



**Faculty of Engineering & Technology Electrical & Computer  
Engineering Department**

**ENCS 3320  
Computer Networks**

**Project #1 - Report**

---

**Prebroad by:**

Ghazi Haj Qassem	1210778	sec 3
Diaa Badaha	1210478	sec 1
Omar Husain	1212739	sec 3

**Instructor:**

Dr. Abdelkarim Awad

**Date:** 22-12-2023

## **Abstract**

The project is is mainly focused on the use of the application layer protocols, primarily HTTP and DNS. In addition to that, socket programming is employed to create basic client and server programs, including a web server. Some commands that are entered through the terminal may be useful in gaining knowledge about the path to a destination or some information about it. Wireshark, a common packet tracer application, is also employed to capture some packets, mainly DNS messages.

## Table of Contents

<b>Abstract .....</b>	<b>II</b>
<b>Table of figures .....</b>	<b>IV</b>
<b>Part I.....</b>	<b>V</b>
Point 1.....	V
Point 2.1 .....	V
Point 2.2.....	VI
Point 2.3.....	VII
Point 2.4.....	VIII
Point 3.....	1
<b>Part II.....</b>	<b>3</b>
Tests.....	3
<b>Part III.....</b>	<b>4</b>
Point 0.....	4
Point 1.....	4
Point 2.....	6
Point 3.....	7
Point 4.....	7
Point 5.....	8
Point 6.....	8
Point 7.....	9
Point 8.....	10
Point 9.....	10
<b>Appendix .....</b>	<b>11</b>

## Table of figures

Figure 1: results of pinging a device in the same network .....	V
Figure 2: results of pinging <code>www.cornell.edu</code> .....	VI
Figure 3: Results of using the command " <code>tracert www.cornell.edu</code> " .....	VII
Figure 4: results of using the command " <code>nslookup www.cornell.edu</code> " .....	VIII
Figure 5: DNS query message captured by Wireshark .....	1
Figure 6: DNS query response message captured by Wireshark .....	2
Figure 7: Testing the server and client in part 2 when entering an invalid ID .....	3
Figure 8: Testing the server and client in part 2 when entering a valid ID .....	3
Figure 11: Summarization of point 0 .....	4
Figure 12: Requesting "/" through the browser and printing the HTTP response in the terminal window .....	4
Figure 13: Requesting <code>"/main_en"</code> through the browser and printing the HTTP response in the terminal window.....	5
Figure 14: Requesting <code>"/index.html"</code> through the browser and printing the HTTP response in the terminal window.....	5
Figure 15: Requesting <code>"/en"</code> through the browser and printing the HTTP response in the terminal window.....	6
Figure 16: Requesting <code>"/ar"</code> through the browser and printing the HTTP response in the terminal window.....	6
Figure 17: Requesting a file <code>.html</code> " <code>secondary.html</code> " through the browser and printing the HTTP response in the terminal window .....	7
Figure 18: Requesting a file <code>.css</code> " <code>main.css</code> " through the browser and printing the HTTP response in the terminal window.....	7
Figure 19: Requesting a png image through the browser and printing the HTTP response in the terminal window.....	8
Figure 20: Requesting a jpg image through the browser and printing the HTTP response in the terminal window.....	8
Figure 21: Requesting redirection to <code>cornell.edu</code> through <code>/cr</code> and printing the HTTP response in the terminal window.....	9
Figure 22: Requesting redirection to <code>stackoverflow.com</code> through <code>/so</code> and printing the HTTP response in the terminal window.....	9
Figure 23: Requesting redirection to <code>ritaj.birzeit.edu</code> through <code>/rt</code> and printing the HTTP response in the terminal window.....	10
Figure 24: Requesting an invalid file and printing the HTTP response in the terminal window .....	10
Figure 9: Client code .....	11
Figure 10: Server code .....	11
Figure 25: Webserver code 1.....	12
Figure 26: Webserver code 2.....	13

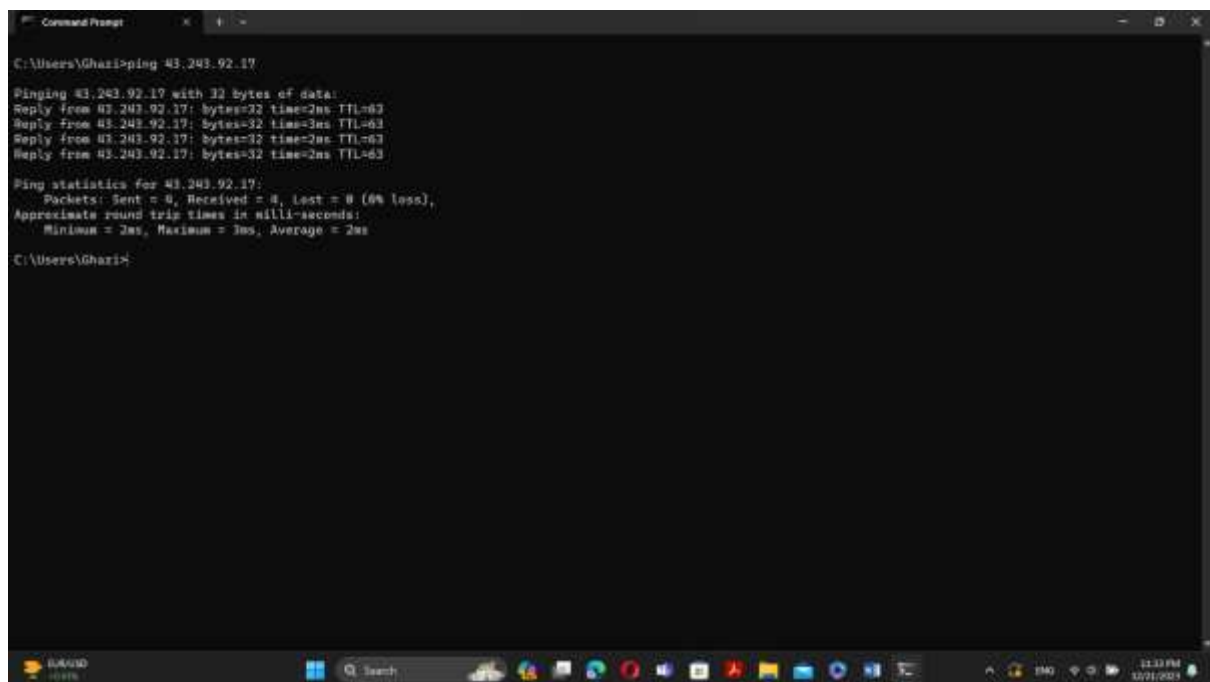
# Part I

## Point 1

The following commands are given with their brief description:

- ping: A command used to test the reachability of a host by computing the round-trip time, time to live (TTL) of packets and statistics for packet loss.
- tracer: A command used to trace the route from a source to a destination by sending three packets along each hop (to each device on the path) and computing the round-trip time for each packet.
- nslookup: A command used to query the DNS server for DNS records.
- telnet: A text-based network protocol for 8-bit bidirectional communication for remote access.

## Point 2.1



```
C:\Users\Ghazi>ping 43.243.92.17

Pinging 43.243.92.17 with 32 bytes of data:
Reply from 43.243.92.17: bytes=32 time=2ms TTL=63
Reply from 43.243.92.17: bytes=32 time=3ms TTL=63
Reply from 43.243.92.17: bytes=32 time=2ms TTL=63
Reply from 43.243.92.17: bytes=32 time=2ms TTL=63

Ping statistics for 43.243.92.17:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2ms, Maximum = 3ms, Average = 2ms

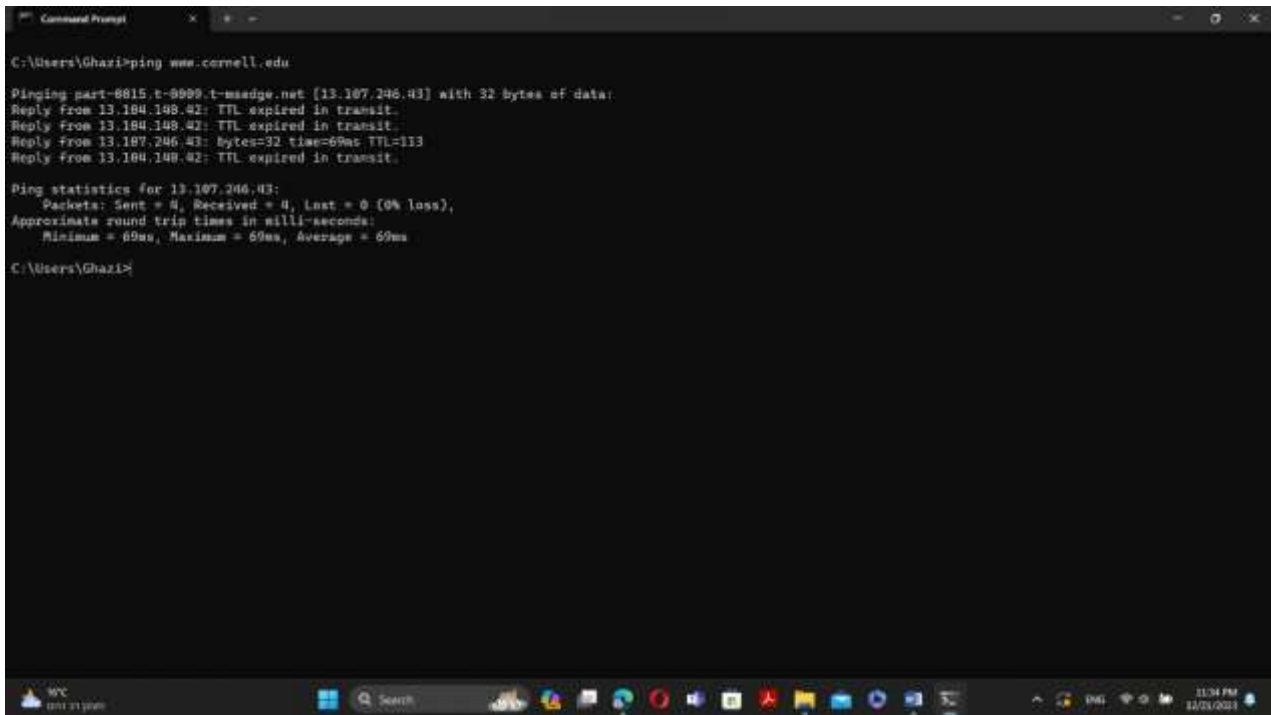
C:\Users\Ghazi>
```

*Figure 1: results of pinging a device in the same network*

As seen, after I pinged my smartphone, the results show that 4 packets, each 32 bytes in size, were sent to my smart phone, and none was lost. The time 2-3 ms displayed is the round-trip time, which is the time required for a packet to reach the destination added to the time required to receive a reply from the destination to acknowledge the reception of packets. TTL is the time to live for packet, which is the number of network devices along the path that the

packet can pass through. If a packet travels beyond TTL, then it is discarded to prevent it from indefinitely circulating in the network. Thus, a packet here is allowed to pass through 63 devices.

## Point 2.2



```
C:\Users\Ghazi>ping www.cornell.edu

Pinging part-8815.t-0009.t-msedge.net [13.107.246.43] with 32 bytes of data:
Reply from 13.184.148.42: TTL expired in transit.
Reply from 13.184.148.42: TTL expired in transit.
Reply from 13.187.246.43: bytes=32 time=69ms TTL=113
Reply from 13.184.148.42: TTL expired in transit.

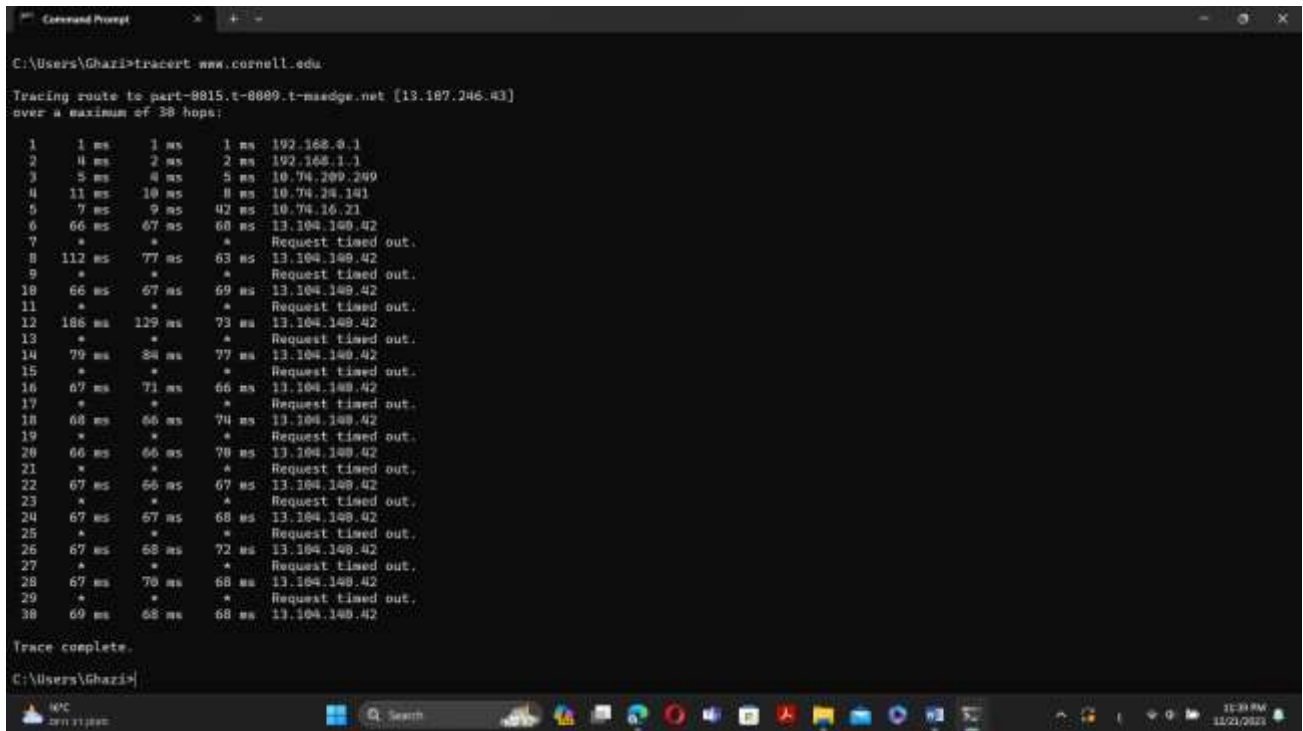
Ping statistics for 13.107.246.43:
    Packets: Sent = 4, Received = 1, Lost = 3 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 69ms, Maximum = 69ms, Average = 69ms

C:\Users\Ghazi>
```

Figure 2: results of pinging [www.cornell.edu](http://www.cornell.edu)

In contrast to pinging a device in the same network, pinging [www.cornell.edu](http://www.cornell.edu), which is a more distant location, yielded higher time results. Three packets were discarded, since they passed through devices more than their specified TTL. However, one packet arrived, and its round-trip time (RTT) is 69 ms, and TTL = 113. This means that it took a lot of time and passed through much more devices compared than the device in the same network that was pinged. The results show that the destination is likely to be in the USA, since RTT is 35 times the RTT that resulted from pinging a device in the same network, and “TTL expired in transit” is yet another indicator. The IP address was also checked to confirm the case, and it turned out to be in the USA.

### Point 2.3



```
C:\Users\Ghazi>tracert www.cornell.edu

Tracing route to part-8815.t-8889.t-msedge.net [13.187.246.43]
over a maximum of 30 hops:

  0  1 ms  1 ms  1 ms  192.168.0.1
  1  4 ms  2 ms  2 ms  192.168.1.1
  2  5 ms  4 ms  5 ms  10.74.200.209
  3 11 ms 10 ms  8 ms  10.74.24.141
  4  7 ms  9 ms 42 ms  10.74.16.21
  5 66 ms 67 ms 68 ms 13.104.140.42
  6 * * * Request timed out.
  7 112 ms 77 ms 63 ms 13.104.140.42
  8 * * * Request timed out.
  9 66 ms 67 ms 69 ms 13.104.140.42
 10 * * * Request timed out.
 11 186 ms 129 ms 73 ms 13.104.140.42
 12 * * * Request timed out.
 13 79 ms 84 ms 77 ms 13.104.140.42
 14 * * * Request timed out.
 15 67 ms 71 ms 66 ms 13.104.140.42
 16 * * * Request timed out.
 17 68 ms 66 ms 74 ms 13.104.140.42
 18 * * * Request timed out.
 19 66 ms 66 ms 70 ms 13.104.140.42
 20 * * * Request timed out.
 21 67 ms 66 ms 67 ms 13.104.140.42
 22 * * * Request timed out.
 23 67 ms 67 ms 68 ms 13.104.140.42
 24 * * * Request timed out.
 25 67 ms 68 ms 72 ms 13.104.140.42
 26 * * * Request timed out.
 27 67 ms 70 ms 68 ms 13.104.140.42
 28 * * * Request timed out.
 29 69 ms 68 ms 68 ms 13.104.140.42
 30

Trace complete.

C:\Users\Ghazi>
```

Figure 3: Results of using the command "tracert [www.cornell.edu](http://www.cornell.edu)"

As shown above, the packets took 30 hops to reach the destination, meaning in the last hop, the three packets had to pass through 30 network devices to reach the destination host. The first column represents the hop number. The next three columns contain the round-trip time (RTT) for each of three packets sent in this hop. The last column is the destination (be it a host or a network device) that each of the three packets in a certain hop are sent to. Some hops display "\*" and "Request timed out.". This means that the router did not respond with a message within the specified time limit in the three cases. Another thing to note is that in the last 25 hops, the destination IP addresses are the same. This may be a security measure to obscure IP addresses of each device as having the same IP address of the host destination.

## Point 2.4

```
C:\Users\Ghazi>nslookup www.cornell.edu
Server: Unknown
Address: 192.168.0.1

Non-authoritative answer:
Name:   part-0015.t-0009.t-msedge.net
Address: 2628:1ec:bdf::43
        2628:1ec:48::43
        13.107.246.43
        13.107.213.43
Aliases: www.cornell.edu
        cornell-edge-ekhdhg6czdab2bf.s01.azurefd.net
        star-azurefd-prod.trafficmanager.net
        shod.dual-lw.part-0015.t-0009.t-msedge.net

C:\Users\Ghazi>
```

Figure 4: results of using the command "nslookup [www.cornell.edu](http://www.cornell.edu)"

It is seen that the DNS server sends two types of DNS records, A and CNAME. An A record is for the IP addresses of that hostname, while a CNAME record is for aliases of that hostname. These records came from a non-authoritative DNS server. An authoritative DNS server is the kind of server that stores these types of records that are mapped to this hostname. Records maybe were cached at this non-authoritative server, since the answer was obtained from it instead of an authoritative one.



### Point 3

The following captured DNS messages are as a result of pinging [www.cornell.edu](http://www.cornell.edu).

The below captured message using Wireshark shows some DNS query, requesting the IP address mapped to URL [www.cornell.edu](http://www.cornell.edu) as a result of being pinged. Of course, it queries the DNS server, which turns out to be a DNS server inside my local network, for an A record, which stores the IP address of that hostname.

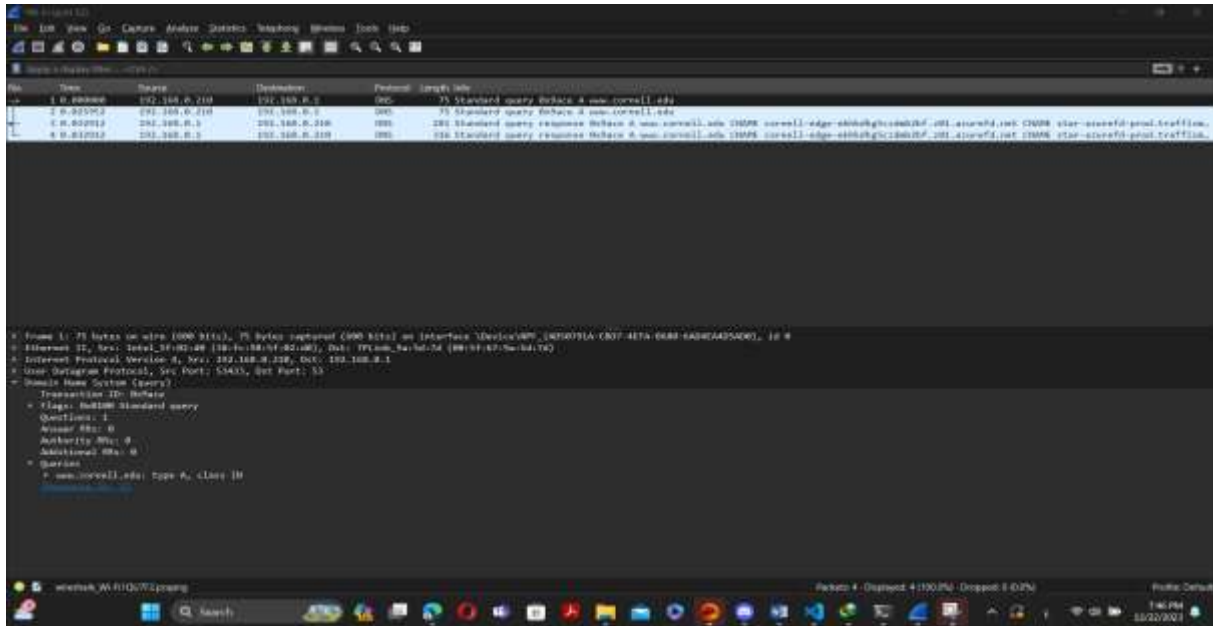


Figure 5: DNS query message captured by Wireshark

The screenshot shows a Kali Linux terminal window with a Wireshark capture of a DNS query and response. The terminal output is as follows:

```

# Frame 3: 281 bytes on wire (2248 bits), 281 bytes captured (2248 bits) on interface /dev/virtio-lan0 (08:00:27:00:00:00), 1 bit
# Ethernet II, Src: VirtualBox Adapter (08:00:27:00:00:00), Dst: Intel E1000 (08:00:27:00:00:00)
# Internet Protocol Version 4, Src: 192.168.0.1, Dest: 192.168.0.10
# User Datagram Protocol, Src Port: 53, Dst Port: 53433
# Domain Name System (response)
  Transaction ID: 4096
  Flags: 0x00 Standard query response, No error
  Questions: 1
  Answer RRs: 6
  Authority RRs: 0
  Additional RRs: 0
  Query:
    > www.corvill.edu: type A, class IN
  Answers:
    > www.corvill.edu: type AAAA, class IN, ttl=300, www.corvill.edu: 2001:db8::1
    > ns1.corvill.edu: type AAAA, class IN, ttl=300, ns1.corvill.edu: 2001:db8::1
    > ns2.corvill.edu: type AAAA, class IN, ttl=300, ns2.corvill.edu: 2001:db8::1
    > www.corvill.edu: type A, class IN, ttl=300, www.corvill.edu: 192.168.0.1
    > ns1.corvill.edu: type A, class IN, ttl=300, ns1.corvill.edu: 192.168.0.1
    > ns2.corvill.edu: type A, class IN, ttl=300, ns2.corvill.edu: 192.168.0.1
  Timing: 0.000000 seconds

```

Figure 6: DNS query response message captured by Wireshark

## Part II

### Tests

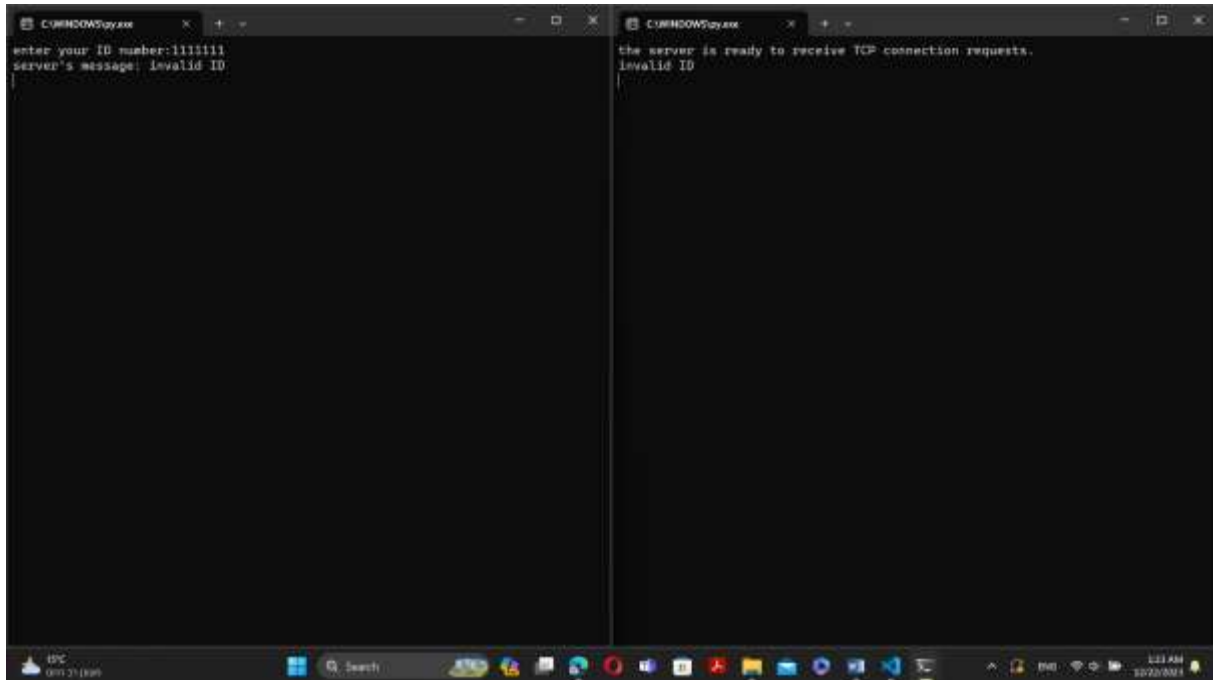


Figure 7: Testing the server and client in part 2 when entering an invalid ID

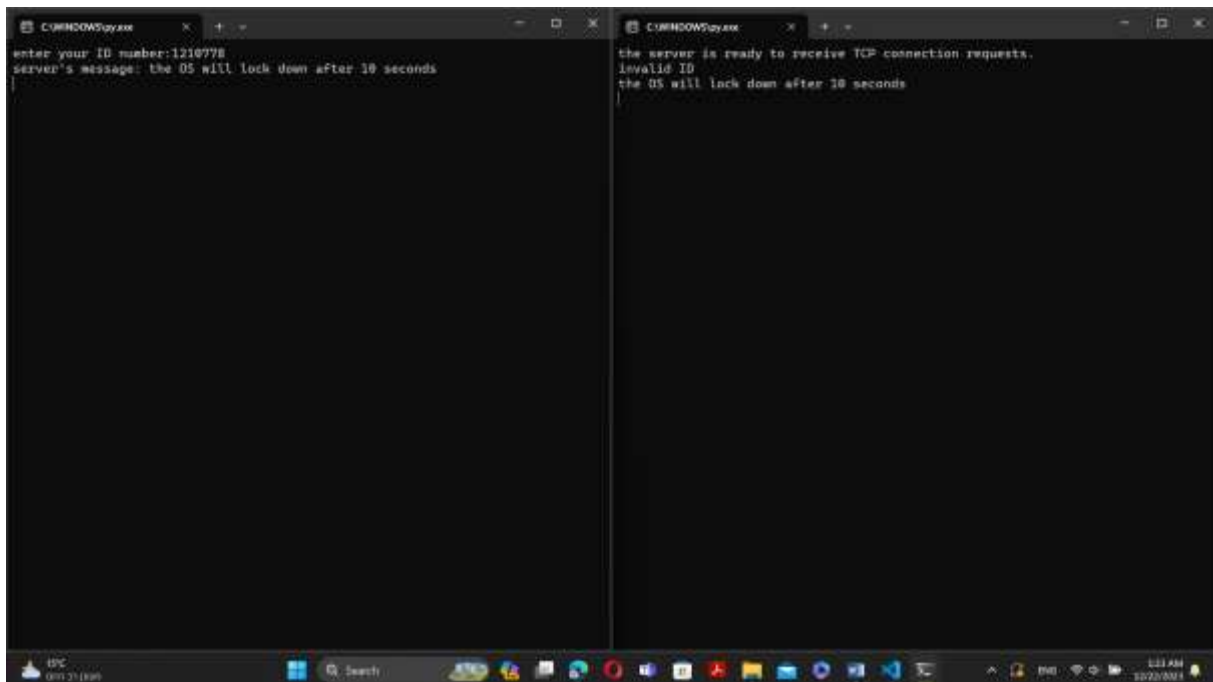


Figure 8: Testing the server and client in part 2 when entering a valid ID

## Part III

### Point 0

The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET. And we summarized it in the web page as shown in figure 11.

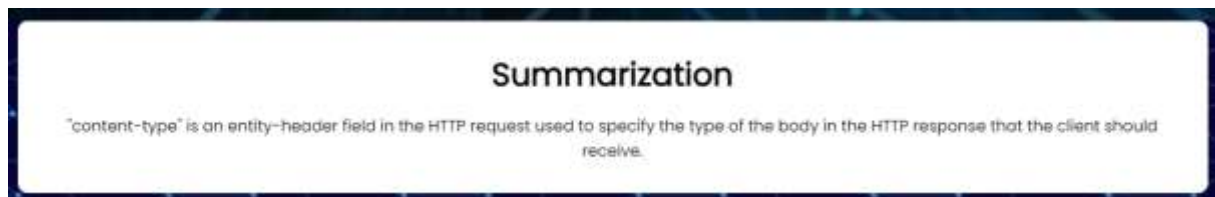


Figure 9: Summarization of point 0

### Point 1

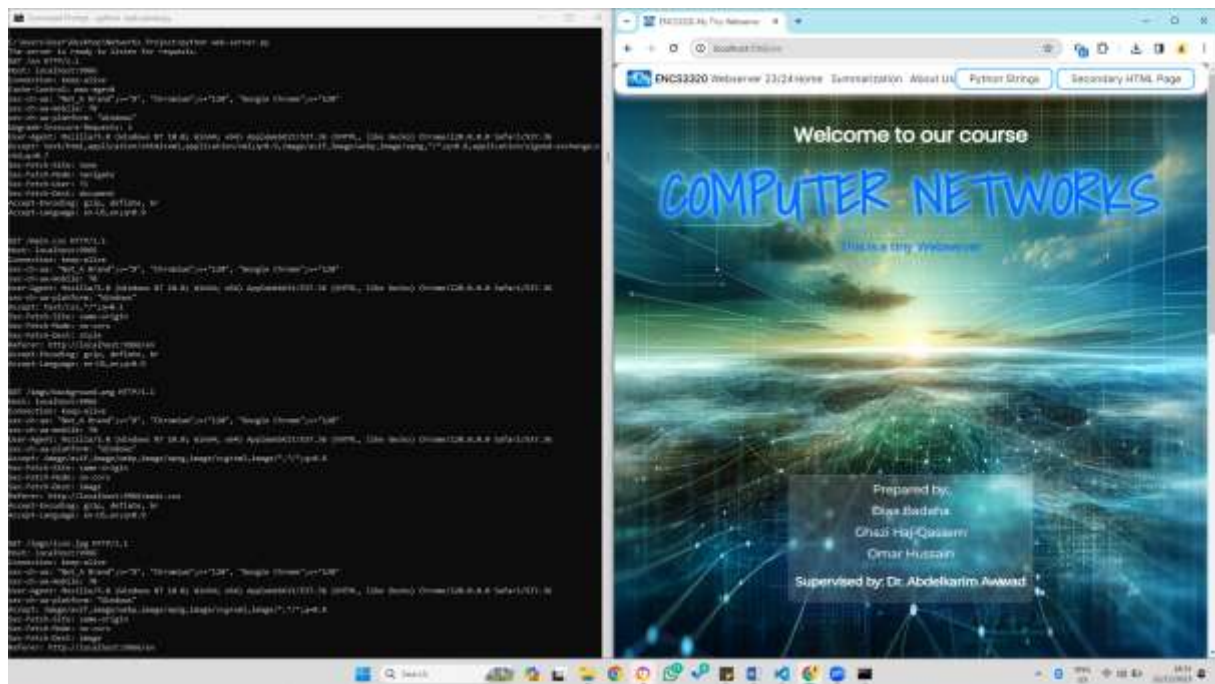


Figure 10: Requesting "/" through the browser and printing the HTTP response in the terminal window

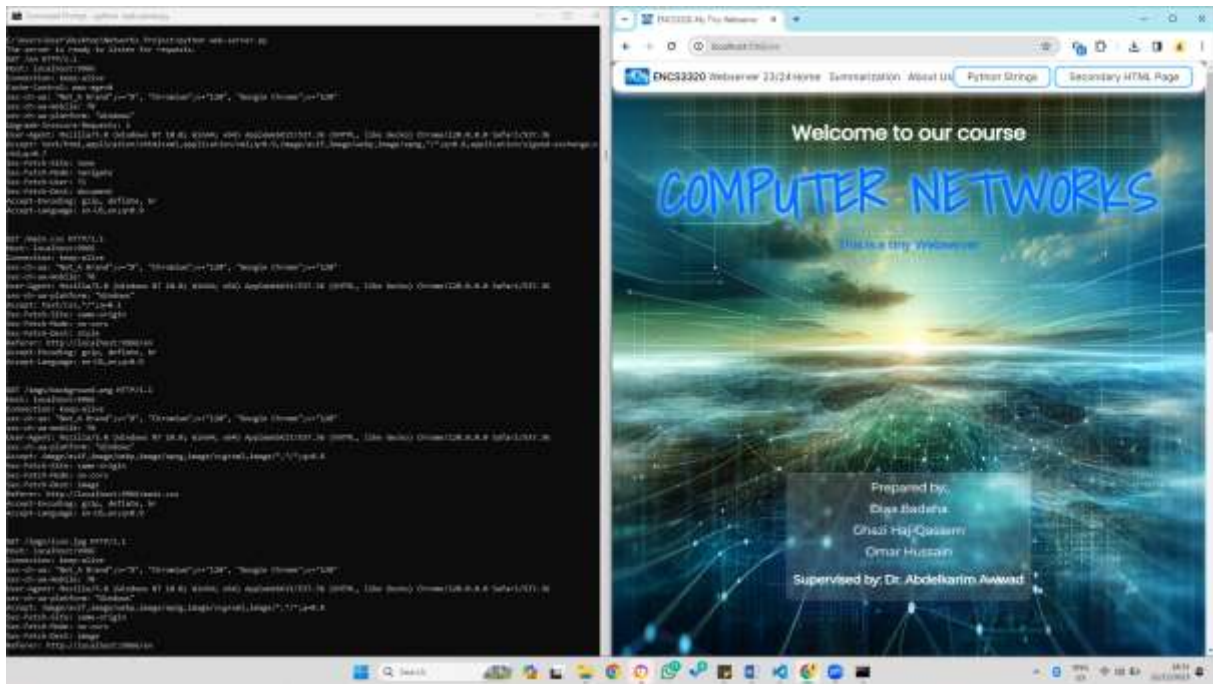


Figure 12: Requesting "/"index.html" through the browser and printing the HTTP response in the terminal window

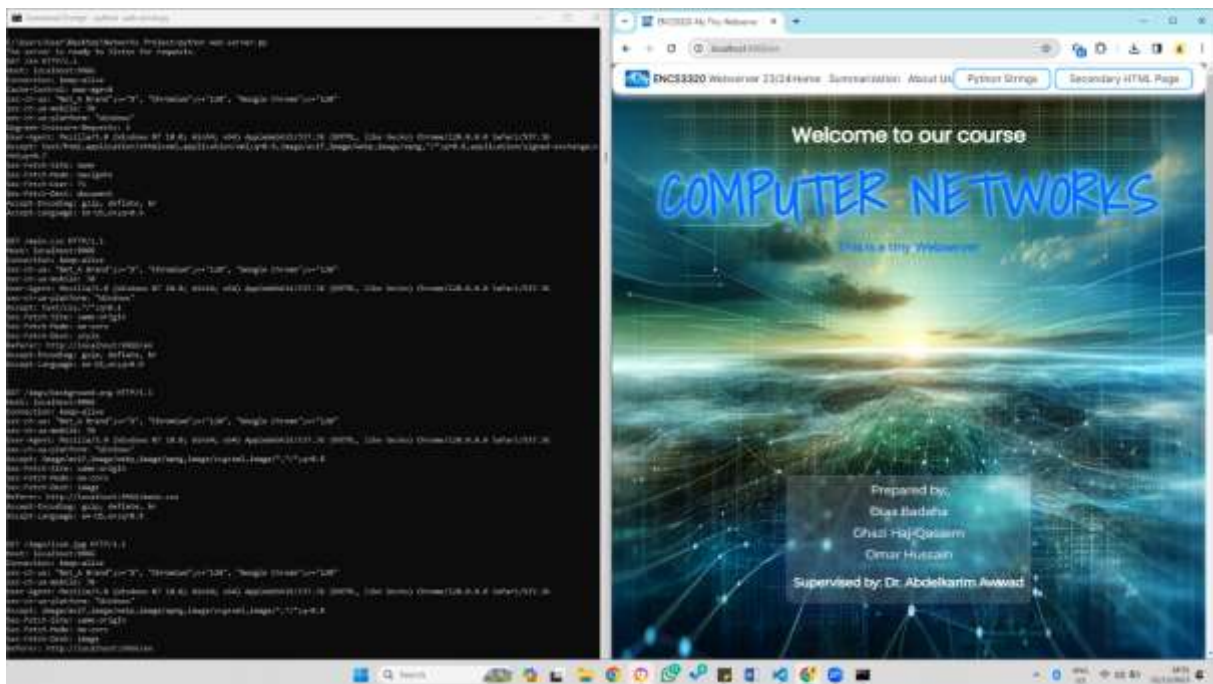


Figure 11: Requesting "/main\_en" through the browser and printing the HTTP response in the terminal window





### Point 3



Figure 15: Requesting a file .html "secondary.html" through the browser and printing the HTTP response in the terminal window

### Point 4

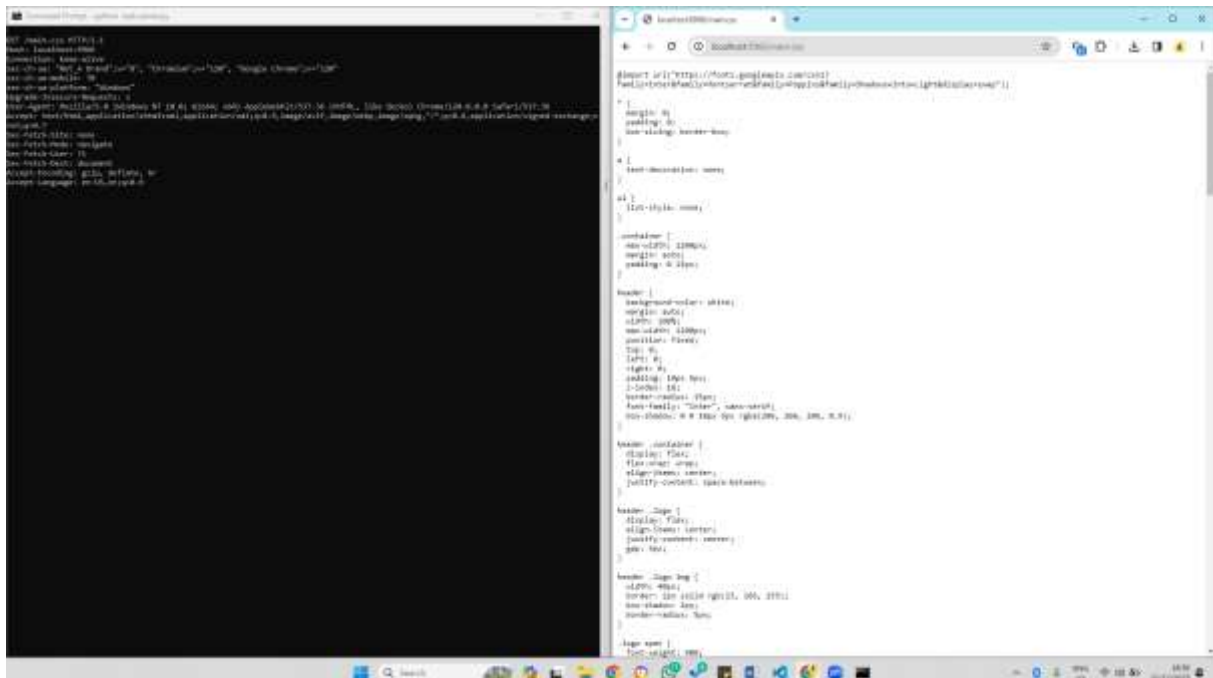


Figure 16: Requesting a file .css "main.css" through the browser and printing the HTTP response in the terminal window

## Point 5

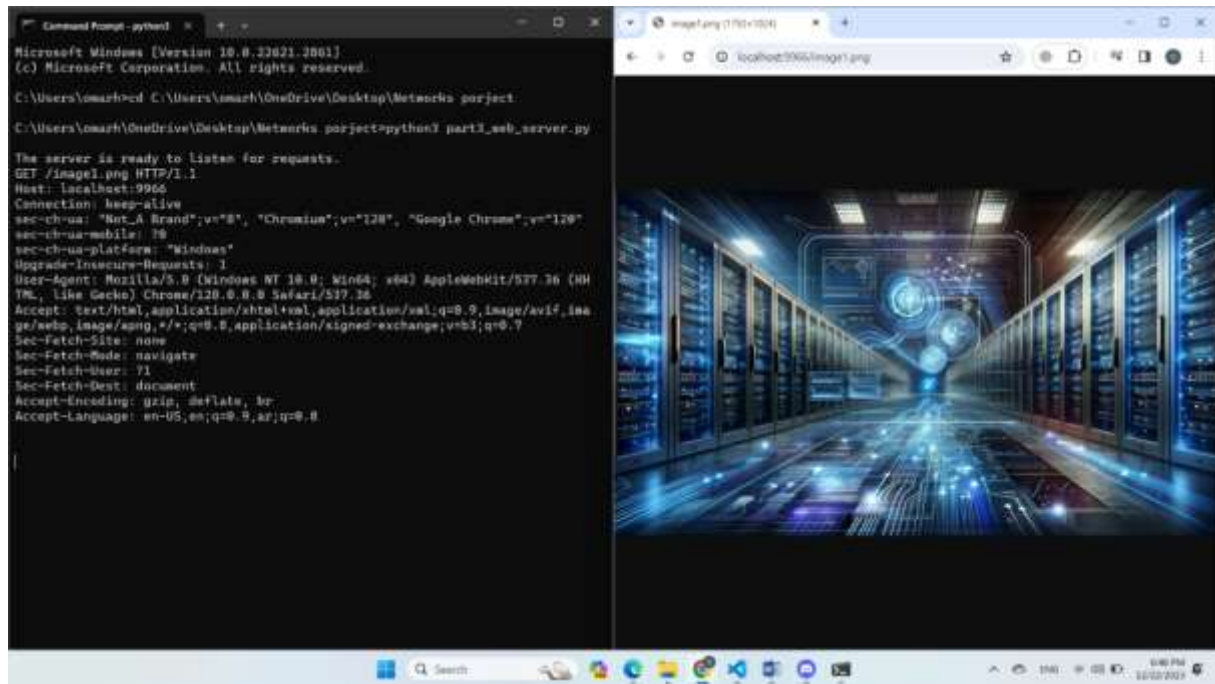


Figure 17: Requesting a png image through the browser and printing the HTTP response in the terminal window

## Point 6

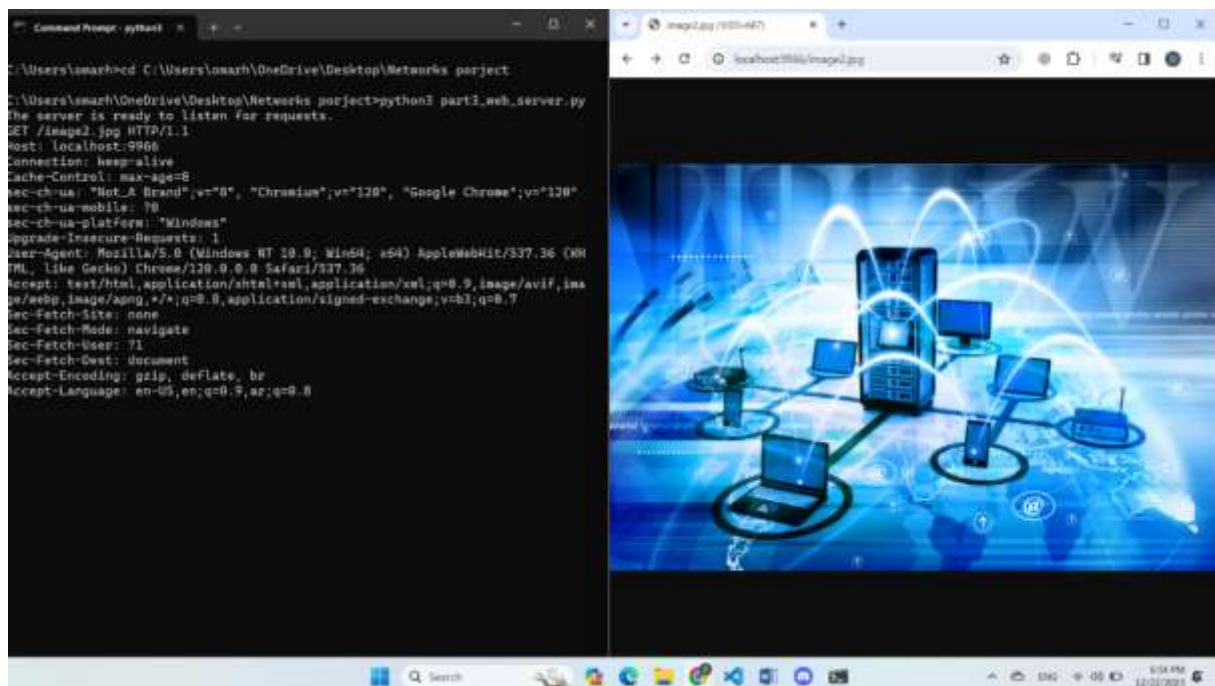


Figure 18: Requesting a jpg image through the browser and printing the HTTP response in the terminal window



## Point 7

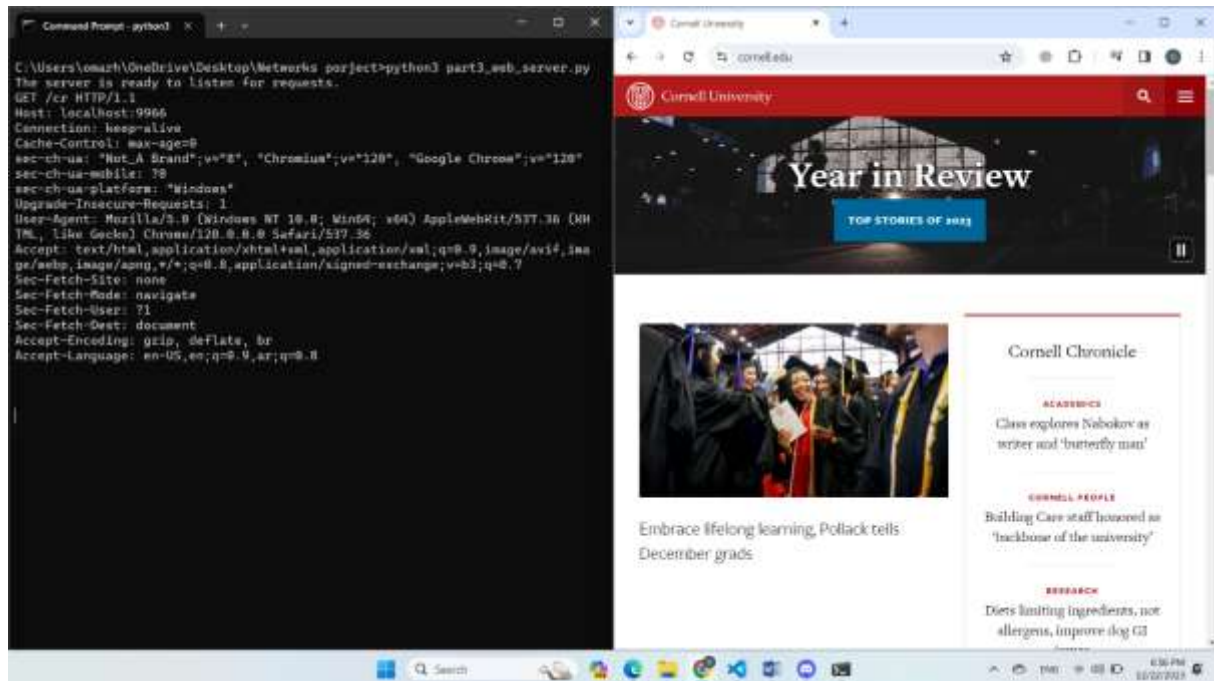


Figure 19: Requesting redirection to `cornell.edu` through `/cr` and printing the HTTP response in the terminal window

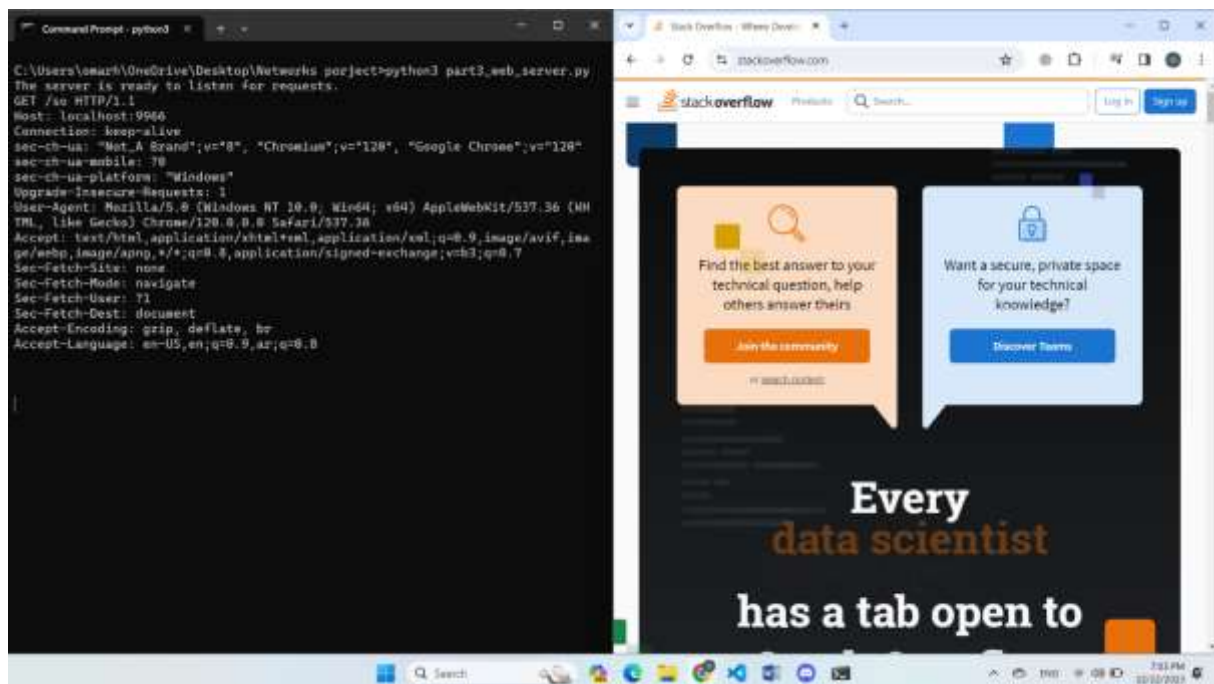


Figure 20: Requesting redirection to `stackoverflow.com` through `/so` and printing the HTTP response in the terminal window

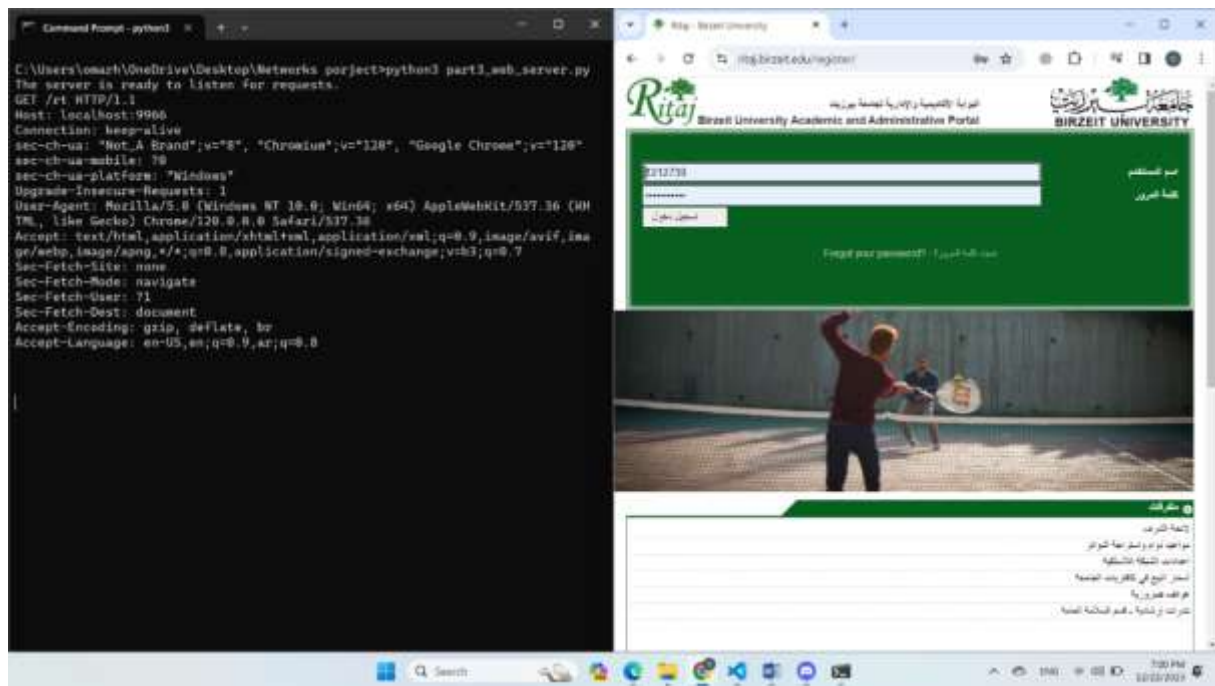


Figure 21: Requesting redirection to `ritaj.birzeit.edu` through `/rt` and printing the HTTP response in the terminal window

## Point 8

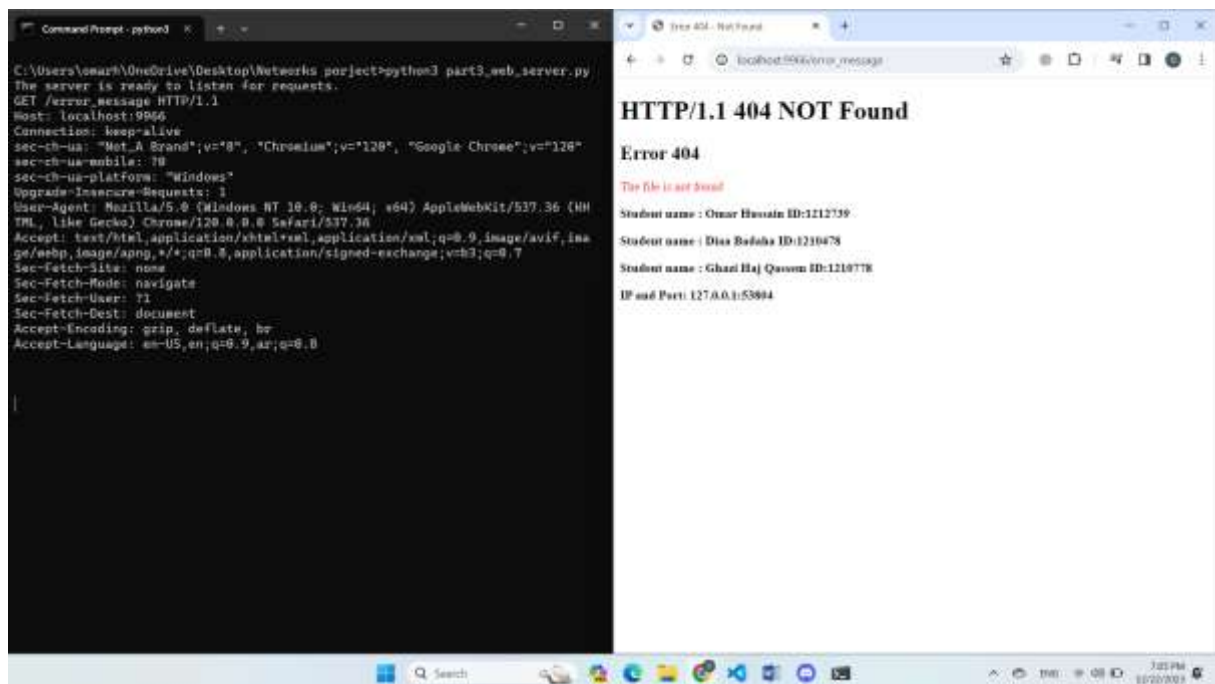


Figure 22: Requesting an invalid file and printing the HTTP response in the terminal window

## Point 9

Done in the previous points...

## Appendix

```
1 from socket import *
2
3 server_name = 'localhost' # The host is the same device running both server and client programs.
4 server_port = 8955
5
6 client_socket = socket(AF_INET, SOCK_STREAM) # Create a client socket: AF_INET is for IPv4, and SOCK_STREAM for TCP
7 client_socket.connect((server_name, server_port)) # Request a TCP connection to the server process specified with the given IP address and port number
8
9 message = input('Enter your ID number:') # Prompt the user to enter his/her ID number
10 client_socket.send(message.encode()) # Send the message to the server
11
12 new_message = client_socket.recv(2048) # Receive a message from the server
13
14 print('Server\'s message:', new_message.decode()) # Decode the message received from the server and print it
15
16 client_socket.close()
```

Figure 23: Client code

```
1 from socket import *
2 import ctypes
3 import time
4
5 lockMarkStation = ctypes.windll.user32.LockWorkStation # The method that locks the OS imported from ctypes library
6
7 server_port = 8955
8
9 server_socket = socket(AF_INET, SOCK_STREAM) # The server socket that receives the TCP connection requests from the client: AF_INET is for IPv4, and SOCK_STREAM for TCP
10 server_socket.bind(('', server_port)) # Bind the server socket with this port
11 server_socket.listen(1) # Listen for at most one client request
12 print('The server is ready to receive TCP connection requests.')
13
14 while True:
15     connection_socket, client_addr = server_socket.accept() # Accept the TCP connection request issued by the client, and create a new connection socket
16     client_message = connection_socket.recv(1024).decode() # Receive the message from the client and decode it
17
18     # Check for ID validity
19     if client_message == '1210778' or client_message == '1210478' or client_message == '1212738':
20         server_message = 'The OS will lock down after 10 seconds'
21     else:
22         server_message = 'Invalid ID'
23     # Print the message on the server side and send it to the client
24     print(server_message)
25     connection_socket.send(server_message.encode())
26     # Close the connection socket
27     connection_socket.close()
28     # Lock the OS after 10 sec
29     if client_message == '1210778' or client_message == '1210478' or client_message == '1212738':
30         time.sleep(10) # Sleep for 10 seconds
31         lockMarkStation() # Lock the OS
32
```

Figure 24: Server code

```

1  from socket import *
2
3  # Set the port number for the server to listen on
4  server_port = 9966
5
6  server_socket = socket(AF_INET, SOCK_STREAM) # Create a TCP socket
7  server_socket.bind(('', server_port)) # Bind the server
8  server_socket.listen(1) # Listen for incoming connections with a maximum queue size of 1
9  print('The server is ready to listen for requests.')
10
11 while True:
12     connection_socket, client_addr = server_socket.accept() # Get the connection
13     client_ip, client_port = client_addr # Get the IP address & the Port of the client
14     sentence = connection_socket.recv(1024).decode() # receive the message and decode it
15     print(sentence) # Print the message to the terminal window
16     list_of_words = sentence.split(' ') # Split the sentence into words
17
18     request = list_of_words[1][1:] # extract the name of the request file from the HTTP request
19     # Check what is the type of the request and send the appropriate file based on the type
20     if request == 'en' or request == 'index.html':
21         with open('main_en.html', 'rb') as file:
22             connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
23             connection_socket.send("Content-Type: text/html\r\n".encode())
24             connection_socket.send("\r\n".encode())
25             content = file.read()
26             connection_socket.send(content)
27     elif request == 'ar':
28         with open('main_ar.html', 'rb') as file:
29             connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
30             connection_socket.send("Content-Type: text/html\r\n".encode())
31             connection_socket.send("\r\n".encode())
32             content = file.read()
33             connection_socket.send(content)
34     elif request.endswith('.html'):
35         with open(request, 'rb') as file:
36             connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
37             connection_socket.send("Content-Type: text/html\r\n".encode())
38             connection_socket.send("\r\n".encode())
39             content = file.read()
40             connection_socket.send(content)
41     elif request.endswith('.css'):
42         with open(request, 'rb') as file:
43             connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
44             connection_socket.send("Content-Type: text/css\r\n".encode())
45             connection_socket.send("\r\n".encode())
46             content = file.read()
47             connection_socket.send(content)
48     elif request.endswith('.png'):
49         with open(request, 'rb') as file:
50             connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
51             connection_socket.send("Content-Type: image/png\r\n".encode())
52             connection_socket.send("\r\n".encode())
53             content = file.read()
54             connection_socket.send(content)
55     elif request.endswith('.jpg'):
56         with open(request, 'rb') as file:
57             connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
58             connection_socket.send("Content-Type: image/jpg\r\n".encode())
59             connection_socket.send("\r\n".encode())
60             content = file.read()
61             connection_socket.send(content)

```

Figure 25: Webserver code 1



```

1 elif request.endswith('.svg'):
2     with open(request, 'rb') as file:
3         connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
4         connection_socket.send("Content-Type: image/svg+xml\r\n".encode())
5         connection_socket.send("\r\n".encode())
6         content = file.read()
7         connection_socket.send(content)
8 elif request == '.cr':
9     connection_socket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
10    connection_socket.send("Content-Type: text/html\r\n".encode())
11    connection_socket.send("Location: http://cornell.edu\r\n".encode())
12    connection_socket.send("\r\n".encode())
13 elif request == '.sp':
14    connection_socket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
15    connection_socket.send("Content-Type: text/html\r\n".encode())
16    connection_socket.send("Location: http://stackoverflow.com\r\n".encode())
17    connection_socket.send("\r\n".encode())
18 elif request == '.rt':
19    connection_socket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
20    connection_socket.send("Content-Type: text/html\r\n".encode())
21    connection_socket.send("Location: http://ritaj.birzeit.edu\r\n".encode())
22    connection_socket.send("\r\n".encode())
23 else: # If the requests are other than these, then the file is not found and send an HTTP response saying that.
24     html_content="""
25     <!DOCTYPE html>
26     <html lang="en">
27     <head>
28     <title>Error 404 - Not Found</title>
29     </head>
30     <body>
31     <h1>HTTP/1.1 404 NOT Found</h1>
32     <h2>Error 404</h2>
33     <p style="color:red;">The file is not found</p>
34     <p><strong>Student name : Omar Hussain ID:1212739 </strong></p>
35     <p><strong>Student name : Oiaa Badaha ID:1210478</strong></p>
36     <p><strong>Student name : Ghazi Haj Qassem ID:1210770 </strong></p>
37     <p><strong>IP and Port: {client_ip}:{client_port}</strong></p>
38     </body>
39     </html>"""
40     response = "HTTP/1.1 404 Not Found\r\n"
41     response += "Content-Type: text/html\r\n"
42     response += "\r\n"
43     response += html_content
44     connection_socket.send(response.encode())
45 # Close the connection
46 connection_socket.close()
47
48

```

Figure 26: Webserver code 2