FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COMPUTER NETWORKS

ENCS3320

Project 1

Socket Programming

Prepared by

Raneem Daqa - Jana Abu Nasser

1202093 - 1201110

Instructor

Dr. Abdelkarim Awwad

BIRZEIT

January – 2023

Contents

Table of figures

1 Part 1

Ping Command

Ping is a command-line utility, available on virtually any operating system with network connectivity, which acts as a test to see if a networked device is reachable in other words it is used as a simple way to verify that a computer can communicate over the network with another computer or network device.

The ping command sends a request over the network to a specific device, it sends Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waits for a response, the two most important pieces of information that we can take from this command is How many of those responses are returned, and how long it takes for them to return from the destination computer.

Traceroute (tracert) Command

The Traceroute command (tracert) is a utility command used to measure and display the time it takes for the packets to travel between the user's computer and the destination IP address or domain and it gives the details about the path that a packet takes.

Name Server Lookup (Nslookup) Command

Name Server Lookup (Nslookup) is a network administration command-line tool for querying the Domain Name System (DNS) to obtain the mapping between a domain name and IP address or other DNS records, it is used in troubleshooting DNS-related problems.

1.1 Ping a device in the same network

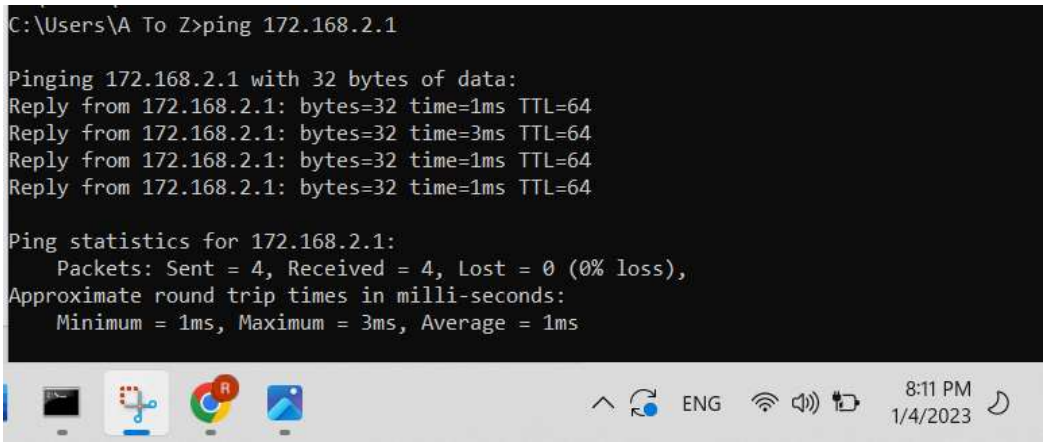


Figure 1: Ping a device in the same network

In the above figure, we pinged a device in the same network which resulted in 0% loss in the packets we sent and all 4 packets were received successfully by the other device and because the other device is in the same network we got the min, max and average time of 0ms.

1.2 ping www.yale.edu

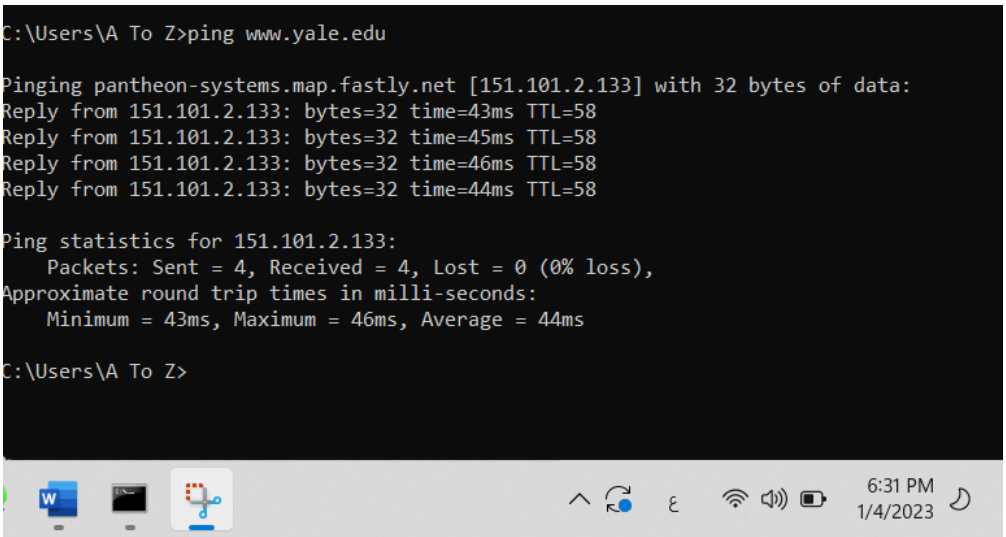


Figure 2: ping www.yale.edu

From figure 2 we can see that we have 0% loss in the packets (4 packets) we sent which means that each packet we sent reached www.yale.edu successfully and returned and that means that the connection of this network is a good use.

1.3 tracert www.yale.edu

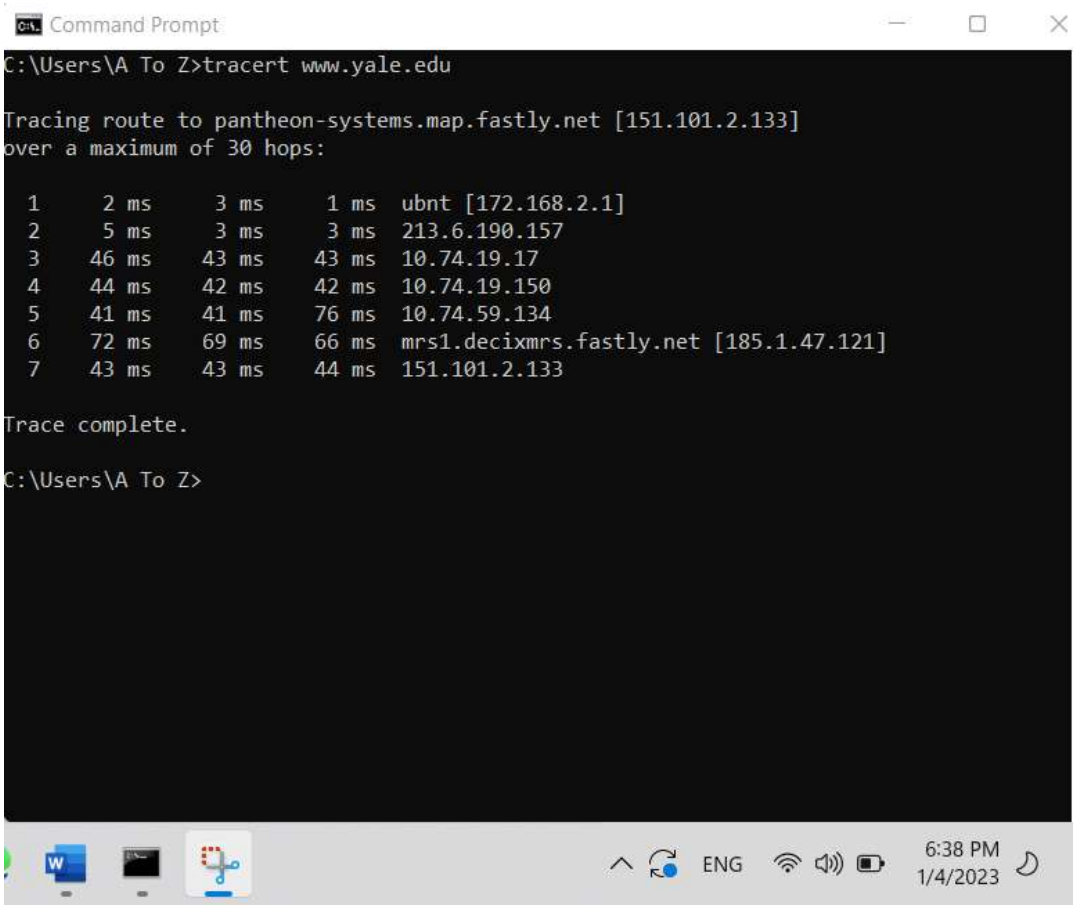


Figure 3: tracert www.yale.edu

From figure 3 we can see that the tracert command has identified 5 network devices including the router of our network with the IP address 172.168.2.1 and through to the target www.yale.edu with the IP 151.101.2.133.

1.4 nslookup www.yale.edu

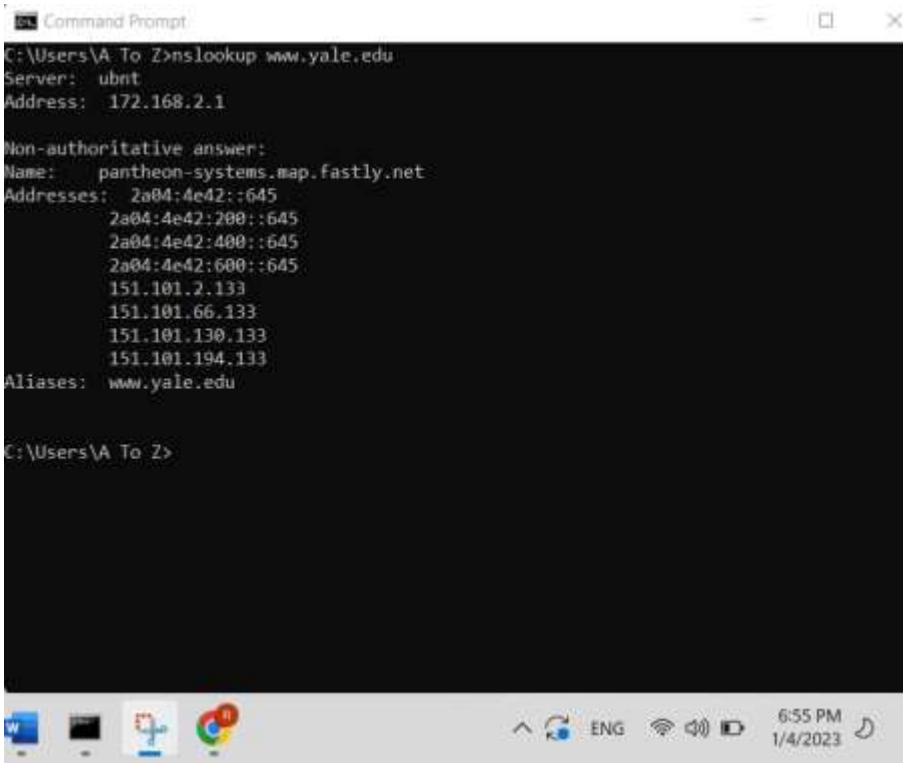


Figure 4: nslookup www.yale.edu

From figure 4 we first get the non-authoritative answer which means that the DNS server that provided the answer to my query isn't directly responsible for this domain name but it knows the name because it was previously resolved and it revived that information from a cached record and we get the name of the server which is www.yale.edu and the IP address of the server which is 151.101.194.133.

2 Part2

2.1 TCP

2.1.1 Same computer

Here the client sends the numbers from 0 to 1000,000 to a TCP server listening on port 5566 on the same computer (localhost). Also measured the time required to send the packets using python codes.

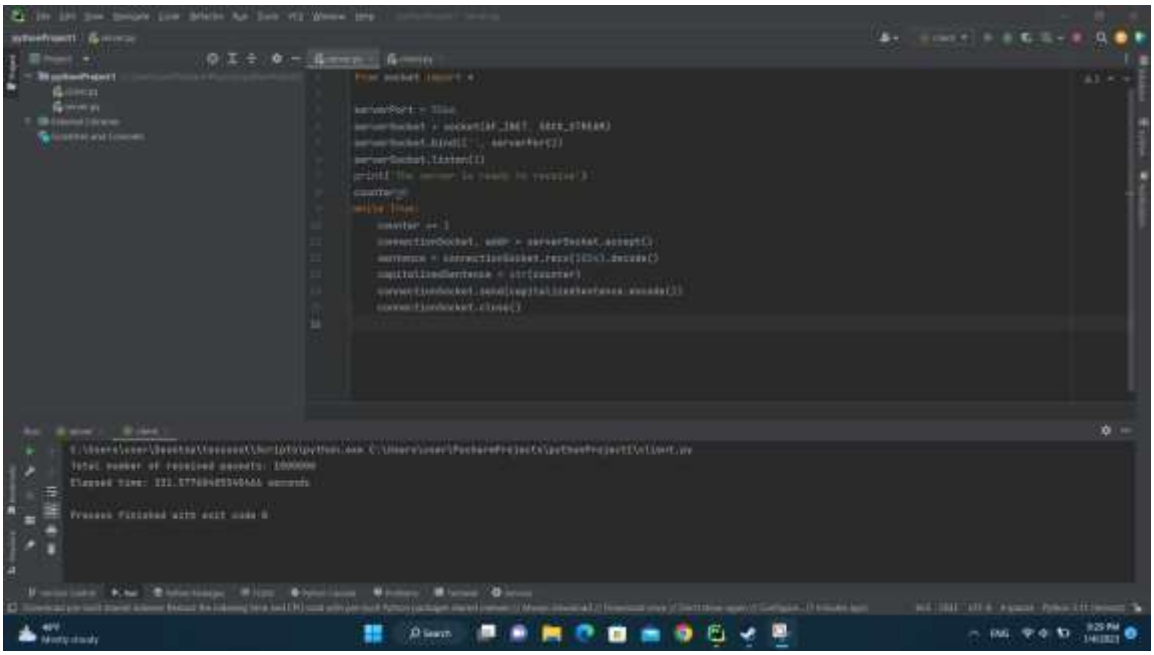


Figure 5: TCP Server code receives packets from client on the same computer

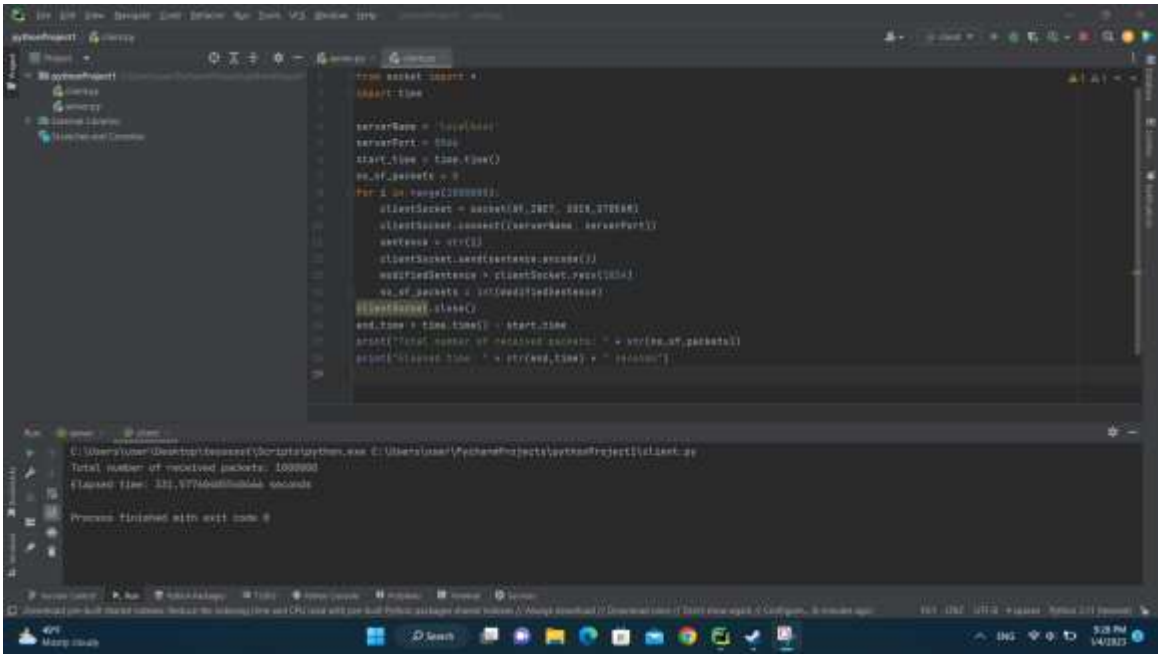


Figure 6: TCP Client code sends packets to the server on the same computer

2.1.2 Two different computers connected by a cable directly or through a switch

Here the client sends the numbers from 0 to 1000,000 to a TCP server listening on port 5566 5566 on the Two different computers connected by a cable directly. Also measured the time required to send the packets using python codes.

As shown , all the 1000000 numbers received to the server without any loss.

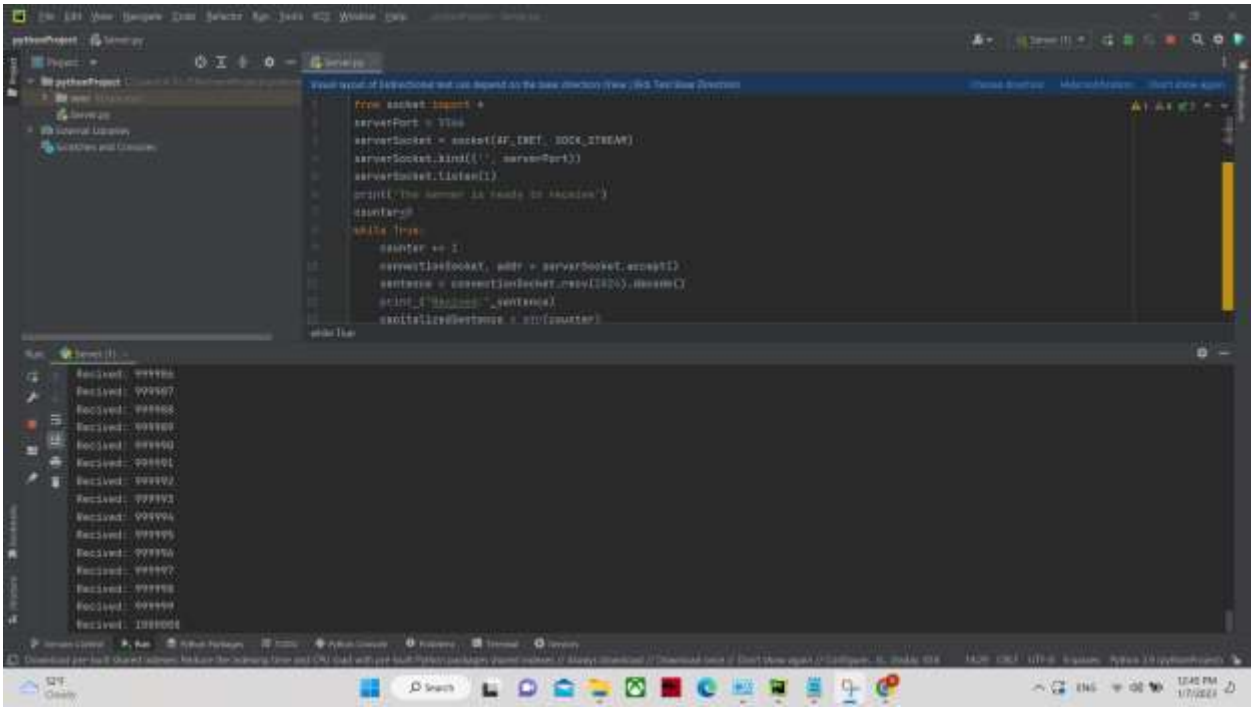


Figure 7: TCP Server code receives packets from client on two different computers connected by a cable directly

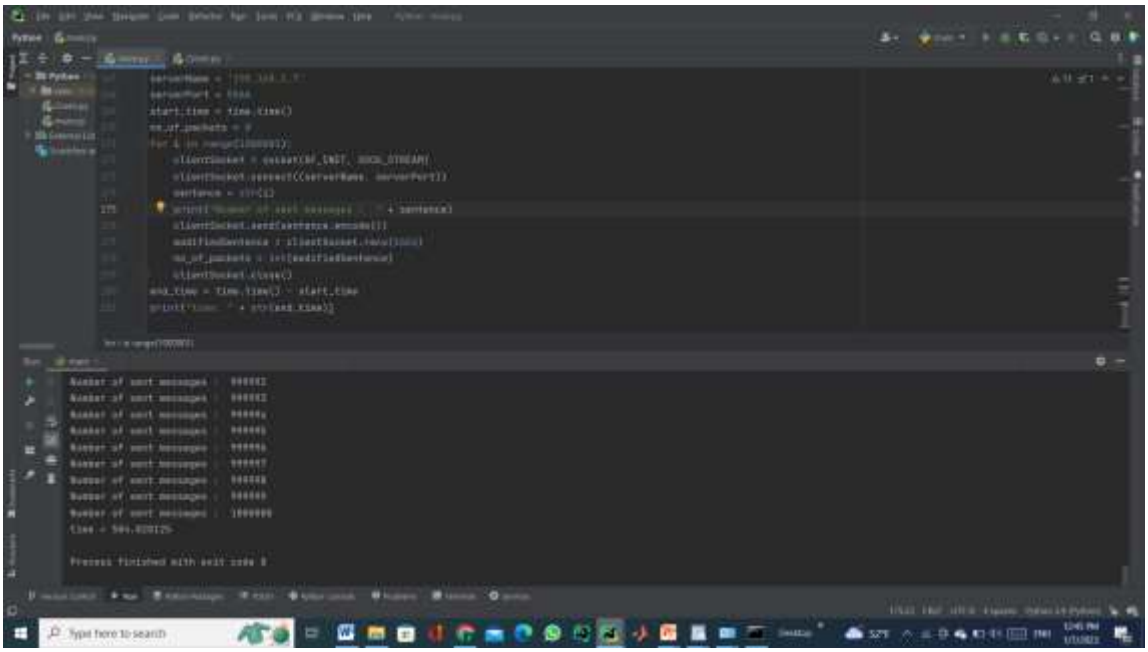


Figure 8: TCP Client code sends packets to the server on two different computers connected by a cable directly

2.1.3 Two different computers connected through WiFi

Here, the client sends the numbers from 0 to 1000,000 to a TCP server listening on port 5566 on the Two different computers connected through WiFi. Also measured the time required to send the packets using python codes.

As shown , the 1000000 numbers received to the server without loss.

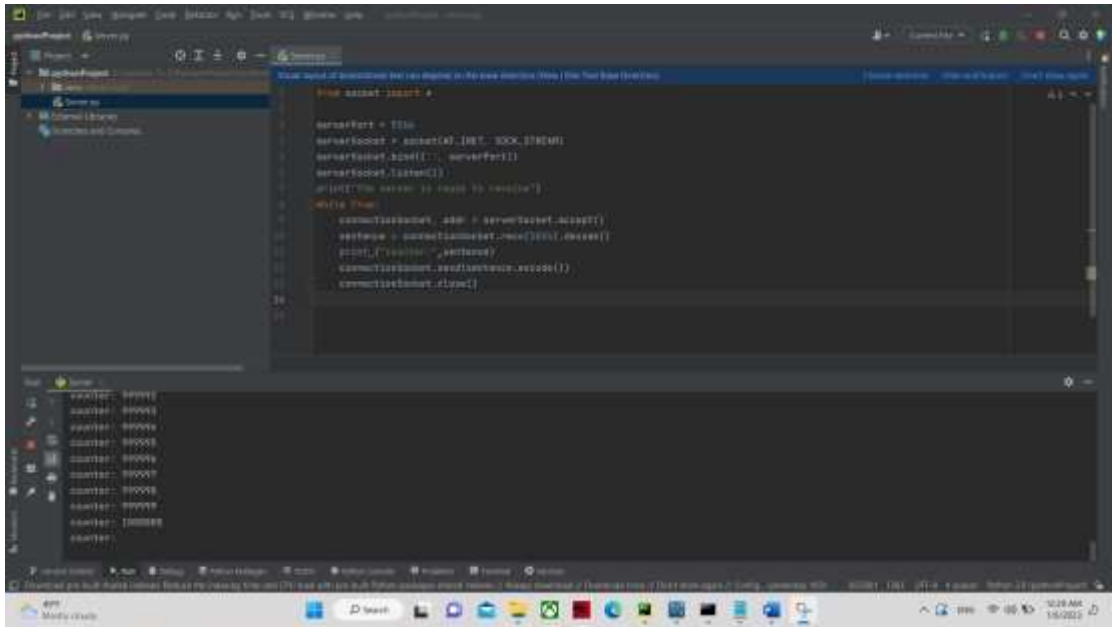


Figure 9: TCP Server code receives packets from client on two different computers connected through WiFi

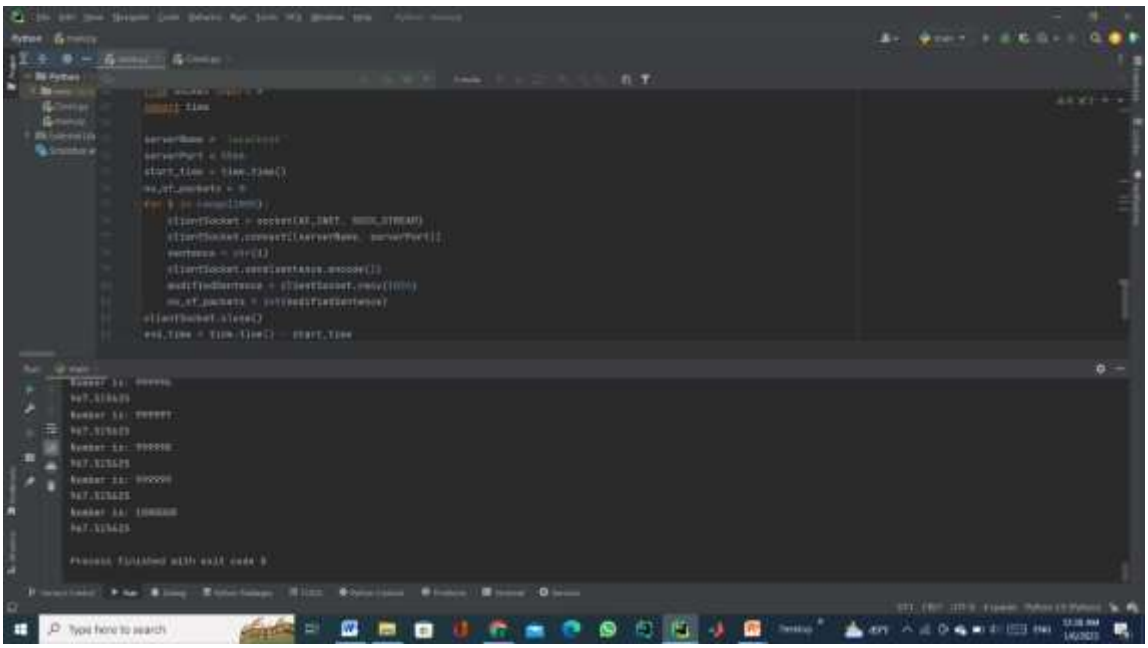


Figure 10: TCP Client code sends packets to the server on two different computers connected through WiFi

We noticed after trying the three cases that sending in the same computer (localhost) is the fastest and the slowest one when we send the packets between two different computers connected through WiFi.

2.2 UDP

2.2.1 On Same computer

Here, the client sends the numbers from 0 to 1000,000 to a UDP server listening on port 5566 on the same computer (localhost). Also measured the time required to send the packets using python codes.

As shown , all the 1000000 numbers received to the server without any loss.

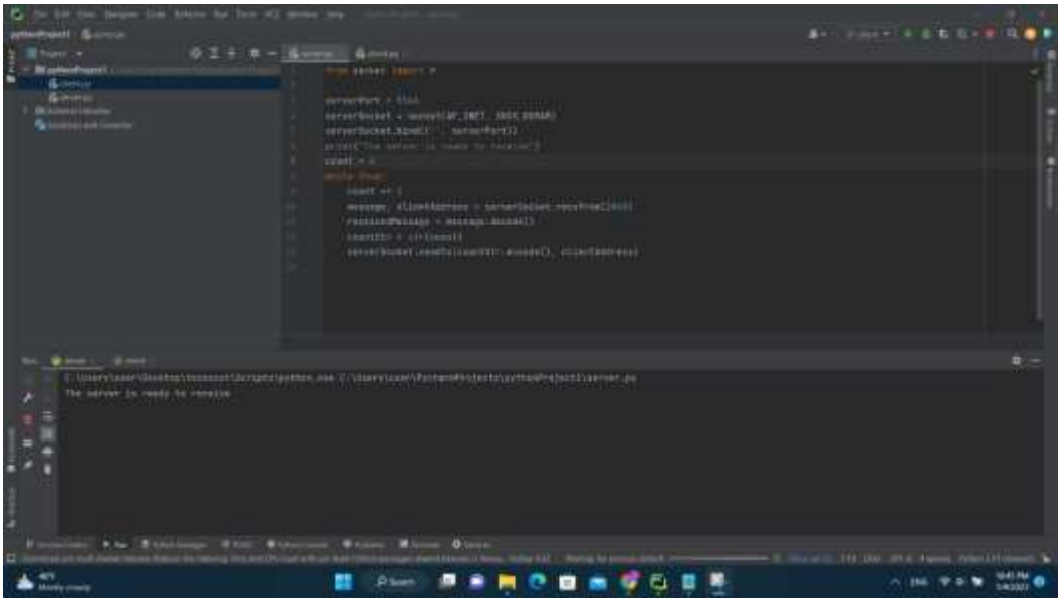


Figure 11: UDP Server code receives packets from client on the same computer

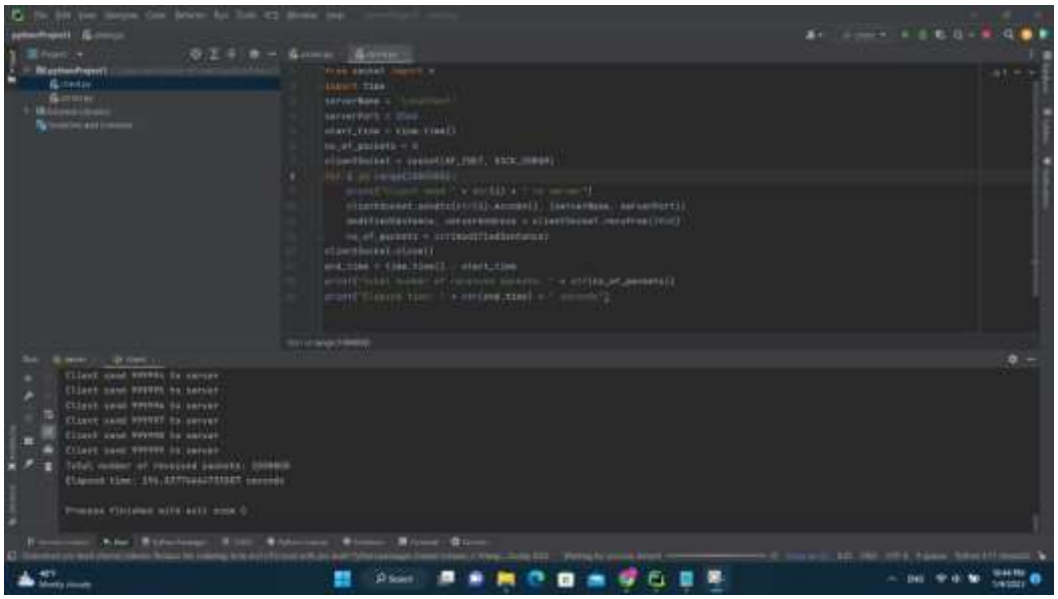


Figure 12: UDP Client code sends packets to the server on the same computer

2.2.2 Two different computers connected by a cable directly or through a switch

Here, the client sends the numbers from 0 to 1000,000 to a UDP server listening on port 5566 on the Two different computers connected by a cable directly. Also measured the time required to send the packets using python codes.

As shown , the 1000000 numbers received to the server with some loss but it's normal while using UDP server.

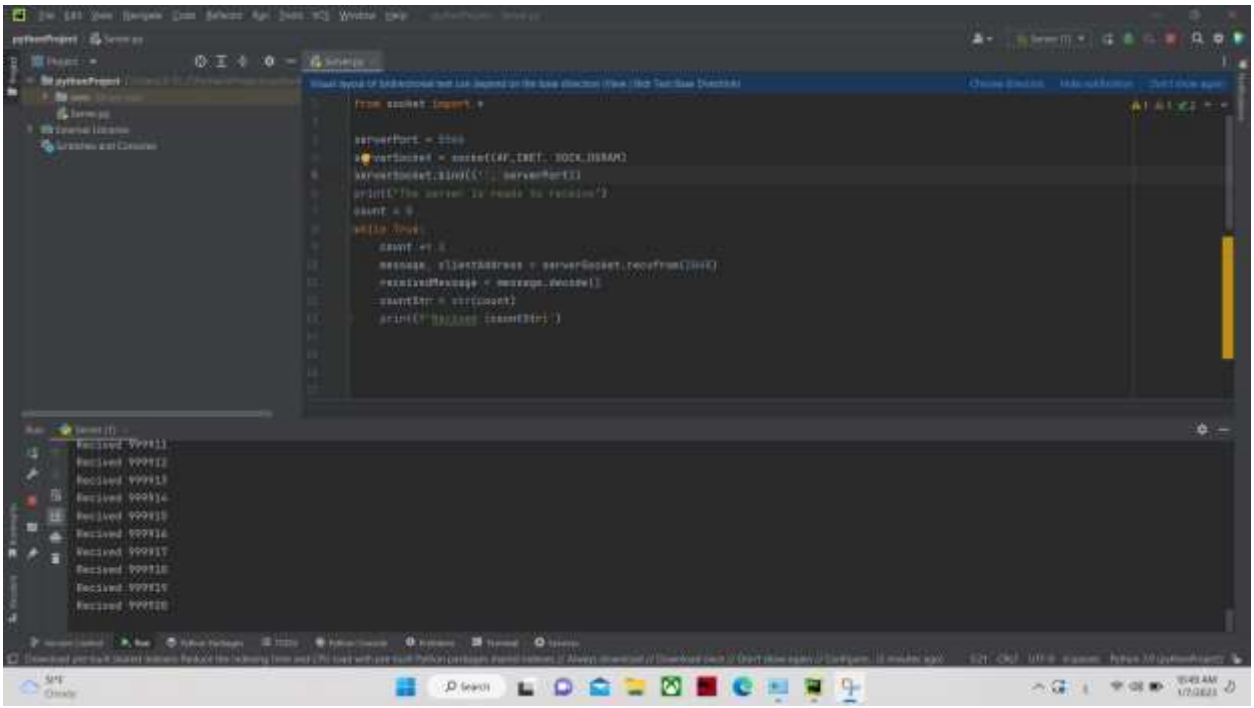


Figure 13: UDP Server code receives packets from client on two different computers connected by a cable directly

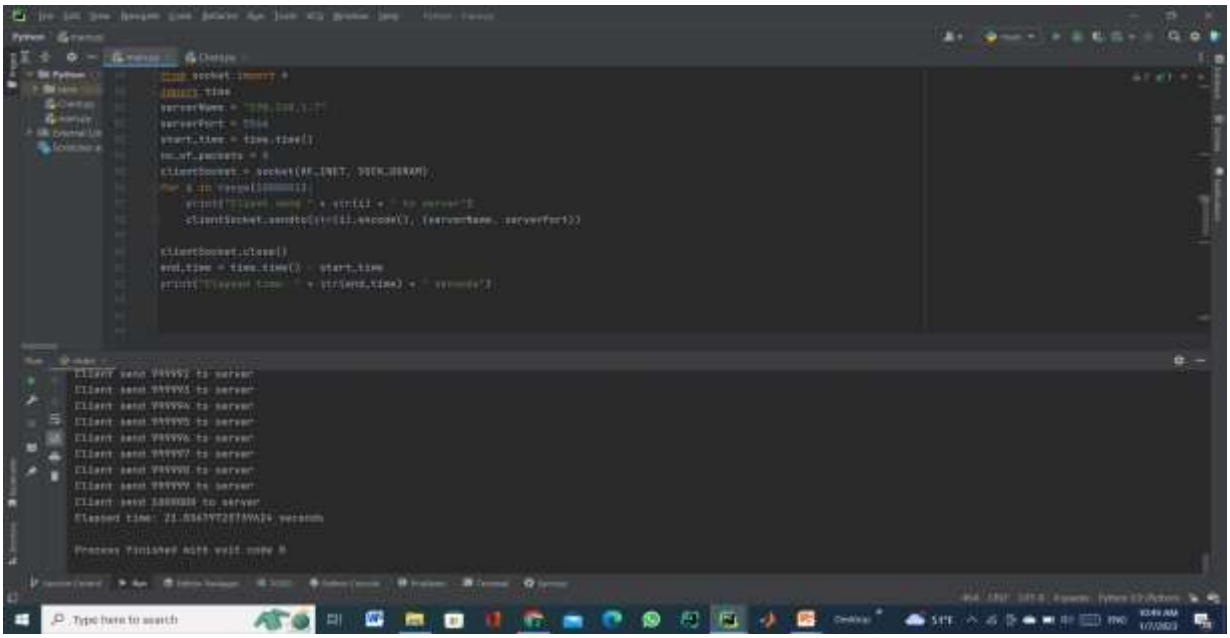


Figure 14: UDP Client code sends packets to the server on two different computers connected by a cable directly

2.2.3 Two different computers connected through WiFi

Here, the client sends the numbers from 0 to 1000,000 to a UDP server listening on port 5566 on the Two different computers connected through WiFi. Also measured the time required to send the packets using python codes.

As shown , the 1000000 numbers received to the server with some loss but it's normal while using UDP server.

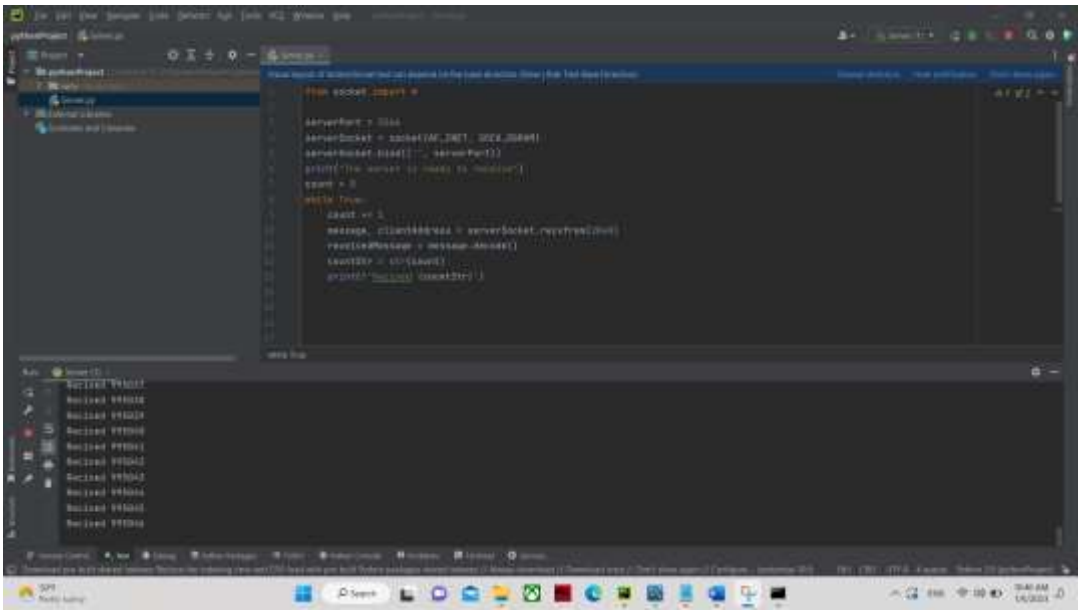


Figure 15: UDP Server code receives packets from client on two different computers connected through WiFi

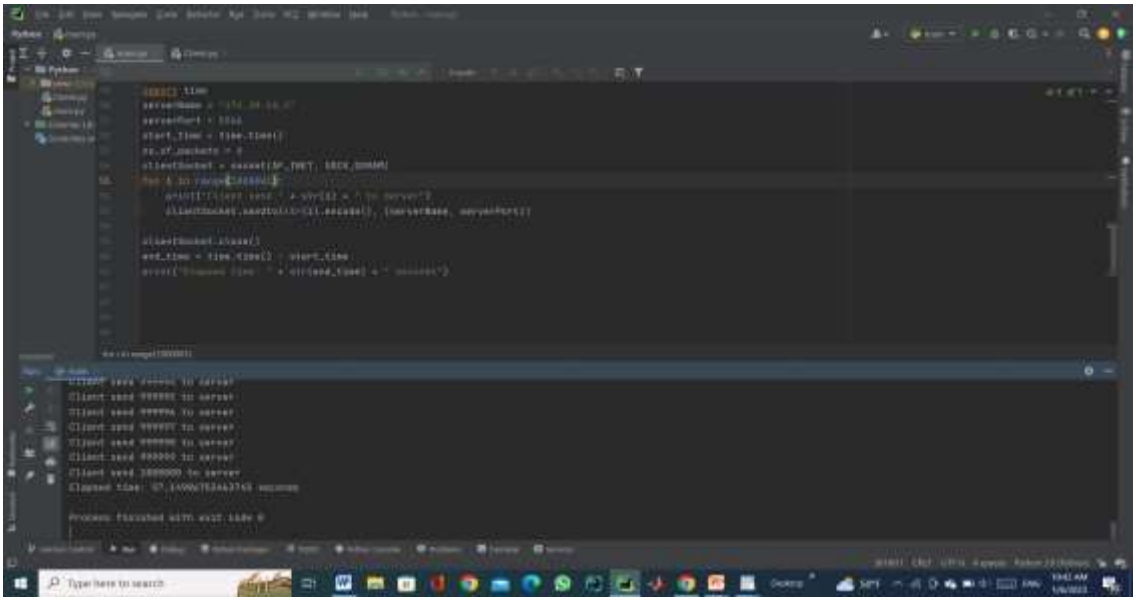


Figure 16: UDP Client code sends packets to the server on two different computers connected through WiFi

We noticed after trying TCP server and UDP server to send the numbers from 0 to 1000000 that the UDP is faster and takes less time than the TCP.

3 Part 3

In this part, the aim was to design a web page using HTML and CSS basics to act as a client with a local server created using socket programming over Python.

3.1 Main web page request

When http://localhost:7788/index.html , or http://localhost:7788/main_en.html , or http://localhost:7788/en or http://localhost:7788/ is opened, a request is sent from the browser (client) to the server requesting main_en.html. The figure below shows the requested information.

```
The server is ready to receive


IP: 127.0.0.1, Port: 54099
GET /main_en.html HTTP/1.1
Host: localhost:7788
Connection: keep-alive
sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108", "Microsoft Edge";v="108"
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/108.0.0.0 Safari/537.36 Edg/108.0.1462.54
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Purpose: prefetch
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
```

Figure 17: Browser main_ar.html request shown on terminal

main_en.html contains multiple references such as .css file, another .html file, and two pictures, these objects are also requested and printed on the terminal.

This is the title of the web page, “ENCS3320-Simple Webserver”.

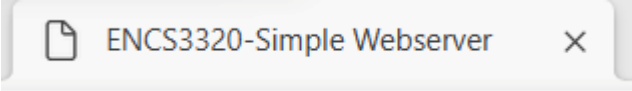


Figure 18: Web page title

This is the welcoming message on our web page which contains some of the requirements such as the shown message and colored texts. The web page shows up after the browser sends a request to the server and gets a response containing it.

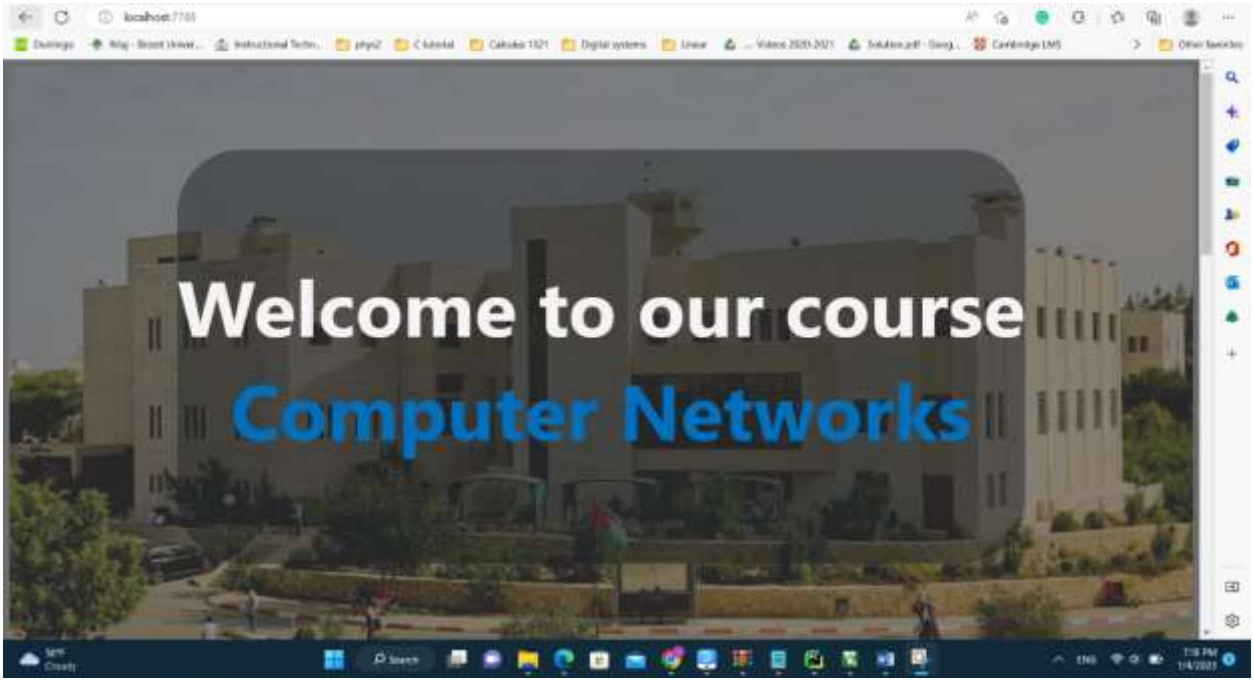


Figure 19: Web page welcoming message

The below figure shows an example of a group member and its data. However, each group member is shown on a separate page.

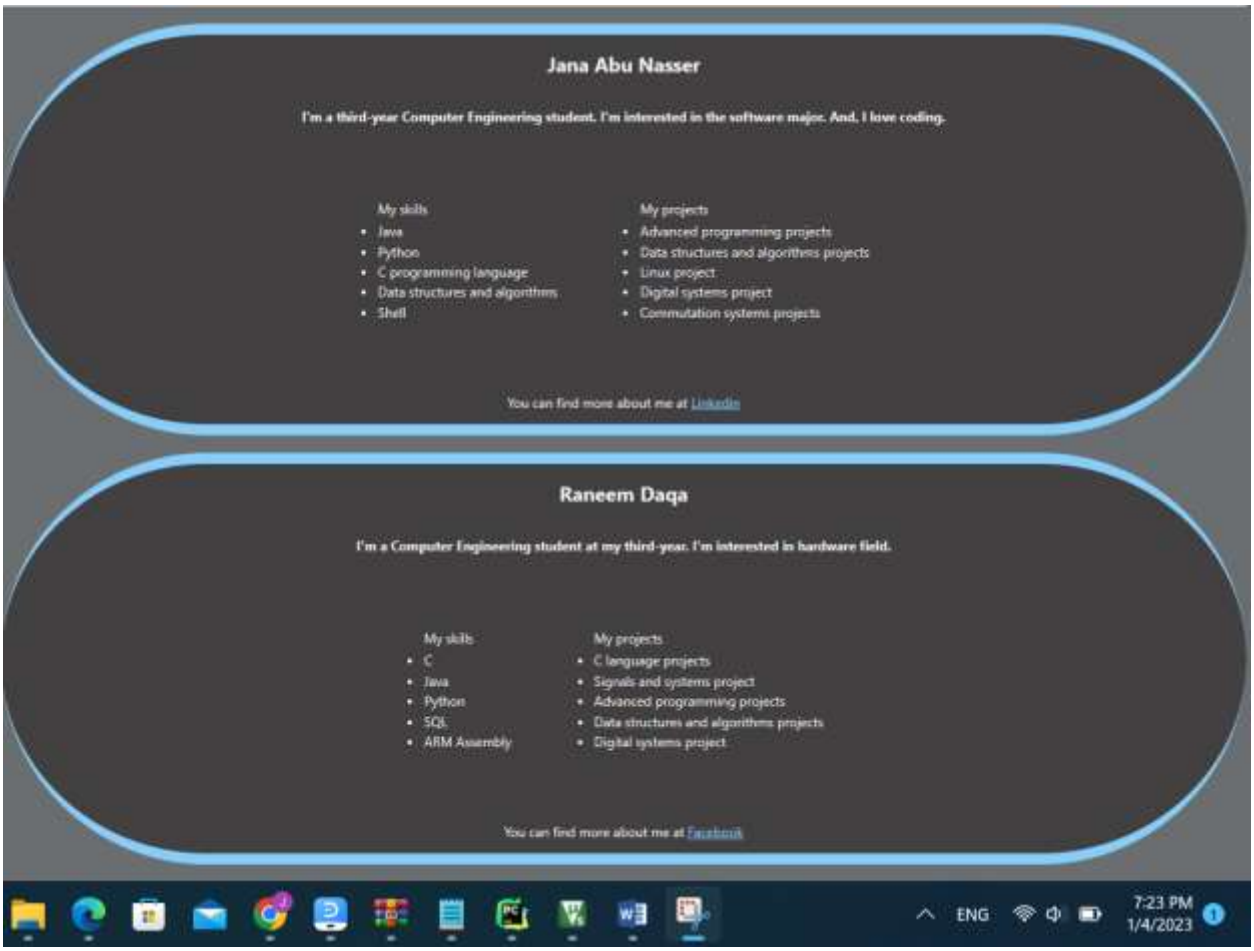


Figure 20: Group member representation on the Web page

Figure 21: Web page footer

This figure displays the footer of the web page, and it contains a link to an HTML file and a URL.

3.2 Main web Arabic version

When http://localhost:7788/main_ar.html , or http://localhost:7788/ar is opened, a request is sent from the browser (client) to the server requesting main_ar.html which is the Arabic version of main_en.html. The figure below shows the requested information.

Figure 22: Browser main.html request shown on terminal for Arabic version

main_ar.html contains multiple references such as .css file, another .html file, and two pictures, these objects are also requested and printed on the terminal.

This is the welcoming message on our Arabic web page which contains some of the requirements such as the shown message and colored texts. The web page shows up after the browser sends a request to the server and gets a response containing it.

Figure 23: Web Arabic page welcoming message

The below figure shows an example of a group member and its data in Arabic. However, each group member is shown on a separate page.

Figure 24: Group member representation on the Web Arabic page

Figure 25: Web Arabic page footer

This figure displays the footer of the web page, and it contains a link to an HTML file and a URL.

3.2 When clicking on the “Visit us!” request

It was required to add a link to a local HTML file. When the user clicks on Visit us, the browser requests “visit_us.html” which is a local HTML page referenced by main.html. The figure below is the request message information of the browser.

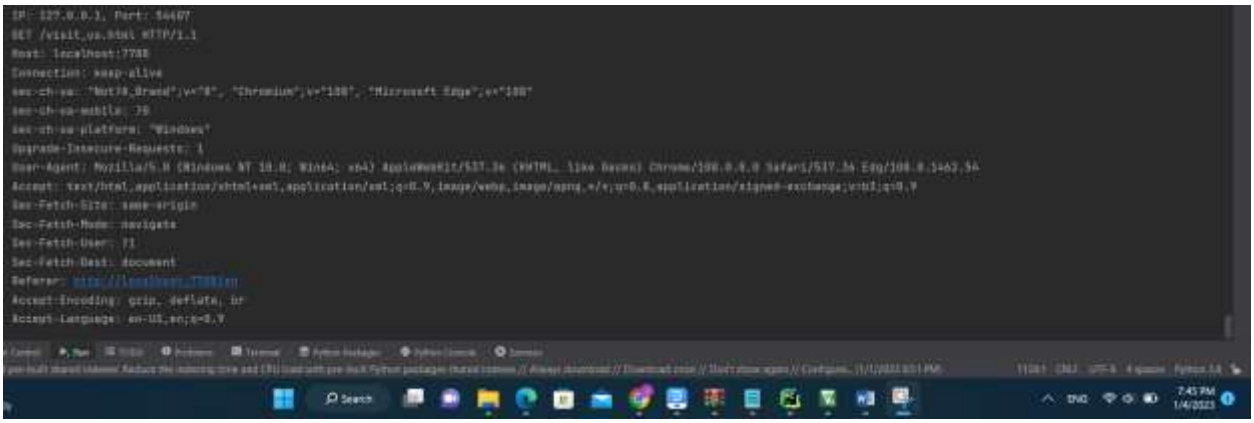


Figure 26: Browser request when clicking on "Visit us!"

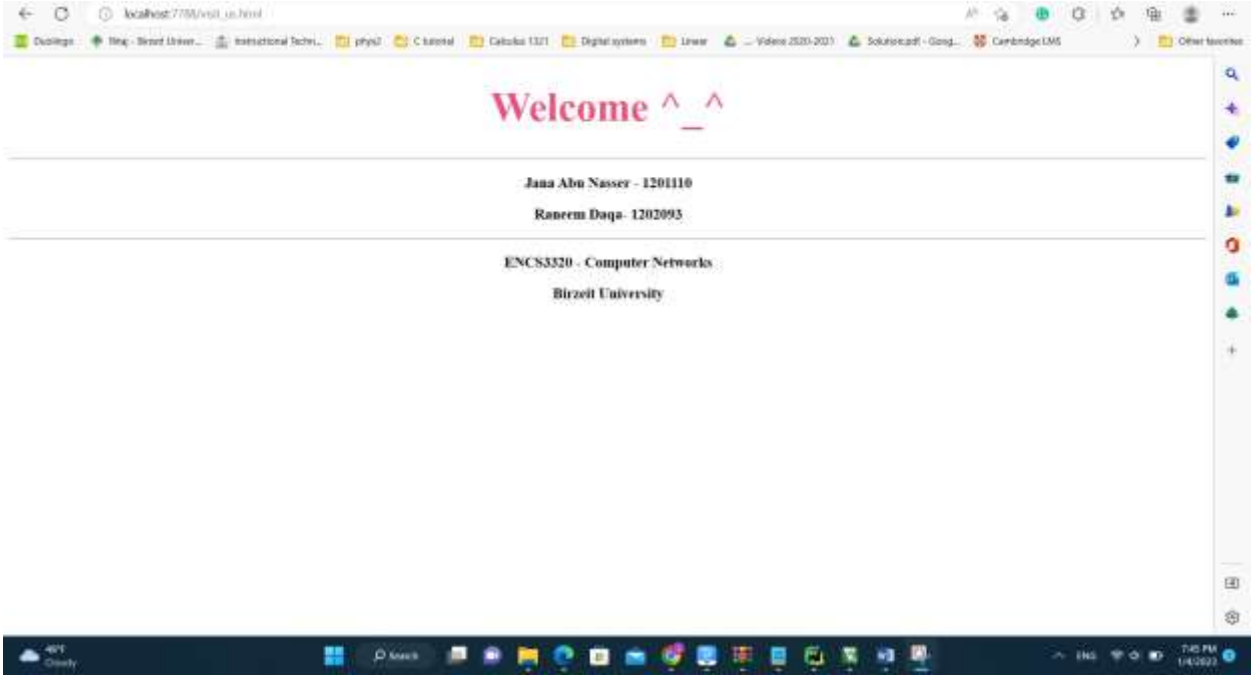


Figure 27: Visit us! Page

The figure above is the Visit us! HTML page which contains a message and group member names, IDs. And, course ID and name, and university name.

3.3 Requesting HTML file

Sometimes client tends to request any HTML file he wants. It was covered in the server that the user can request any HTML file. Hence, the below figure is the request information of a browser requesting the “empty.html” file.

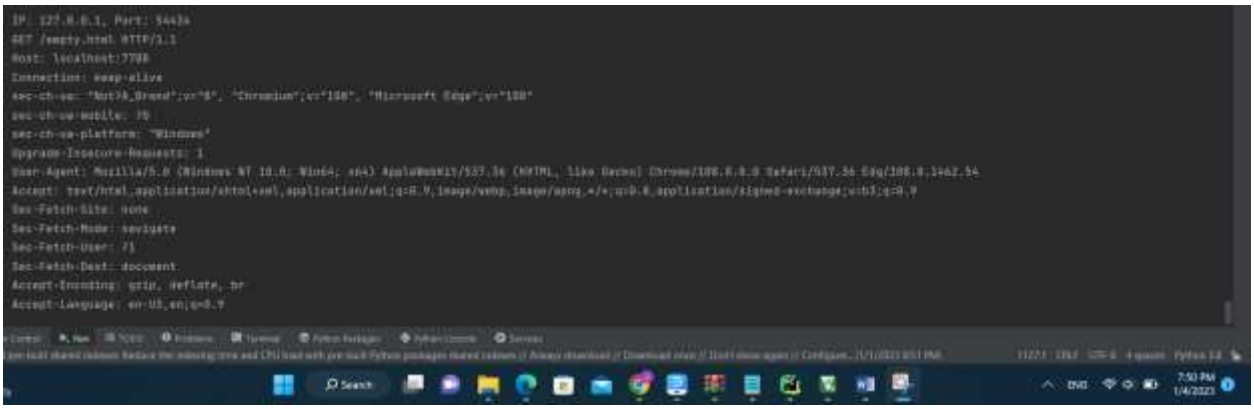


Figure 28: Browser request information when requesting HTML file

The figure below is the empty.html page which is an empty page that contains “empty” text only. It was used to ensure that the server works correctly.

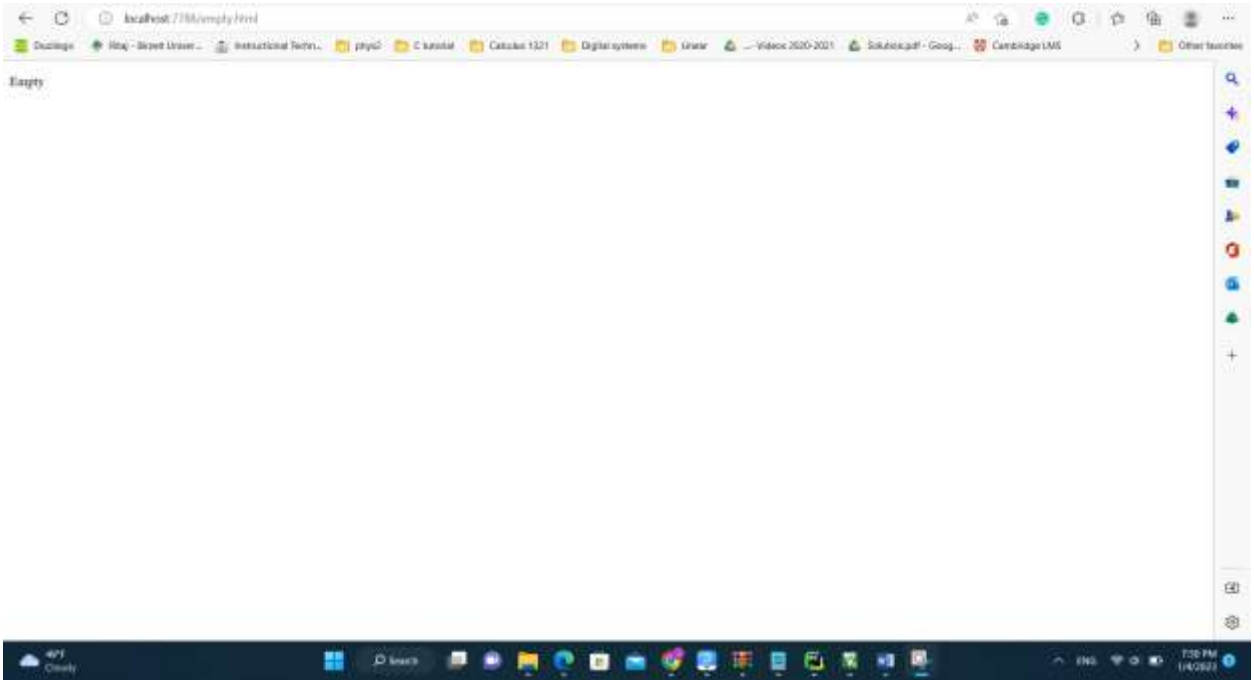


Figure 29: empty.html page

3.4 Requesting CSS file

Same as HTML file. The user can request and .css file he wants through the server.

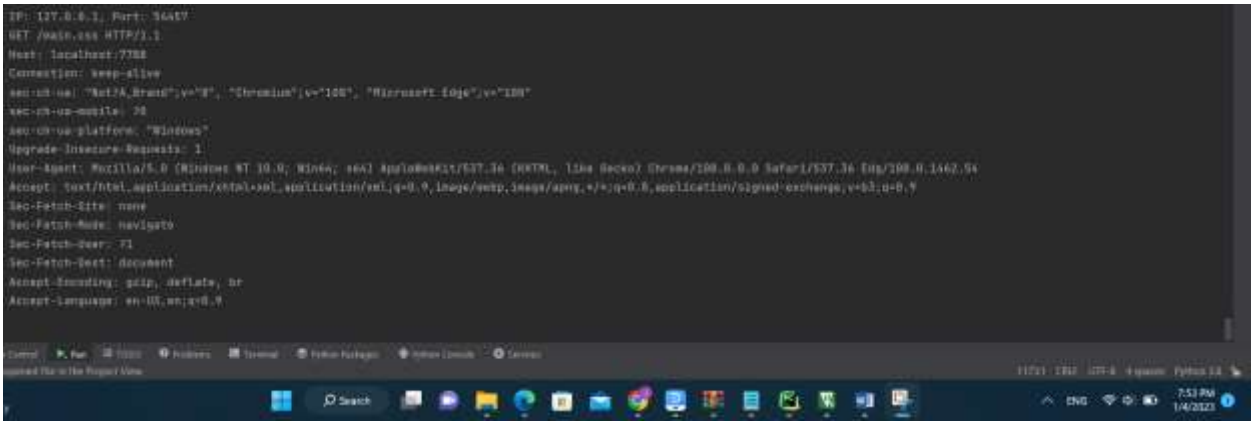
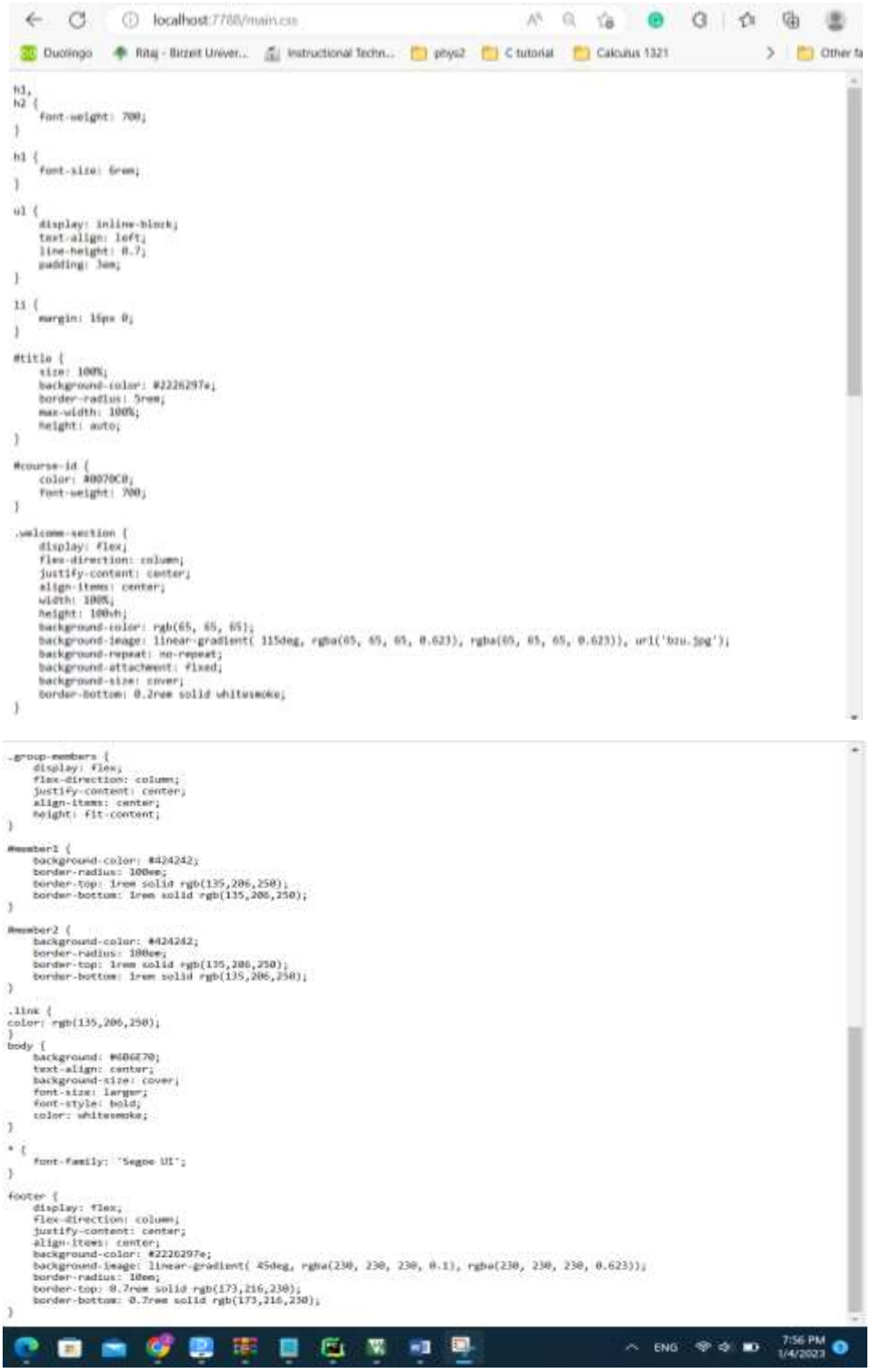


Figure 30: Browser request message when requesting .css file

The figure below shows that the server responded with a .css file that contains styles only, and it is represented in plan text.



```
h1,
h2 {
    font-weight: 700;
}

h1 {
    font-size: 6rem;
}

ul {
    display: inline-block;
    text-align: left;
    line-height: 0.7;
    padding: 3em;
}

li {
    margin: 15px 0;
}

#title {
    size: 100%;
    background-color: #2226297e;
    border-radius: 5rem;
    max-width: 100%;
    height: auto;
}

#course-id {
    color: #0070C0;
    font-weight: 700;
}

.welcome-section {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 100%;
    height: 100vh;
    background-color: rgb(65, 65, 65);
    background-image: linear-gradient( 115deg, rgba(65, 65, 65, 0.623), rgba(65, 65, 65, 0.623)), url('bzu.jpg');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-size: cover;
    border-bottom: 0.2rem solid whitesmoke;
}
```

```
.group-members {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    height: fit-content;
}

#member1 {
    background-color: #424242;
    border-radius: 100em;
    border-top: 1rem solid rgb(135,206,250);
    border-bottom: 1rem solid rgb(135,206,250);
}

#member2 {
    background-color: #424242;
    border-radius: 100em;
    border-top: 1rem solid rgb(135,206,250);
    border-bottom: 1rem solid rgb(135,206,250);
}

.link {
color: rgb(135,206,250);
}
body {
    background: #6D6E70;
    text-align: center;
    background-size: cover;
    font-size: larger;
    font-style: bold;
    color: whitesmoke;
}

* {
    font-family: "Segoe UI";
}

footer {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    background-color: #2226297e;
    background-image: linear-gradient( 45deg, rgba(230, 230, 230, 0.1), rgba(230, 230, 230, 0.623));
    border-radius: 10em;
    border-top: 0.7rem solid rgb(173,216,230);
    border-bottom: 0.7rem solid rgb(173,216,230);
}
```

Figure 31: CSS code

3.5 Requesting .png image

A client may request an image of type .png, this case is also treated as other requests. Meanwhile, the server responses with the image to display it on the browser. The figure below shows the browser's request information.

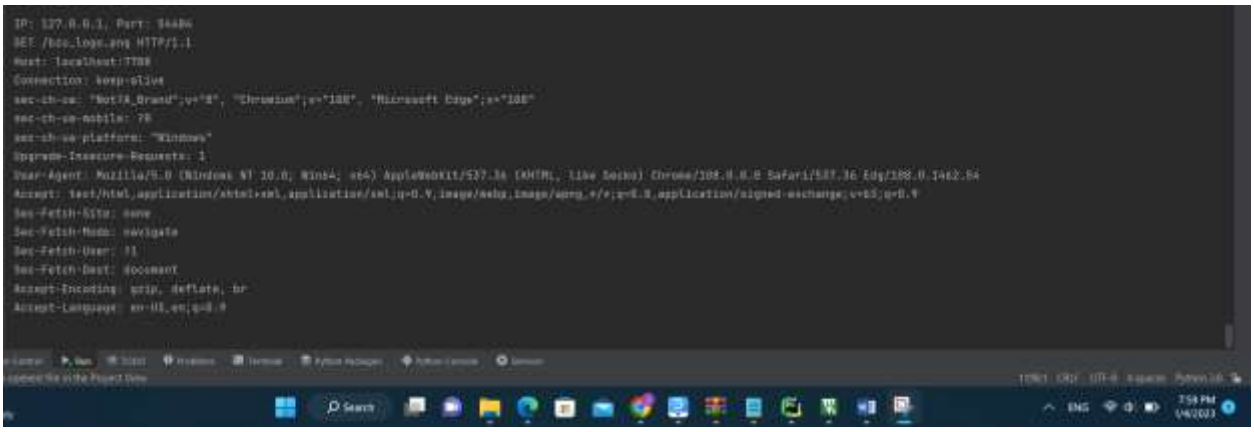


Figure 32: Browser request message when requesting .png image

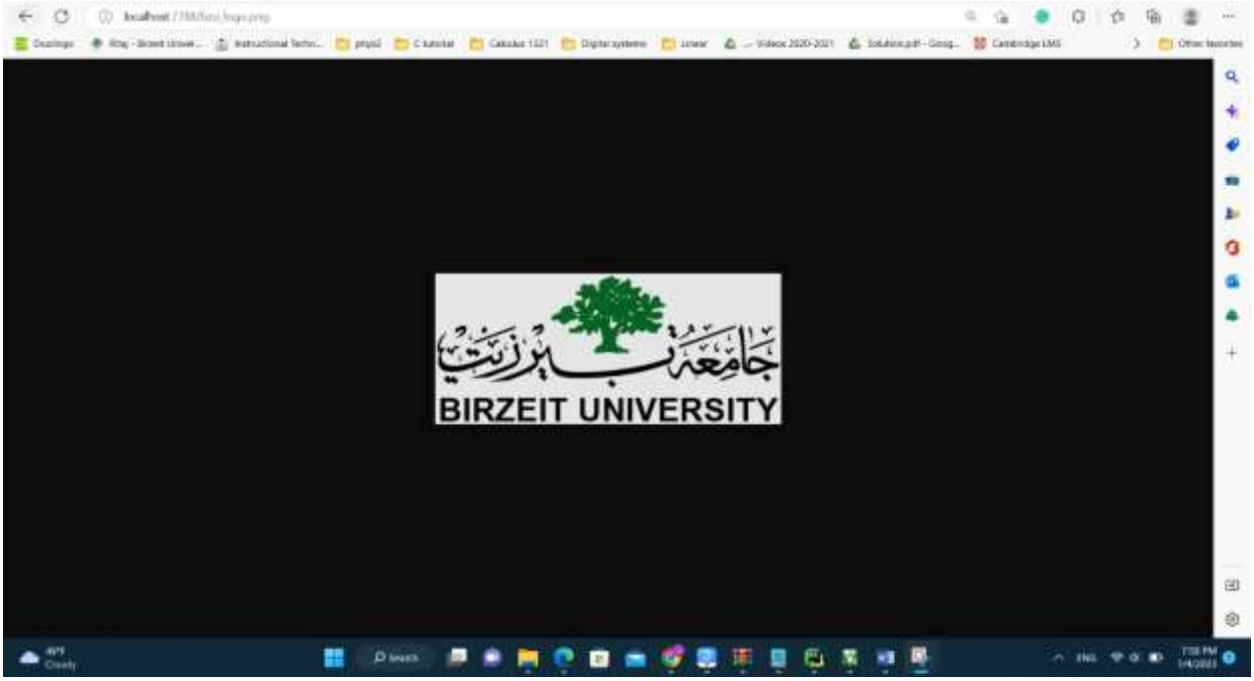


Figure 33: .png image displayed on the browser

3.6 Requesting .jpg image

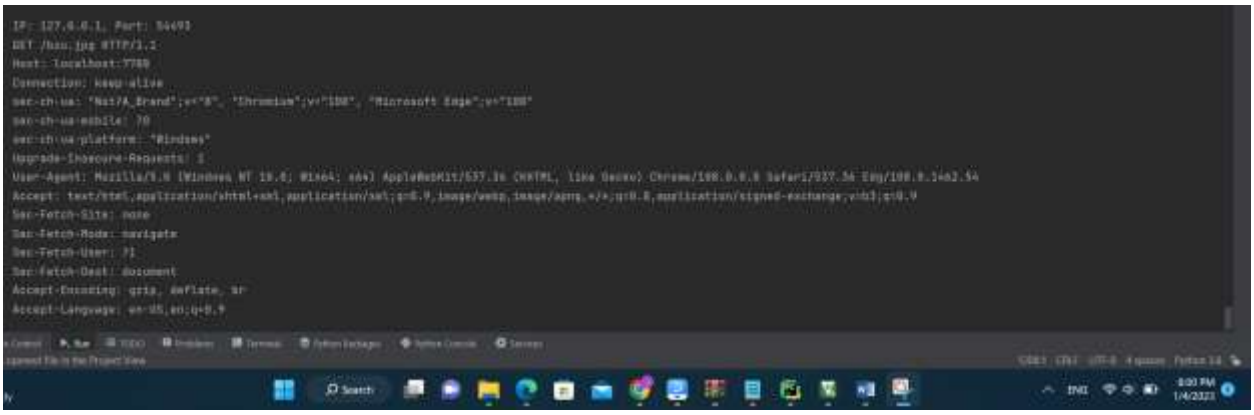


Figure 34: : Browser request message when requesting .jpg image

Figure 35: .jpg image displayed on the browser

3.7 Requesting non-existing file

When the browser (client) requests a non-existing file, we're required to handle that case by responding with an error.html page that contains an error message and our names and IDs.

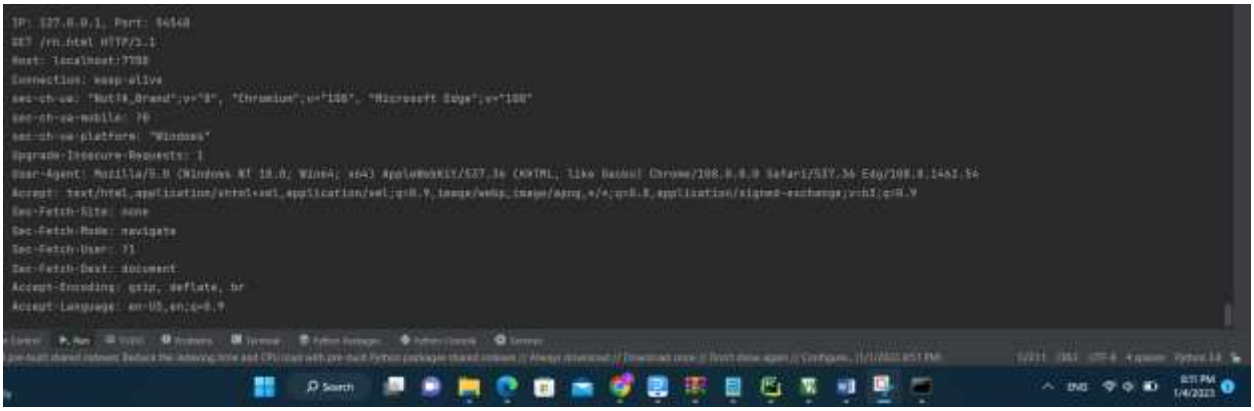


Figure 36: Browser request when the file doesn't exist

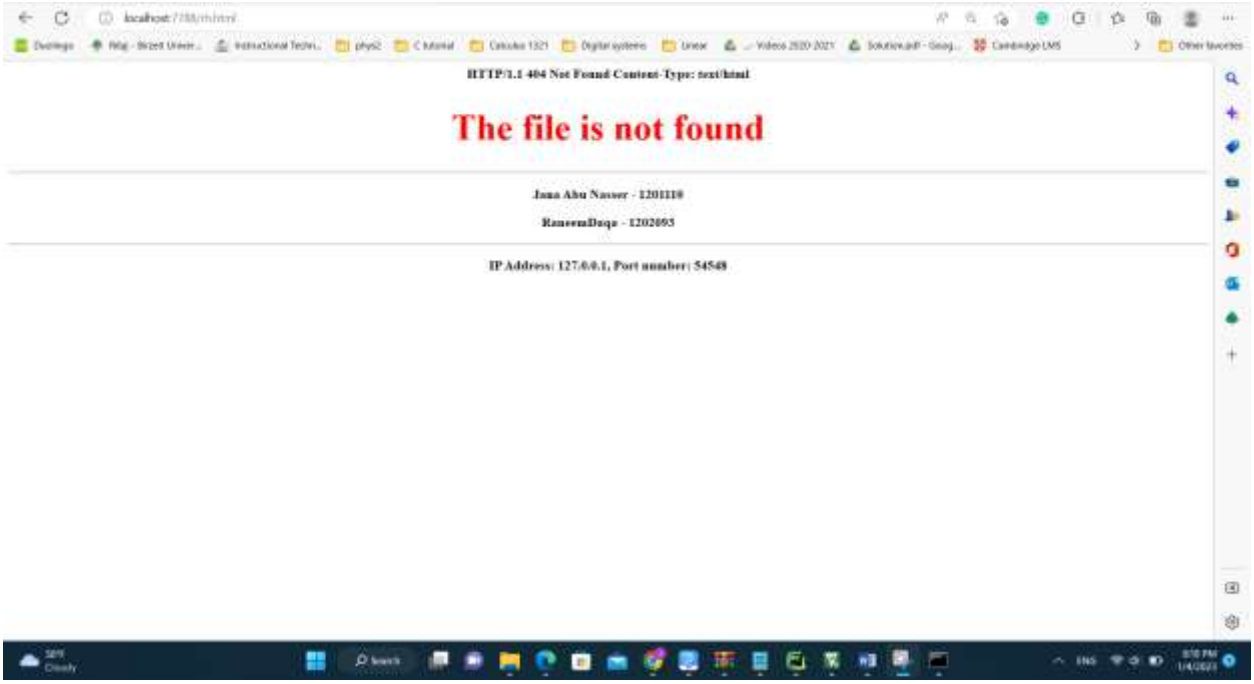


Figure 37: error.html page when a file doesn't exist

3.8 307 Temporary Redirect

Here , we the status code 307 Temporary Redirect to redirect the client request

3.8.1 Redirect to Google website

If the request is /go then the request will be temporary redirect to google website as shown below:

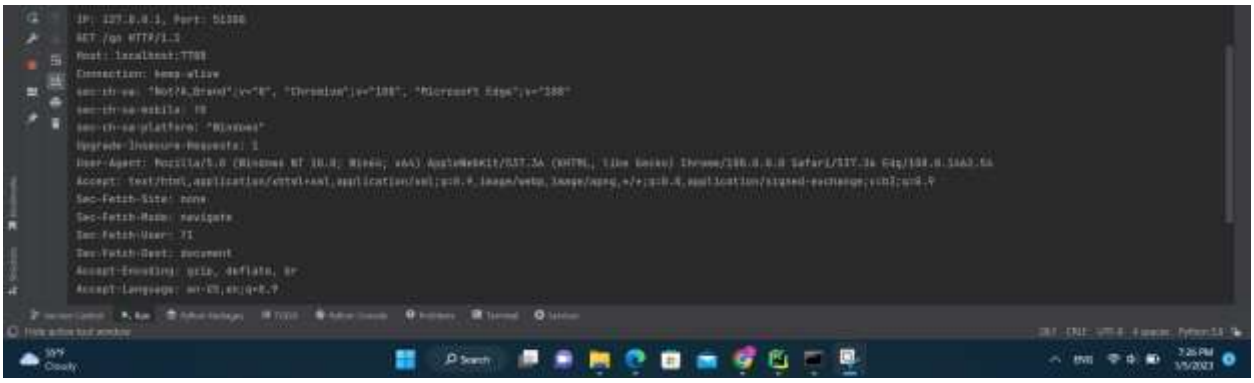


Figure 38: Browser request message when redirect the request to Google website

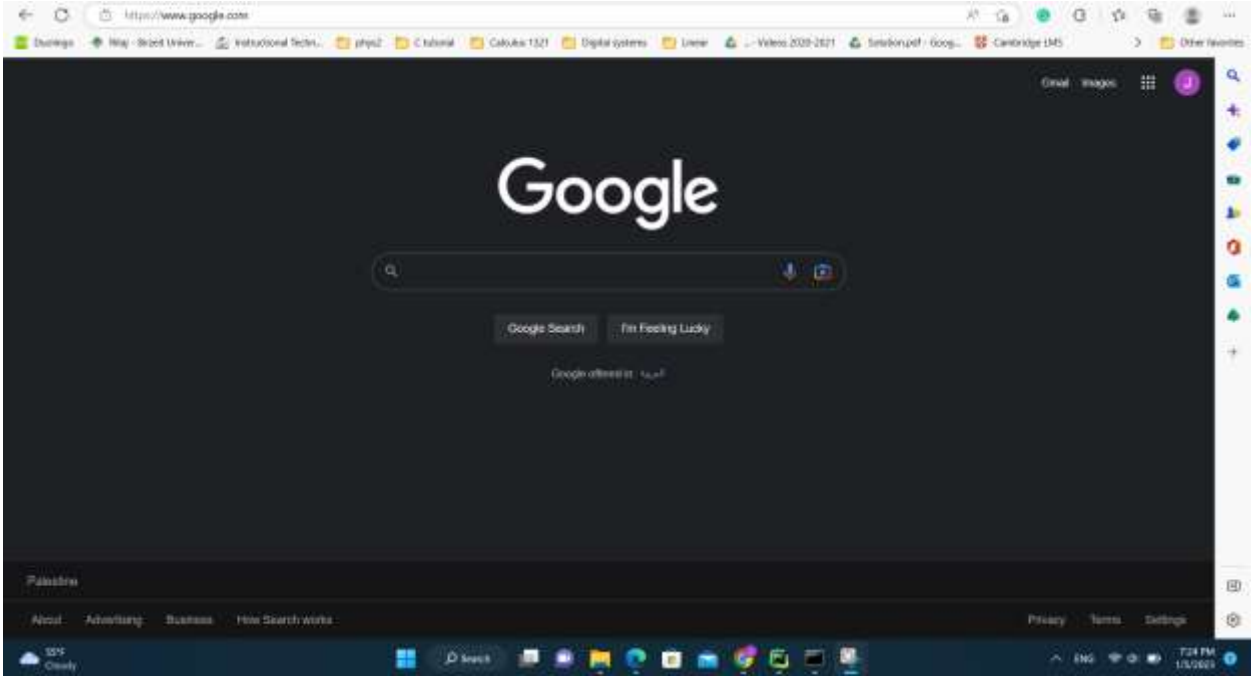


Figure 39: Google website opened when the request is redirected

3.8.2 Redirect to stackoverflow.com website

If the request is /so then the request will be temporary redirect to stackoverflow website as shown below:

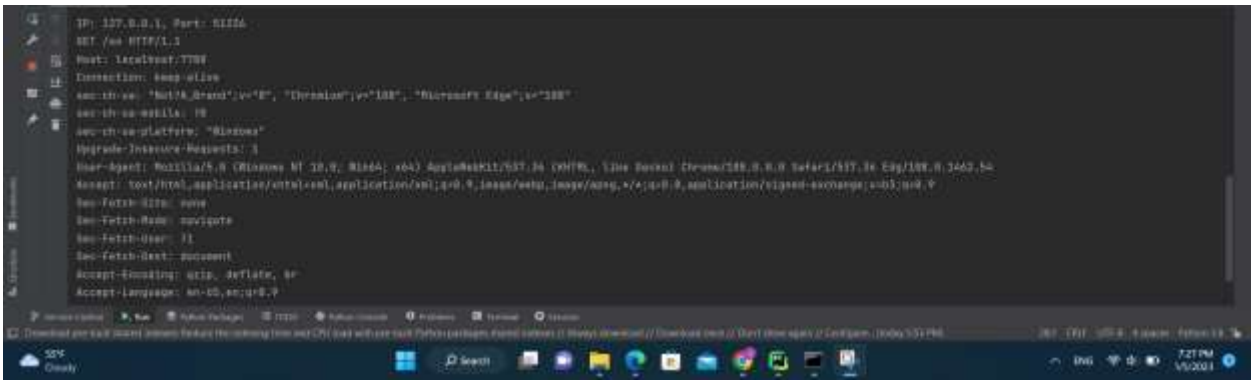


Figure 40: Browser request message when redirect the request to stackoverflow website

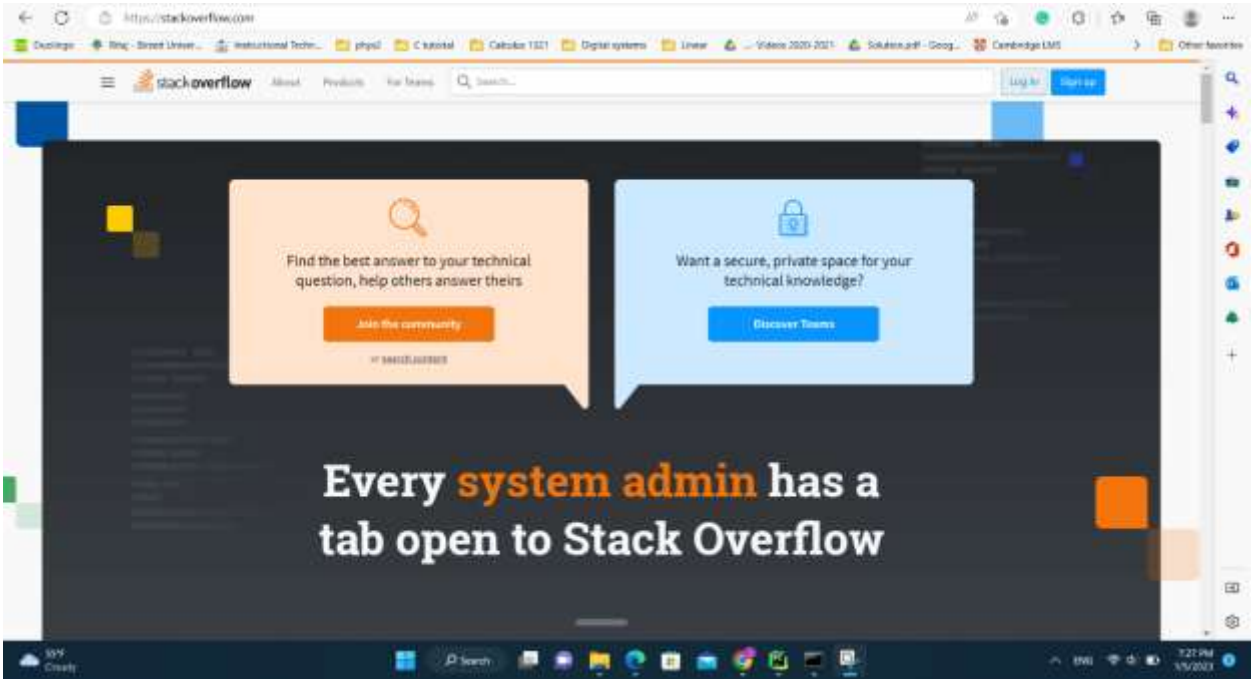


Figure 41: Stackoverflow website opened when the request is redirected

3.8.3 Redirect to birzeit university website

If the request is /bzu then the request will be temporary redirect to birzeit university website as shown below:

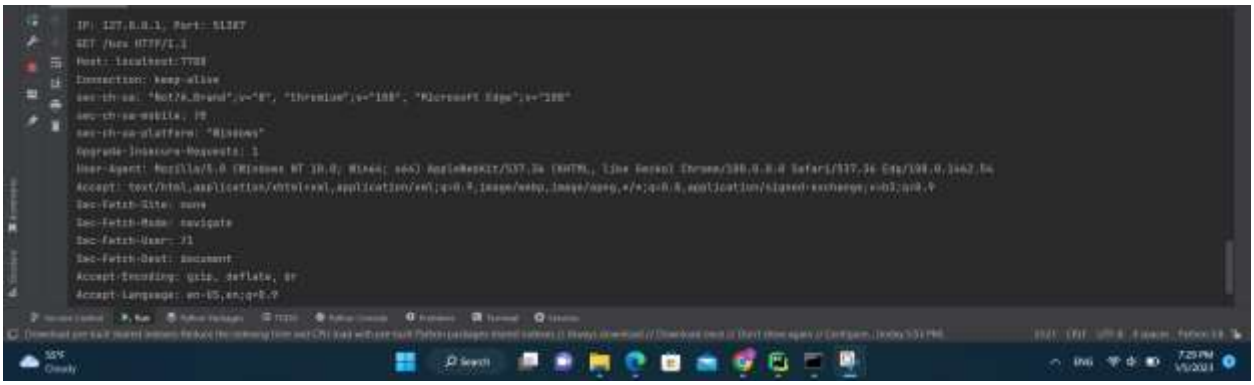


Figure 42: Browser request message when redirect the request to birzeit university website

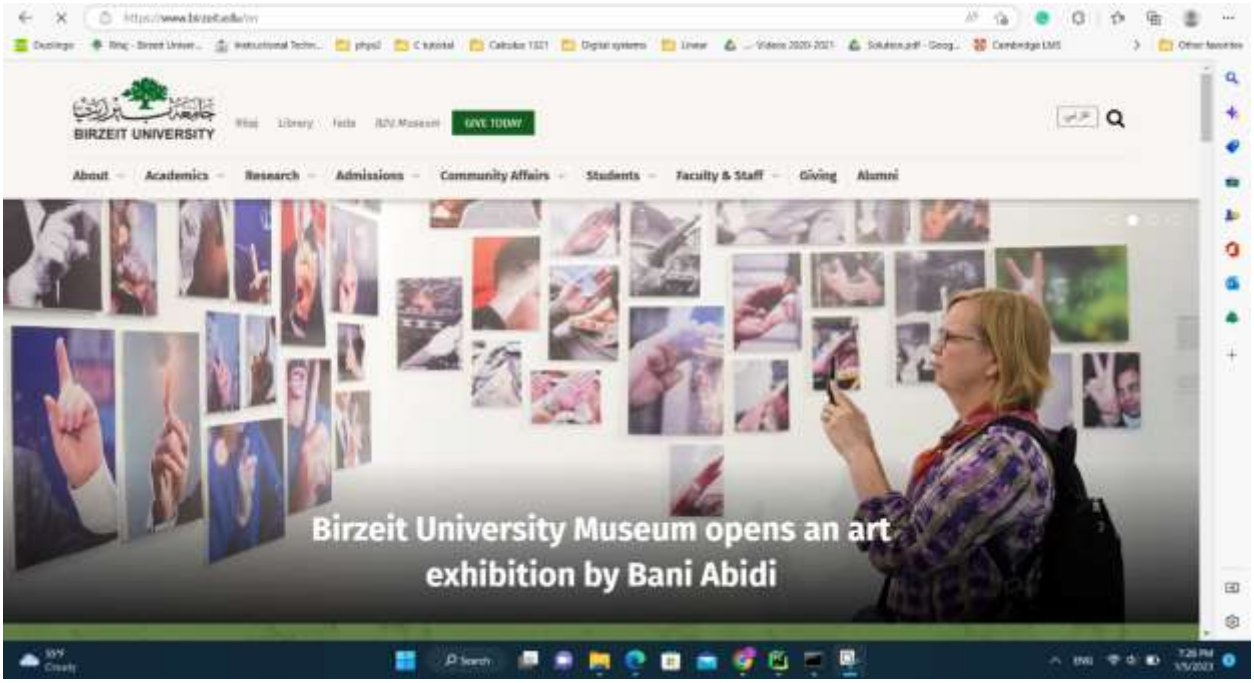


Figure 43: birzeit university website opened when the request is redirected

3.9 Requesting from another device “mobile”

3.9.1 Web Arabic page

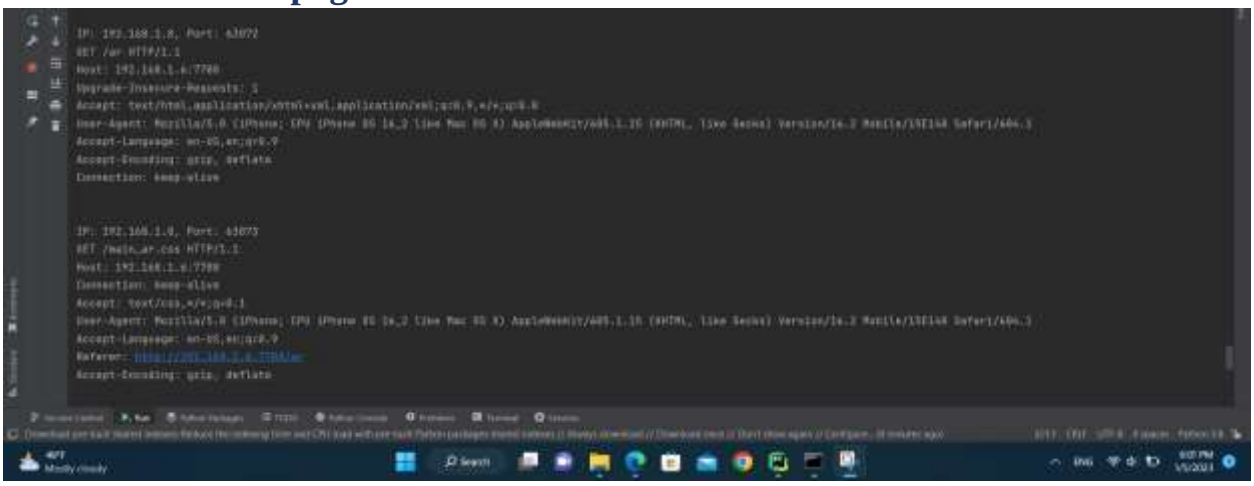


Figure 44: Requesting the Arabic page from another device “mobile”

Figure 45: Group member displayed on mobile

Figure 46: Main web Arabic page on mobile

As well as, the Arabic web page was running on localhost, we've to try to run it over another device on the same host. The figures show that the web and server work correctly.

3.9.2 Web English page (main page)

Figure 47: Requesting the English page (main page) from another device "mobile" using main.html

Figure 48: Requesting the English page (main page) from another device "mobile" using \en

Figure 49: Main web page on mobile

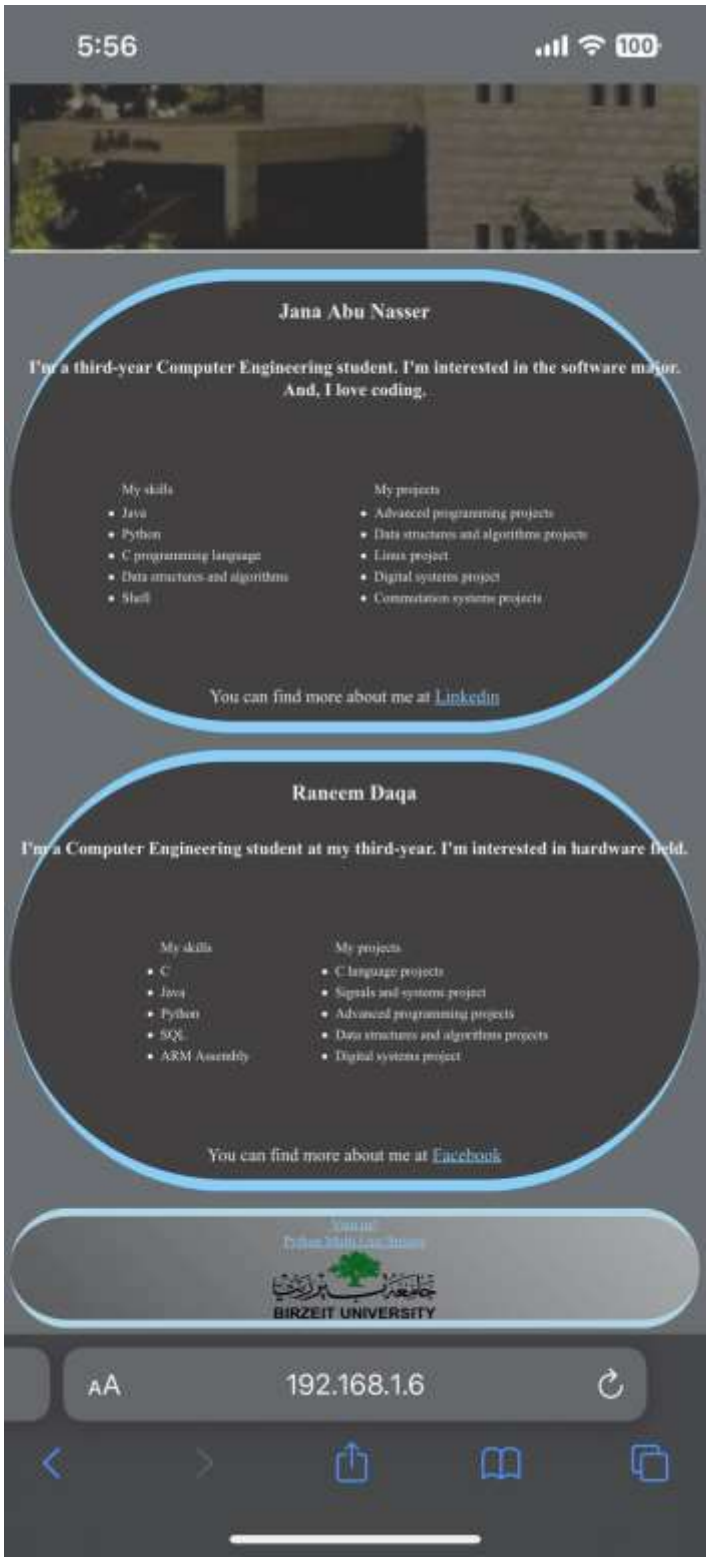


Figure 50: Group member displayed on mobile

As well as, the web page was running on localhost, we've to try to run it over another device on the same host. The figures show that the web and server work correctly.

4 Codes

4.1 Part 3

4.1.1 HTML

4.1.1.1 visit_us.html

Figure 51:Visit us code

4.1.1.2 main.html

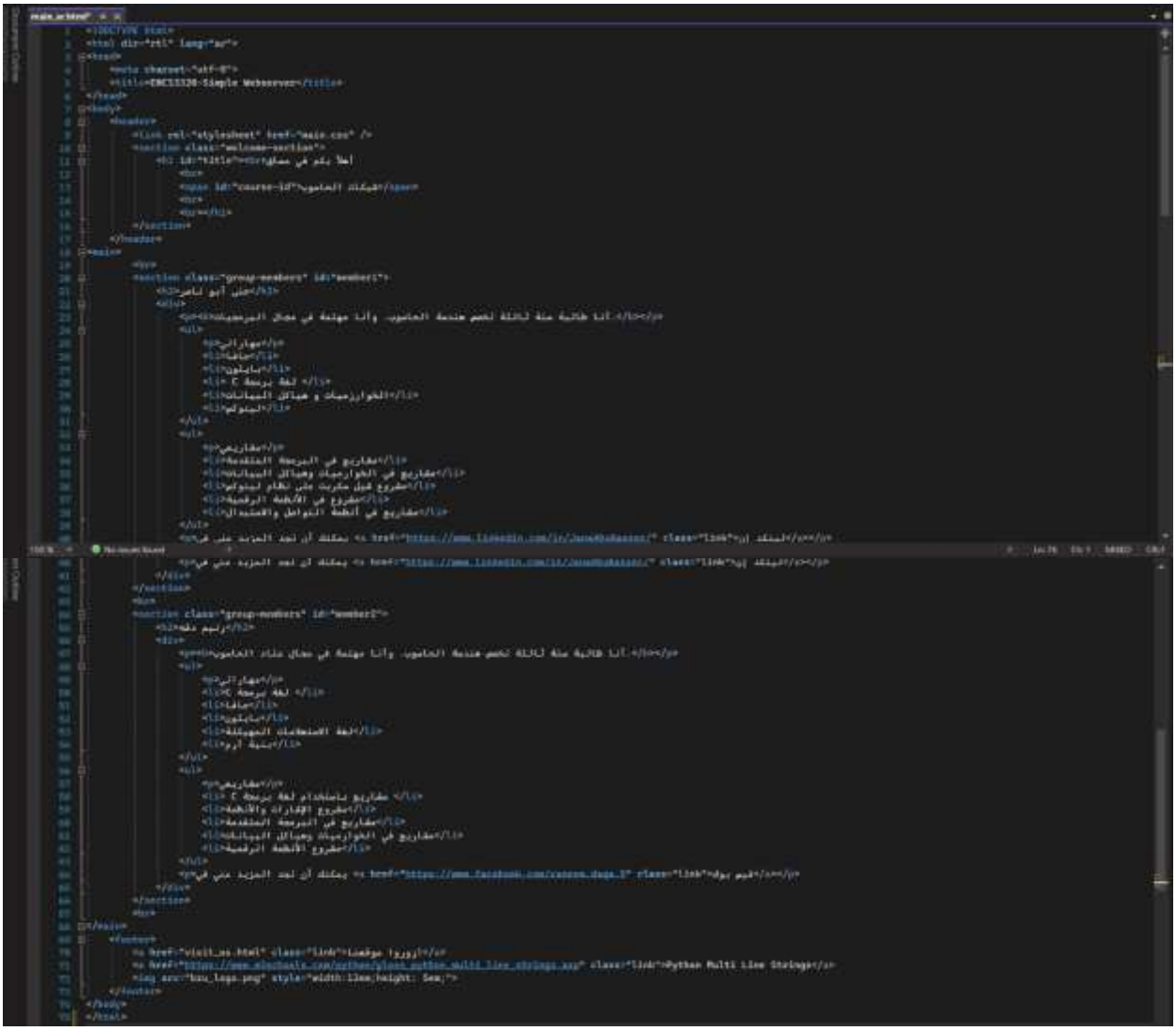


Figure 52:main_en.html code

4.1.1.3 Empty.html

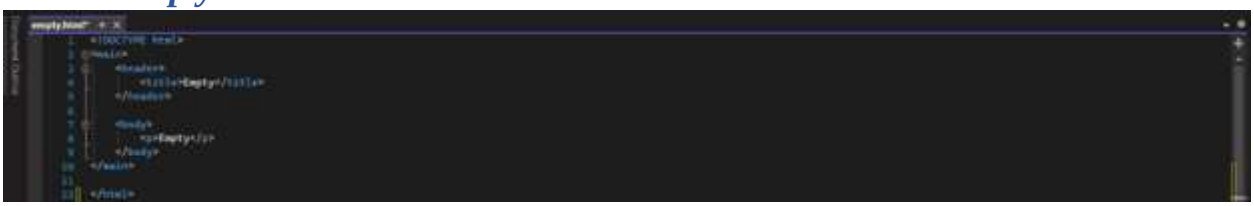


Figure 53: empty html code

4.1.2 CSS

4.1.2.1 main.css

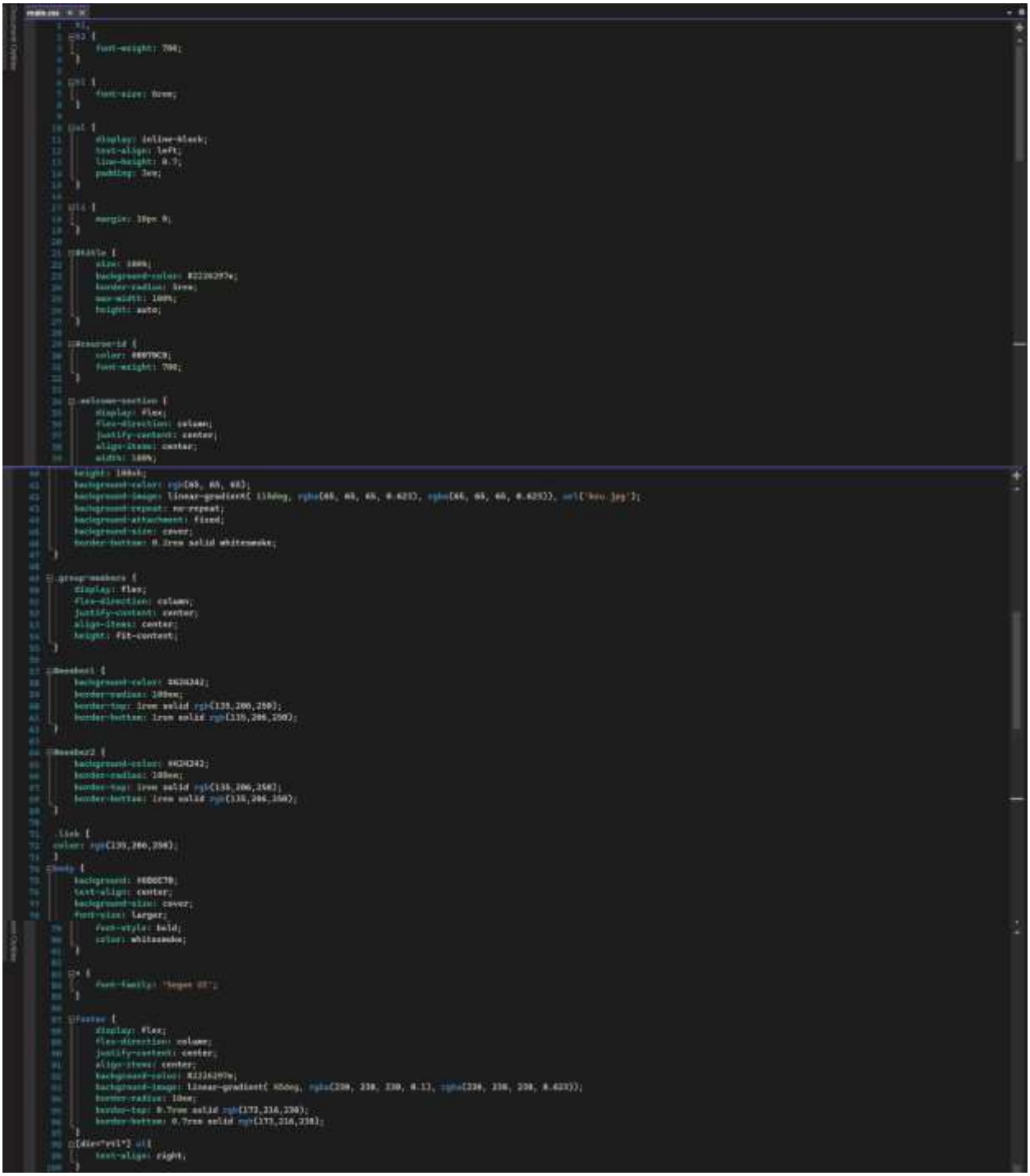


Figure 54:main.css code

4.1.3 Server (python code)

```python
from socket import *
import os
from operator import itemgetter
# server port, socket initialization and type
serverPort = 7788
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
print("The server is ready to receive\n\n")
# start getting requests
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    ip = addr[0]
    port = addr[1]

    print('IP: ' + str(ip) + ', Port: ' + str(port))
    print(sentence)
    # If the sentence is not empty, the requested file is gotten from request header
    if sentence != '':
        requested_file = sentence.split(' ')[1].replace('/', '')
        print(requested_file + "text 555")
    else:
        # if the request is empty the connection is closed
        connectionSocket.close()
        continue
        # is not found then exception is raised
        # if the requested file is main.html or index.html
        if requested_file == '' or requested_file == 'main.html' or requested_file == 'index.html' or requested_file == 'en':
            connectionSocket.send(b"HTTP/1.1 200 OK\r\n")
            connectionSocket.send(b"Content-Type: text/html \r\n")
            connectionSocket.send(b"\r\n")
            mhtml = open('main_en.html', 'rb')
            connectionSocket.send(mhtml.read())
            mhtml.close()
        elif requested_file == 'main_ar.html' or requested_file == 'ar':
            connectionSocket.send(b"HTTP/1.1 200 OK\r\n")
            connectionSocket.send(b"Content-Type: text/html \r\n")
            connectionSocket.send(b"\r\n")
            mhtml = open('main_ar.html', 'rb')
            connectionSocket.send(mhtml.read())
            mhtml.close()
        elif requested_file == 'go':
            connectionSocket.send(f"HTTP/1.1 307 Temporary Redirect\r\n".encode())
            connectionSocket.send(f"Location: https://www.google.com/\r\n".encode())
        elif requested_file == 'so':
            connectionSocket.send(f"HTTP/1.1 307 Temporary Redirect\r\n".encode())
            connectionSocket.send(f"Location: https://stackoverflow.com/\r\n".encode())
        elif requested_file == 'bzu':
            connectionSocket.send(f"HTTP/1.1 307 Temporary Redirect\r\n".encode())
            connectionSocket.send(f"Location: https://www.birzeit.edu/en\r\n".encode())
        # if the client requests any html file
        elif '.html' in requested_file:
            connectionSocket.send(b"HTTP/1.1 200 OK\r\n")
```

```python
            connectionSocket.send(b"Content-Type: text/html \r\n")
            connectionSocket.send(b"\r\n")
            print('Response status: 200 OK\n\n')
            file = open(str(requested_file), "rb")
            connectionSocket.send(file.read())
            file.close()
        # if the client requests any css file
        elif '.css' in requested_file:
            connectionSocket.send(b"HTTP/1.1 200 OK\r\n")
            connectionSocket.send(b"Content-Type: text/css \r\n")
            connectionSocket.send(b"\r\n")
            print('Response status: 200 OK\n\n')
            file = open(str(requested_file), "rb")
            connectionSocket.send(file.read())
            file.close()
        # if the request contains .png image
        elif '.png' in requested_file:
            connectionSocket.send(b"HTTP/1.1 200 OK\r\n")
            connectionSocket.send(b"Content-Type: image/png \r\n")
            connectionSocket.send(b"\r\n")
            print('Response status: 200 OK\n\n')
            file = open(str(requested_file), "rb")
            connectionSocket.send(file.read())
            file.close()
        # if the request contains .jpg image
        elif '.jpg' in requested_file:
            connectionSocket.send(b"HTTP/1.1 200 OK\r\n")
            connectionSocket.send(b"Content-Type: image/jpeg \r\n")
            connectionSocket.send(b"\r\n")
            print('Response status: 200 OK\n\n')
            file = open(str(requested_file), "rb")
            connectionSocket.send(file.read())
            file.close()
        # this is a handler only in order not to get not found error
        elif 'favicon.ico' == requested_file:
            print()
        else:
            print(requested_file + "test")
            raise Exception('Not Found')
    # if the file is not found in the project folder
    except Exception as e:
        connectionSocket.send(b"HTTP/1.1 404 Not Found \r\n")
        connectionSocket.send(b"Content-Type: text/html \r\n")
        connectionSocket.send(b"\r\n")
        print(requested_file + "test")
        print('\n\bResponse status: 404 Not Found')
        file = '<!DOCTYPE html><style>* {text-align: center;font-size: large;font-weight: bold;}h1 {font-size: ' \
               '50px;}#error-message {color: red;}</style><main><header><title>Error</title></header><body><div ' \
               'id="error-message"><h1>The file is not found</h1></div><hr><div><p>Jana Abu Nasser - 1201119</p><p>Ranem' \
               'Daas - 1202093</p></div><hr><div><p>IP Address: ' + str(ip) + \
               ', Port number: ' + str(port) + '</p></div></body></main></html> '
        connectionSocket.send(file.encode())
    connectionSocket.close()
```

Figure 55: server python code