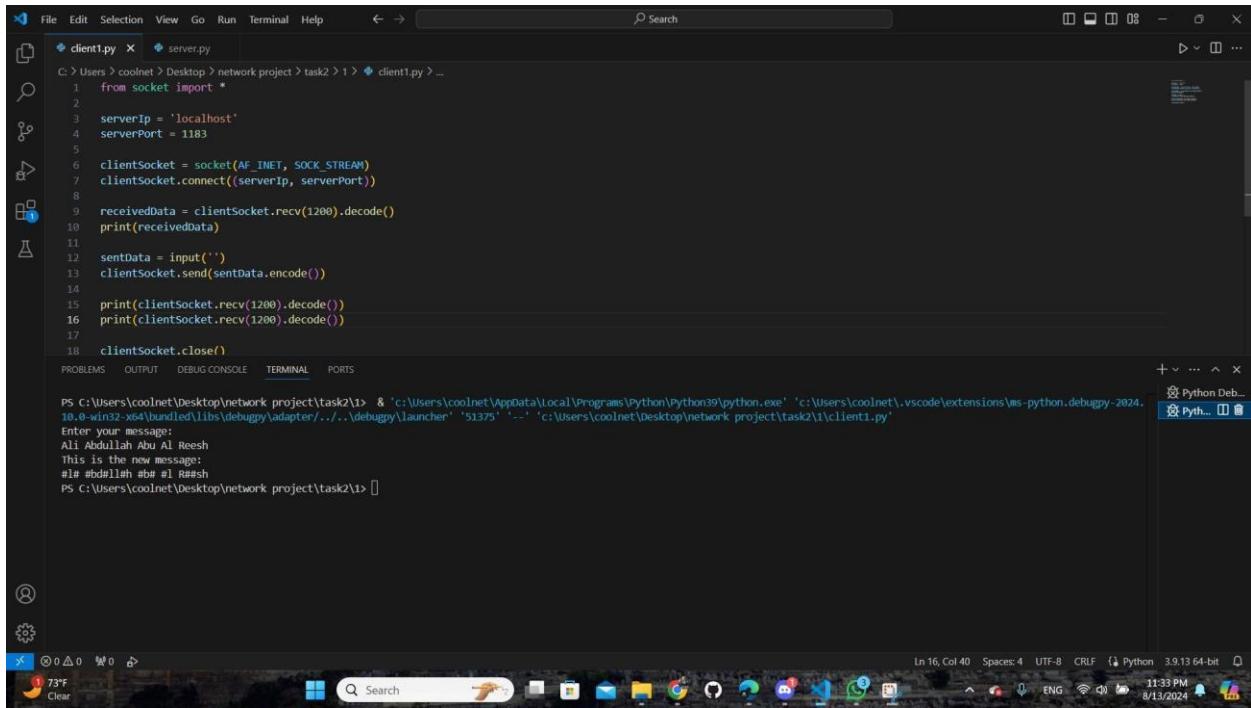
The terminal below the code shows the command-line interface with several identical entries for 'server.py' and then the server's output:

```
PS C:\Users\coolnet\Desktop\network project\task2\1> & c:\Users\coolnet\AppData\Local\Programs\Python\Python39\python.exe "c:\Users\coolnet\.vscode\extensions\ms-python.python.debug-2024.10.0-win32-x64\bundled\libs\debugpy\adapter/../debugpy\launcher" 51368 -- -- c:\Users\coolnet\Desktop\network project\task2\1\server.py
Server is on and waiting for a connection...
Connected to ('127.0.0.1', 51378)
```

Figure 18: first server output message

The server prints a message that it is up and running and is waiting for a client to connect to it (one client, no more). After connecting to that client, it prints the client's IP and its port to confirm that it is connected.

**- The Client Output Message:**



The screenshot shows the Visual Studio Code interface. On the left, there are two tabs: 'client1.py' and 'server.py'. The 'client1.py' tab is active, displaying the following Python code:

```
1 from socket import *
2
3 serverIp = 'localhost'
4 serverPort = 1183
5
6 clientSocket = socket(AF_INET, SOCK_STREAM)
7 clientSocket.connect((serverIp, serverPort))
8
9 receivedData = clientSocket.recv(1200).decode()
10 print(receivedData)
11
12 sentData = input('')
13 clientSocket.send(sentData.encode())
14
15 print(clientSocket.recv(1200).decode())
16 print(clientSocket.recv(1200).decode())
17
18 clientSocket.close()
```

Below the code editor is a terminal window showing the execution of the script and its output:

```
PS C:\Users\coolnet\Desktop\network project\task2\b> & "c:\Users\coolnet\AppData\Local\Programs\Python\Python39\python.exe" 'c:\Users\coolnet\.vscode\extensions\ms-python.debugpy-2024.10.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '51375' '--' 'c:\Users\coolnet\Desktop\network project\task2\b\client1.py'
Enter your message:
Ali Abdullah Abu Al Reesh
This is the new message:
#I# #bdellah #b# #I# #Reesh
PS C:\Users\coolnet\Desktop\network project\task2\b>
```

The terminal also displays system status icons at the bottom.

Figure 19: Client output message

After connecting to the server, the client is asked to enter a sentence to be sent to the server, which in turn converts the vowels to # and then returns it to the client, and the final sentence is printed as we see above.

*- The Second Server Output Message:*

The screenshot shows a Windows desktop environment. In the center is a terminal window titled 'Terminal' from the VS Code interface. The window displays the following text:

```
C:\> Users > coolnet > Desktop > network project > task2 > 1 > client1.py > ...
1   from socket import *
2
3   serverIp = 'localhost'
4   serverPort = 1183
5
6   clientSocket = socket(AF_INET, SOCK_STREAM)
7   clientSocket.connect((serverIp, serverPort))
8
9   receivedData = clientSocket.recv(1200).decode()
10  print(receivedData)
11
12  sendData = input('')
13  clientSocket.send(sendData.encode())
14
15  print(clientSocket.recv(1200).decode())
16  print(clientSocket.recv(1200).decode())
17
18  clientSocket.close()

PS C:\Users\coolnet\Desktop\network project\task2\1> & 'c:\Users\coolnet\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\coolnet\vscode\extensions\ms-python.python.debugpy-2024.0.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '51368' '--' 'c:\Users\coolnet\Desktop\network project\task2\1\server.py'
Server is on and waiting for a connection...
Connected to ('127.0.0.1', 51378)
Message from client: Ali Abdullah Abu Al Reesh
Connection closed.

PS C:\Users\coolnet\Desktop\network project\task2\1>
```

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The status bar at the bottom shows 'Ln 16, Col 40' and 'Python 3.9.13 64-bit'. The system tray icons include battery level (73%), a small orange icon, a search bar, and other standard Windows icons.

Figure 20: Second server output

In this image, the server shows the sentence it received from the client by printing it, then performs the necessary operations on it, sends it, then stops the connection and prints the sentence "Connection closed."

## Part 2:

### - The Server Code:

The screenshot shows a Python code editor interface with the following details:

- File Explorer:** Shows three files: client1.py, client2.py, and server.py.
- Code Editor:** Displays the contents of server.py. The code is a UDP server that binds to port 1833 and listens for messages from clients. It uses a dictionary to track messages from each client and handles them sequentially.
- Terminal:** Shows the output "Server is on and ready to receive messages..."
- Bottom Status Bar:** Includes icons for battery (73°F), search, file operations, and system status (Wi-Fi, battery, time 11:21 PM, date 8/13/2024).

```
from socket import *
import threading

# Define server parameters
serverPort = 1833
serverIp = 'localhost'

# Create UDP socket
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind((serverIp, serverPort))
print("Server is on and ready to receive messages...")

# Dictionary to keep track of the last message from each client
clients = {}

def handle_client(clientMessage, clientAddress):
    clientNumber = clients.get(clientAddress, len(clients) + 1)
    clients[clientAddress] = clientNumber

    print(f"\nMessage from client {clientNumber}:")
    print(clientMessage.decode())

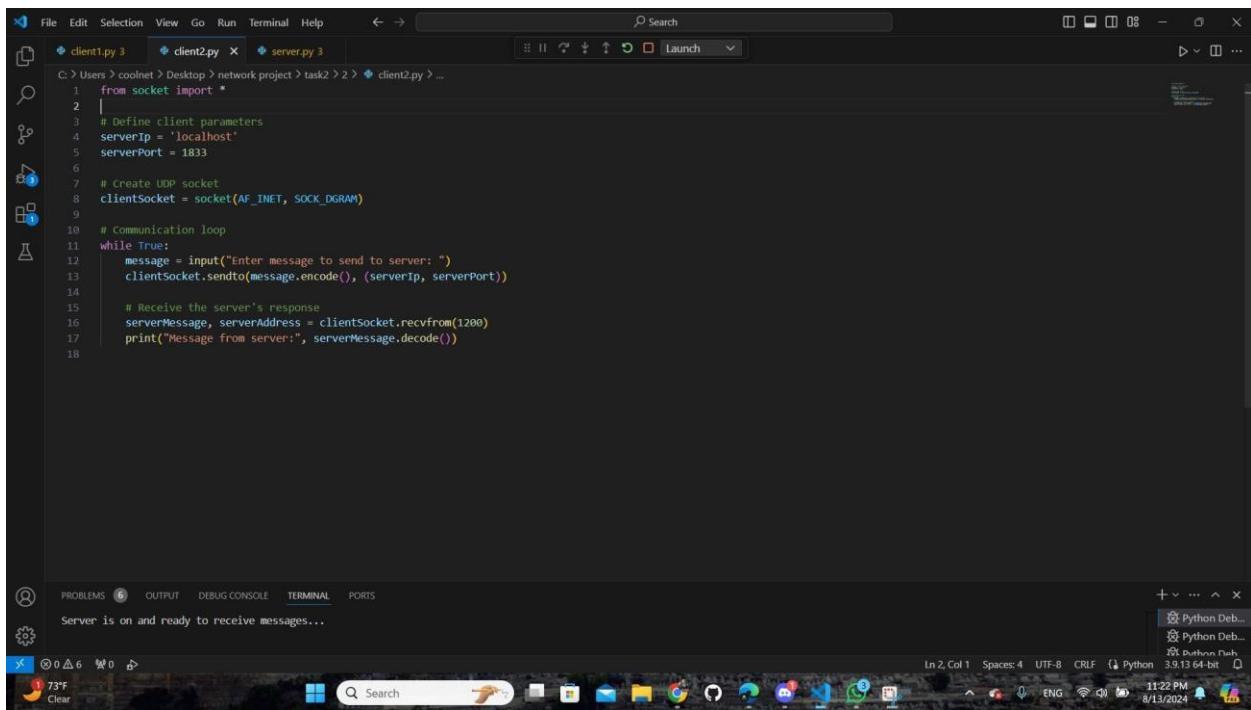
    # Prompt server user to send a message back to the client
    messageToClient = input("Enter your message to client " + str(clientNumber) + ": ")
    serversocket.sendto(messageToClient.encode(), clientAddress)

while True:
    clientMessage, clientAddress = serverSocket.recvfrom(1200)
    threading.Thread(target=handle_client, args=(clientMessage, clientAddress)).start()
```

Figure 21: server code for part2

This image shows the server code, which is different from the previous task, because in this task the server will open a connection with the rest of the clients, and the server can respond to messages (it works as a client, not a server).

**- The Client1 && Client2 Code:**

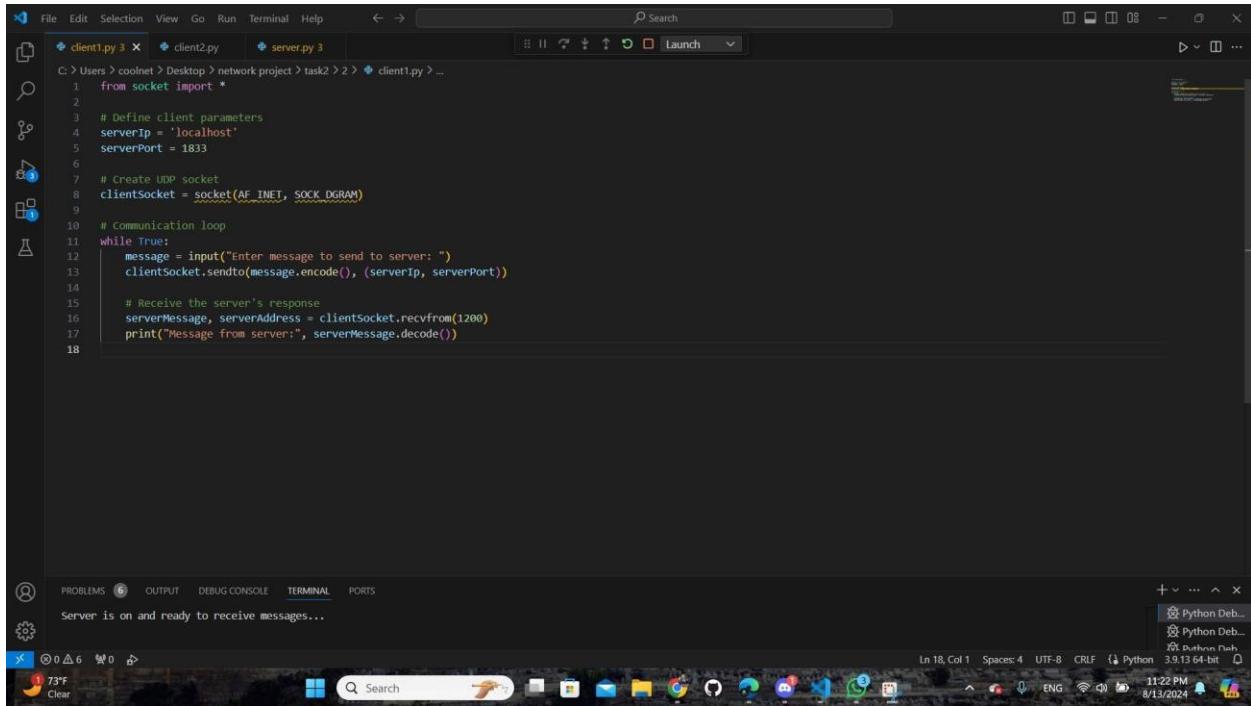


The screenshot shows a code editor window with three tabs: client1.py, client2.py (which is the active tab), and server.py. The code in client2.py is as follows:

```
C:\> Users > coolnet > Desktop > network project > task2 > 2 > client2.py > ...
1  from socket import *
2
3  # Define client parameters
4  serverIp = 'localhost'
5  serverPort = 1833
6
7  # Create UDP socket
8  clientSocket = socket(AF_INET, SOCK_DGRAM)
9
10 # Communication loop
11 while True:
12     message = input("Enter message to send to server: ")
13     clientSocket.sendto(message.encode(), (serverIp, serverPort))
14
15     # Receive the server's response
16     serverMessage, serverAddress = clientSocket.recvfrom(1280)
17     print("Message from server:", serverMessage.decode())
18
```

The terminal below the code editor shows the message "Server is on and ready to receive messages...".

Figure 22:client 2 code



The screenshot shows a code editor window with three tabs: client1.py (active), client2.py, and server.py. The code in client1.py is as follows:

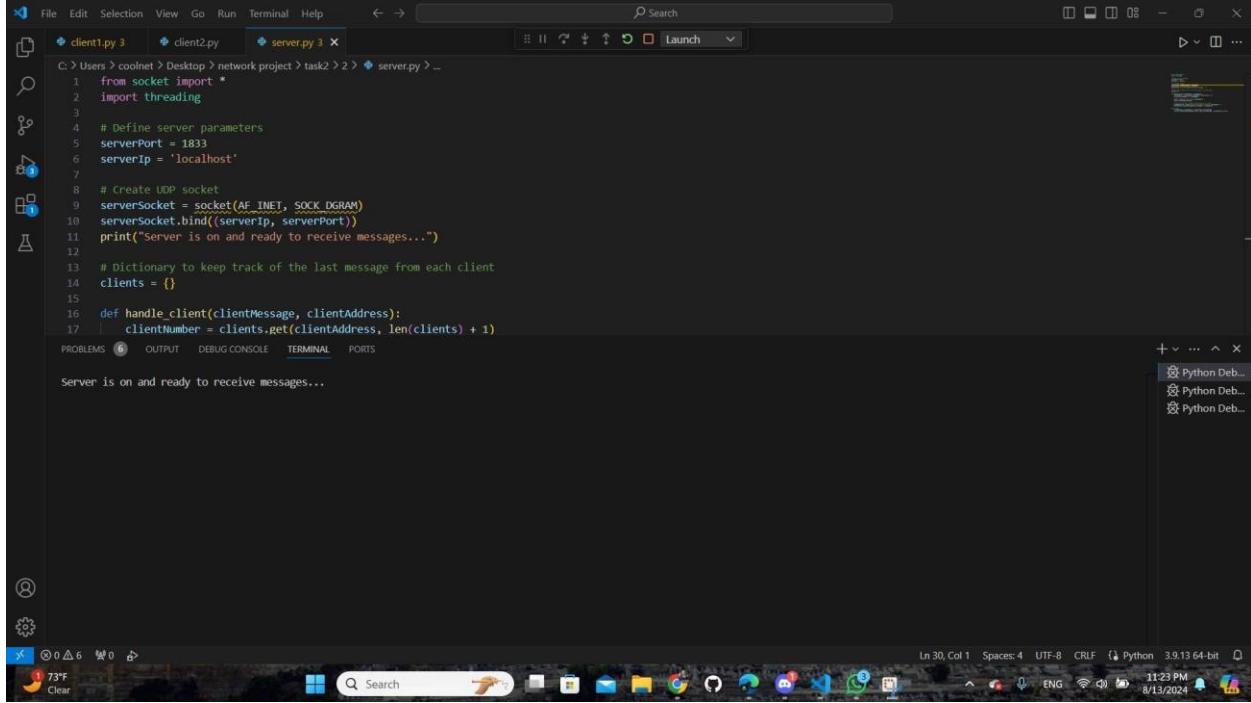
```
C:\> Users > coolnet > Desktop > network project > task2 > 2 > client1.py > ...
1  from socket import *
2
3  # Define client parameters
4  serverIp = 'localhost'
5  serverPort = 1833
6
7  # Create UDP socket
8  clientSocket = socket(AF_INET, SOCK_DGRAM)
9
10 # Communication loop
11 while True:
12     message = input("Enter message to send to server: ")
13     clientSocket.sendto(message.encode(), (serverIp, serverPort))
14
15     # Receive the server's response
16     serverMessage, serverAddress = clientSocket.recvfrom(1280)
17     print("Message from server:", serverMessage.decode())
18
```

The terminal below the code editor shows the message "Server is on and ready to receive messages...".

Figure 23:client 1 code

At the same time, the client, after connecting to the server, can send and receive (it works as a server and a client at the same time), and the client cannot send to other clients.

**- The First Server Output Message:**



The screenshot shows a code editor interface with several tabs open. The active tab is 'server.py' (version 3). The code is a UDP server setup:

```
C:\> Users > coolnet > Desktop > network project > task2 > 2 > server.py > ...
1  from socket import *
2  import threading
3
4  # Define server parameters
5  serverPort = 1833
6  serverIp = 'localhost'
7
8  # Create UDP socket
9  serverSocket = socket(AF_INET, SOCK_DGRAM)
10 serverSocket.bind((serverIp, serverPort))
11 print("Server is on and ready to receive messages...")
12
13 # Dictionary to keep track of the last message from each client
14 clients = {}
15
16 def handle_client(clientMessage, clientAddress):
17     clientNumber = clients.get(clientAddress, len(clients) + 1)
```

In the terminal tab, the output is:

```
Server is on and ready to receive messages...
```

The status bar at the bottom indicates the file is 3.9.13 64-bit, and the system status shows 73°F, ENG, 11:23 PM, 8/13/2024.

After the server is started and configured to connect to more than one client, it prints a message that it is ready to connect with clients.

### - The Client Output:

The screenshot shows a terminal window with three tabs: client1.py, client2.py, and server.py. The client1.py tab is active, displaying Python code for a UDP client. The terminal output shows the client sending a message to the server and receiving a response. The system tray at the bottom indicates it's 73°F and shows other icons.

```
C:\> Users > coolnet > Desktop > network project > task2 > 2 > client1.py > ...
1 from socket import *
2
3 # Define client parameters
4 serverIp = 'localhost'
5 serverPort = 1833
6
7 # Create UDP socket
8 clientSocket = socket(AF_INET, SOCK_DGRAM)
9
10 # Communication loop
11 while True:
12
13     # Enter message to send to server: hi, my name is mouat
14     Enter message to send to server: hi, my name is mouat
15     Message from server: hello, and my is baha
16     Enter message to send to server: [ ]
```

Figure 24:client 1 message

The screenshot shows a terminal window with three tabs: client1.py, client2.py, and server.py. The client2.py tab is active, displaying Python code for a UDP client. The terminal output shows the client sending a message to the server and receiving a response. The system tray at the bottom indicates it's 73°F and shows other icons.

```
C:\> Users > coolnet > Desktop > network project > task2 > 2 > client2.py > ...
1 from socket import *
2
3 # Define client parameters
4 serverIp = 'localhost'
5 serverPort = 1833
6
7 # Create UDP socket
8 clientSocket = socket(AF_INET, SOCK_DGRAM)
9
10 # Communication loop
11 while True:
12
13     # Enter message to send to server: my name is zaid mousa
14     Enter message to send to server: my name is zaid mousa
15     Message from server: hello zaid, how do you do?
16     Enter message to send to server: [ ]
```

Figure 25: client 2 message

After the client connects to the server, it requests the server's message, and when it arrives there, the server responds to the message (exchanging messages like Messenger, etc.).

#### **- The Second Server Output Message:**

The screenshot shows a Microsoft Windows desktop with a dark theme. In the center is a Microsoft Visual Studio Code (VS Code) window. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar contains icons for file operations like Open, Save, Find, and others. The main editor area has three tabs: client1.py 3, client2.py x, and server.py 3. The client2.py tab is active, displaying Python code for a UDP client. The code imports socket, defines client parameters (serverIp='localhost', serverPort=1833), creates a UDP socket, and enters a loop to receive messages from the server. Below the editor are tabs for PROBLEMS (with 6 errors), OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The terminal pane shows the message "Server is on and ready to receive messages...". The status bar at the bottom shows file information (ln 2, col 1), encoding (UTF-8), line endings (CRLF), language (Python), version (3.9.13), and bit width (64-bit). The taskbar at the bottom includes icons for File Explorer, Task View, Taskbar settings, Start button, Search, and several pinned application icons. The system tray shows the date (8/13/2024), time (11:24 PM), battery level (75%), and connectivity status.

```
C:\> Users > coolnet > Desktop > network project > task2 > 2 > client2.py ...
1  from socket import *
2
3  # Define client parameters
4  serverIp = 'localhost'
5  serverPort = 1833
6
7  # Create UDP socket
8  clientSocket = socket(AF_INET, SOCK_DGRAM)
9
10 # Communication loop
11 while True:
12     message = input("Enter message to send to server: ")
```

*Figure 26:server receive client 1 message*

*Figure 27:server receive the client 2 message*

After the message arrives from the client, it is printed who is the client who sent the message and what is the message sent, and he can reply to the sent message, but the client who sent the message cannot resend it unless the server replies to him.



## Task 3:

1-

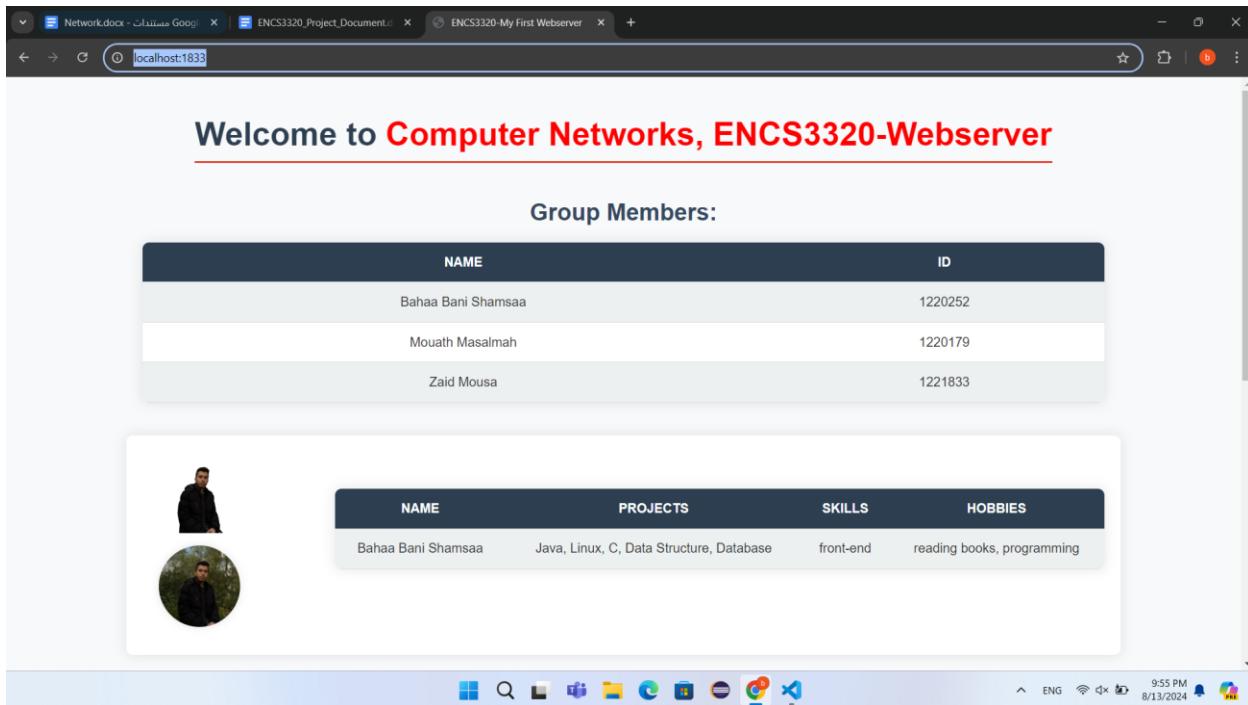


Figure 28:request /

As we see here when I type link <http://localhost:1833/> it goes to the english page

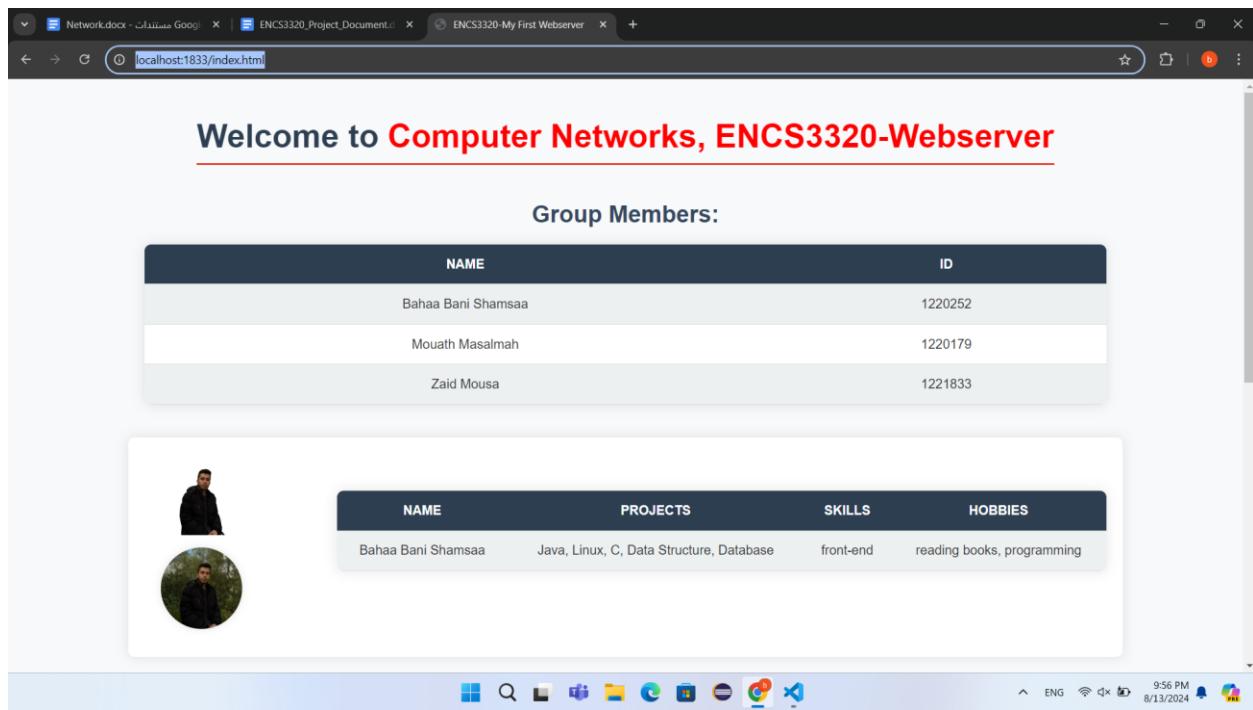


Figure 29: index.html request

As we see here when I type link <http://localhost:1833/index.html> it goes to the english page

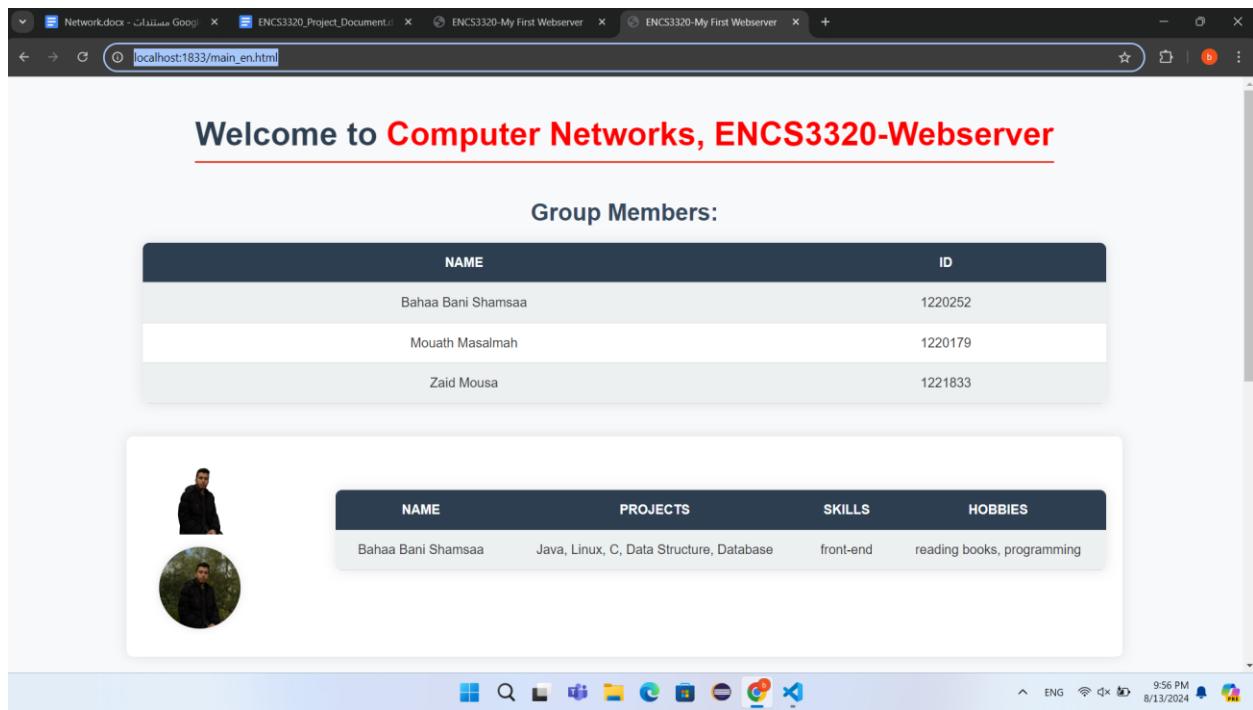


Figure 30:main\_en.html request

As we see here when I type link [http://localhost:1833/main\\_en.html](http://localhost:1833/main_en.html) it goes to the english page

a.

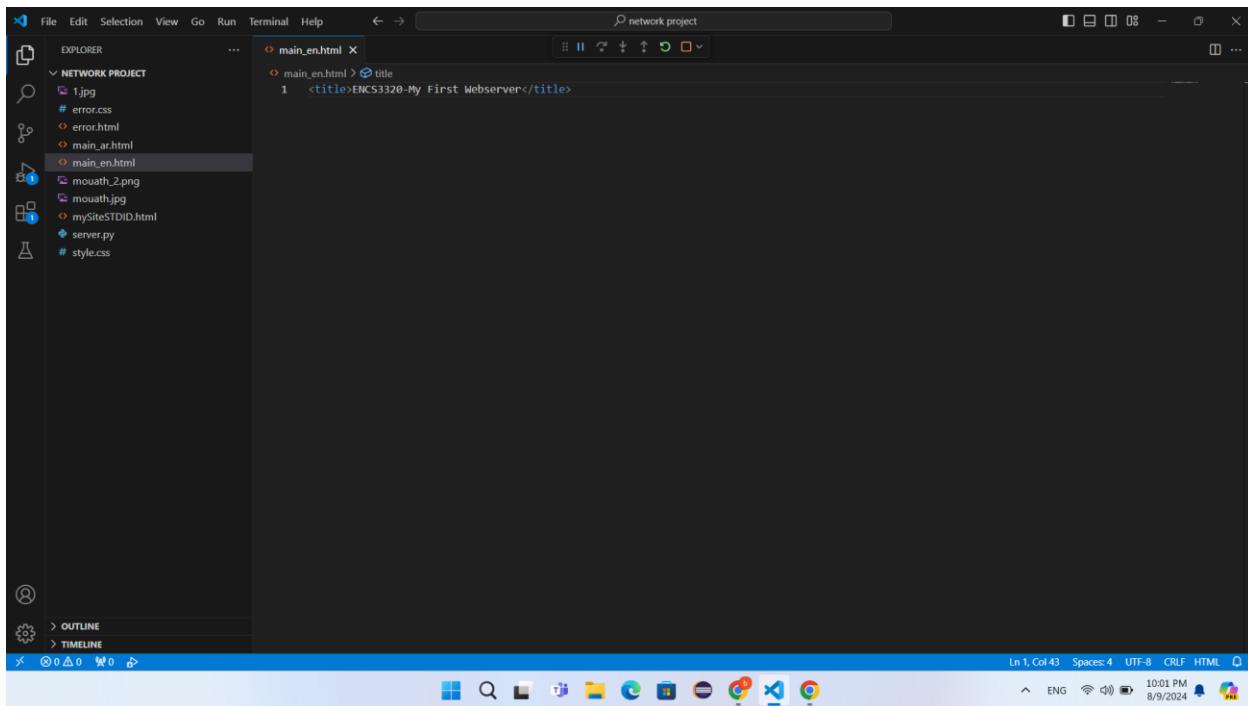


Figure 31:title of the page

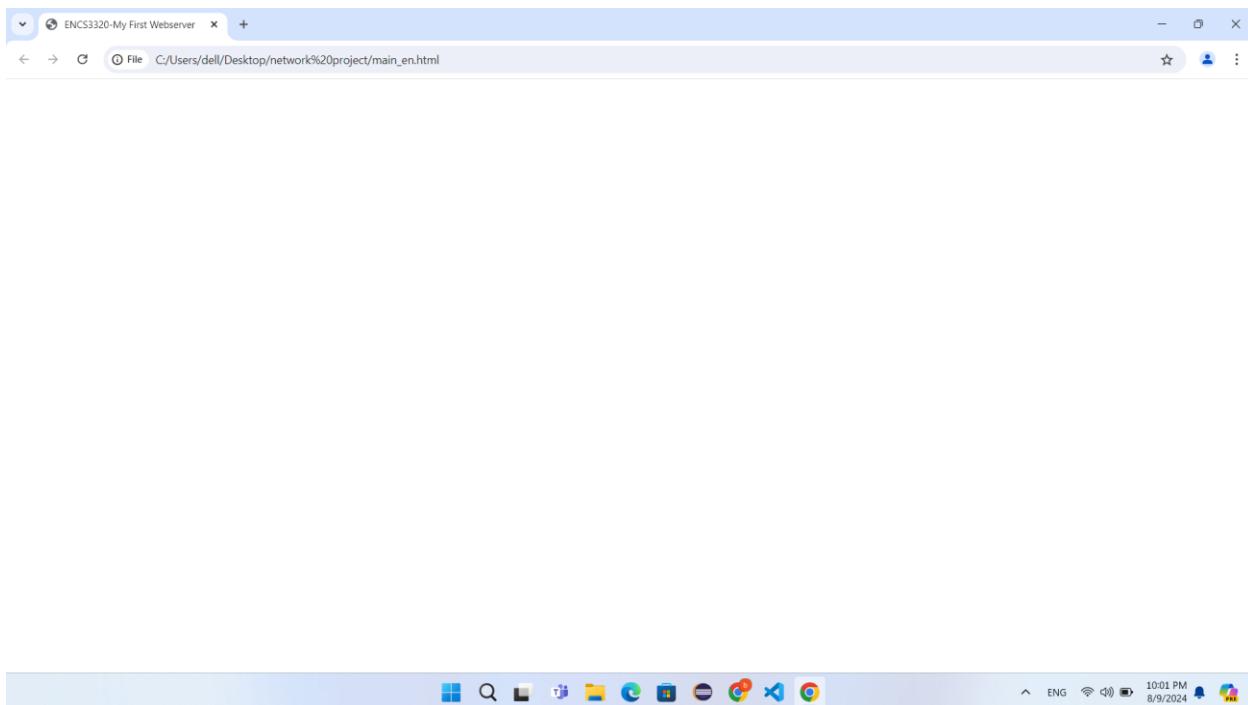


Figure 32:browser title

Here we put the title ENCS-My First Webserver

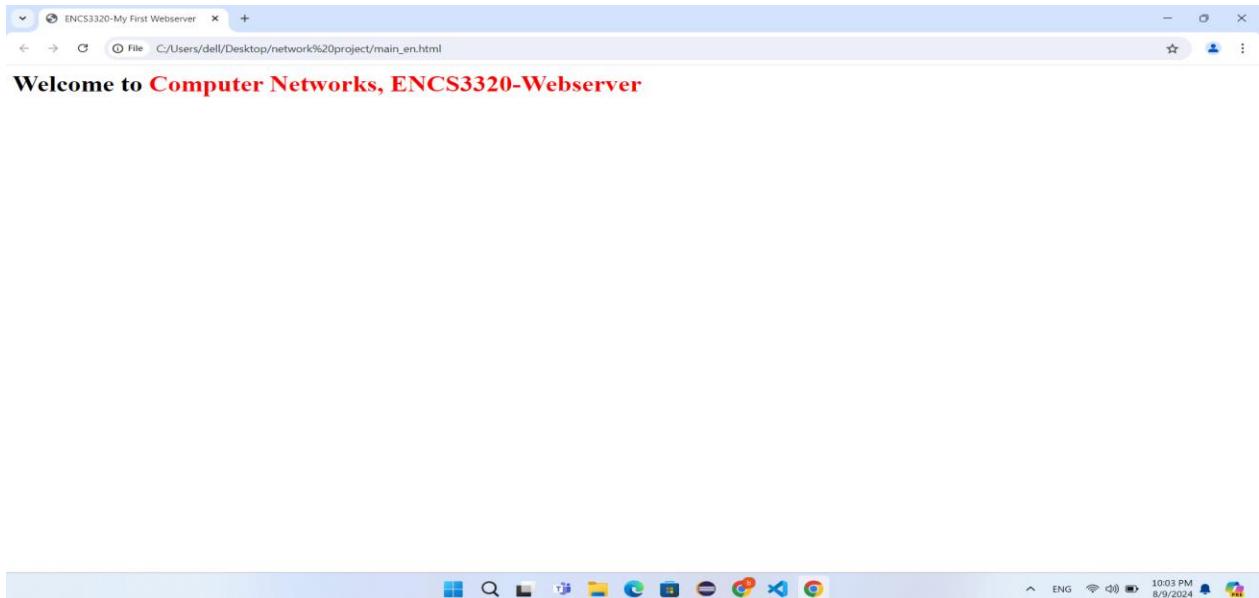
b.

The screenshot shows a code editor interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** network project
- Toolbar:** Includes icons for back, forward, search, and file operations.
- Explorer:** Shows the project structure:
  - NETWORK PROJECT
    - 1.jpg
    - # error.css
    - error.html
    - main\_ar.html
    - main\_en.html**
    - mouath.2.png
    - mouath.jpg
    - mySiteSTID.html
    - server.py
    - # style.css
- Code Editor:** The main window displays the content of `main_en.html`.

```
<title>ENCS3320-My First Webserver</title>
<h1>Welcome to<span style="color: red;"> Computer Networks, ENCS3320-Webserver</span></h1>
```
- Bottom Status Bar:** Ln 3, Col 1, Spaces: 4, UTF-8, CRLF, HTML.
- Bottom Icons:** Includes icons for file operations, search, and browser tabs.
- Sidebar:** Includes sections for OUTLINE and TIMELINE.
- Bottom Navigation:** Includes icons for file operations, search, and browser tabs.

*Figure 33:welcome message*



*Figure 34: browser welcome*

Here I use span to make the words with different colors

c.

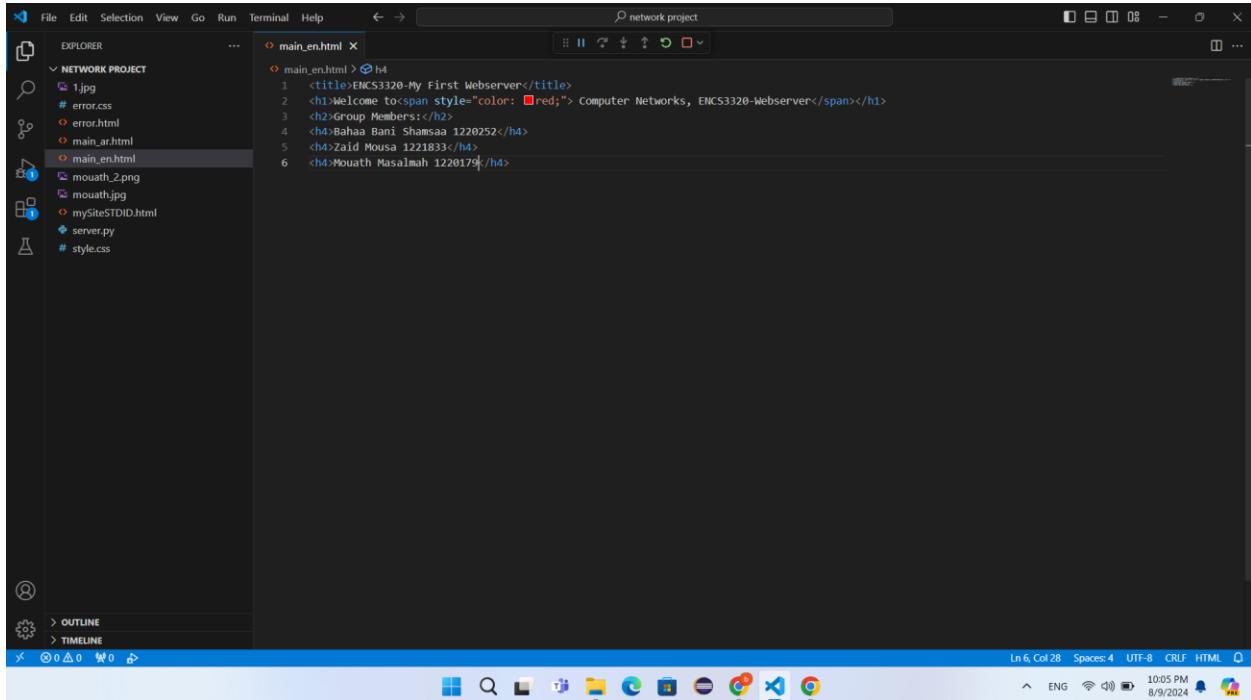


Figure 35:group members

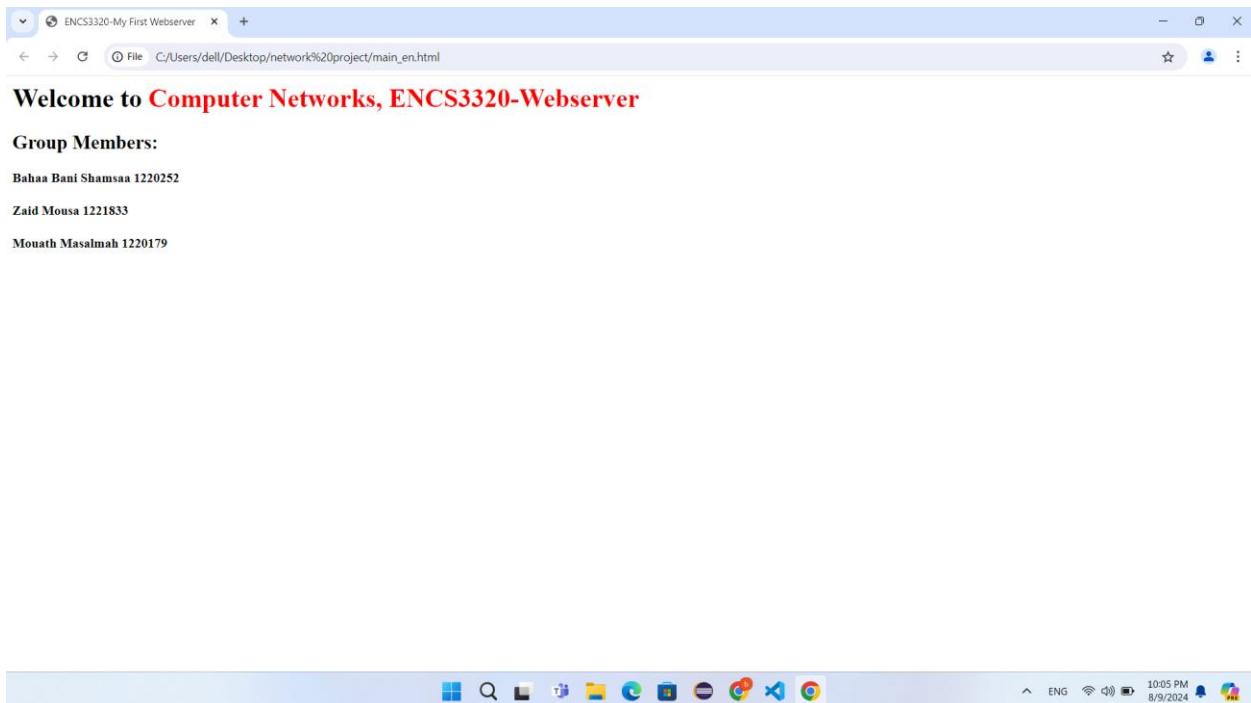
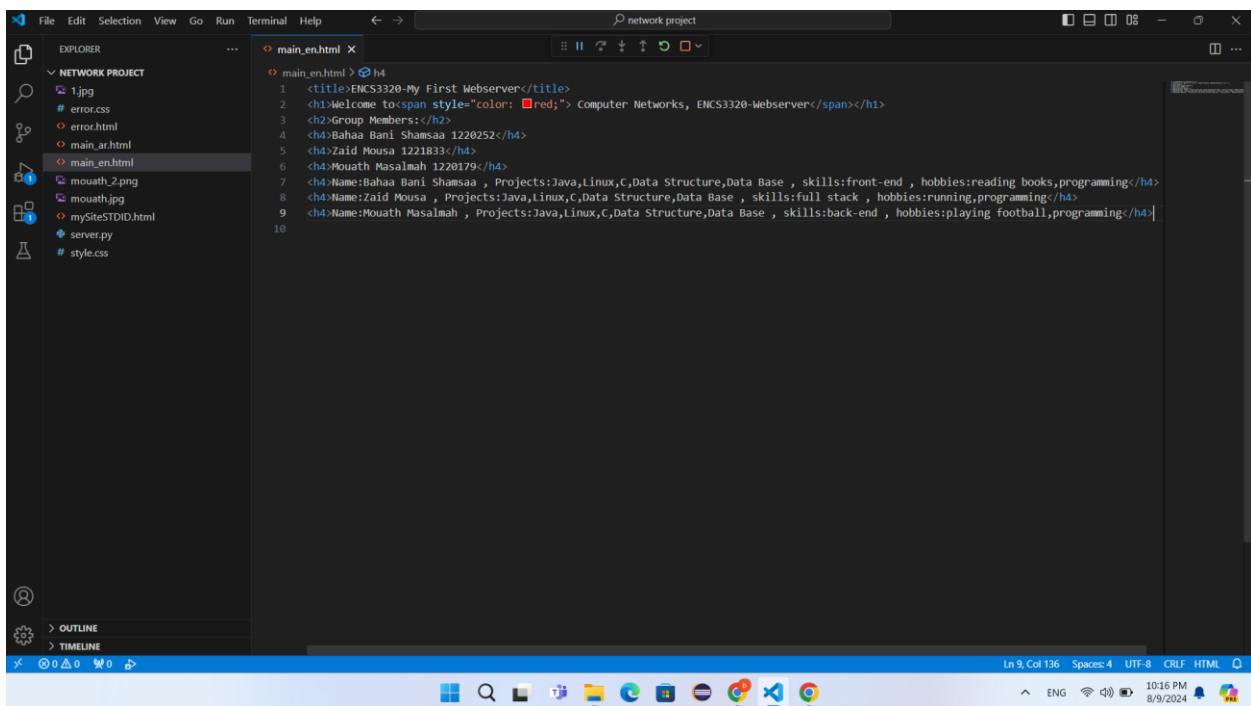


Figure 36:browser group members

Here we type the group members names with the numbers

d.



The screenshot shows a code editor window with the following details:

- File Path:** network project\main\_en.html
- Content:**

```
<title>ENCS3320-My First Webserver</title>
<h1>Welcome to<span style="color: red;> Computer Networks, ENCS3320-Webserver:</span></h1>
<h2>Group Members:</h2>
<h3>Bahaa Bani Shamsaa 1220252</h3>
<h3>Zaid Mousa 1221833</h3>
<h3>Mouath Masalmah 1220179</h3>
Name:Bahaa Bani Shamsaa , Projects:Java,Linux,C,Data Structure,Data Base , skills:front-end , hobbies:reading books,programming
Name:Zaid Mousa , Projects:Java,Linux,C,Data Structure,Data Base , skills:full stack , hobbies:running,programming
Name:Mouath Masalmah , Projects:Java,Linux,C,Data Structure,Data Base , skills:back-end , hobbies:playing football,programming
```
- Tools Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help, a search bar, and icons for zoom and refresh.
- Sidebar:** EXPLORER, OUTLINE, and TIMELINE.
- Status Bar:** Ln 9, Col 136, Spaces: 4, UTF-8, CRLF, HTML, and a date/time stamp: 8/9/2024 10:16 PM.

Figure 37: info about group members

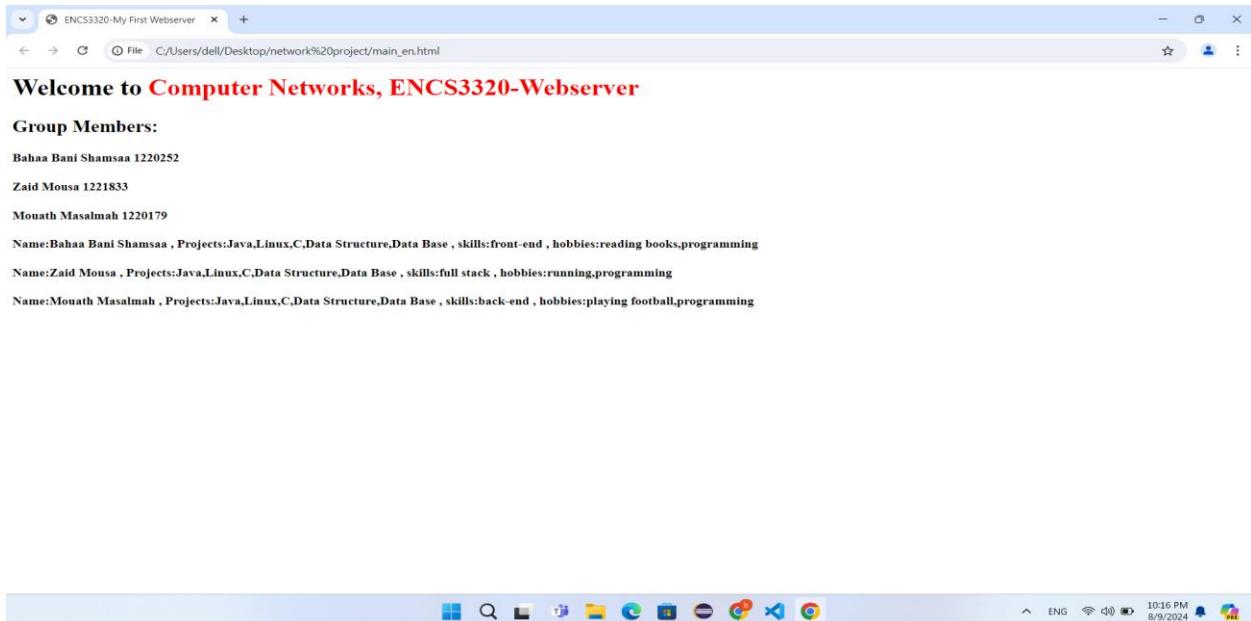
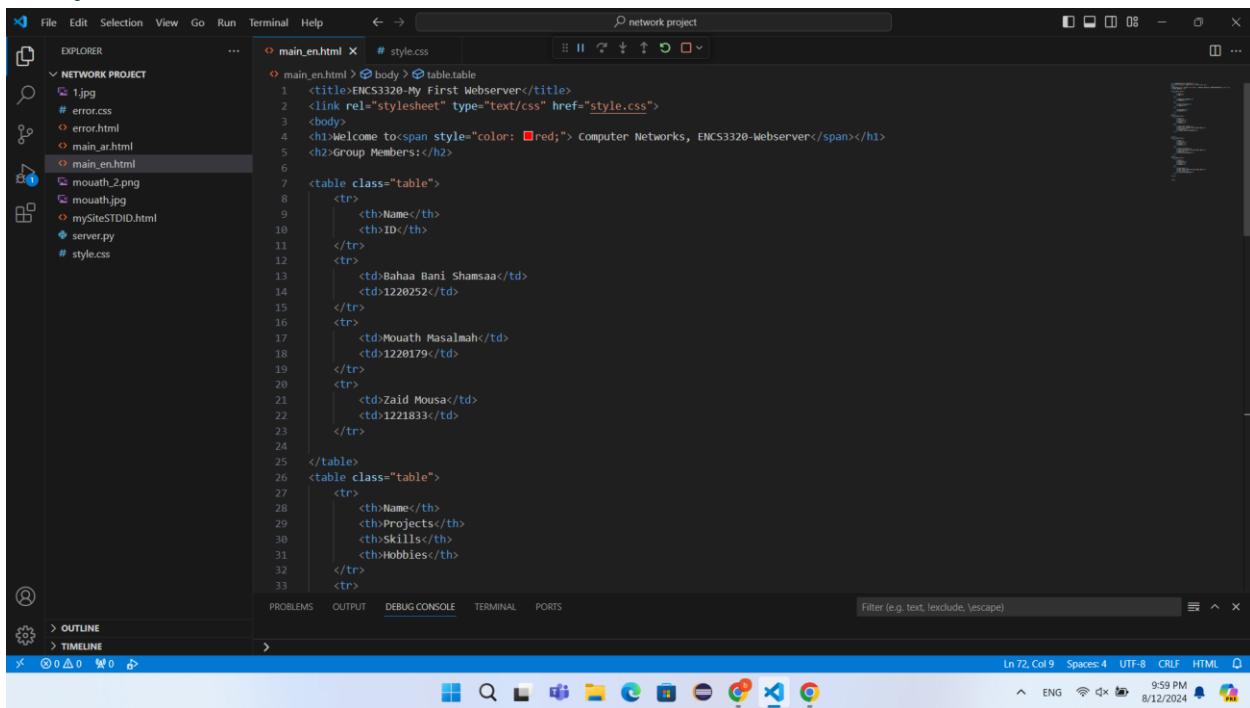


Figure 38:browser info group members

Here we type the names, projects , skills and hobbies

e and f:

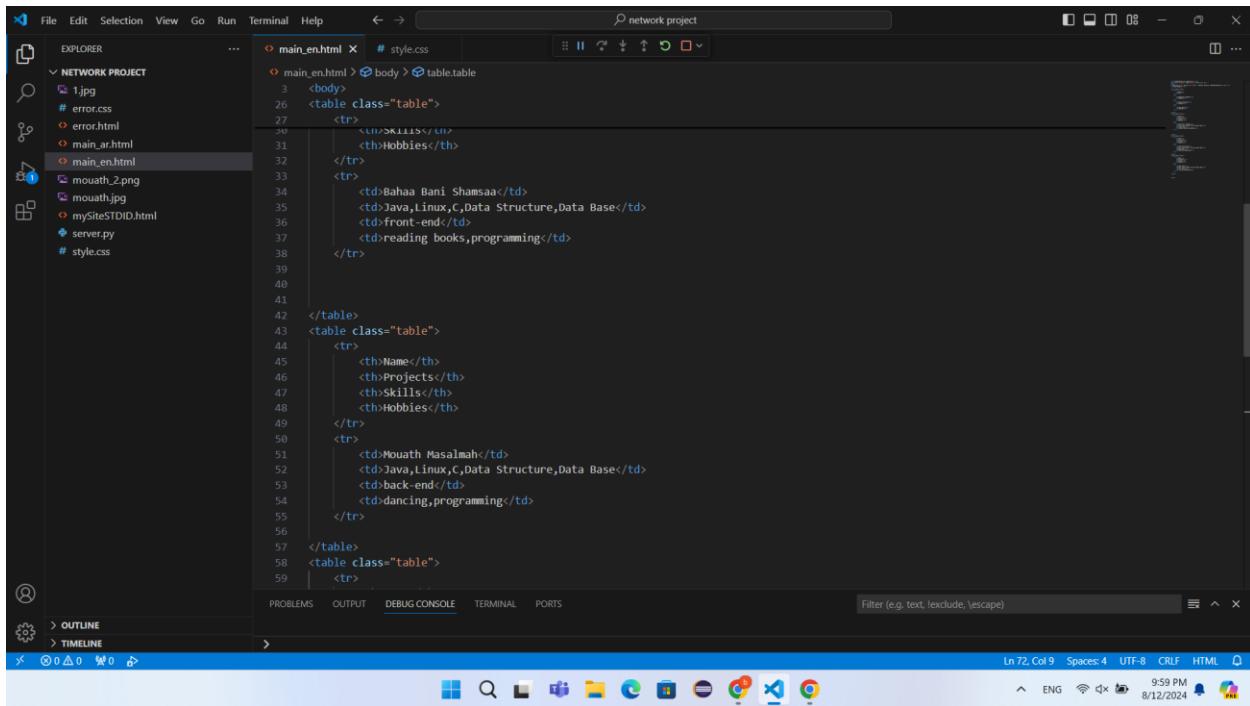


```
File Edit Selection View Go Run Terminal Help network project

EXPLORER NETWORK PROJECT
1.jpg
# error.css
error.html
main_ar.html
main_en.html
mouath_2.png
mouath.jpg
mySiteSTUDID.html
server.py
# style.css

main_en.html # style.css
main_en.html > body > table.table
1 <title>ENCS3320-My First Webserver</title>
2 <link rel="stylesheet" type="text/css" href="style.css">
3 <body>
4 <h1>Welcome to Computer Networks, ENCS3320-Webserver</span></h1>
5 <h2>Group Members:</h2>
6
7 <table class="table">
8   <tr>
9     <th>Name</th>
10    <th>ID</th>
11  </tr>
12  <tr>
13    <td>Bahaa Bani Shamsaa</td>
14    <td>1220252</td>
15  </tr>
16  <tr>
17    <td>Mouath Masalmah</td>
18    <td>1220179</td>
19  </tr>
20  <tr>
21    <td>Zaid Mousa</td>
22    <td>1221833</td>
23  </tr>
24
25 </table>
26 <table class="table">
27   <tr>
28     <th>Name</th>
29     <th>Projects</th>
30     <th>Skills</th>
31     <th>Hobbies</th>
32   </tr>
33   <tr>
34     <td>Bahaa Bani Shamsaa</td>
35     <td>Java,Linux,C,Data Structure,Data Base</td>
36     <td>front-end</td>
37     <td>reading books,programming</td>
38   </tr>
39
40 </table>
41 <table class="table">
42   <tr>
43     <th>Name</th>
44     <th>Projects</th>
45     <th>Skills</th>
46     <th>Hobbies</th>
47   </tr>
48   <tr>
49     <td>Mouath Masalmah</td>
50     <td>Java,Linux,C,Data Structure,Data Base</td>
51     <td>back-end</td>
52     <td>dancing,programming</td>
53   </tr>
54
55 </table>
56 <table class="table">
57   <tr>
58
59 </tr>
```

Figure 39:table 1

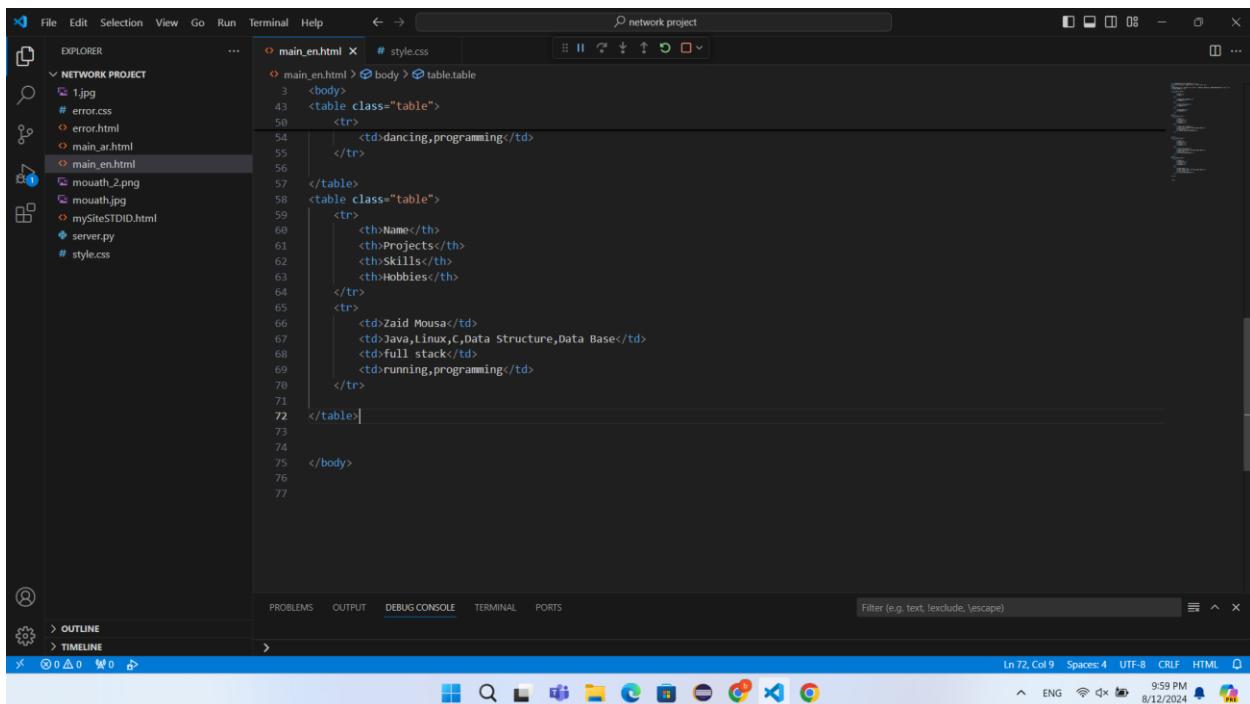


```
File Edit Selection View Go Run Terminal Help network project

EXPLORER NETWORK PROJECT
1.jpg
# error.css
error.html
main_ar.html
main_en.html
mouath_2.png
mouath.jpg
mySiteSTUDID.html
server.py
# style.css

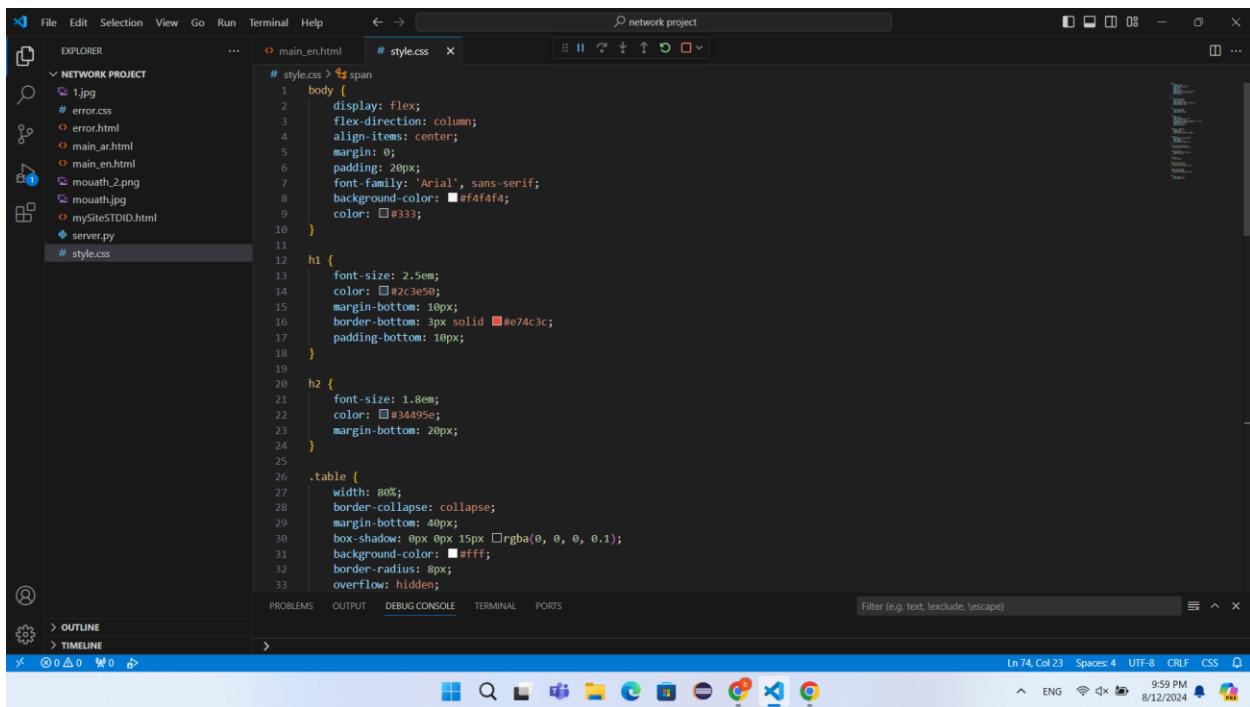
main_en.html # style.css
main_en.html > body > table.table
3 <body>
26 <table class="table">
27   <tr>
28     <th>Skills</th>
29     <th>Hobbies</th>
30   </tr>
31   <tr>
32     <td>Bahaa Bani Shamsaa</td>
33     <td>Java,Linux,C,Data Structure,Data Base</td>
34   </tr>
35   <tr>
36     <td>Mouath Masalmah</td>
37     <td>Java,Linux,C,Data Structure,Data Base</td>
38   </tr>
39
40 </table>
41 <table class="table">
42   <tr>
43     <th>Name</th>
44     <th>Projects</th>
45     <th>Skills</th>
46     <th>Hobbies</th>
47   </tr>
48   <tr>
49     <td>Bahaa Bani Shamsaa</td>
50     <td>Java,Linux,C,Data Structure,Data Base</td>
51     <td>front-end</td>
52     <td>reading books,programming</td>
53   </tr>
54
55 </table>
56 <table class="table">
57   <tr>
58
59 </tr>
```

Figure 40:table 2



```
<body>
  <table class="table">
    <tr>
      <td>dancing,programming</td>
    </tr>
    <tr>
      <th>Name</th>
      <th>Projects</th>
      <th>Skills</th>
      <th>Hobbies</th>
    </tr>
    <tr>
      <td>zaid Mousa</td>
      <td>Java,Linux,C,Data Structure,Data Base</td>
      <td>full stack</td>
      <td>running,programming</td>
    </tr>
  </table>
</body>
```

Figure 41:table 3



```
# style.css > 4 span
body {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: 0;
  padding: 20px;
  font-family: 'Arial', sans-serif;
  background-color: #f4f4f4;
  color: #333;
}

h1 {
  font-size: 2.5em;
  color: #2c3e50;
  margin-bottom: 10px;
  border-bottom: 3px solid #e74c3c;
  padding-bottom: 10px;
}

h2 {
  font-size: 1.8em;
  color: #34495e;
  margin-bottom: 20px;
}

.table {
  width: 80%;
  border-collapse: collapse;
  margin-bottom: 40px;
  box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.1);
  background-color: #fff;
  border-radius: 8px;
  overflow: hidden;
```

Figure 42:css for table 1

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files like 1.jpg, #error.css, error.html, main\_ar.html, main\_en.html, mousath\_2.png, mousath.jpg, mySiteSTID.html, server.py, and #style.css.
- Code Editor:** Displays the following CSS code for #style.css:

```
# style.css > span
  .table {
    background-color: #fff;
    border-radius: 8px;
    overflow: hidden;
  }

  .table th, .table td {
    padding: 15px;
    text-align: center;
    border-bottom: 1px solid #ddd;
  }

  .table th {
    background-color: #34495e;
    color: #fff;
    font-weight: bold;
    text-transform: uppercase;
  }

  .table tr:nth-child(even) {
    background-color: #f2f2f2;
  }

  .table tr:hover {
    background-color: #f9c5c5;
    cursor: pointer;
  }

  .table td {
    color: #555;
  }

  .table th:first-child,
```

The code editor has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The bottom status bar shows "Ln 74, Col 23" and "Filter (e.g. text, 'exclude', \escape)".

Figure 43:css for table 2

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files like 1.jpg, #error.css, error.html, main\_ar.html, main\_en.html, mousath\_2.png, mousath.jpg, mySiteSTID.html, server.py, and #style.css.
- Code Editor:** Displays the following CSS code for #style.css:

```
# style.css > span
  .table tr:hover {
    cursor: pointer;
  }

  .table td {
    color: #555;
  }

  .table th:first-child,
  .table td:first-child {
    border-top-left-radius: 8px;
  }

  .table th:last-child,
  .table td:last-child {
    border-top-right-radius: 8px;
  }

  span {
    color: #e74c3c;
    font-weight: bold;
  }
```

The code editor has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The bottom status bar shows "Ln 74, Col 23" and "Filter (e.g. text, 'exclude', \escape)".

Figure 44:css for table 3

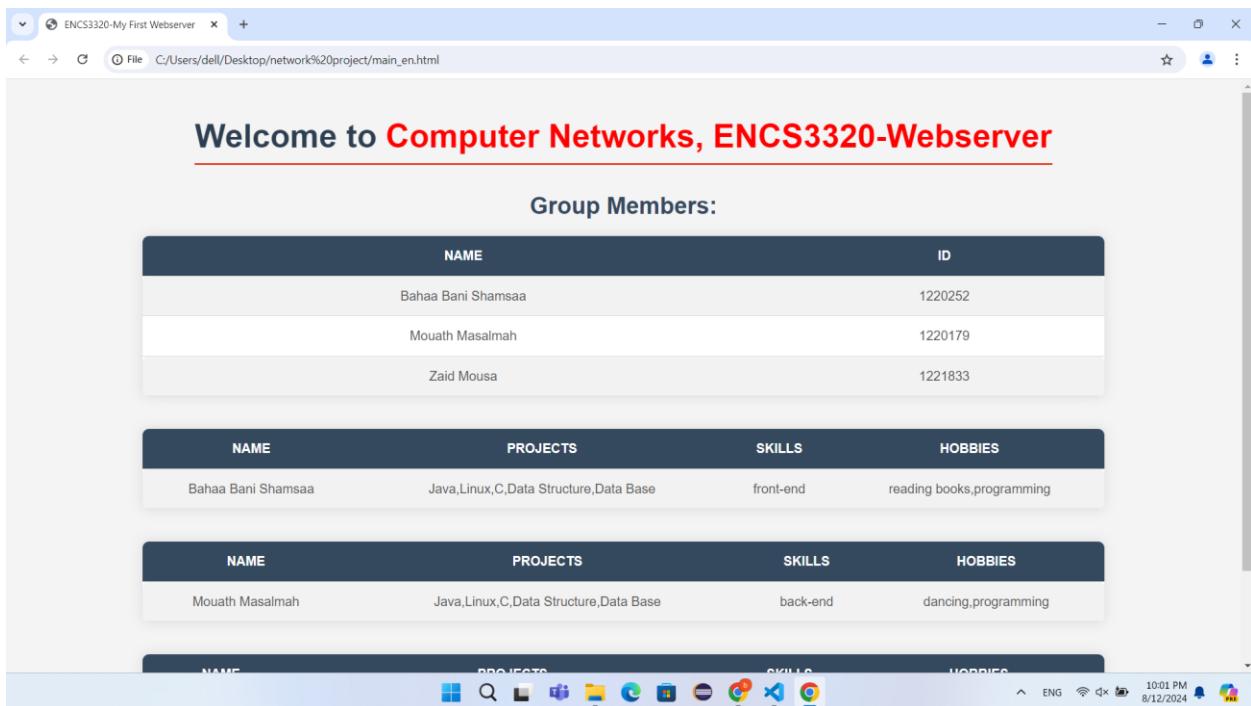


Figure 45:browser for table 1

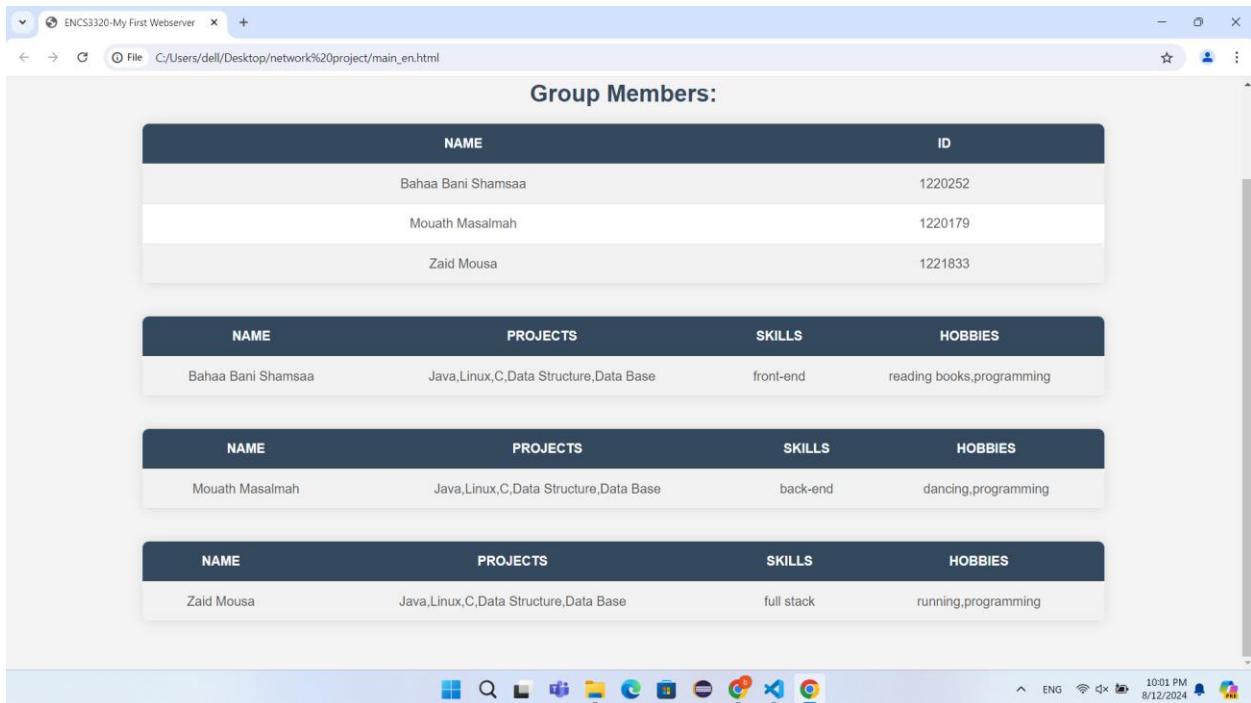
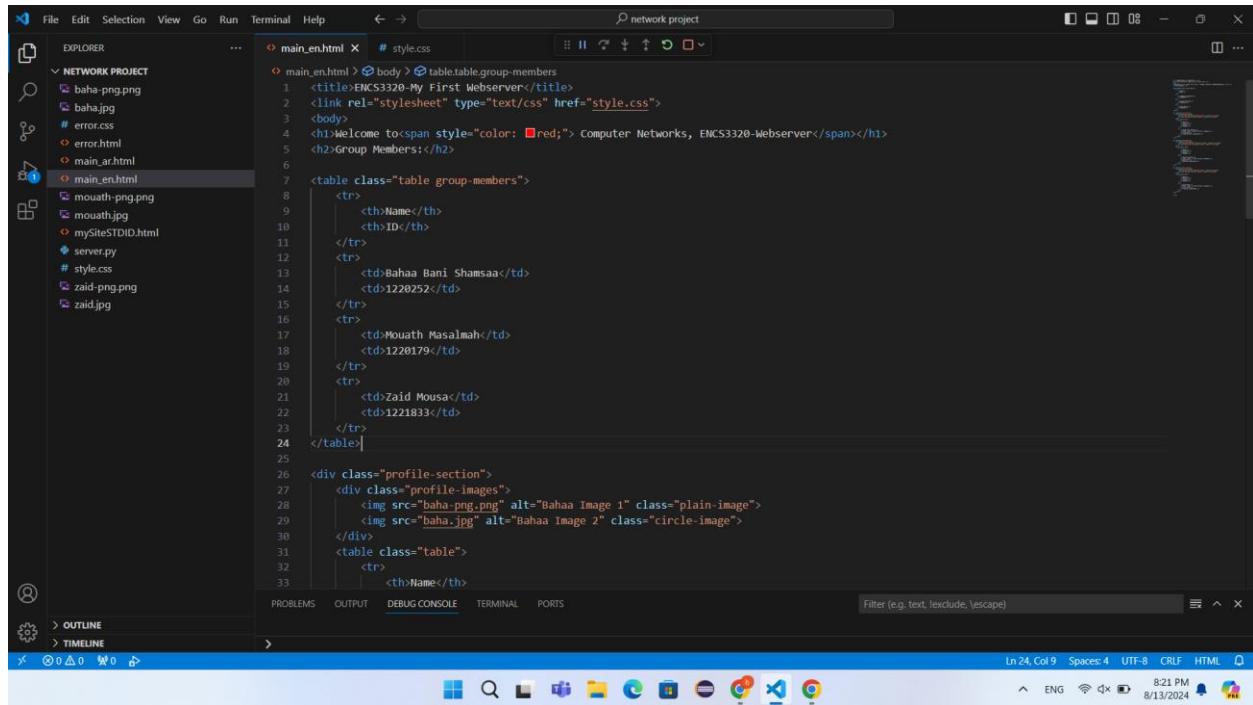


Figure 46:browser for table 2

Here we divide the group members names and IDs in table and each member with his information in tables, and we use a good CSS to make the page look nice, we use things like hover, background , border

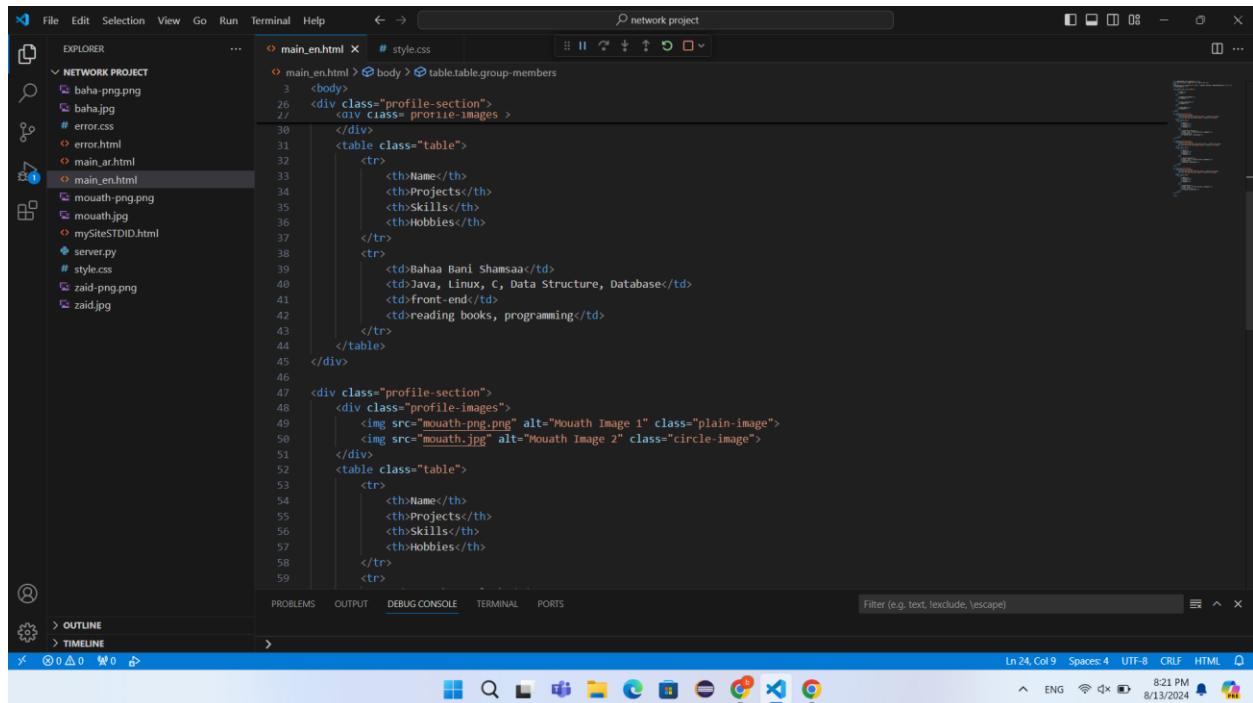
h.



```
<table class="table group-members">
    <tr>
        <th>Name</th>
        <th>ID</th>
    </tr>
    <tr>
        <td>Bahaa Bani Shamsaa</td>
        <td>1220252</td>
    </tr>
    <tr>
        <td>Mouath Masalmah</td>
        <td>1220179</td>
    </tr>
    <tr>
        <td>Zaid Mousa</td>
        <td>1221833</td>
    </tr>
</table>

<div class="profile-section">
    <div class="profile-images">
        
        
    </div>
    <table class="table">
        <tr>
            <th>Name</th>
        </tr>
```

Figure 47:png and jpg 1



```
<table class="table">
    <tr>
        <th>Name</th>
        <th>Projects</th>
        <th>Skills</th>
        <th>Hobbies</th>
    </tr>
    <tr>
        <td>Bahaa Bani Shamsaa</td>
        <td>Java, Linux, C, Data Structure, Database</td>
        <td>front-end</td>
        <td>reading books, programming</td>
    </tr>
</table>

<div class="profile-section">
    <div class="profile-images">
        
        
    </div>
    <table class="table">
        <tr>
            <th>Name</th>
            <th>Projects</th>
            <th>Skills</th>
            <th>Hobbies</th>
        </tr>
        <tr>
```

Figure 48:png and jpg 2

The screenshot shows the Visual Studio Code interface with the 'main\_en.html' file open in the editor. The code displays two profile sections for 'Mouath Masalmah' and 'Zaid Mousa'. Each section includes a table with columns for Name, Projects, Skills, and Hobbies, and a row of two images labeled 'plain-image' and 'circle-image'. The 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS' tabs are visible at the bottom.

```
<body>
  <div class="profile-section">
    <div class="profile-images">
      
      
    </div>
    <table class="table">
      <tr>
        <th>Name</th>
        <th>Projects</th>
        <th>Skills</th>
        <th>Hobbies</th>
      </tr>
      <tr>
        <td>Mouath Masalmah</td>
        <td>Java, Linux, C, Data Structure, Database</td>
        <td>back-end</td>
        <td>dancing, programming</td>
      </tr>
    </table>
  </div>
  <div class="profile-section">
    <div class="profile-images">
      
      
    </div>
    <table class="table">
      <tr>
        <th>Name</th>
        <th>Projects</th>
        <th>Skills</th>
        <th>Hobbies</th>
      </tr>
      <tr>
        <td>Zaid Mousa</td>
        <td>Java, Linux, C, Data Structure, Database</td>
        <td>full stack</td>
        <td>running, programming</td>
      </tr>
    </table>
  </div>
</body>
```

Figure 49:png and jpg 3

The screenshot shows the Visual Studio Code interface with the 'main\_en.html' file open in the editor. The code has been modified to reflect the changes made in Figure 49. The 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS' tabs are visible at the bottom.

```
<body>
  <div class="profile-section">
    <div class="profile-images">
      
      
    </div>
    <table class="table">
      <tr>
        <th>Name</th>
        <th>Projects</th>
        <th>Skills</th>
        <th>Hobbies</th>
      </tr>
      <tr>
        <td>Zaid Mousa</td>
        <td>Java, Linux, C, Data Structure, Database</td>
        <td>full stack</td>
        <td>running, programming</td>
      </tr>
    </table>
  </div>
</body>
```

Figure 50:png and jpg 4

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a "NETWORK PROJECT" folder containing files: baha-png.png, baha.jpg, #error.css, error.html, main\_ar.html, main\_en.html, mouauth-png.png, mouauth.jpg, mySiteSTID.html, and server.py.
- Editor:** The main editor window displays the content of style.css. The code includes styles for body, h1, h2, and a table.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. It also shows the current file path (main\_en.html), line number (Ln 8, Col 32), and encoding (UTF-8).
- System Taskbar:** Shows icons for File Explorer, Search, Task View, File Manager, Edge, and other system applications.

Figure 51:css png and jpg 1

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a "NETWORK PROJECT" folder containing files: baha-png.png, baha.jpg, #error.css, error.html, main\_ar.html, main\_en.html, mouauth-png.png, mouauth.jpg, mySiteSTID.html, and server.py.
- Editor:** The main editor window displays the content of style.css. The code includes styles for .table, .table th, .table td, .table tr, .table tr:hover, and .profile-section.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. It also shows the current file path (main\_en.html), line number (Ln 8, Col 32), and encoding (UTF-8).
- System Taskbar:** Shows icons for File Explorer, Search, Task View, File Manager, Edge, and other system applications.

Figure 52:css png and jpg 2

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a "NETWORK PROJECT" folder containing files: baha-png.png, baha.jpg, #error.css, error.html, main\_ar.html, main\_en.html, mounath-png.png, mounath.jpg, mySiteSTID.html, server.py, and #style.css.
- Editor:** The main pane displays the content of the #style.css file. The visible code includes:

```
# style.css > body
  .profile-section {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: 80px;
    background-color: #ffffff;
    border-radius: 8px;
    padding: 20px;
    margin-bottom: 20px;
    box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.1);
  }
  .profile-section:not(:last-child) {
    margin-bottom: 40px;
  }
  .profile-images {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    margin-left: 20px;
  }
  .circle-image {
    width: 100px;
    height: 100px;
    border-radius: 50%;
    margin-bottom: 15px;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease;
  }
  .circle-image:hover {
    transform: scale(1.05);
  }
  .plain-image {
    width: 100px;
    height: 100px;
    margin-bottom: 15px;
    transition: transform 0.3s ease;
  }
  .plain-image:hover {
    transform: scale(1.05);
  }
```

The bottom status bar shows: Ln 8, Col 32 | Spaces: 4 | UTF-8 | CRLF | CSS | Filter (e.g. text, !exclude, \escape).

Figure 53:css png and jpg 3

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a "NETWORK PROJECT" folder containing files: baha-png.png, baha.jpg, #error.css, error.html, main\_ar.html, main\_en.html, mounath-png.png, mounath.jpg, mySiteSTID.html, server.py, and #style.css.
- Editor:** The main pane displays the content of the #style.css file. The visible code includes:

```
# style.css > body
  .profile-images {
    justify-content: center;
    margin-left: 20px;
  }
  .circle-image {
    width: 100px;
    height: 100px;
    border-radius: 50%;
    margin-bottom: 15px;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s ease;
  }
  .circle-image:hover {
    transform: scale(1.05);
  }
  .plain-image {
    width: 100px;
    height: 100px;
    margin-bottom: 15px;
    transition: transform 0.3s ease;
  }
  .plain-image:hover {
    transform: scale(1.05);
  }
```

The bottom status bar shows: Ln 8, Col 32 | Spaces: 4 | UTF-8 | CRLF | CSS | Filter (e.g. text, !exclude, \escape).

Figure 54:css png and jpg 4

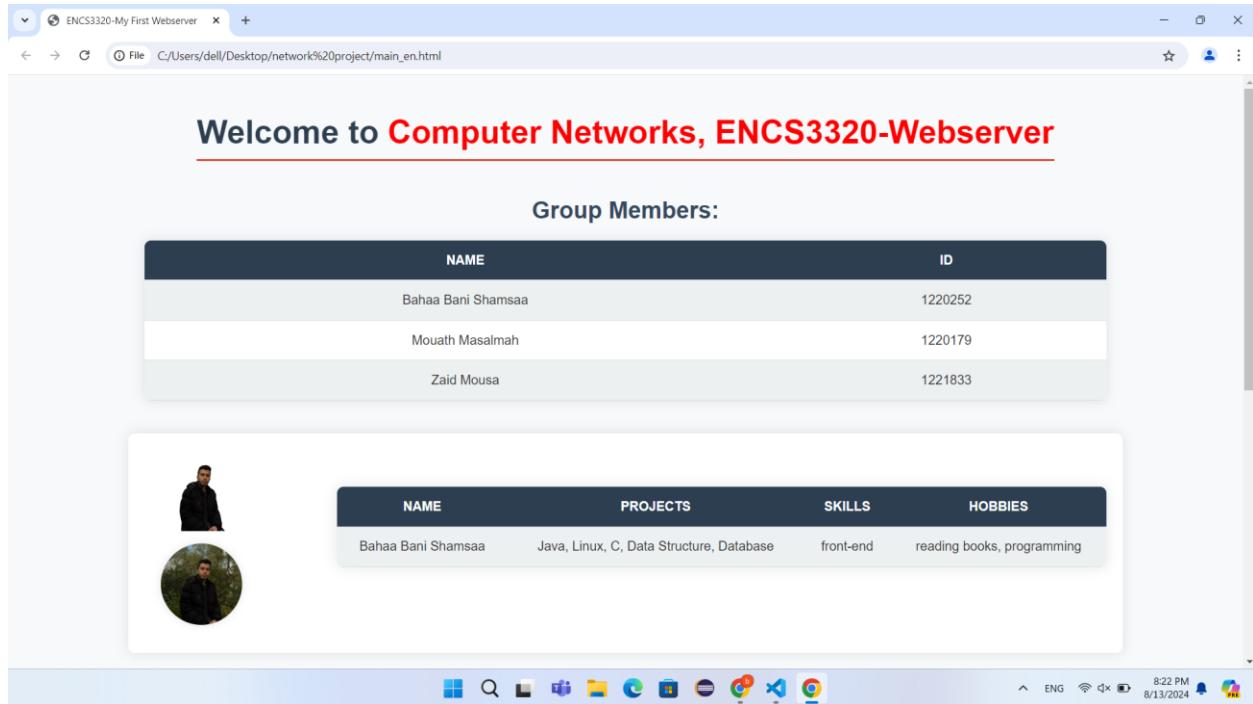


Figure 55:browser for png and jpg 1

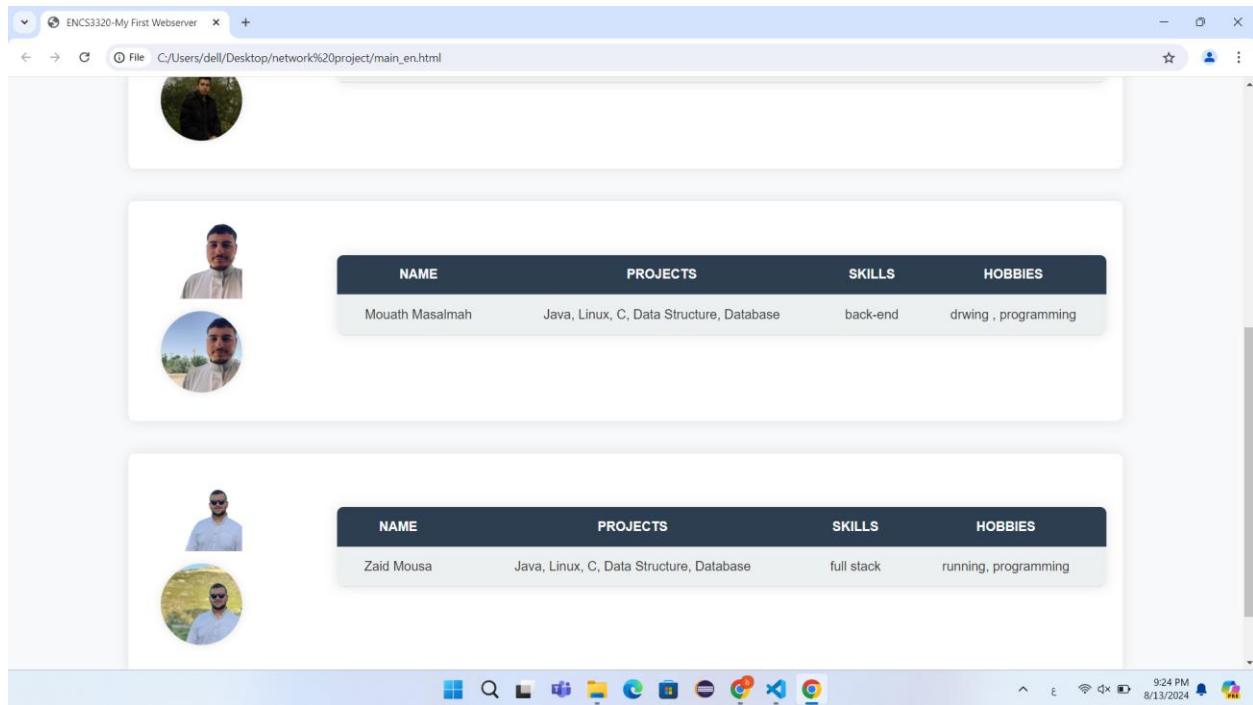
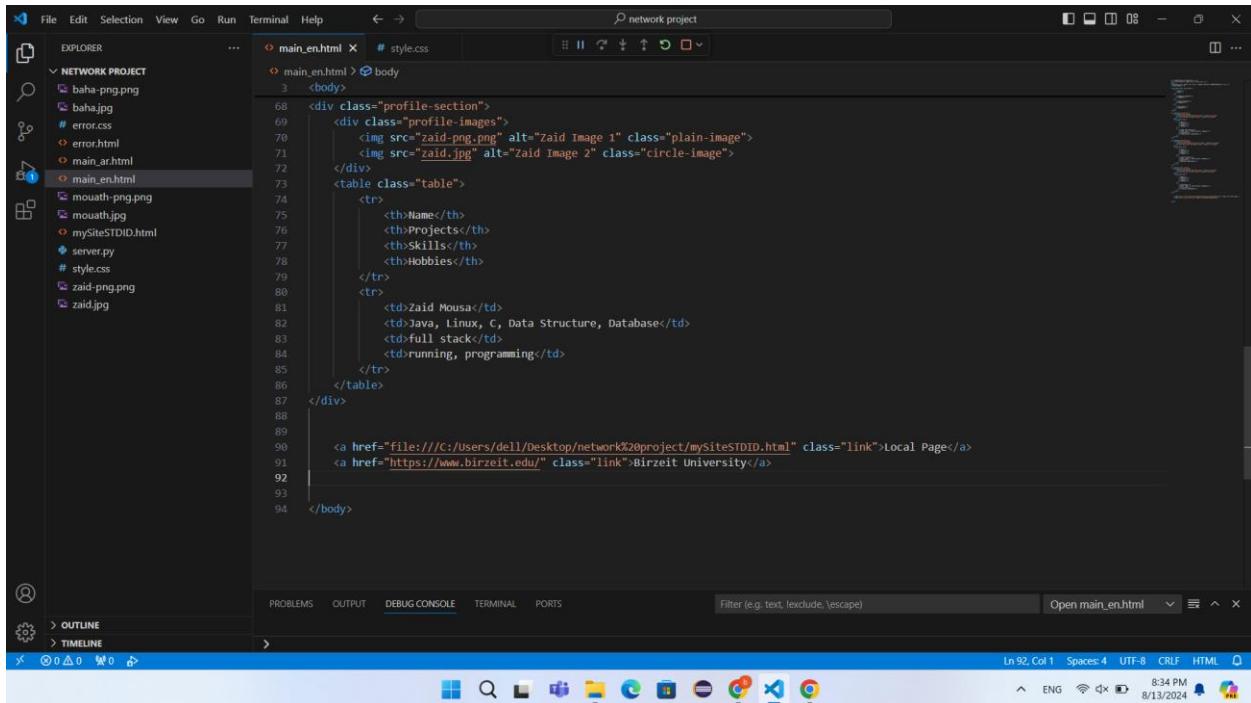


Figure 56:browser for png and jpg 2

As we see here, we put 6 images 3 .jpg and 3 .png , each member we put his images at his table , we use border-radius to make the img circle

*i and j:*



The screenshot shows the Visual Studio Code interface with the 'main\_en.html' file open in the editor. The code displays two profiles: 'Mouath Masalmah' and 'Zaid Mousa'. Each profile section includes a header image (circle or plain), a table with columns for Name, Projects, Skills, and Hobbies, and links to a local page and Birzeit University.

```
<div class="profile-section">
  <div class="profile-images">
    
    
  </div>
  <table class="table">
    <tr>
      <th>Name</th>
      <th>Projects</th>
      <th>Skills</th>
      <th>Hobbies</th>
    </tr>
    <tr>
      <td>Zaid Mousa</td>
      <td>Java, Linux, C, Data Structure, Database</td>
      <td>full stack</td>
      <td>running, programming</td>
    </tr>
  </table>
</div>

<a href="file:///C:/Users/dell/Desktop/network%20project/mySiteSTDID.html" class="link">Local Page</a>
<a href="https://www.birzeit.edu/" class="link">Birzeit University</a>
```

Figure 57:link to local html and Birzeit web

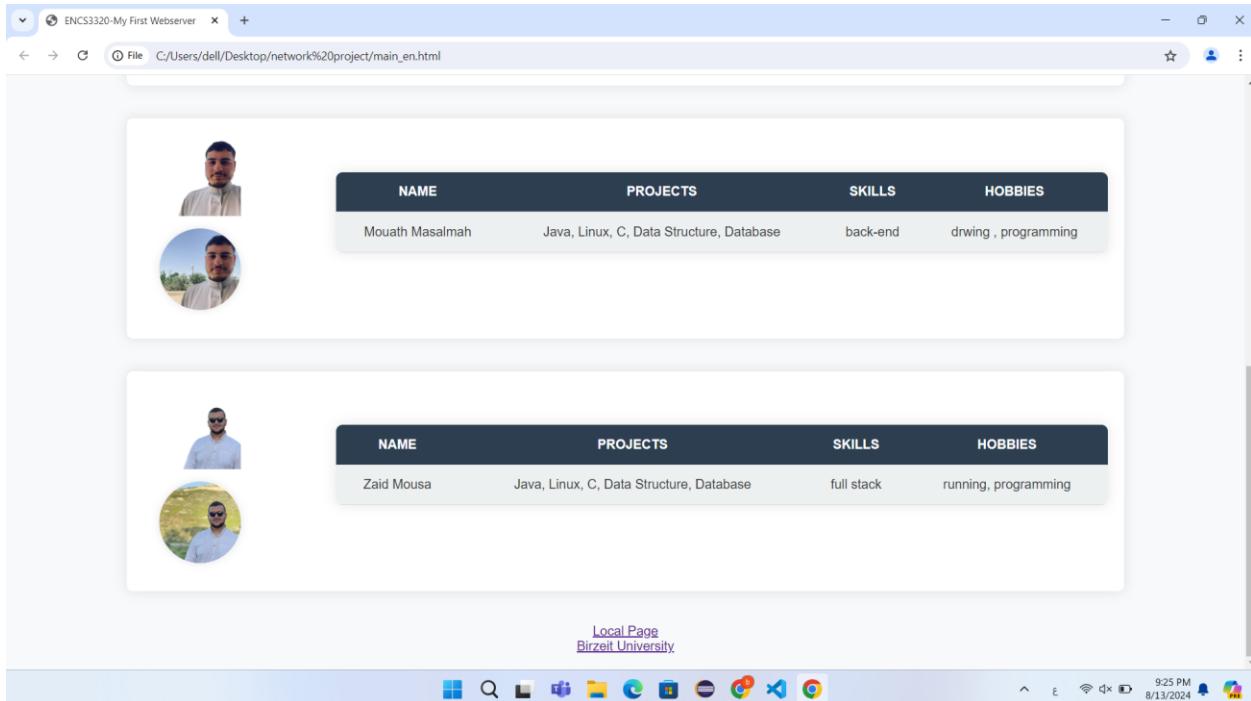


Figure 58:browser for link to local html and Birzeit web

Here we put 2 links one for Birzeit and another one for a local page that it is mySiteSTDID.html

## 2.

| الاسم         | الرقم الجامعي |
|---------------|---------------|
| بهاء بنى شمسه | 1220252       |
| معاذ مسالمة   | 1220179       |
| زيد موسى      | 1221833       |

|  |                               |   |                                     |  |
|--|-------------------------------|---|-------------------------------------|--|
|  | <b>الاسم</b><br>بهاء بنى شمسه | <b>المشروع</b><br>جيادا , لينوكس , سي , بنية المعلومات , قواعد البيانات | <b>المهارات</b><br>الواجهة الامامية | <b>الهوايات</b><br>قراءة الكتب , الترجمة |
|--|-------------------------------|---|-------------------------------------|--|

Figure 59: /ar request

| الاسم         | الرقم الجامعي |
|---------------|---------------|
| بهاء بنى شمسه | 1220252       |
| معاذ مسالمة   | 1220179       |
| زيد موسى      | 1221833       |

|  |                               |   |                                     |  |
|--|-------------------------------|---|-------------------------------------|--|
|  | <b>الاسم</b><br>بهاء بنى شمسه | <b>المشروع</b><br>جيادا , لينوكس , سي , بنية المعلومات , قواعد البيانات | <b>المهارات</b><br>الواجهة الامامية | <b>الهوايات</b><br>قراءة الكتب , الترجمة |
|--|-------------------------------|---|-------------------------------------|--|

Figure 60:main\_ar.html request

As we see here when I search using this link <http://localhost:1833/ar> it shows the arabic version of main\_en.html which is main\_ar.html

### 3.

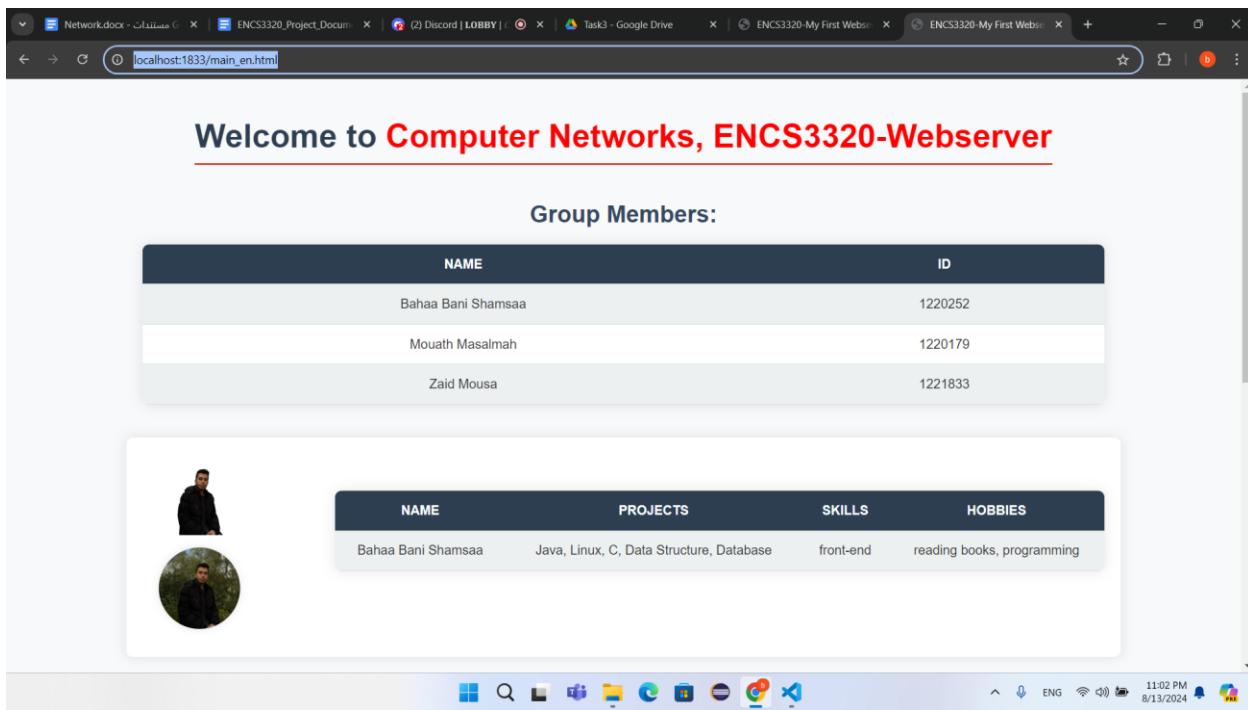
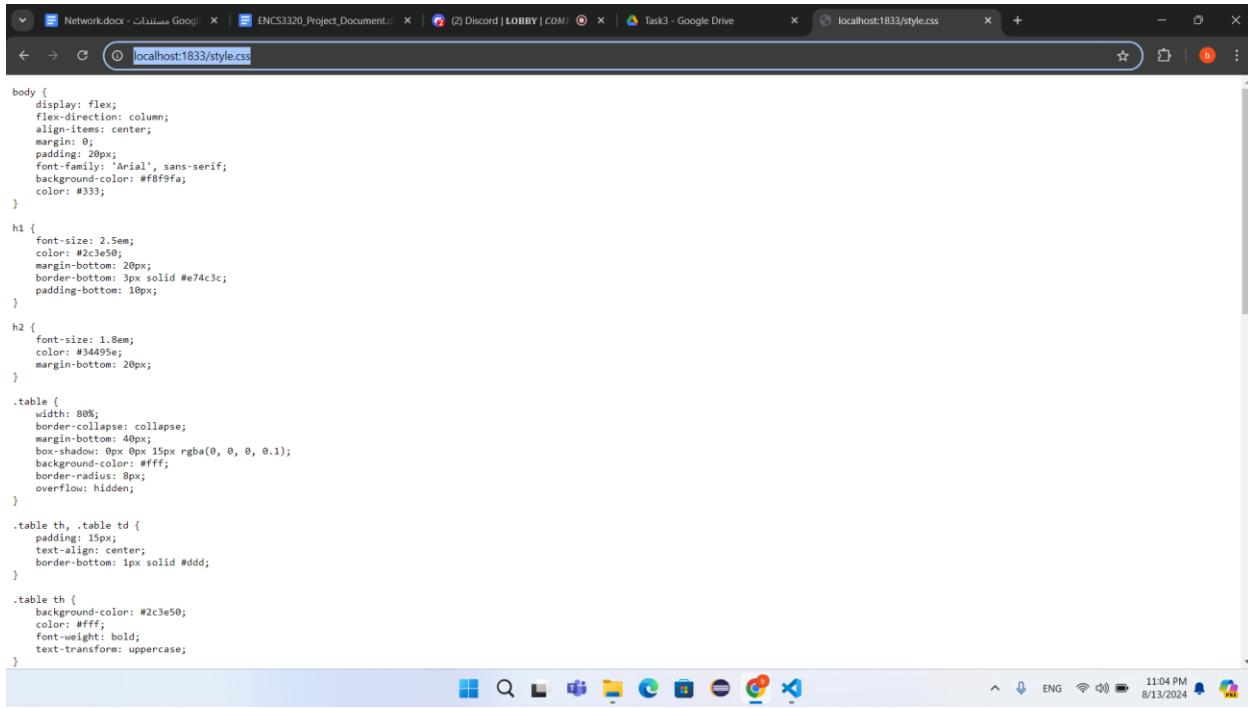


Figure 61:main\_en.html request

Here we tried to search for an .html file and we search for main\_en.html using the link [http://localhost:1833/main\\_en.html](http://localhost:1833/main_en.html) and it shows the contents of main\_en.html file

#### 4.



```
body {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin: 0;
    padding: 20px;
    font-family: 'Arial', sans-serif;
    background-color: #f0f5fa;
    color: #333;
}

h1 {
    font-size: 2.5em;
    color: #2c3e50;
    margin-bottom: 20px;
    border-bottom: 3px solid #e74c3c;
    padding-bottom: 10px;
}

h2 {
    font-size: 1.8em;
    color: #34495e;
    margin-bottom: 20px;
}

.table {
    width: 80%;
    border-collapse: collapse;
    margin-bottom: 40px;
    box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.1);
    background-color: #fff;
    border-radius: 8px;
    overflow: hidden;
}

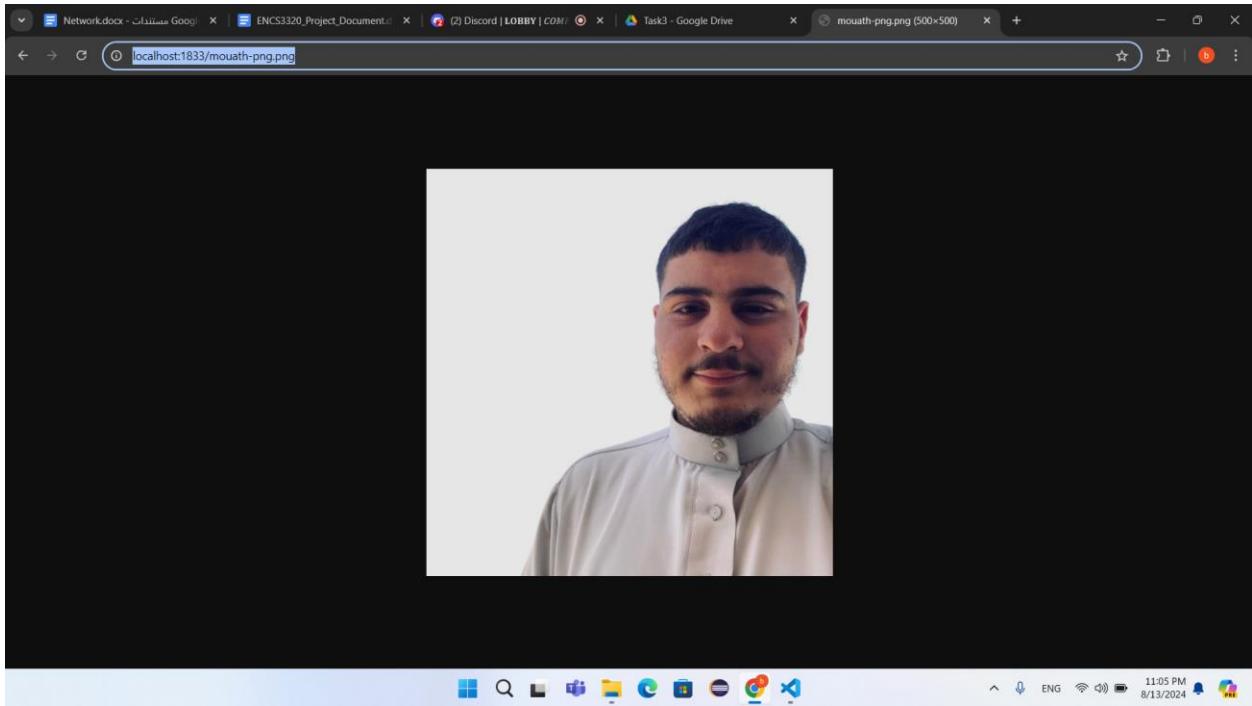
.table th, .table td {
    padding: 15px;
    text-align: center;
    border-bottom: 1px solid #ddd;
}

.table th {
    background-color: #2c3e50;
    color: #fff;
    font-weight: bold;
    text-transform: uppercase;
}
```

Figure 62:style.css request

Here we tried to search for and .css file and we search for style.css using the link <http://localhost:1833/style.css> and it shows the content of style.css file

## 5.



*Figure 63:mouauth.png request*

Here we tried to search for .png img and we search for mouauth-png.png using the link <http://localhost:1833/mouauth-png.png> and it shows the content of mouauth-png.png

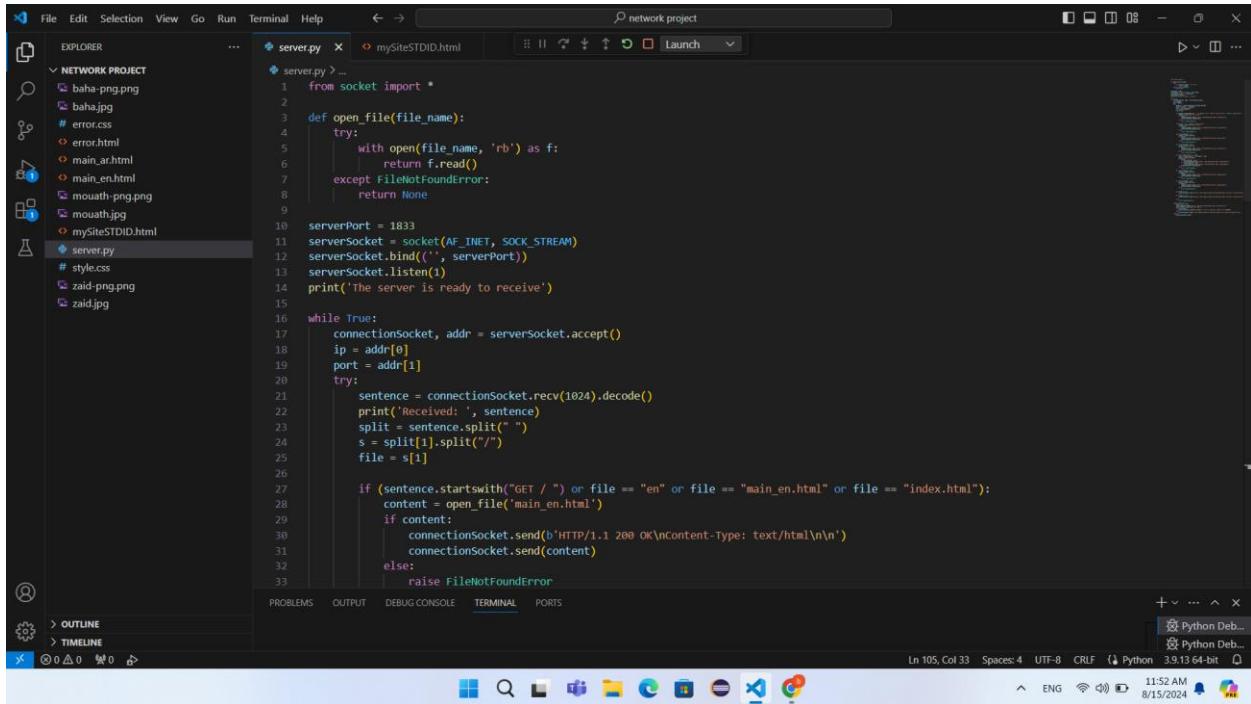
## 6.



*Figure 64:zaid.jpg request*

Here we tried to search for .jpg img and we search for zaid.jpg using the link <http://localhost:1833/zaid.jpg> and it shows the content of zaid.jpg

## 7 and 8:



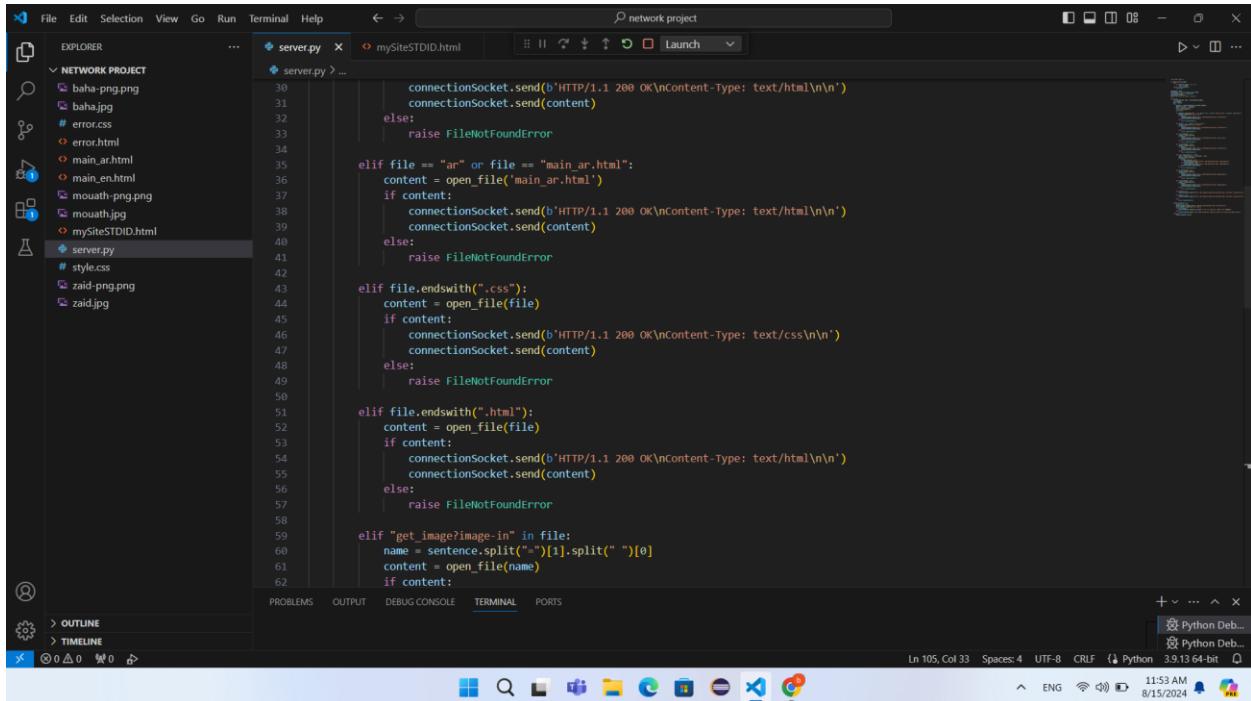
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a "NETWORK PROJECT" folder containing files: baha-png.png, baha.jpg, # error.css, error.html, main\_ar.html, main\_en.html, mouauth-png.png, mouauth.jpg, mySiteSTDID.html, server.py, # style.css, zaid-png.png, and zaid.jpg.
- Code Editor:** The active file is `server.py`, which contains Python code for a socket-based web server. It handles GET requests for files like "index.html", "main\_en.html", "main\_ar.html", and "style.css". It also handles a "get\_image?image-in" query by reading files from the directory.
- Terminal:** Shows the command `network project`.
- Bottom Status Bar:** Displays "Ln 105, Col 33" and "8/15/2024".

```

    1  from socket import *
    2
    3  def open_file(file_name):
    4      try:
    5          with open(file_name, 'rb') as f:
    6              return f.read()
    7      except FileNotFoundError:
    8          return None
    9
   10 serverPort = 1833
   11 serverSocket = socket(AF_INET, SOCK_STREAM)
   12 serverSocket.bind(("", serverPort))
   13 serverSocket.listen(1)
   14 print('The server is ready to receive')
   15
   16 while True:
   17     connectionSocket, addr = serverSocket.accept()
   18     ip = addr[0]
   19     port = addr[1]
   20     try:
   21         sentence = connectionSocket.recv(1024).decode()
   22         print('Received: ', sentence)
   23         split = sentence.split(" ")
   24         s = split[1].split("/")
   25         file = s[1]
   26
   27         if (sentence.startswith("GET / ") or file == "en" or file == "main_en.html" or file == "index.html"):
   28             content = open_file('main_en.html')
   29             if content:
   30                 connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
   31                 connectionSocket.send(content)
   32             else:
   33                 raise FileNotFoundError
   34
   35         elif file == "ar" or file == "main_ar.html":
   36             content = open_file('main_ar.html')
   37             if content:
   38                 connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
   39                 connectionSocket.send(content)
   40             else:
   41                 raise FileNotFoundError
   42
   43         elif file.endswith(".css"):
   44             content = open_file(file)
   45             if content:
   46                 connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/css\n\n')
   47                 connectionSocket.send(content)
   48             else:
   49                 raise FileNotFoundError
   50
   51         elif file.endswith(".html"):
   52             content = open_file(file)
   53             if content:
   54                 connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
   55                 connectionSocket.send(content)
   56             else:
   57                 raise FileNotFoundError
   58
   59         elif "get_image?image-in" in file:
   60             name = sentence.split(" ")[1].split(" ")[0]
   61             content = open_file(name)
   62             if content:
   63
   
```

Figure 65:server 1



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

- File Explorer:** Shows a "NETWORK PROJECT" folder containing files: baha-png.png, baha.jpg, # error.css, error.html, main\_ar.html, main\_en.html, mouauth-png.png, mouauth.jpg, mySiteSTDID.html, server.py, # style.css, zaid-png.png, and zaid.jpg.
- Code Editor:** The active file is `server.py`, which contains Python code for a socket-based web server. It handles GET requests for files like "index.html", "main\_en.html", "main\_ar.html", and "style.css". It also handles a "get\_image?image-in" query by reading files from the directory.
- Terminal:** Shows the command `network project`.
- Bottom Status Bar:** Displays "Ln 105, Col 33" and "8/15/2024".

```

    30         connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
    31         connectionSocket.send(content)
    32     else:
    33         raise FileNotFoundError
    34
    35     elif file == "ar" or file == "main_ar.html":
    36         content = open_file('main_ar.html')
    37         if content:
    38             connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
    39             connectionSocket.send(content)
    40         else:
    41             raise FileNotFoundError
    42
    43     elif file.endswith(".css"):
    44         content = open_file(file)
    45         if content:
    46             connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/css\n\n')
    47             connectionSocket.send(content)
    48         else:
    49             raise FileNotFoundError
    50
    51     elif file.endswith(".html"):
    52         content = open_file(file)
    53         if content:
    54             connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: text/html\n\n')
    55             connectionSocket.send(content)
    56         else:
    57             raise FileNotFoundError
    58
    59     elif "get_image?image-in" in file:
    60         name = sentence.split(" ")[1].split(" ")[0]
    61         content = open_file(name)
    62         if content:
    63
    
```

Figure 66:server 2

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

- File Explorer:** Shows a "NETWORK PROJECT" folder containing files: baha-png.png, baha.jpg, # error.css, error.html, main\_ar.html, main\_en.html, msmouth-png.png, msmouth.jpg, mySiteSTDID.html, server.py, # style.css, zaid-png.png, and zaid.jpg.
- Code Editor:** The active file is `server.py`, displaying Python code for a web server. The code handles file requests and sends HTTP responses. It includes logic for image types (.jpg, .png) and specific files like "so" and "itc". Error handling is also present.
- Terminal:** Shows the command `network project`.
- Status Bar:** Displays "Ln 105, Col 33" and "Python 3.9.13 64-bit".

Figure 67:server 3

This screenshot continues the code from Figure 67, showing the completion of the `server.py` script. The code now includes exception handling for 404 errors and sends an HTML error page back to the client. The code editor shows lines 80 through 105.

```

    content = open_file(file)
    if content:
        connectionSocket.send(b'HTTP/1.1 200 OK\nContent-Type: image/png\n\n')
        connectionSocket.send(content)
    else:
        raise FileNotFoundError

    elif file == "so":
        connectionSocket.send(b'HTTP/1.1 307 Temporary Redirect\r\nContent-Type: text/html; charset=utf-8\r\nLocation: https://stacko')

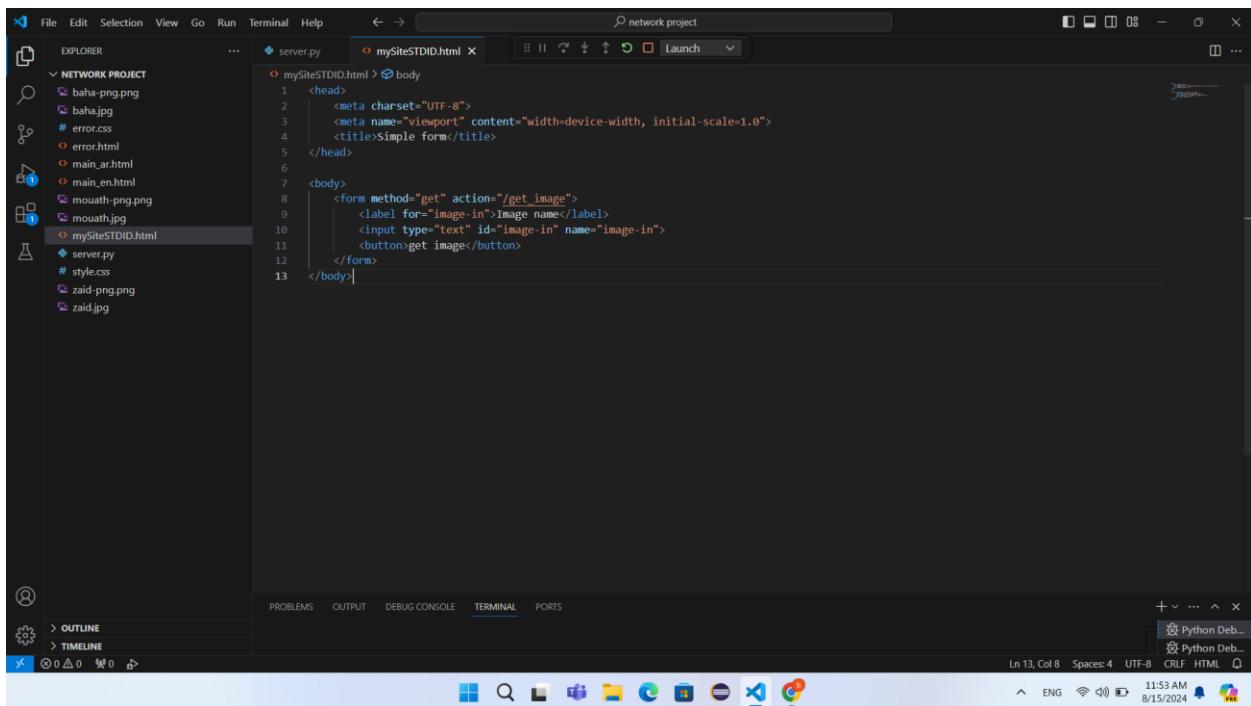
    elif file == "itc":
        connectionSocket.send(b'HTTP/1.1 307 Temporary Redirect\r\nContent-Type: text/html; charset=utf-8\r\nLocation: https://itc.bi')

    else:
        raise FileNotFoundError

except Exception as e:
    print(f'Error: {e}')
    connectionSocket.send(b'HTTP/1.1 404 Not Found\nContent-Type: text/html\n\n')
    error_content = open_file('error.html')
    if error_content:
        connectionSocket.send(error_content + f"<p> ip: {ip} port: {port}</p>".encode())
    else:
        connectionSocket.send(b'<html><body><h1>404 Not Found</h1><p>File not found.</p></body></html>')
finally:
    connectionSocket.close()

```

Figure 68:server 4



```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>simple form</title>
</head>
<body>
    <form method="get" action="/get_image">
        <label for="image-in">Image name</label>
        <input type="text" id="image-in" name="image-in">
        <button type="submit">Get Image</button>
    </form>
</body>
```

Figure 69:mySiteSTDID.html

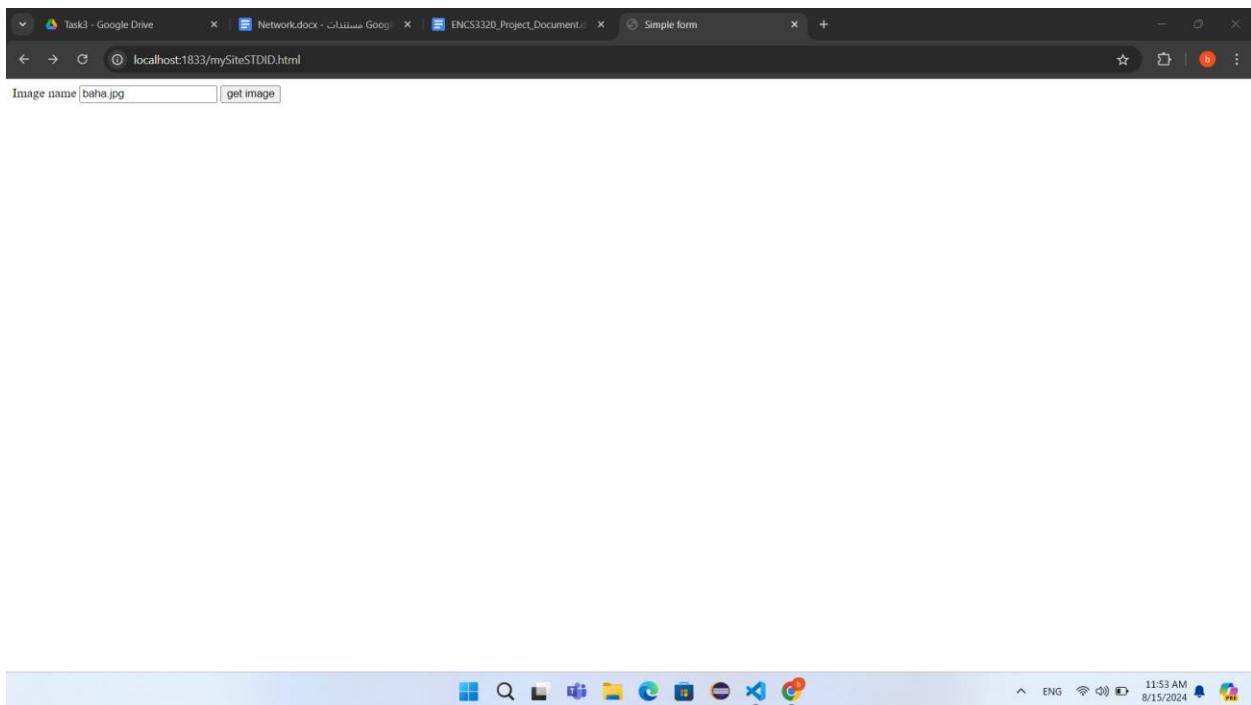


Figure 70:browesr for mySiteSTDID.html

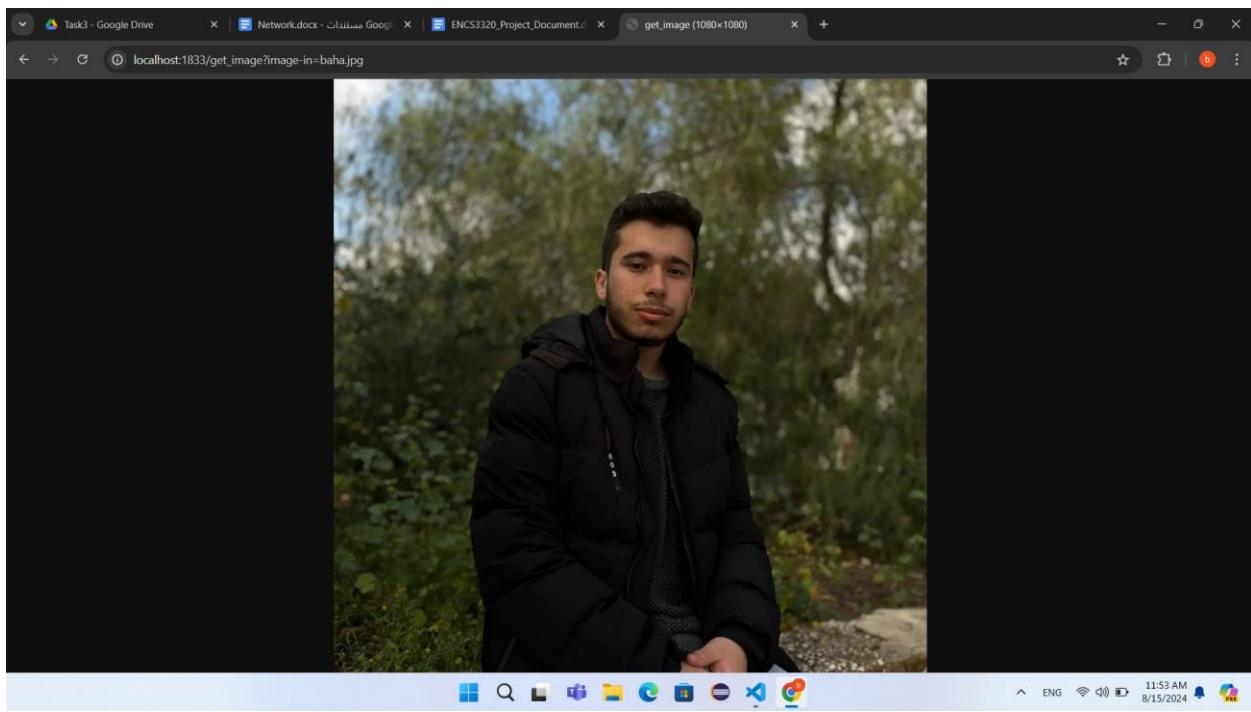


Figure 71:image browser for mySiteSTDID.html

As we see in images we use zaid ID which is 1221833 and we use last 4 numbers 1833 as a port number then we create socket then we set it up , then we see if the sentence we have start with / or en or main\_en.html or index.html then open file main\_en.html, else if sentence start with ar or main\_ar.html so open main\_ar.html file , else if sentence end with .css so it open the file requested .css , else if sentence end with .html it open the file requested .html , else if sentence contains get\_image?image\_in this asking for an specific image , else if file ends with .jpg serach for that file , else if file end with .png search for that file

## 9.

a.

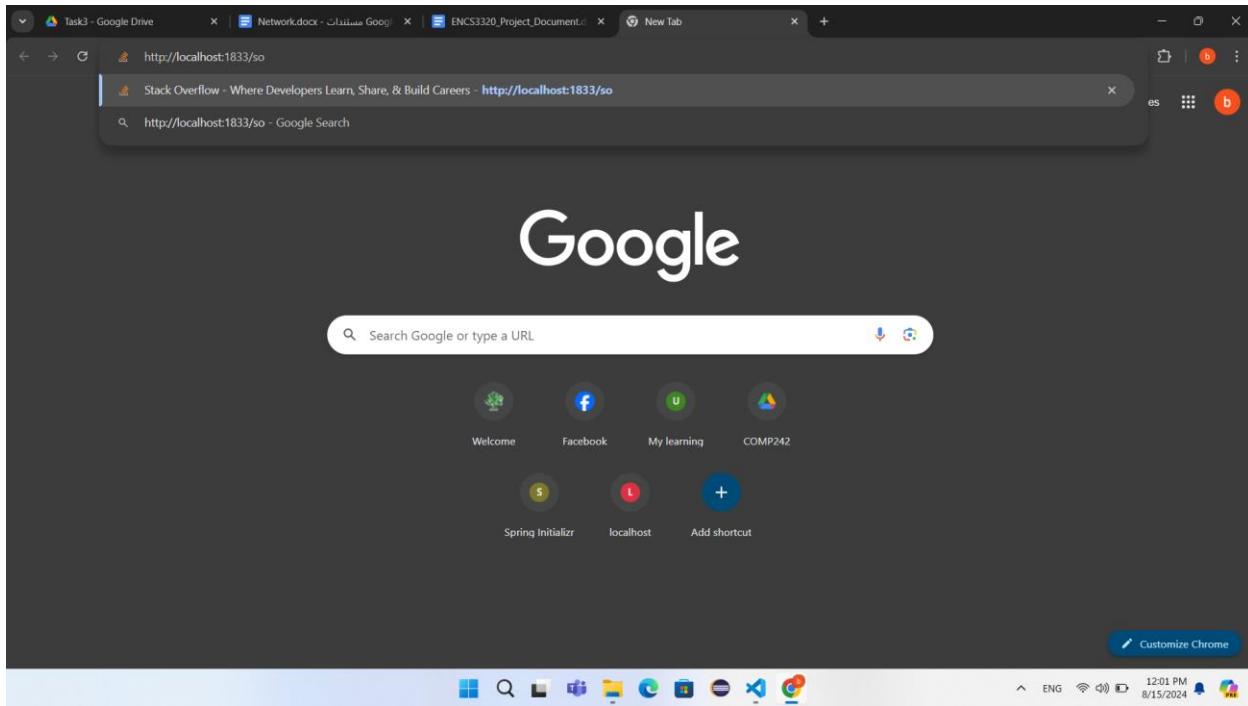


Figure 72: /so request

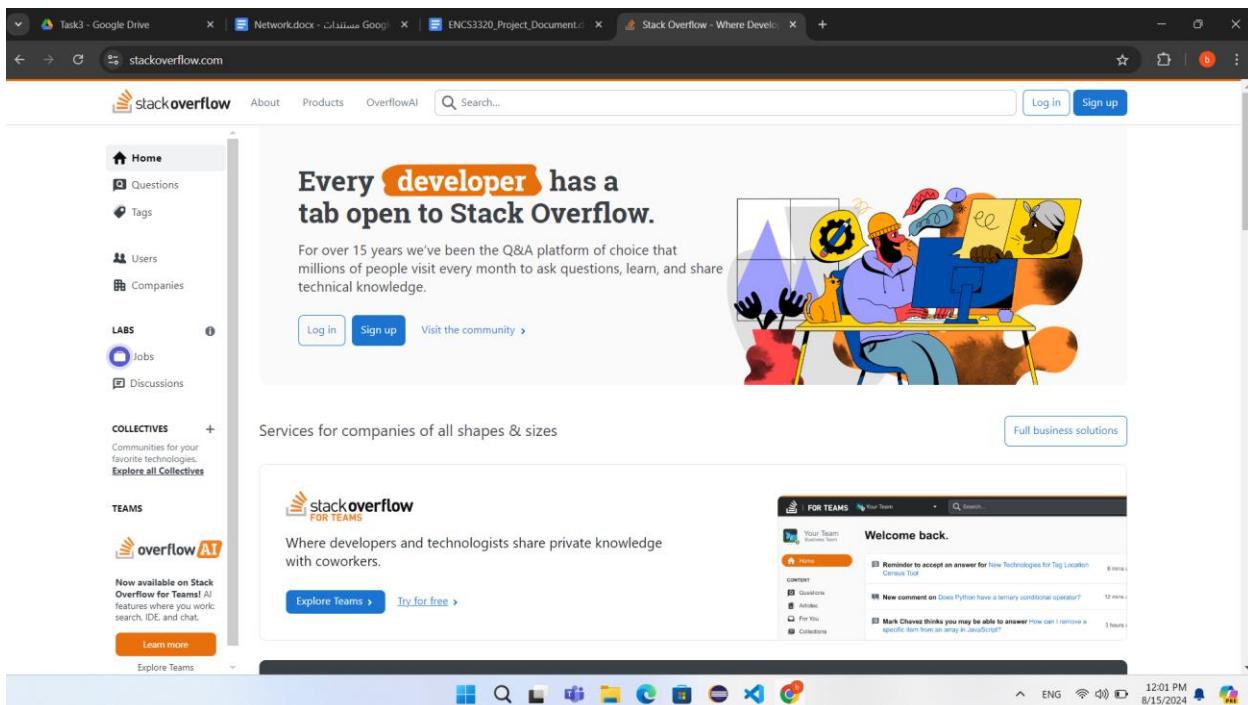


Figure 73:solution for /so

Here we search for <http://localhost:1833/so> and it goes to stackoverflow.com

b.

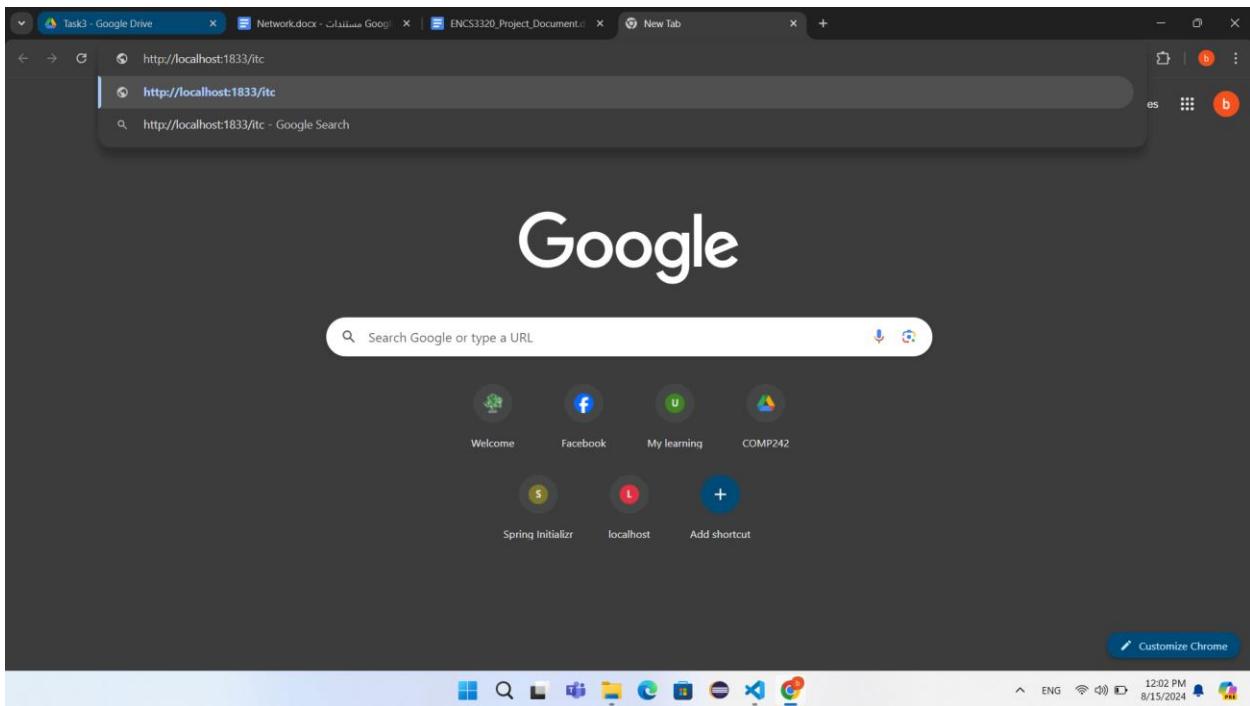


Figure 74: /itc request

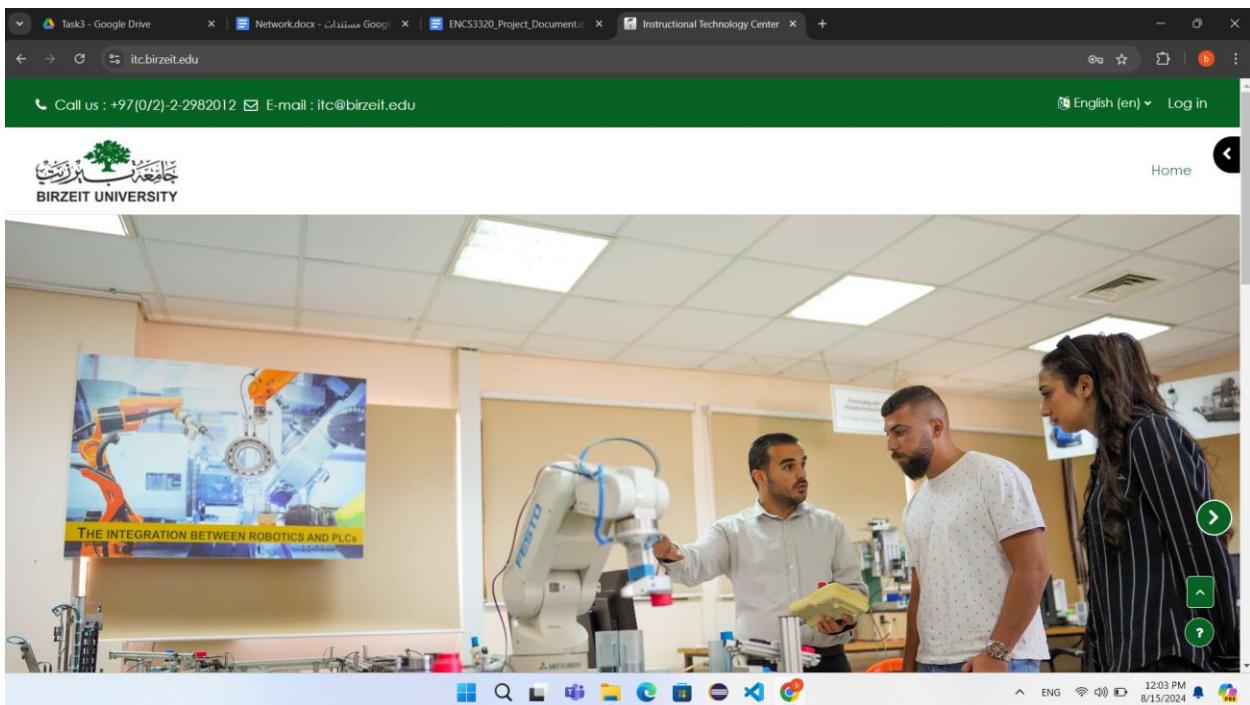


Figure 75: solution for /itc

Here we search for <http://localhost:1833/itc> and it goes to [itc.birzeit.edu](http://itc.birzeit.edu)

## 10.

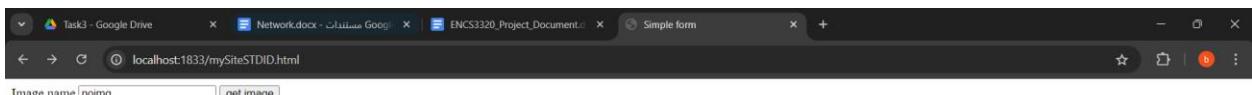
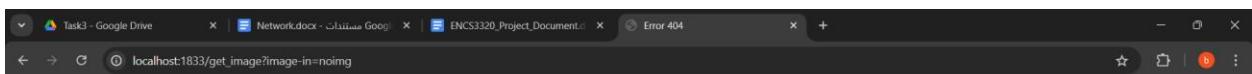


Figure 76: wrong image



### The file is not found

Zaid Mousa 1221833  
Mouath Masalmah 1220179  
Bahaa Bani Shamsaa 1220252  
ip: 127.0.0.1 port: 57495



Figure 77: error.html browser

Here when we search for an image does not exist the error.html page shows

## 11.

```
Received: GET /get_image?image-in-mouauth.jpg HTTP/1.1
Host: localhost:1833
Connection: keep-alive
sec-ch-ua: "Not)A;Brand";v="99", "Google Chrome";v="127", "Chromium";v="127"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:1833/mySiteSTDID.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Idea-2a3d4e=d2af89cd-da14-4c9c-96ac-2e645a6ab28b

Received:
Error: list index out of range
Traceback (most recent call last):
  File "c:\Users\dell\Desktop\Network Project\server.py", line 101, in <module>
    connectionSocket.send(error_content + F'<p> ip: {ip} port: {port} </p>'.encode())
ConnectionAbortedError: [WinError 10053] An established connection was aborted by the software in your host machine
PS C:\Users\dell\Desktop\Network Project|
```

Figure 78: http request on terminal

Here first it is asking the server for a file named mouauth.jpg then the request sent to the server then it tells the server what browser and operating system are used

## Problems and challenges:

We had a problem with the arabic page when we search for it , it shows like that:

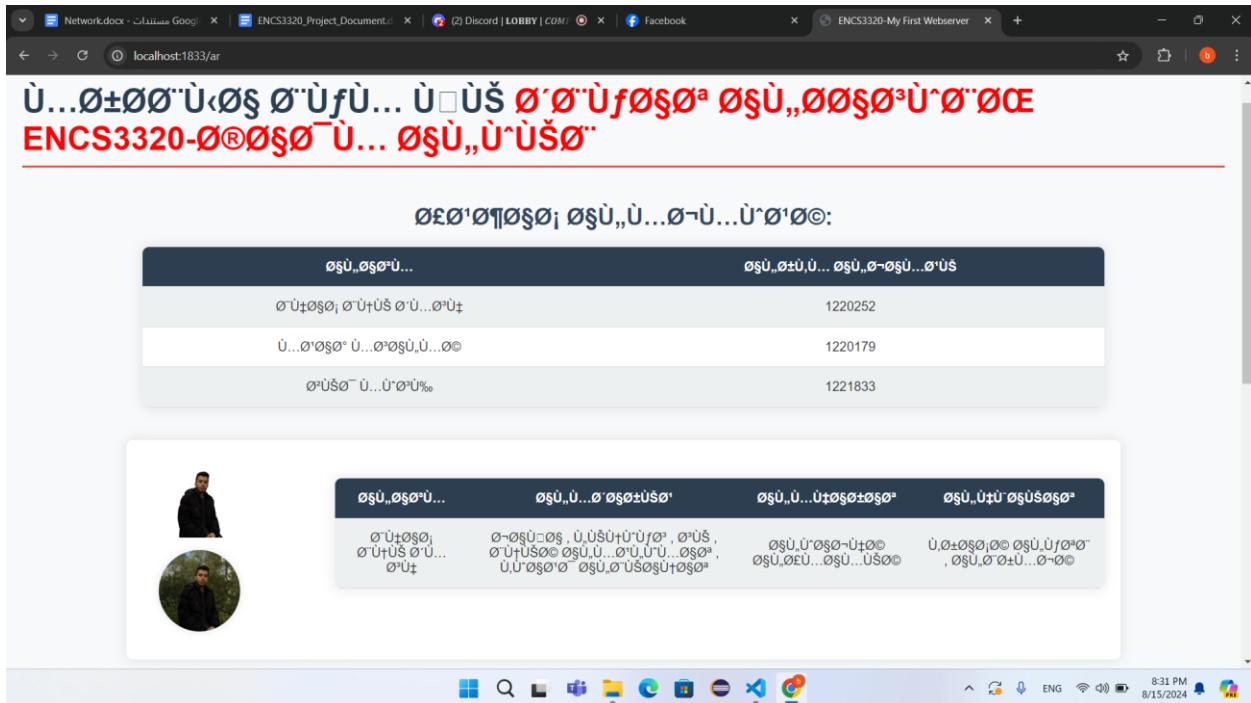
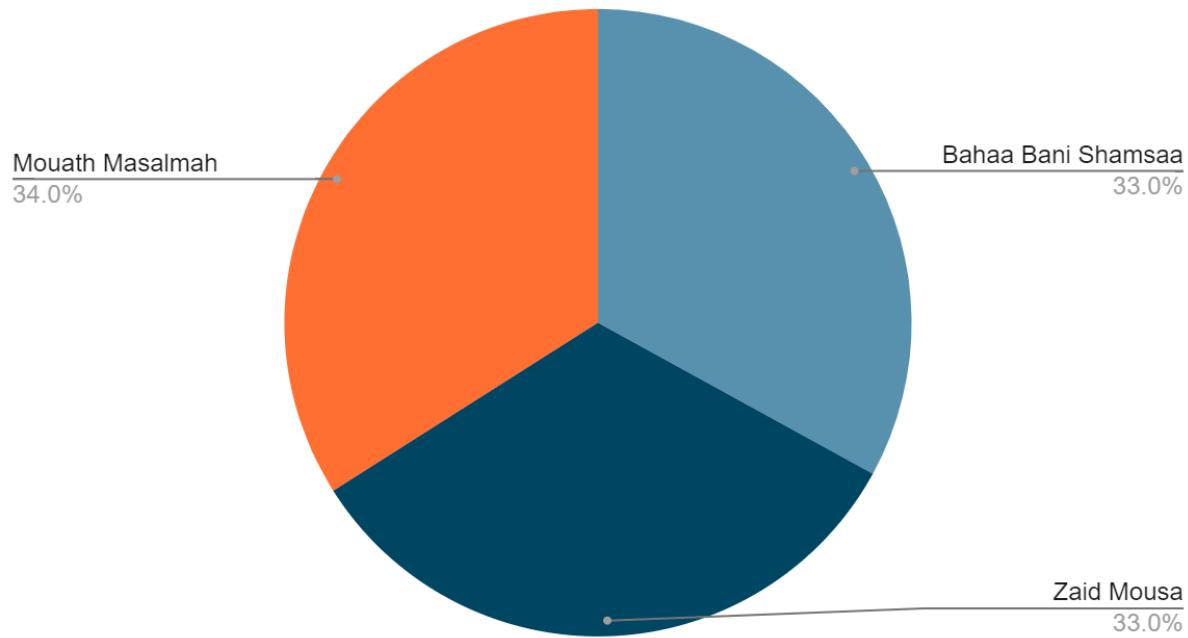


Figure 79: problem on /ar

and we solve it by adding `<meta charset="UTF-8">`

## Work Done:

Work Done



## Code and report:

Task1:

- 1.Mouath
- 2.A.Bahaa
- B.Bahaa
- C.Bahaa
- D.Mouath
- E.Zaid
- 3.Mouath
- 4.Zaid

Task 2:

1. Zaid , Mouath and Bahaa
- 2.Zaid and Mouath

Task 3:

1.Bahaa and Zaid

A.Bahaa

B.Bahaa

C.Bahaa

D.Bahaa

E.Bahaa

F.Bahaa

H.Bahaa

I.Bahaa

J.Bahaa

2.Mouath

3.Mouath

4.Zaid

5.Zaid

6.Bahaa

7.Bahaa

8.Mouath

9.A.Zaid

B.Zaid

10.Mouath

11.Mouath

# References

DNS : <https://www.cloudflare.com/learning/dns/what-is-dns/>

Round-trip time (RTT): <https://www.cloudflare.com/learning/cdn/glossary/round-trip-time-rtt/>

Hop (networking): [https://en.wikipedia.org/wiki/Hop\\_\(networking\)](https://en.wikipedia.org/wiki/Hop_(networking))

"Bad Request" error: <https://www.semrush.com/blog/400-bad-request/>

vowels: <https://www.grammarly.com/blog/vowels/>

port: <https://www.cloudflare.com/learning/network-layer/what-is-a-computer-port/>

html, css & python: <https://www.w3schools.com/>

wireshark:<https://www.comptia.org/content/articles/what-is-wireshark-and-how-to-use-it#:~:text=Wireshark%20is%20a%20network%20protocol,packet%20sniffer%20in%20the%20world.>