



**FACULTY OF ENGINEERING TECHNOLOGY  
ELECTRIC & COMPUTER ENGINEERING DEPARTMENT**

**ENCS3320 – Computer Networks  
Project #1 Socket Programming**

---

**Prepared by:**

<b>Student name</b>	<b>ID</b>	<b>Section</b>
Ahmad Owenah	1193117	4
Layan Salem	1221026	4
Tuleen Rimawi	1220211	4

**Submission date:** 1 December 2024

## Table of content

<b>Table of content</b> .....	2
<b>Table of figures</b> .....	3
<b>Table of tables</b> .....	5
<b>Theory and Procedure</b> .....	6
<b>Task 1:</b> .....	6
<b>Networking Commands and Wireshark:</b> .....	6
<b>Command Details</b> .....	6
<b>Running Network Commands:</b> .....	7
<b>Packet Capture with Wireshark:</b> .....	8
<b>Task 2 +3:</b> .....	8
<b>Task 2: Web server</b> .....	10
<b>Task 3:</b> .....	26
<b>How the game Works:</b> .....	27
<b>Procedure:</b> .....	27
<b>Results and Discussions</b> .....	41
<b>Task one:</b> .....	41
<b>Explanation</b> .....	41
<b>Trying the commands</b> .....	42
<b>Task two:</b> .....	48
<b>Task three:</b> .....	49
<b>Alternative Solutions, Issues, and Limitations</b> .....	53
Task 2:.....	53
Task 3:.....	53
<b>Teamwork Chart analysis:</b> .....	55
<b>References</b> .....	57

## Table of figures

Figure 1 socket programming .....	8
Figure 2 TCP datagram .....	9
Figure 3 UDP datagram .....	9
Figure 4 web server.....	10
Figure 5 task 2 server code 1 .....	10
Figure 6 testing html_en .....	11
Figure 7 server code 2.....	11
Figure 8 testing html_ar .....	12
Figure 9 server code 3.....	12
Figure 10 request html file.....	13
Figure 11 Response of html request.....	13
Figure 12 Request of CSS request .....	14
Figure 13 Response of CSS request.....	14
Figure 14 style of css 1 .....	15
Figure 15 style of css 2 .....	15
Figure 16 style of css 3 .....	16
Figure 17 request image.....	16
Figure 18 image .....	17
Figure 19 Response of png .....	17
Figure 20 request JPG.....	18
Figure 21 JPG image.....	18
Figure 22 Response of image.jpg request .....	19
Figure 23 code to display image in english .....	19
Figure 24 display image in English.....	20
Figure 25 code to display image (Arabic) .....	20
Figure 26 display image in Arabic.....	21
Figure 27 handle redirects .....	21
Figure 28 redirect testing .....	22
Figure 29 redirect code 1 .....	22
Figure 30 testing redirect to itc .....	23
Figure 31 Response of itc .....	23
Figure 32 wrong request handle code.....	24
Figure 33 html code for wrong request.....	25
Figure 34 wrong request page.....	25
Figure 35 UDP datagram .....	26
Figure 36 Setting Up the Server code.....	27
Figure 37 loading the Question Dataset code .....	28
Figure 38 dataset file.....	29
Figure 39 handling Client Connections code.....	30
Figure 40 starting game code 1 .....	31
Figure 41 starting game code 2 .....	32
Figure 42 starting game code 3 .....	33
Figure 43 starting game code 4 .....	34

Figure 44 main server code .....	35
Figure 45 main server code 2 .....	36
Figure 46 client code 1 .....	37
Figure 47 client code 2 .....	38
Figure 48 client code 3 .....	40
Figure 49 running ipconfig .....	42
Figure 50 running ipconfig 2 .....	42
Figure 51 running ipconfig 3 .....	43
Figure 52 testing ping1.....	44
Figure 53 testing ping 2.....	44
Figure 54 testing tracert.....	45
Figure 55 testing nslookup .....	46
Figure 56 testing telnet .....	46
Figure 57 testing telnet 2 .....	47
Figure 58 Wireshark test .....	48
Figure 59 testing task 3 part 1.....	49
Figure 60 testing task 3 part 2.....	49
Figure 61 testing task 3 part 3.....	50
Figure 62 testing task 3 part 4.....	51
Figure 63 testing task 3 part 4.....	51
Figure 64 testing task 3 part 5.....	52
Figure 65 Teamwork Chart analysis.....	55
Figure 66 Teamwork Chart analysis.....	55
Figure 67 Teamwork Chart analysis.....	56
Figure 68 Teamwork Chart analysis.....	Error! Bookmark not defined.

## Table of tables

Table 1 methods used in setting server.....	28
Table 2 methods used in loading dataset .....	29
Table 3 methods used in handling Client Connections code .....	30
Table 4 methods used in starting game1 .....	31
Table 5 methods used in starting game2 .....	32
Table 6 methods used in starting game 3.....	33
Table 7 methods used in starting game 3 .....	34
Table 8 methods used in main server code.....	35
Table 9 methods used in main server code.....	36
Table 10 methods used in client code .....	38
Table 11 methods used in client code 2.....	39
Table 12 methods used in client code3.....	40

# Theory and Procedure

## Task 1:

### **Networking Commands and Wireshark:**

Modern computer networks rely on a combination of tools and protocols for diagnosing connectivity, analyzing data flow, and troubleshooting issues. This project uses essential networking commands and Wireshark, a packet analysis tool, to explore network operations.

#### **Command Details**

- **IPCONFIG:**

The ipconfig command displays the current IP configuration of a system, including IP address, subnet mask, default gateway, and DNS server addresses. For instance, running ipconfig /all shows detailed settings. This is particularly useful for verifying network configuration and resolving connectivity issues. [1]

- **PING:**

The ping command is a fundamental tool for testing connectivity between two devices on a network. It sends Internet Control Message Protocol (ICMP) echo requests to the target and waits for replies., running ping measures round-trip time (RTT) and determines if the host is reachable. This command helps diagnose network delays and packet loss. [1]

- **TRACERT:**

The tracert (or traceroute) command traces the route data packets take to a destination. It displays each hop (router) along the path and their respective IP addresses. executing tracert can reveal delays or failures at specific points in the network. This tool is invaluable for pinpointing network bottlenecks. [1]

- NSLOOKUP:  
The nslookup command queries Domain Name System (DNS) servers to resolve domain names into IP addresses or vice versa. Using nslookup retrieves the IP address of the server hosting the domain. This is crucial for identifying DNS misconfigurations. [1]
- TELNET:  
Telnet is a protocol that establishes a text-based session with a remote host over a specified port. For instance, running telnet example.com 80 verifies connectivity to the host's HTTP service. Although largely replaced by SSH for security reasons, Telnet remains useful for basic port checks. [2]
- Wireshark:  
Wireshark is a packet analyzer that captures and displays network traffic for analysis. It enables detailed inspection of packets, such as DNS queries, aiding in troubleshooting advanced issues like dropped packets or protocol mismatches. [3]

## Running Network Commands:

- Open a command prompt or terminal.
- Use ipconfig to display the system's IP address, subnet mask, gateway, and DNS server. Record this data for future reference.
- Use ping to test:
  1. A local device (your router's IP) to ensure LAN connectivity.
  2. A public domain (e.g., google.com) to check internet connectivity.
  - 3.
- Run tracert for discover.engineering.utoronto.ca to view all intermediary routers. Note any high-latency hops that may indicate network issues.
- Use telnet to connect to discover.engineering.utoronto.ca on port 80. Check if the connection succeeds and the server responds with an HTTP banner or similar.
- Use nslookup for discover.engineering.utoronto.ca to retrieve its associated IP address and DNS server information.

## Packet Capture with Wireshark:

- Install Wireshark and start a new capture on my active network interface.
- Perform a DNS query, visit google.com in a browser while Wireshark is running.
- Stop the capture and apply a DNS filter to isolate the relevant packets.
- Analyze the captured packets, noting details such as, DNS.

## Task 2 +3:

- Socket Programming:

Socket programming is the backbone of network communication, enabling applications to exchange data. In this project, both TCP and UDP protocols are explored, demonstrating their unique characteristics.

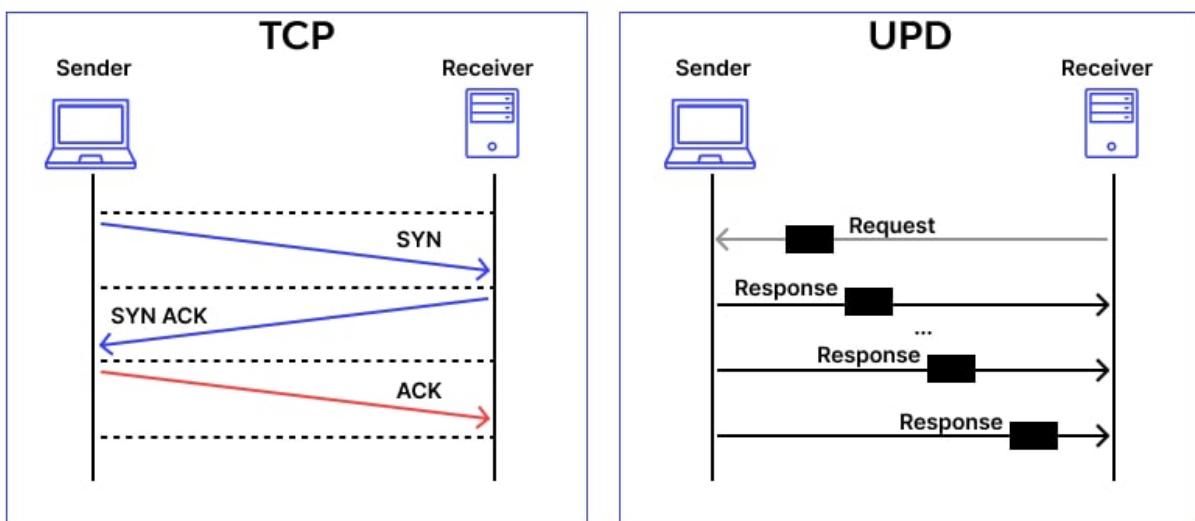


Figure 1 socket programming

- Transmission Control Protocol (TCP):

TCP is a reliable, connection-oriented protocol that ensures data delivery in the correct order. It is ideal for scenarios requiring accuracy, such as web browsing or file transfers. For example, a TCP server binds to a port, listens for incoming connections, and exchanges data with clients. [4]

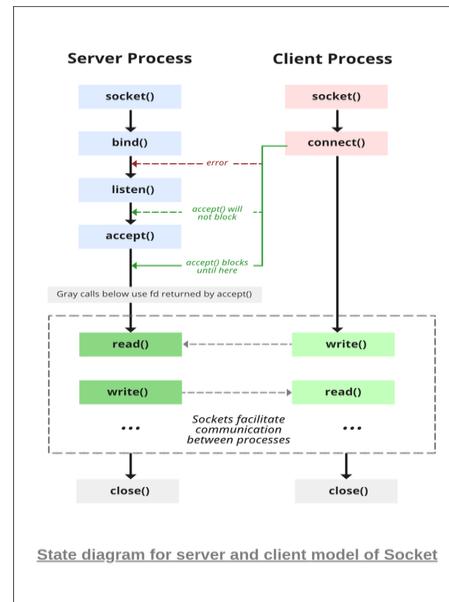


Figure 2 TCP datagram

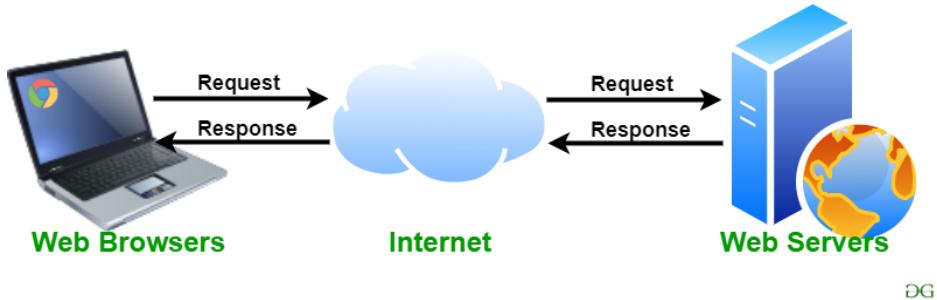
- User Datagram Protocol (UDP):

UDP is a connectionless protocol designed for speed and low latency. It is commonly used in real-time applications, such as video streaming or online gaming, where occasional packet loss is acceptable. [5]



Figure 3 UDP datagram

## Task 2: Web server



*Figure 4 web server*

In this part, we used socket programming to create a complete web server on port 5698 using the Python programming language, as well as HTML, CSS. First, we defined a server port which is 2035, then we created a socket instance and pass it two parameters, the first one is AF\_INET which means to use IPv4 protocol, and the second one is SOCK\_STREAM which means connection-oriented TCP protocol. As shown in figure 5. After creating the socket, we associated it with its local address, allowing clients to connect to the server using that address using bind(). Then we put the socket into listening mode to make it ready for the requests.

The screenshot shows a Windows desktop environment with several open windows. In the center is a Visual Studio Code window displaying Python code for a socket server. The code uses `socket` and `os` modules to handle incoming connections and receive data from clients. The code editor has syntax highlighting and line numbers. A status bar at the bottom shows the file path and some system information. To the right of the code editor, there's a vertical terminal window showing command-line output. At the bottom, the taskbar displays various pinned icons for software like FileZilla, Google Chrome, and Microsoft Edge.

```
error.txt error.css.txt server0.tx main_en.html # error.css
serverPy > ...
1  from socket import *
2
3
4  def open_file(file_name):
5      try:
6          with open(file_name, 'rb') as f:
7              return f.read()
8      except FileNotFoundError:
9          return None
10
11
12 serverPort = 5698
13 serverSocket = socket(AF_INET, SOCK_STREAM)
14 serverSocket.bind(('', serverPort))
15 serverSocket.listen(1)
16 print('The server is ready to receive')
17
18 while True:
19     connectionSocket, addr = serverSocket.accept()
20     ip = addr[0]
21     port = addr[1]
22     try:
23         sentence = connectionSocket.recv(1024).decode()
24         print('Received: ', sentence)
25         split = sentence.split(" ")
26
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... x
de\extensions\ms-python.debugpy-2024.12.0-win32-x64\bundled\Libs\debugpy\adapter\...\debugpy\launcher' '49919' '--' 'C:\Users\Asus\Desktop\Part 2\server0.py' x Python Deb
sers\Asus\Desktop\Part 2\server0.py
In 122, Col 1 Spaces: 4 UTF-8 CR/LF () Python 3.12.0-64-bit 12/1 ENG Wi-Fi 6/20 10:00
```

*Figure 5 task 2 server code 1*

The server will accept and complete the connection by using `accept()`. Then we will receive the http request from the server and decode it to store it in the “sentence” variable.

- English version of web server If the request is / or /index.html or /main\_en.html or / en then the server should send main\_en.html file. If the request is / or main\_ar.html then the main html file will be sent to the client. To send the html file, the server will first open that file then read it. Then it will send the header which contains of the encoded response status. Finally, the server will send the encoded file that it read previously. As shown in figures below :

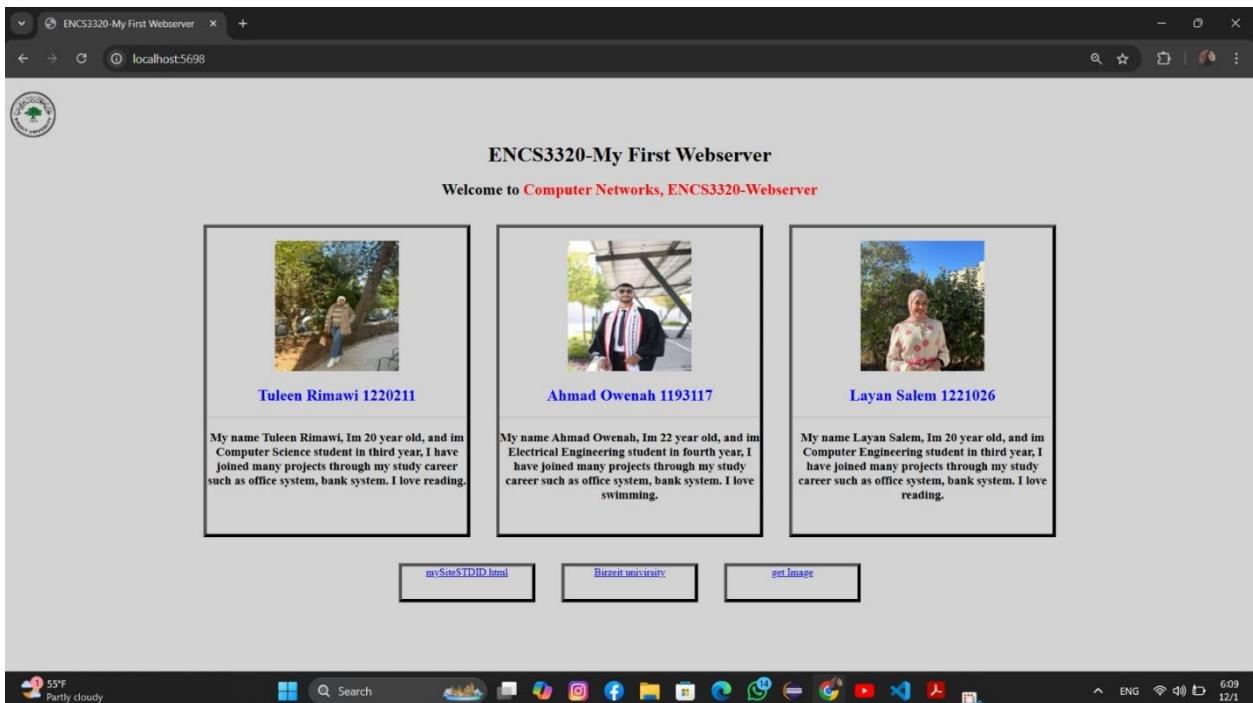


Figure 6 testing *html\_en*

```

32
33 if split[1] == "/" or split[1].startswith("/en") or split[1].startswith("/main_en.html") or split[1].startswith(
34     "/index.html"):
35     connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
36     connectionSocket.send(b'Content-Type: text/html\r\n')
37     connectionSocket.send(b'\r\n')
38     content = open_file('main_en.html')
39     if content:
40         connectionSocket.send(content)
41

```

The code editor interface shows the file is saved as "mySiteSTDID.html", and the status bar indicates "Ln 11, Col 1" and "Python 3.12.0 64-bit".

Figure 7 server code 2

## 2. Arabic version of web server

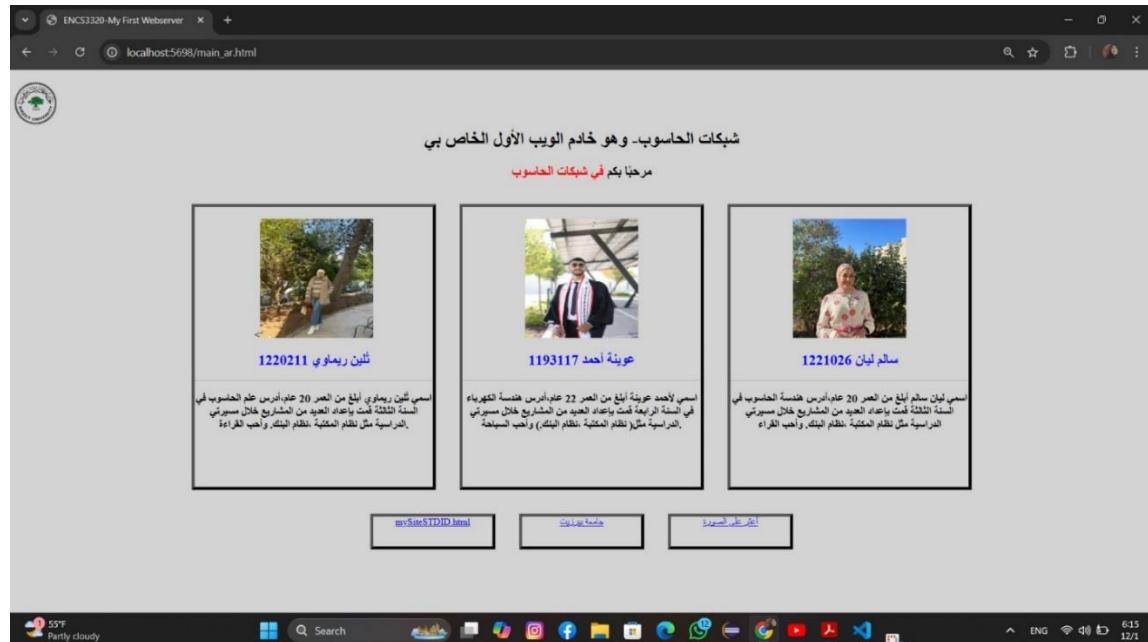


Figure 8 testing html\_ar

```

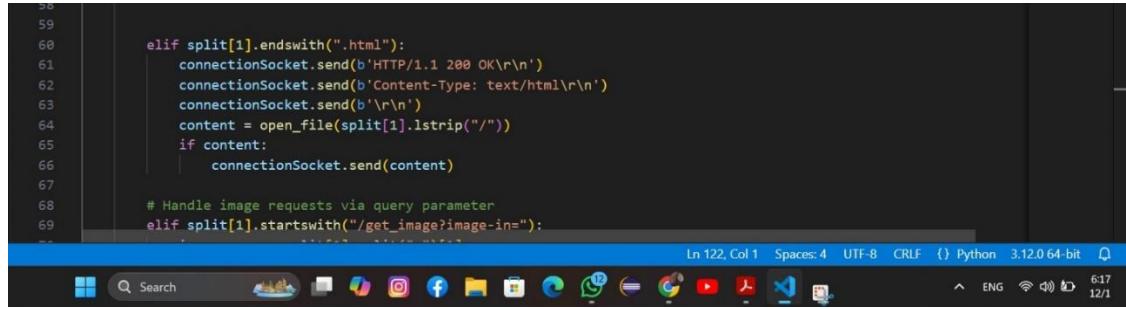
41
42 elif split[1].startswith("/ar") or split[1].startswith("/main_ar.html"):
43     connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
44     connectionSocket.send(b'Content-Type: text/html\r\n')
45     connectionSocket.send(b'\r\n')
46     content = open_file('main_ar.html')
47     if content:
48         connectionSocket.send(content)
49
50
51 elif split[1].endswith(".css"):
52     connectionSocket.send(b'HTTP/1.1 200 OK\r\n')

```

Ln 122, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit 6:16  
12/1

Figure 9 server code 3

### 3. Request of html file:

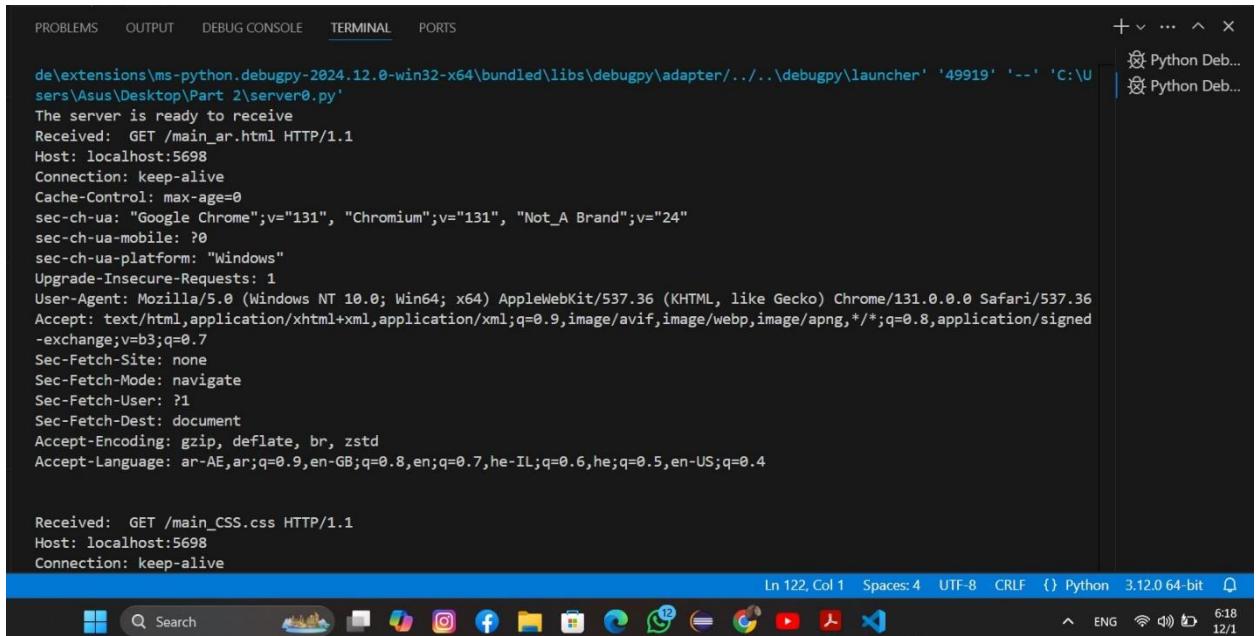


```
58
59
60     elif split[1].endswith(".html"):
61         connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
62         connectionSocket.send(b'Content-Type: text/html\r\n')
63         connectionSocket.send(b'\r\n')
64         content = open_file(split[1].lstrip("/"))
65         if content:
66             connectionSocket.send(content)
67
68     # Handle image requests via query parameter
69     elif split[1].startswith("/get_image?image-in="):
70
```

The screenshot shows a terminal window with Python code for a web server. The code handles requests for HTML files by opening them and sending the content back to the client. It also handles image requests via a query parameter. The terminal window includes status bars at the bottom showing line count (Ln 122), column count (Col 1), spaces (Spaces: 4), encoding (UTF-8), and file type (CRLF). It also shows the Python version (3.12.0 64-bit) and system status (6:17, ENG, WiFi, battery level 12/1).

Figure 10 request html file

If the request is /html then the main html file will be sent to the client. To send the html file, the server will first open that file then read it. Then it will send the header which contains of the encoded response status (which is OK), the encoded content type. Finally, the server will send the encoded file and open the local html file which called (main\_en.html).



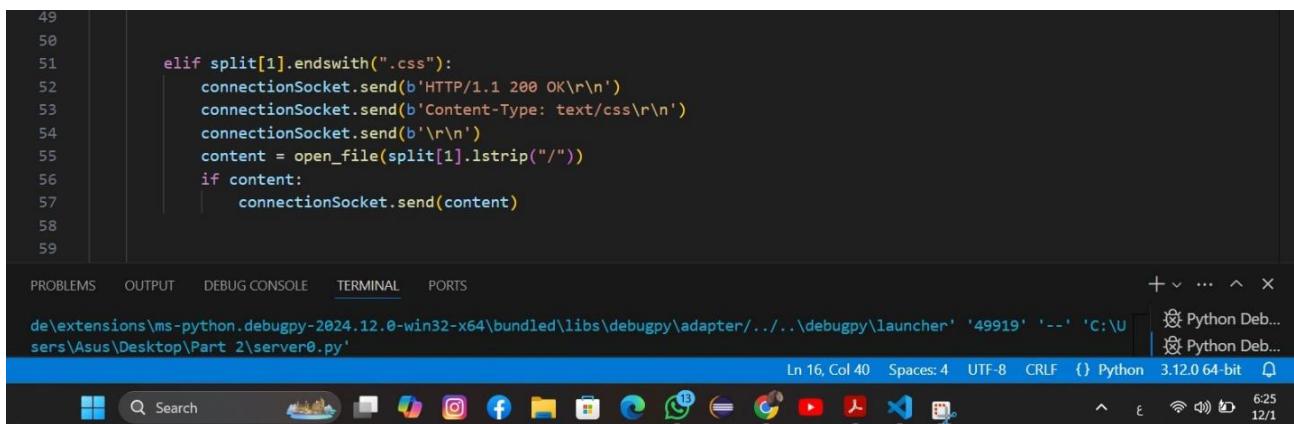
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... x
de\extensions\ms-python.debugpy-2024.12.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '49919' '--' 'C:\U
sers\Asus\Desktop\Part 2\server0.py'
The server is ready to receive
Received: GET /main_ar.html HTTP/1.1
Host: localhost:5698
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A_Brand";v="24"
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/131.0.0.0 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: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ar-AE,ar;q=0.9,en-GB;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4

Received: GET /main_CSS.css HTTP/1.1
Host: localhost:5698
Connection: keep-alive
```

The screenshot shows a terminal window displaying the response of an HTML request. The response includes the server's log message, the received HTTP request (GET /main\_ar.html), and the client's headers (User-Agent, Accept, etc.). The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is active), and PORTS. It also shows the Python version (3.12.0 64-bit) and system status (6:18, ENG, WiFi, battery level 12/1).

Figure 11 Response of html request

4. Request the CSS file: This code handles the case where the file name contains ".css". If true, it sends an HTTP response header indicating success (200 OK) and specifies that the content type is CSS. After that, it sends the contents of the specified CSS file as the response body.



```

49
50
51     elif split[1].endswith(".css"):
52         connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
53         connectionSocket.send(b'Content-Type: text/css\r\n')
54         connectionSocket.send(b'\r\n')
55         content = open_file(split[1].lstrip("/"))
56         if content:
57             connectionSocket.send(content)
58
59

```

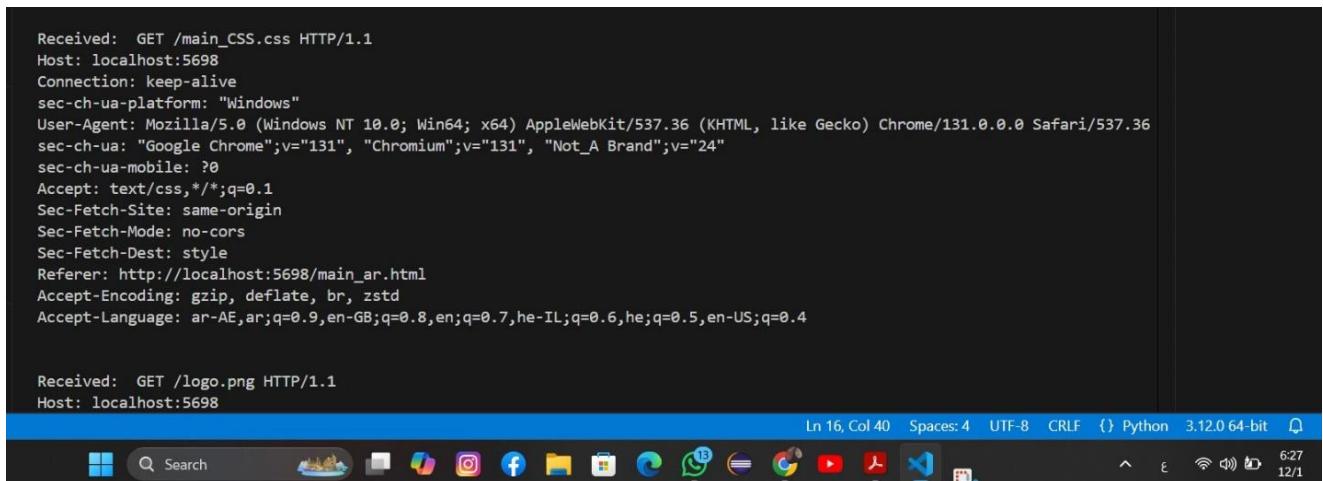
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

de\extensions\ms-python.debugpy-2024.12.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher' '49919' '--' 'C:\U  
sers\Asus\Desktop\Part 2\server0.py'

Ln 16, Col 40 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit

File Explorer Taskbar Search Start button

Figure 12 Request of CSS request



```

Received: GET /main_CSS.css HTTP/1.1
Host: localhost:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A_Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: text/css,*/*;q=0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://localhost:5698/main_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ar-AE,ar;q=0.9,en-GB;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4

Received: GET /logo.png HTTP/1.1
Host: localhost:5698

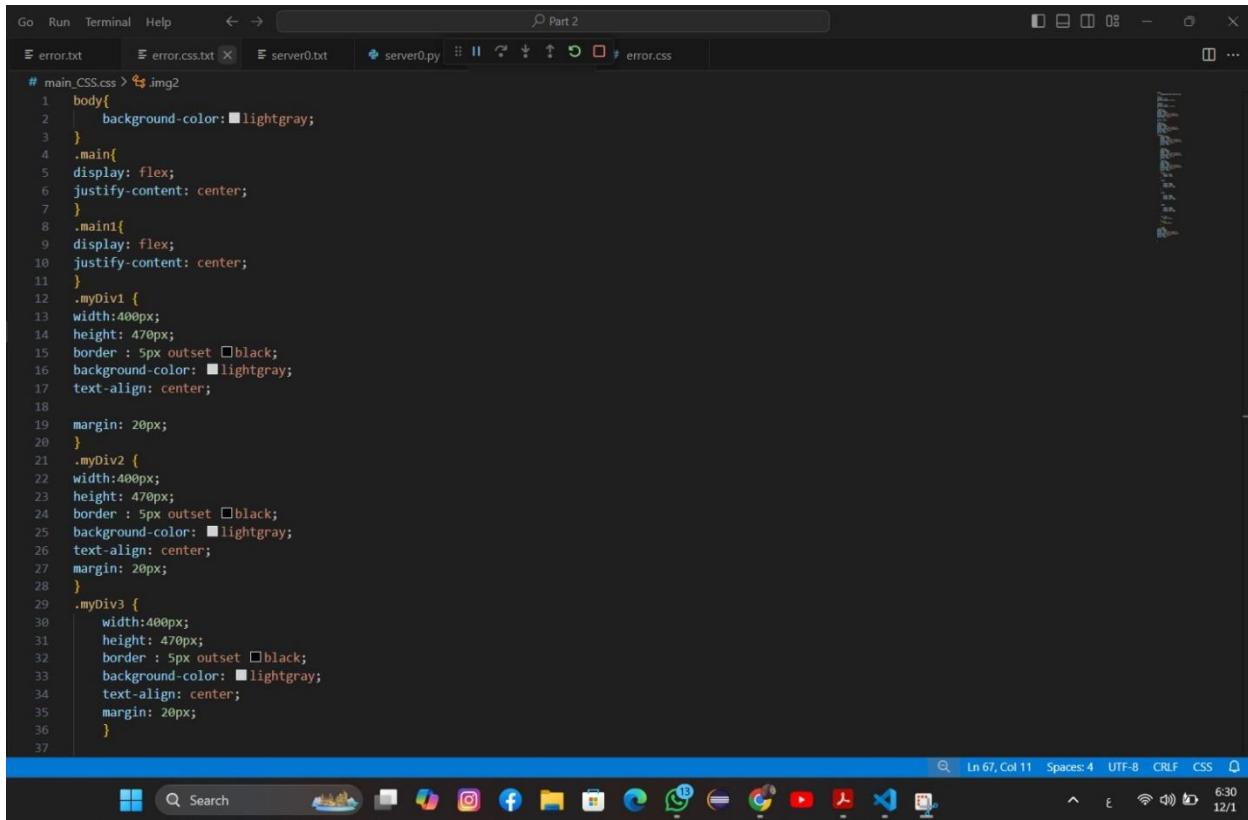
```

Ln 16, Col 40 Spaces: 4 UTF-8 CRLF {} Python 3.12.0 64-bit

File Explorer Taskbar Search Start button

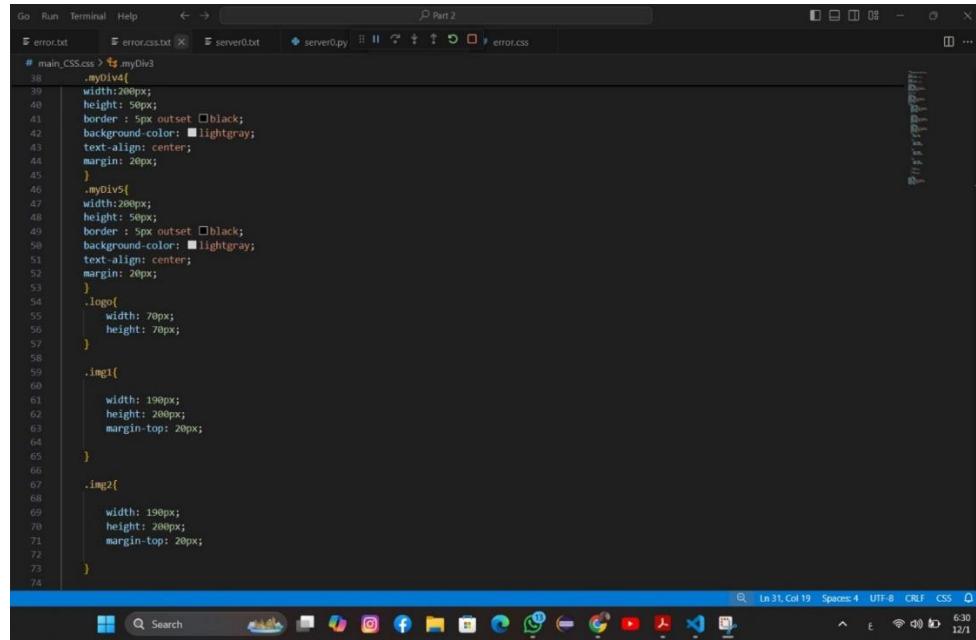
Figure 13 Response of CSS request

## The Style of CSS:



```
# main_CSS.css > ↗ img2
1 body{
2     background-color: lightgray;
3 }
4 .main{
5 display: flex;
6 justify-content: center;
7 }
8 .main1{
9 display: flex;
10 justify-content: center;
11 }
12 .myDiv1 {
13 width:400px;
14 height: 470px;
15 border : 5px outset black;
16 background-color: lightgray;
17 text-align: center;
18
19 margin: 20px;
20 }
21 .myDiv2 {
22 width:400px;
23 height: 470px;
24 border : 5px outset black;
25 background-color: lightgray;
26 text-align: center;
27 margin: 20px;
28 }
29 .myDiv3 {
30 width:400px;
31 height: 470px;
32 border : 5px outset black;
33 background-color: lightgray;
34 text-align: center;
35 margin: 20px;
36
37 }
```

Figure 14 style of css 1



```
# main_CSS.css > ↗ myDiv3
38 .myDiv4{
39     width:200px;
40     height: 50px;
41     border : 5px outset black;
42     background-color: lightgray;
43     text-align: center;
44     margin: 20px;
45 }
46 .myDiv5{
47     width:200px;
48     height: 50px;
49     border : 5px outset black;
50     background-color: lightgray;
51     text-align: center;
52     margin: 20px;
53 }
54 .logo{
55     width: 70px;
56     height: 70px;
57 }
58 .img1{
59     width: 190px;
60     height: 200px;
61     margin-top: 20px;
62 }
63
64 .img2{
65     width: 190px;
66     height: 200px;
67     margin-top: 20px;
68 }
69
70
71
72
73
74 }
```

Figure 15 style of css 2

```

Go Run Terminal Help ← → Part 2
error.txt error.css.txt server0.txt server0.py # error.css
# main_CSS.css > img2
0>
66 .img2{
67     width: 190px;
68     height: 200px;
69     margin-top: 20px;
70 }
71
72 .img3{
73     width: 190px;
74     height: 200px;
75     margin-top: 20px;
76 }
77
78 .red-text{
79     color: red;
80 }
81 .text1{
82     color: blue;
83 }
84
85 .line{
86     width:450px;
87     height: 20px;
88     border : 2px outset black;
89     background-color: lightgray;
90     text-align: center;
91     margin: 50px;
92 }
93
94 }
95
96
97
98 }

```

The screenshot shows a code editor window with several tabs at the top: 'error.txt', 'error.css.txt' (which is the active tab), 'server0.txt', 'server0.py', and '# error.css'. The main area contains CSS code for 'main\_CSS.css'. It includes rules for '.img2' (width: 190px, height: 200px, margin-top: 20px), '.img3' (width: 190px, height: 200px, margin-top: 20px), '.red-text' (color: red), '.text1' (color: blue), and '.line' (width: 450px, height: 20px, border: 2px outset black, background-color: lightgray, text-align: center). There are also margins of 50px above and below the line rule. The bottom status bar shows 'Ln 73, Col 6' and other system information.

Figure 16 style of css 3

- Request an image with PNG format This code checks if the file name contains ".png". If it does, it opens the file, sends an HTTP response header indicating success (200 OK), and specifies that the content type is a PNG image. After that, it sends the contents of the PNG file. As code bellow:

```

81
82     # Handle direct image file requests
83     elif split[1].endswith(".jpg") or split[1].endswith(".png"):
84         file = split[1].lstrip("/")
85         content = open_file(file)
86         if content:
87             if file.endswith(".jpg"):
88                 connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
89                 connectionSocket.send(b'Content-Type: image/jpeg\r\n')
90             elif file.endswith(".png"):
91                 connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
92                 connectionSocket.send(b'Content-Type: image/png\r\n')
93             connectionSocket.send(b'\r\n')
94             connectionSocket.send(content)
95

```

The screenshot shows a code editor window with a dark theme. The code handles image file requests. It checks if the file name ends with '.jpg' or '.png'. If it does, it strips the leading '/' from the file path, opens the file, and then sends an HTTP response header (HTTP/1.1 200 OK) followed by the content type (image/jpeg for jpg or image/png for png). Finally, it sends the file content and a blank line. Lines 81 through 95 are shown.

Figure 17 request image

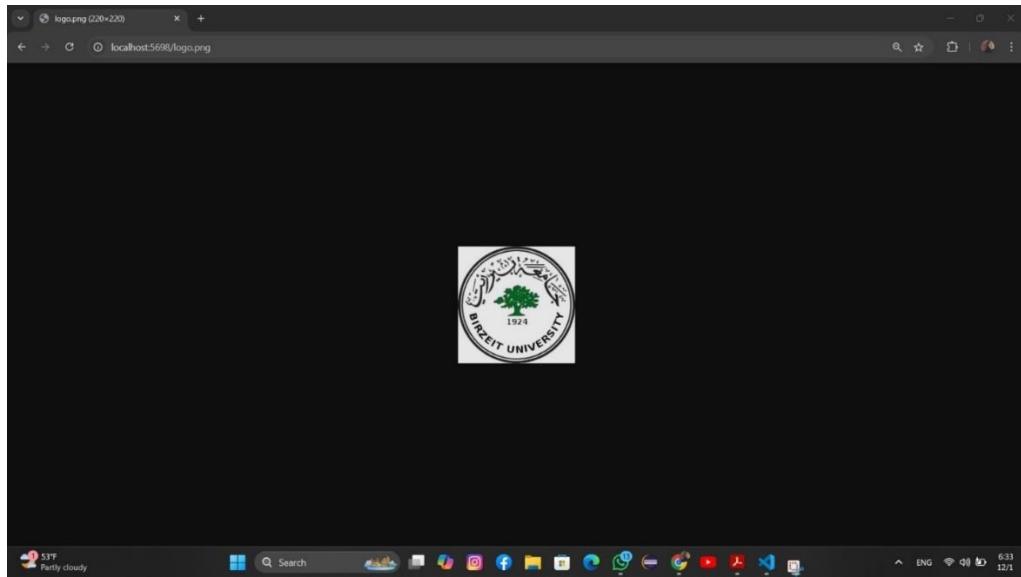


Figure 18 image

```

Received: GET /logo.png HTTP/1.1
Host: localhost:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A_Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:5698/main_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ar-AE,ar;q=0.9,en-GB;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4

Received: GET /img1.jpg HTTP/1.1
Host: localhost:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A_Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:5698/main_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ar-AE,ar;q=0.9,en-GB;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4

```

Figure 19 Response of png

6. Request an image with JPG format: This code checks if the file name contains ".jpg". If it does, it opens the file, sends an HTTP response header indicating success (200 OK), and specifies that the content type is a JPEG image. After that, it sends the contents of the JPEG file as the response body. As code bellow:

```

67     # Handle image requests via query parameter
68     elif split[1].startswith("/get_image?image-in="):
69         image_name = split[1].split("=")[1]
70         content = open_file(image_name)
71         if content:
72             if image_name.endswith(".jpg"):
73                 connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
74                 connectionSocket.send(b'Content-Type: image/jpeg\r\n')
75             elif image_name.endswith(".png"):
76                 connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
77                 connectionSocket.send(b'Content-Type: image/png\r\n')
78                 connectionSocket.send(b'\r\n')
79                 connectionSocket.send(content)
80             connectionSocket.send(content)
81
82     # Handle direct image file requests
83     elif split[1].endswith(".jpg") or split[1].endswith(".png"):
84         file = split[1].strip("/")
85         content = open_file(file)
86         if content:
87             if file.endswith(".jpg"):
88                 connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
89                 connectionSocket.send(b'Content-Type: image/jpeg\r\n')
90             elif file.endswith(".png"):
91                 connectionSocket.send(b'HTTP/1.1 200 OK\r\n')
92                 connectionSocket.send(b'Content-Type: image/png\r\n')
93                 connectionSocket.send(b'\r\n')
94             connectionSocket.send(content)
95
96

```

Figure 20 request JPG

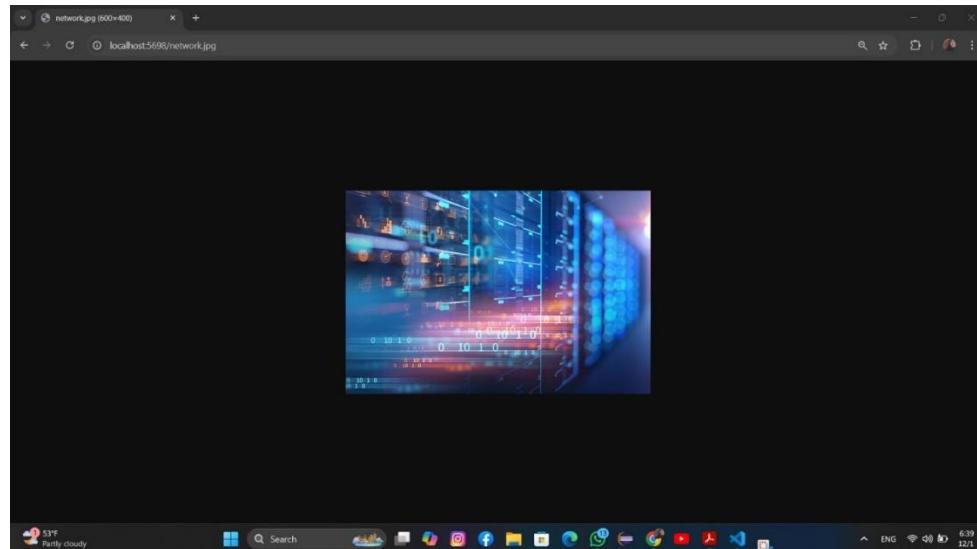


Figure 21 JPG image

```

Received: GET /network.jpg HTTP/1.1
Host: localhost:5698
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A_Brand";v="24"
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/131.0.0.0 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: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ar-AE,ar;q=0.9,en-GB;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4

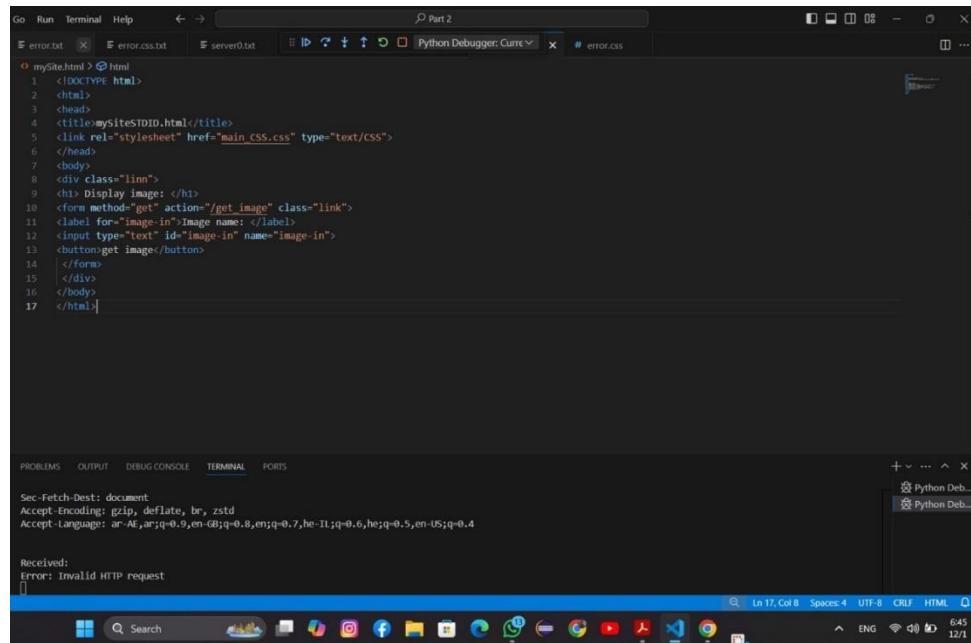
```

Figure 22 Response of image.jpg request

## 7. Store some images in a folder

### 8. Display image :

This site made for getting an images by writing the name of the image in box then select the button (get image) to display.



```

mySite.html > HTML
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>mySite</title>
5 <link rel="stylesheet" href="main.css" type="text/css">
6 </head>
7 <body>
8 <div class="linn">
9 <h1>Display image: </h1>
10 <form method="get" action="/get_image" class="link">
11 <label for="image-in">Image name: </label>
12 <input type="text" id="image-in" name="image-in">
13 <button type="button" value="Get Image"/>
14 </form>
15 </div>
16 </body>
17 </html>

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ar-AE,ar;q=0.9,en-GB;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4

Received:
Error: Invalid HTTP request

```

Figure 23 code to display image in english

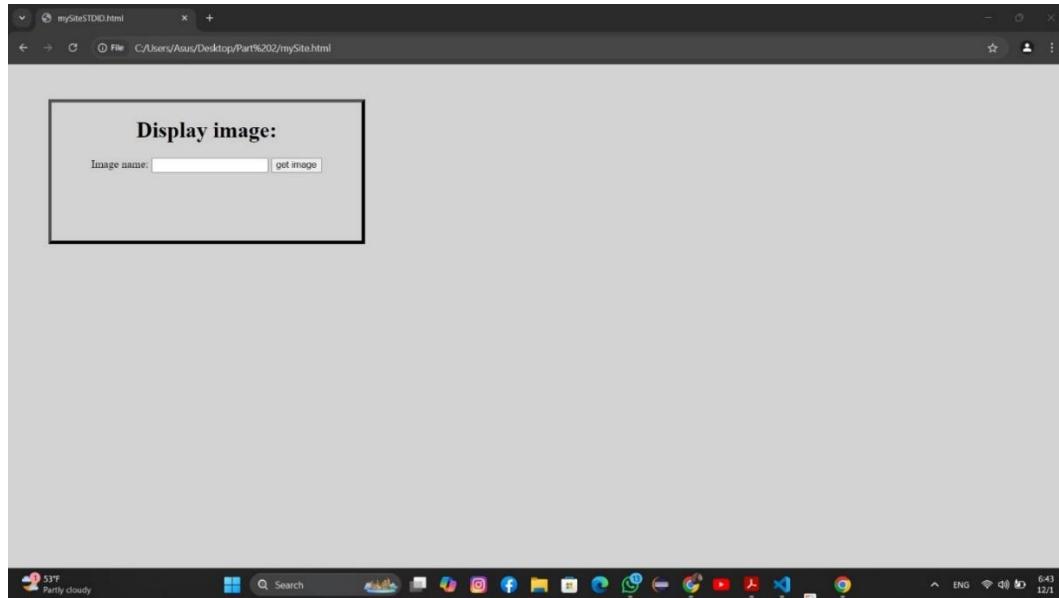


Figure 24 display image in English

```

Go Run Terminal Help ← → Part 2
error.txt error.css.txt server0.txt Python Debugger: Current # error.css
mySite-a.html > html > head
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>mySiteSTDID.html</title>
7   <link rel="stylesheet" href="main_CSS.css" type="text/css">
8 </head>
9 <body>
10 <div class="linn">
11   <h1>عرض الصورة</h1>
12   <form method="get" action="/get_image" class="link">
13
14     <button>إذن لعرض الصورة</button>
15     <input type="text" id="image-in" name="image-in">
16     <label for="image-in">اسم الصورة</label>
17
18   </form>
19 </div>
20 </body>
21 </html>

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Sec-Fetch-Dest: document  
Accept-Encoding: gzip, deflate, br, zstd  
Accept-Language: ar-AE,ar;q=0.9,en-US;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4

Received:  
Error: Invalid HTTP request

Python Deb... Python Deb...

Ln 3, Col 7 Spaces: 2 UTF-8 CRLF HTML

Figure 25 code to display image (Arabic)

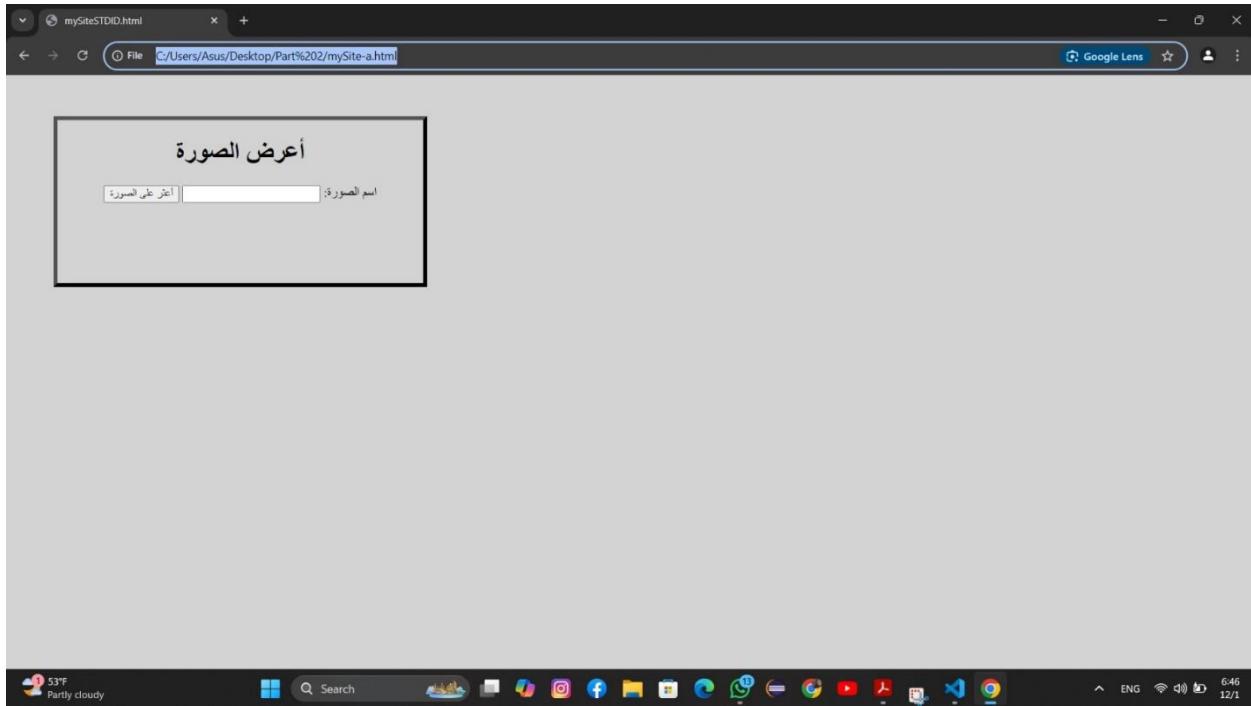


Figure 26 display image in Arabic

9. 307 Temporary Redirect By using the status code 307 Temporary Redirect to redirect the following: a) /bo request : This code checks if the file is equal to "so". If it is, the server sends an HTTP response with a 307 Temporary Redirect status. It indicates that the requested resource is temporarily located at a different URL. The server also sets the content type to HTML and redirects the client to "https://gaia.cs.umass.edu/kurose\_ross/index.php" by specifying the Location header, as bellow:

```
95
96     # Handle redirects
97     elif split[1] == "/bo":
98         connectionSocket.send(b"HTTP/1.1 307 Temporary Redirect\r\n")
99         connectionSocket.send(b"Content-Type: text/html; charset=utf-8\r\n")
100        connectionSocket.send(b"Location: https://gaia.cs.umass.edu/kurose_ross/index.php\r\n")
101        connectionSocket.send(b"\r\n")
```

Figure 27 handle redirects

## Directly transmitted to Computer Networking after sending

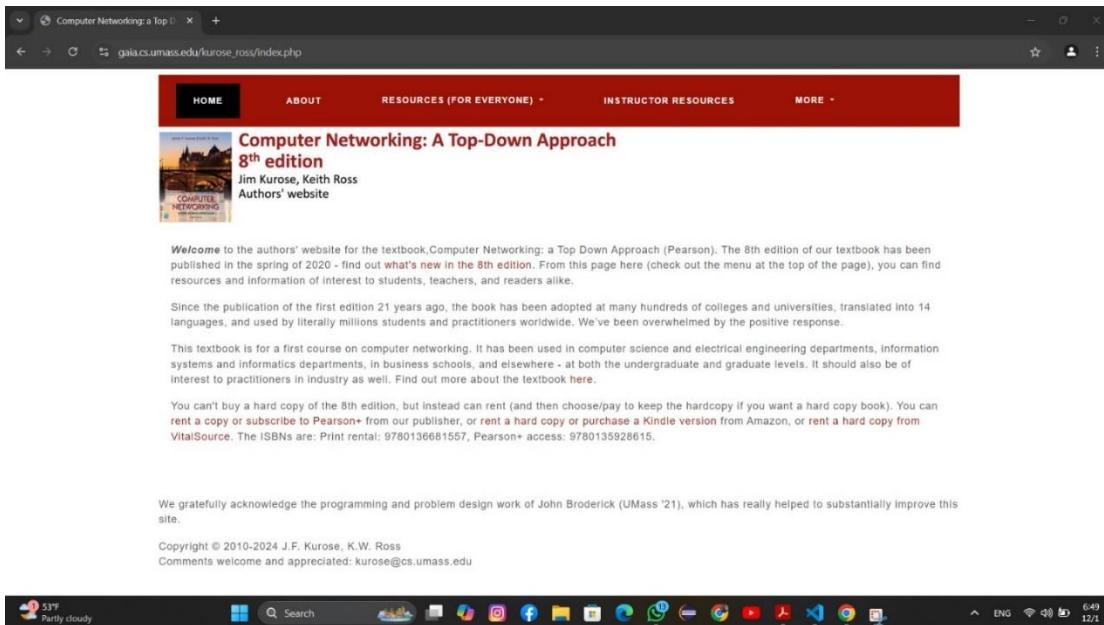


Figure 28 redirect testing

This code in figure 26 checks if the file is equal to "itc". If it is, the server sends an HTTP response with a 307 Temporary Redirect status, indicating a temporary redirection. It sets the content type to HTML and redirects the client to "<https://itc.birzeit.edu/>" by specifying the location header.

```
101 connectionSocket.send(b"\r\n")
102
103 elif split[1] == "/itc":
104     connectionSocket.send(b"HTTP/1.1 307 Temporary Redirect\r\n")
105     connectionSocket.send(b"Content-Type: text/html; charset=utf-8\r\n")
106     connectionSocket.send(b"Location: https://itc.birzeit.edu/\r\n")
107     connectionSocket.send(b"\r\n")
108
109 elif split[1] == "/vi":
```

Figure 29 redirect code 1

- Directly transmitted to itc website after sending

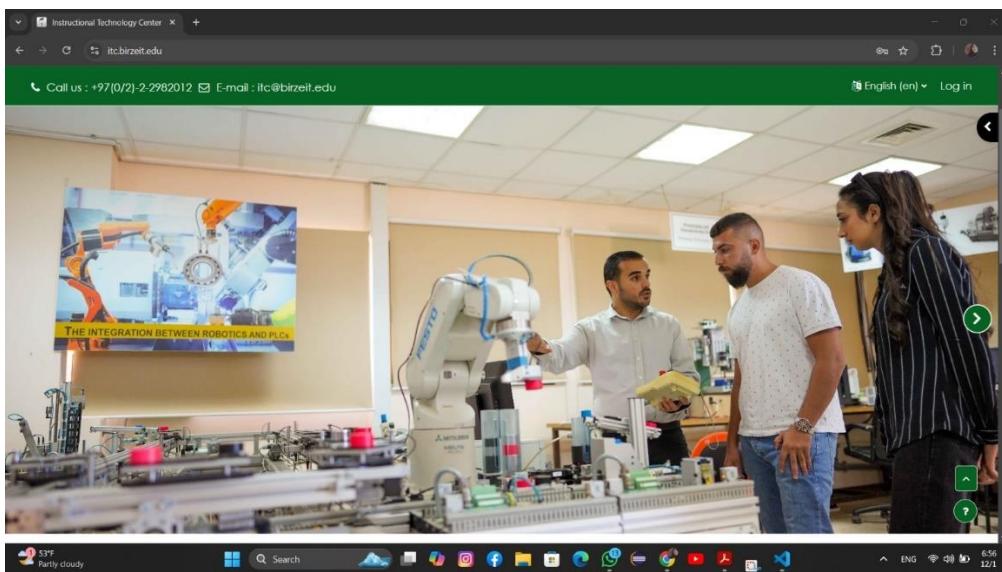


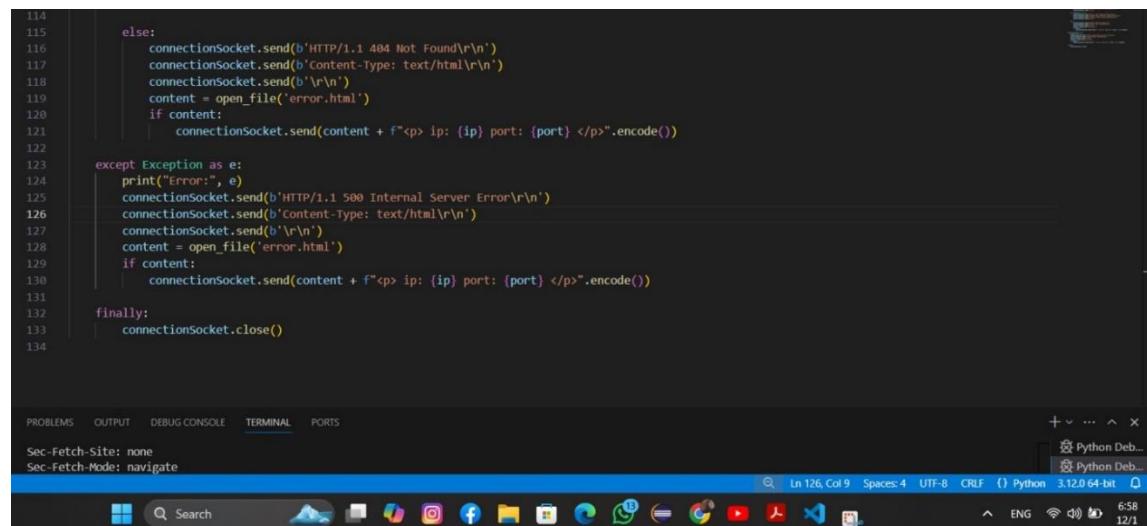
Figure 30 testing redirect to itc

```
Received: GET /itc HTTP/1.1
Host: localhost:5698
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: ar-AE,ar;q=0.9,en-GB;q=0.8,en;q=0.7,he-IL;q=0.6,he;q=0.5,en-US;q=0.4
```

Figure 31 Response of itc

## 10. Wrong request:

This code handles cases where the requested file does not match any of the previous conditions. In such cases, the server sends an HTTP response with a 404 Not Found status, indicating that the requested resource is unavailable. The content type is set to HTML, and the server sends the contents of an "error.html" file along with a custom message that includes the client's IP address and port number.



```
114
115     else:
116         connectionSocket.send(b'HTTP/1.1 404 Not Found\r\n')
117         connectionSocket.send(b'Content-Type: text/html\r\n')
118         connectionSocket.send(b'\r\n')
119         content = open_file('error.html')
120         if content:
121             connectionSocket.send(content + f"<p> ip: {ip} port: {port} </p>".encode())
122
123     except Exception as e:
124         print("Error:", e)
125         connectionSocket.send(b'HTTP/1.1 500 Internal Server Error\r\n')
126         connectionSocket.send(b'Content-Type: text/html\r\n')
127         connectionSocket.send(b'\r\n')
128         content = open_file('error.html')
129         if content:
130             connectionSocket.send(content + f"<p> ip: {ip} port: {port} </p>".encode())
131
132     finally:
133         connectionSocket.close()
```

The screenshot shows a terminal window in a development environment. The code is displayed in a code editor with line numbers from 114 to 134. The code handles an else block for a request that doesn't match previous conditions. It sends an HTTP 404 Not Found response with a Content-Type of text/html. It then attempts to read the content of 'error.html' and send it back to the client, including a custom message with the client's IP address and port number. If an exception occurs, it instead sends an HTTP 500 Internal Server Error response. Finally, it closes the connection socket. The terminal window also shows some system information at the bottom, including the Python version (3.12.0) and the current time (6:58).

Figure 32 wrong request handle code

The screenshot shows a code editor window titled "Part 2". The tab bar includes "error.txt", "error.css.txt", "server0.txt", "server0.py", "# error.css", and "error.html". The "error.html" tab is active, displaying the following HTML code:

```
error.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title> Error 404 </title>
5  <link rel="stylesheet" href="error.css" type="text/css">
6  </head>
7  <body>
8  <h1 class="text"> HTTP/1.1 404 Not Found" in the response status </h1>
9  <h1 class="error"> The file is not found </h1>
10 <p class="text">Tuleen Rimawi 1220211</p>
11 <p class="text">Ahmad Owenah 1193117</p>
12 <p class="text">Layan Salem 1221026</p>
13
14 </body>
15 </html>
```

The status bar at the bottom shows "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL", and "PORTS". The taskbar below the editor includes icons for File Explorer, Task View, Start, Search, Edge, File, Photos, Instagram, Facebook, Mail, Calendar, OneDrive, Google Sheets, Google Slides, Google Docs, YouTube, File Explorer, and Task View. The system tray shows "Sec-Fetch-Site: none", "Sec-Fetch-Mode: navigate", "Ln 15, Col 8", "Spaces 4", "UTF-8", "CR LF", "HTML", "7:01", "ENG", "Wi-Fi", "Battery", and "12/1".

Figure 33 html code for wrong request

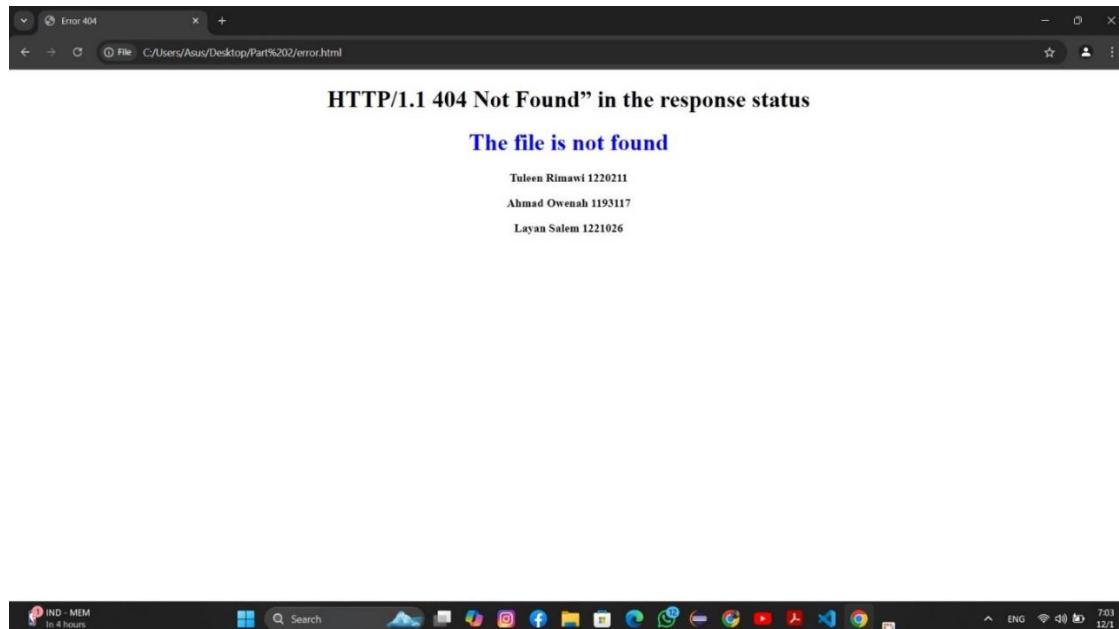


Figure 34 wrong request page

### Task 3:

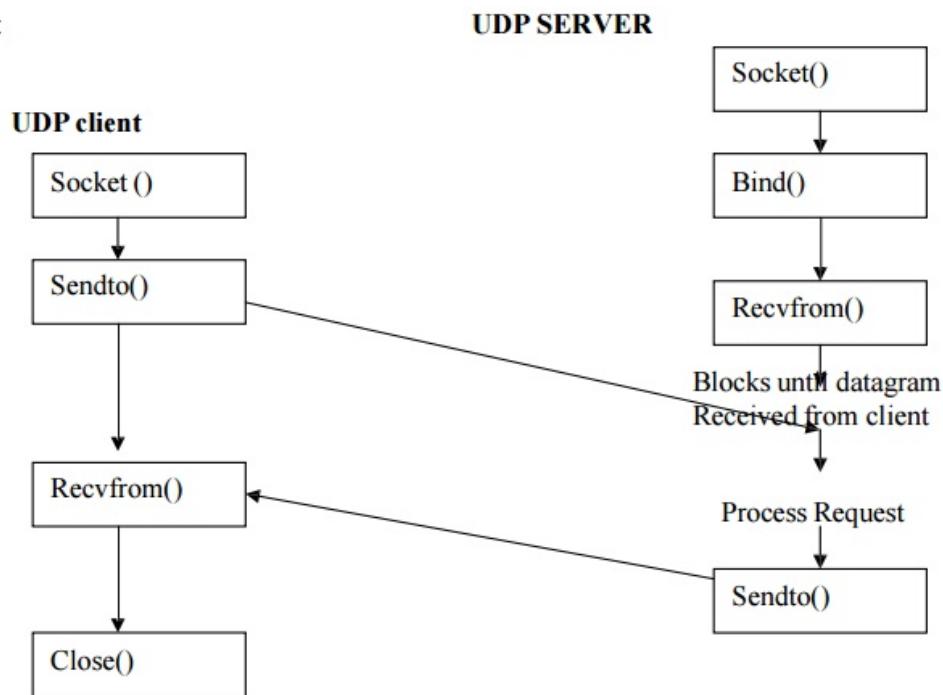


Figure 35 UDP datagram

This task is about creating an engaging multiplayer trivia game that uses UDP communication between a server and multiple clients. The goal is to implement a real-time game where players compete to answer trivia questions, with the server managing the game and scoring.

UDP (User Datagram Protocol) is fast and efficient, which makes it perfect for a trivia game where quick responses are more important than guaranteed delivery of every packet.

Unlike TCP, UDP doesn't waste time establishing connections or resending lost messages. This keeps the game flowing smoothly even if a few responses are missed.

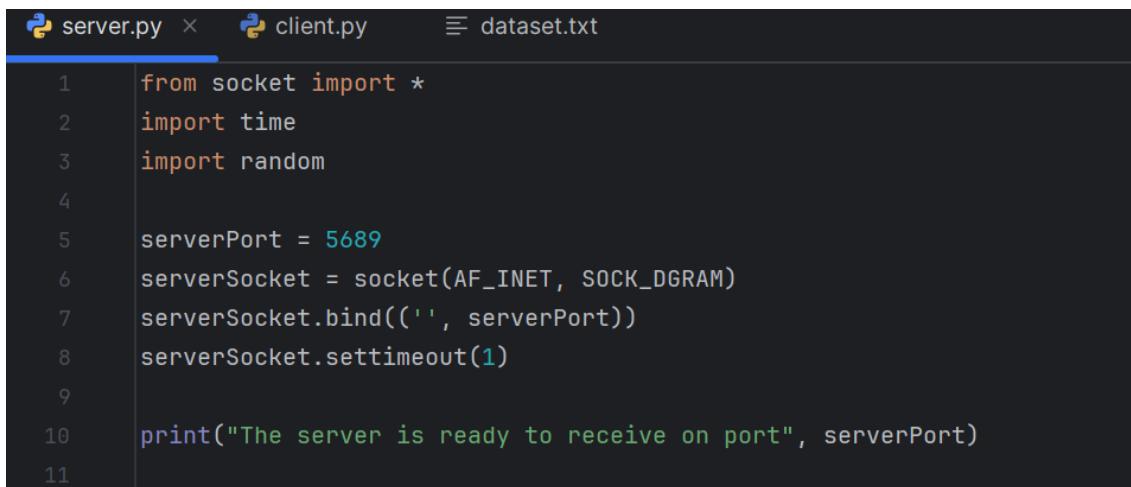
## How the game Works:

- The server is like the host of a quiz show. It sends out questions, collects answers, keeps track of scores, and announces winners.
- The clients are the players. They connect to the server, receive questions, and send their answers back, hoping to outsmart and outrun their opponents.

## Procedure:

### The server is the brain of the game:

#### 1. Setting Up the Server



A screenshot of a code editor window titled "server.py". The code is a Python script for a UDP server. It imports socket, time, and random modules. It defines a server port of 5689 and creates a socket object. The socket is bound to the root IP address and the specified port. A timeout of 1 second is set for receiving data. Finally, a message is printed to the console indicating the server is ready to receive on port 5689.

```
1 from socket import *
2 import time
3 import random
4
5 serverPort = 5689
6 serverSocket = socket(AF_INET, SOCK_DGRAM)
7 serverSocket.bind(('', serverPort))
8 serverSocket.settimeout(1)
9
10 print("The server is ready to receive on port", serverPort)
11
```

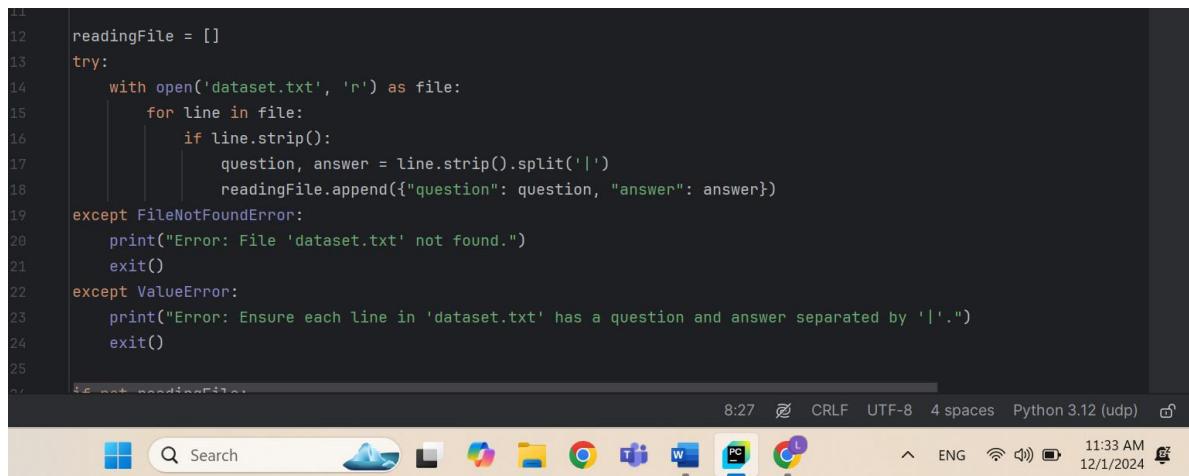
Figure 36 Setting Up the Server code

This is in the server code sets up a UDP server on port 5689 to listen for incoming datagrams. It sets a 1-second timeout for receiving data.

Table 1 methods used in setting server

Method	purpose
socket(AF_INET, SOCK_DGRAM)	Creates a UDP socket for communication between the server and clients. UDP is used for faster data transfer with minimal overhead.
bind((), serverPort))	Binds the socket to port 5689 to allow the server to listen for incoming messages from clients.
settimeout(1)	Ensures the server doesn't wait indefinitely for client messages, allowing it to handle other tasks if no data is received within 1 second.

## 2. Loading the Question Dataset



The screenshot shows a Windows desktop environment. In the foreground, a code editor window displays Python code for reading a dataset.txt file. The code uses a try-except block to handle FileNotFoundError and ValueError exceptions. It reads lines from the file, strips whitespace, splits each line by '|', and appends the resulting question and answer pairs to a list named readingFile. The code editor interface includes syntax highlighting for Python keywords and punctuation. In the background, the Windows taskbar is visible, showing various pinned icons for apps like File Explorer, Google Chrome, and Microsoft Word. The system tray at the bottom right shows the date and time (12/1/2024, 11:33 AM), battery status, and connectivity indicators.

```

11
12     readingFile = []
13
14     try:
15         with open('dataset.txt', 'r') as file:
16             for line in file:
17                 if line.strip():
18                     question, answer = line.strip().split('|')
19                     readingFile.append({"question": question, "answer": answer})
20     except FileNotFoundError:
21         print("Error: File 'dataset.txt' not found.")
22         exit()
23     except ValueError:
24         print("Error: Ensure each line in 'dataset.txt' has a question and answer separated by '|'.")
25         exit()

```

Figure 37 loading the Question Dataset code

```
server.py client.py dataset.txt
1 What is the capital of Germany?|Berlin
2 What is 15 - 6?|9
3 What is the smallest prime number?|2
4 What is the capital of palestine?|jerusalem
5 What is the chemical symbol for water?|H2O
6 How many hours are there in a day?|24
7 What is 7 x 6?|42
8 What is the atomic number of hydrogen?|1
9 What is the square of 9?|81
10 What is the capital of Spain?|Madrid
11 What is the primary color of the sun?|Yellow
12 What is 100 divided by 4?|25
13 How many legs does a spider have?|8
14 What is 11 x 11?|121
15 How many days are there in a leap year?|366
16
```

Figure 38 dataset file

Table 2 methods used in loading dataset

Method	Purpose
open('dataset.txt', 'r')	Opens the dataset file containing questions and answers for the game.
strip()	Removes unnecessary spaces or newlines to ensure clean data processing.
append({"question": question, "answer": answer})	Stores each question and answer in a list for random selection during the game.
FileNotFoundException	Handles the case where the dataset file is missing, ensuring the server exits gracefully.
ValueError	Ensures that the dataset is properly formatted (each line should have exactly one question and one answer separated by pipe “   ”)

### 3. Handling Client Connections

The screenshot shows a code editor window with three tabs: 'server.py', 'client.py', and 'dataset.txt'. The 'server.py' tab is active, displaying Python code for handling client connections. The code uses `recvfrom(2048)` to receive messages from clients, checks if they start with 'username:', and adds them to a list of connected clients. It also handles disconnections. The taskbar at the bottom shows various icons for system and application windows.

```

52     print("Game reset. Waiting for new players...")
53 def handle_client():
54     active = True
55     while active:
56         try:
57             message, clientAddress = serverSocket.recvfrom(2048)
58             decoded_message = message.decode().strip()
59             if decoded_message.startswith("username:"):
60                 answer = decoded_message.split(sep=":", maxsplit=1)[1].strip().lower()
61                 if clientAddress in clients:
62                     if answer:
63                         print(f"Client {clients[clientAddress]} answered: {answer}")
64                     else:
65                         print(f"Client {clients[clientAddress]} provided an empty answer.")
66                 elif decoded_message == "disconnect":
67                     handle_client_disconnection(clientAddress)
68                     active = False
69             except Exception:
70                 pass
71
72 def handle_client_disconnection(clientAddress):
73     if clientAddress in clients:
74         print(f"Client {clients[clientAddress]} ({clientAddress}) has disconnected.")
75         del clients[clientAddress]
76         del scores[clientAddress]

```

Figure 39 handling Client Connections code

Table 3 methods used in handling Client Connections code

Method	Purpose
recvfrom(2048)	Waits for client messages, such as username submissions or answers, and extracts the client's address.
startswith("username:")	Checks if a message contains a username, indicating that a new client is trying to join the game.
clients[clientAddress] = username	Adds the new player to the list of connected clients.
sendto(data.encode(), client)	Sends confirmation messages to clients, such as "You have joined the game!" or instructions for the game.

<code>len(clients)</code>	Counts the number of connected players to ensure that at least 2 players are present before starting the game.
<code>del clients[clientAddress]</code>	Removes disconnected players from the clients and scores dictionaries.
<code>print()</code>	Logs client connection events and disconnections for server monitoring.

#### 4. Starting the Game

The screenshot shows a code editor with two tabs open: 'server.py' and 'client.py'. The 'server.py' tab is active and displays a Python script for a game server. The code defines a function 'start\_game()' that handles the game logic, including sending start messages to clients, selecting questions from a file, and broadcasting them to clients. It also manages client connections and tracks time for rounds and questions. The code uses standard Python libraries like socket and random. The code editor interface includes a status bar at the bottom showing file paths, line numbers, and system information.

```
79 def start_game(): 1usage
80     global scores, game_start_timer , accepting_new_players
81     print("Starting the game...")
82     for client in clients:
83         serverSocket.sendto("The game is starting now!".encode(), client)
84         serverSocket.sendto("This game contains 3 rounds and each round with 3 questions , be ready!!!!".encode(), client)
85     game_start_timer = False
86     for round_num in range(1, 4): #3rounds
87         selected_questions = random.sample(readingFile, k: 3)
88         print(f"Round {round_num} starting...")
89         active_clients = set()
90         for client in clients:
91             active_clients.add(client)
92         for q_index, q in enumerate(selected_questions):
93             question = q["question"]
94             correct_answer = q["answer"].lower()

95             for client in clients:
96                 serverSocket.sendto(f"Round {round_num} Question: {q_index + 1}: {question}".encode(), client)
97             print(f"Broadcasting Question {q_index + 1}: {question}")
98             answers = {}
99             question_start_time = time.time()
100
101             while time.time() - question_start_time < 30: # Wait 30 seconds|
102                 try:
103                     for client in active_clients:
```

*Figure 40 starting game code 1*

*Table 4 methods used in starting game1*

Method	Purpose
<code>time.time()</code>	Tracks the countdown to start the game (30 second timer) and question-answer periods.

time.sleep(30)	Pauses for a set period (30 seconds) to give players time to prepare for the next round.
random.sample(readingFile, 3)	Randomly selects 3 questions for each round to keep the game engaging and unpredictable.

## 5. Processing Player Answers

```

79     def start_game():  usage
80
81         while time.time() - question_start_time < 30:  # Wait 30 seconds
82             try:
83                 answer, clientAddress = serverSocket.recvfrom(2048)
84                 if clientAddress in clients and clientAddress not in answers:
85                     decoded_answer = answer.decode().strip()
86                     if decoded_answer.startswith("answer:"):
87                         answers[clientAddress] = decoded_answer[7:].strip().lower()
88                         active_clients.add(clientAddress)
89                         print(f"Received answer from {clientAddress}: {answers[clientAddress]}")
90
91                 except Exception:
92                     continue
93
94             correct_clients = []
95             for client, answer in answers.items():
96                 if answer == correct_answer:
97                     correct_clients.append(client)
98
99             for idx, client in enumerate(correct_clients):
100                 points = 1 - (idx / len(correct_clients))
101                 scores[client] += points
102                 serverSocket.sendto(f"Correct! You earned {points:.2f} points.".encode(), client)
103             for client in answers:
104                 if client not in correct_clients:
105                     serverSocket.sendto("Wrong!".encode(), client)
106
107             correct_answer_message = f"The correct answer was: {correct_answer.capitalize()}"
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125

```

Figure 41 starting game code 2

Table 5 methods used in starting game2

Method	Purpose
recvfrom(2048)	Collects answers from players during the question-answer period.
startswith("answer:")	Ensures the client message contains an answer for evaluation.
lower()	Converts both the player's answer and the correct answer to lowercase for case-insensitive comparison.
if answer == correct_answer	Checks if a player's answer matches the correct one.

<code>scores[client] += points</code>	Updates the player's score based on how quickly they answered correctly.
---------------------------------------	--

## 6. Sending Feedback and Round Results and Managing Game State

A screenshot of a code editor displaying a Python script named `server.py`. The code implements a game server that handles multiple clients, manages scores, and handles rounds. It includes logic for sending messages to clients, determining a leader, and managing active players. The code uses standard Python libraries like `socket` and `time`.

```
79     def start_game(): 1 usage
80
81         round_results = "Round Results:\n" + "\n".join([f"{clients[client]}: {scores[client]:.2f} points" for client in active_clients])
82         for client in clients:
83             serverSocket.sendto(round_results.encode(), client)
84
85         if scores:
86             leader = max(scores, key=scores.get) # Find the client with the highest score
87             leader_username = clients[leader]
88             leader_message = f"The current leader is {leader_username} with {scores[leader]:.2f} points."
89             for client in clients:
90                 serverSocket.sendto(leader_message.encode(), client)
91             print(leader_message)
92
93         if round_num < 3: # Only add delay if it's not the final round
94             print("Waiting 30 seconds before starting the next round...")
95             for client in active_clients:
96                 serverSocket.sendto("Prepare for the next round. Starting in 30 seconds!".encode(), client)
97             time.sleep(30)
98
99         if len(active_clients) < 2:
100             print("Not enough active players to continue.")
101             for client in clients:
102                 serverSocket.sendto("Not enough active players. The game is ending.".encode(), client)
103             break
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
```

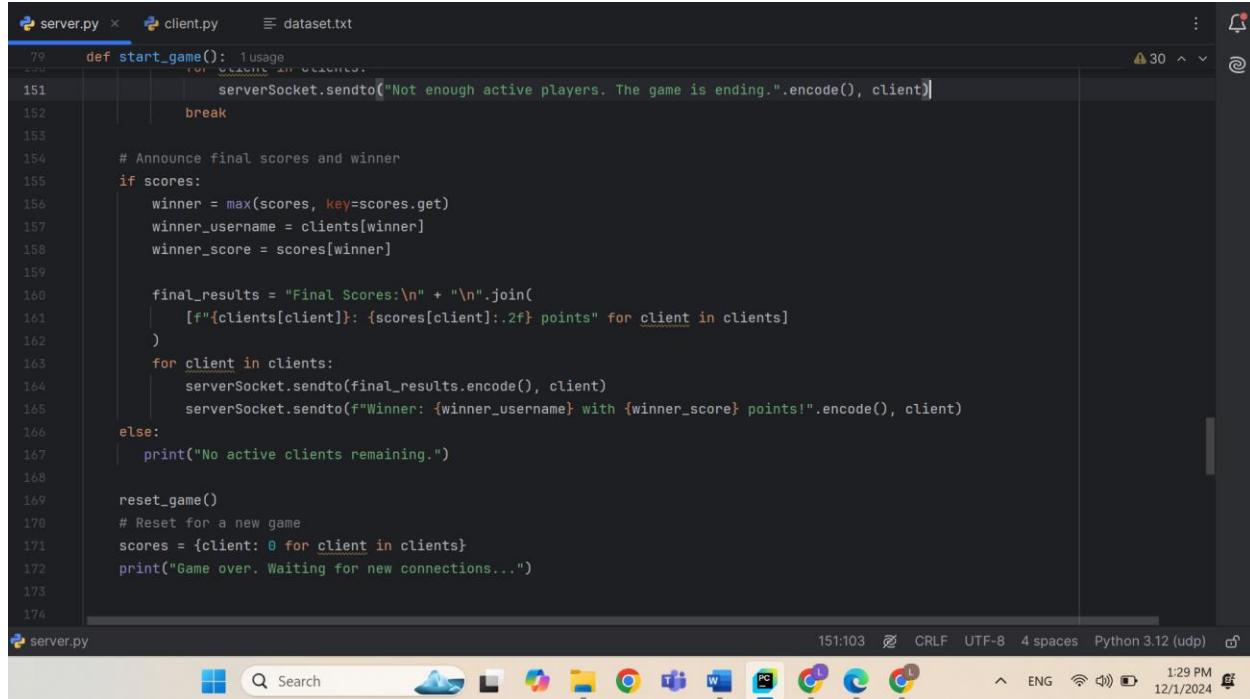
*Figure 42 starting game code 3*

*Table 6 methods used in starting game 3*

Method	Purpose
sendto(data.encode(), client)	Sends feedback to players, such as "Correct!", "Wrong!", and the correct answer for each question.
join()	Formats the list of player scores into a readable message for broadcasting results.
max(scores, key=scores.get)	Determines the player with the highest score to announce as the leader or winner.
reset_game()	Resets all game variables (e.g., scores, clients) for a new game.

global	Allows functions like <code>reset_game</code> to modify global variables (scores, clients, etc.).
<code>if len(active_clients) &lt; 2</code>	Checks if there are enough players to continue the game. Ends the game early if fewer than 2 players are active.

## 7. Error Handling



```

  server.py x  client.py  dataset.txt
  79  def start_game():
100      usage = "usage: python3 server.py <dataset> <clients>\n"
101      if len(sys.argv) != 3:
102          print(usage)
103          sys.exit()
104
105      # Announce final scores and winner
106      if scores:
107          winner = max(scores, key=scores.get)
108          winner_username = clients[winner]
109          winner_score = scores[winner]
110
111          final_results = "Final Scores:\n" + "\n".join(
112              [f"{clients[client]}: {scores[client]:.2f} points" for client in clients]
113          )
114
115          for client in clients:
116              serverSocket.sendto(final_results.encode(), client)
117              serverSocket.sendto(f"Winner: {winner_username} with {winner_score} points!".encode(), client)
118
119      else:
120          print("No active clients remaining.")
121
122      reset_game()
123      # Reset for a new game
124      scores = {client: 0 for client in clients}
125      print("Game over. Waiting for new connections...")
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174

```

Figure 43 starting game code 4

Table 7 methods used in starting game 3

Method	Purpose
<code>try and except</code>	Prevents server crashes when errors occur (e.g., file not found, invalid client input, timeout while waiting for data).
<code>exit()</code>	Stops the server gracefully if critical errors occur (e.g., missing or invalid dataset).

## 8. Server Waiting for Players

A screenshot of a code editor displaying a Python script named `server.py`. The code implements a simple game server that waits for players to join, then starts a game after 20 seconds. It handles new player connections and sends a welcome message to each client. The code uses `socket` and `time` modules. The editor interface shows tabs for `server.py`, `client.py`, and `dataset.txt`. A status bar at the bottom indicates the file is saved, the current time (2024-01-12 14:46 PM), and file format details (CRLF, UTF-8, 4 spaces, Python 3.12 (udp)).

```
179 while True:
180     try:
181
182         print("Waiting for players to join...")
183         time.sleep(1) # Sleep for 1 second to prevent the loop from running too fast
184         while game_start_timer and time.time() - timer_start_time >= 20:
185             if accepting_new_players:
186                 accepting_new_players = False # Stop accepting new players after 20 seconds
187                 print("20 seconds have passed, starting the game now!")
188                 start_game()
189                 game_start_timer = False
190
191             message, clientAddress = serverSocket.recvfrom(2048)
192             decoded_message = message.decode().strip()
193
194             if clientAddress not in clients:
195                 if decoded_message.startswith("username:"):
196                     if accepting_new_players:
197                         username = decoded_message.split(sep=":", maxsplit=1)[1]
198                         clients[clientAddress] = username
199                         scores[clientAddress] = 0
200                         print(f"Client {username} ({clientAddress}) connected.")
201                         serverSocket.sendto("You have joined the game!".encode(), clientAddress)
202                     for client in clients:
```

*Figure 44 main server code*

*Table 8 methods used in main server code*

Component/Method	Purpose
Main Loop (while True)	Keeps the server running, continuously waiting for players to join.
print("Waiting for players...")	Logs server activity for monitoring.
recvfrom(2048)	Waits to receive client messages and their addresses.
startswith("username:")	Checks if a client message contains a username (indicating a new player joining).
clients[clientAddress] = username	Adds the player's username and address to the active players list.
sendto("...".encode(), client)	Sends confirmation messages to players, such as "You have joined!" or "Game in progress, please wait."

## 9. Starting Game Countdown and Handling Errors and Late Joiners

A screenshot of a Windows desktop environment. The main window is a code editor displaying Python code for a server application. The code handles client connections, manages player counts, and initiates a 90-second countdown for game start. The code editor has tabs for 'server.py' and 'client.py'. The status bar at the bottom shows the file path 'C:\Users\user\PycharmProjects\game\server.py', the current time '22:13', and file statistics like 'CRLF', 'UTF-8', '4 spaces', and 'Python 3.12 (udp)'. The taskbar at the bottom includes icons for File Explorer, Task View, and several open browser windows.

```
server.py x dataset.txt
197
198     username = message.split()[0]
199     clients[clientAddress] = username
200     scores[clientAddress] = 0
201     print(f"Client {username} ({clientAddress}) connected.")
202     serverSocket.sendto("You have joined the game!".encode(), clientAddress)
203
204     for client in clients:
205
206         serverSocket.sendto(f"{username} has joined the game\nNumber of players: {len(clients)}".encode(), client)
207
208     if game_start_timer:
209         serverSocket.sendto("The game will start soon. You are in the game!".encode(), clientAddress)
210     else:
211         print(f"Rejected connection from {clientAddress}. Game in progress.")
212         serverSocket.sendto(
213             "The game is in progress. You cannot join now. Please wait for the next game.".encode(),
214             clientAddress)
215
216     # Start the 90-second countdown if there are at least 2 clients
217     if len(clients) >= 2 and not game_start_timer:
218         game_start_timer = True
219         timer_start_time = time.time()
220         print("90-second countdown started. Waiting for more players...")
221
222         for client in clients:
223             serverSocket.sendto("90 seconds until the game starts! Invite more players.".encode(), client)
224
225     except Exception as e:
226         pass
```

*Figure 45 main server code 2*

*Table 9 methods used in main server code*

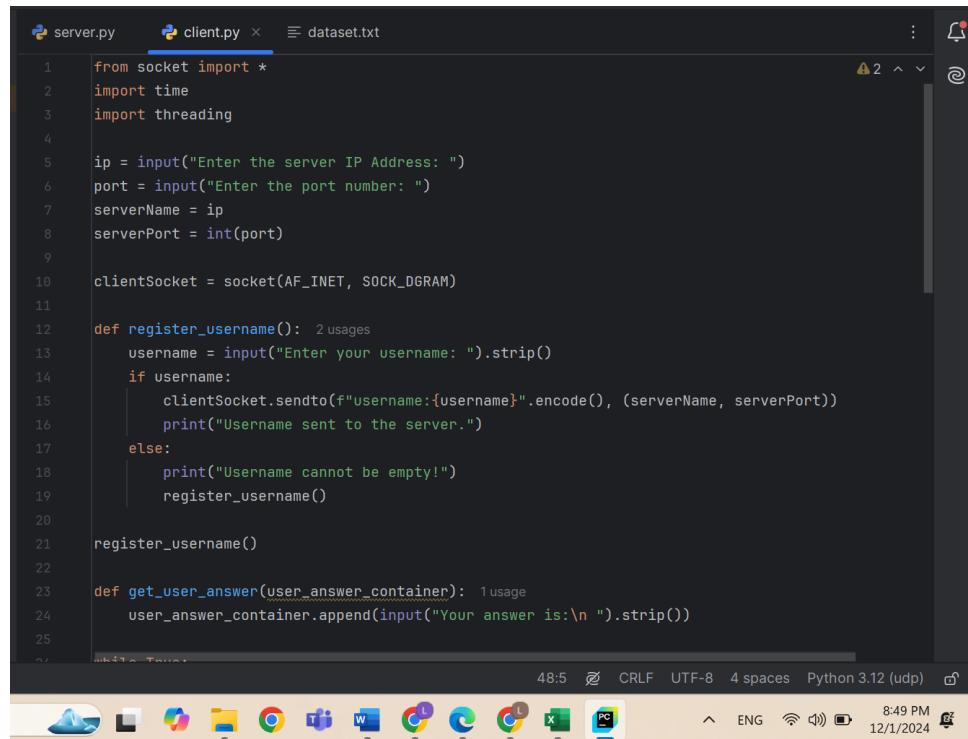
Component/Method	Purpose
if len(clients) >= 2:	Begins the game countdown only when there are at least 2 players.
game_start_timer = True	Activates the countdown timer for game initiation.
timer_start_time = time.time()	Tracks when the countdown began to limit player entry after 20 seconds.
sendto("90 seconds until the game starts...".encode(), client)	Notifies all players about the countdown and encourages them to invite more players.
accepting_new_players = False	Stops accepting new players when the countdown ends, locking the player list.
start_game()	Initiates the actual gameplay process after the countdown ends.
if clientAddress not in clients:	Ensures new players can only join if the game hasn't started.

sendto("Game in progress...".encode(), clientAddress)	Notifies late joiners that they cannot participate and must wait for the next game.
except Exception as e:	Catches and suppresses errors (e.g., invalid messages, timeout) to ensure the server remains stable.
pass	Allows the loop to continue uninterrupted even if an error occurs.

### Player is the clients who joining the game and connect to the server.

The client code allows a user to connect to a game server via UDP, register their username, and participate in a timed question-answer game. It listens for server messages, submits answers, and handles disconnection when the game ends or if there are not enough active players. The client operates in a loop, ensuring smooth communication with the server throughout the game.

#### 1- Setup and Username Registration



```

server.py client.py x dataset.txt
1 from socket import *
2 import time
3 import threading
4
5 ip = input("Enter the server IP Address: ")
6 port = input("Enter the port number: ")
7 serverName = ip
8 serverPort = int(port)
9
10 clientSocket = socket(AF_INET, SOCK_DGRAM)
11
12 def register_username(): 2 usages
13     username = input("Enter your username: ").strip()
14     if username:
15         clientSocket.sendto(f"username:{username}".encode(), (serverName, serverPort))
16         print("Username sent to the server.")
17     else:
18         print("Username cannot be empty!")
19         register_username()
20
21 register_username()
22
23 def get_user_answer(user_answer_container): 1 usage
24     user_answer_container.append(input("Your answer is:\n").strip())
25

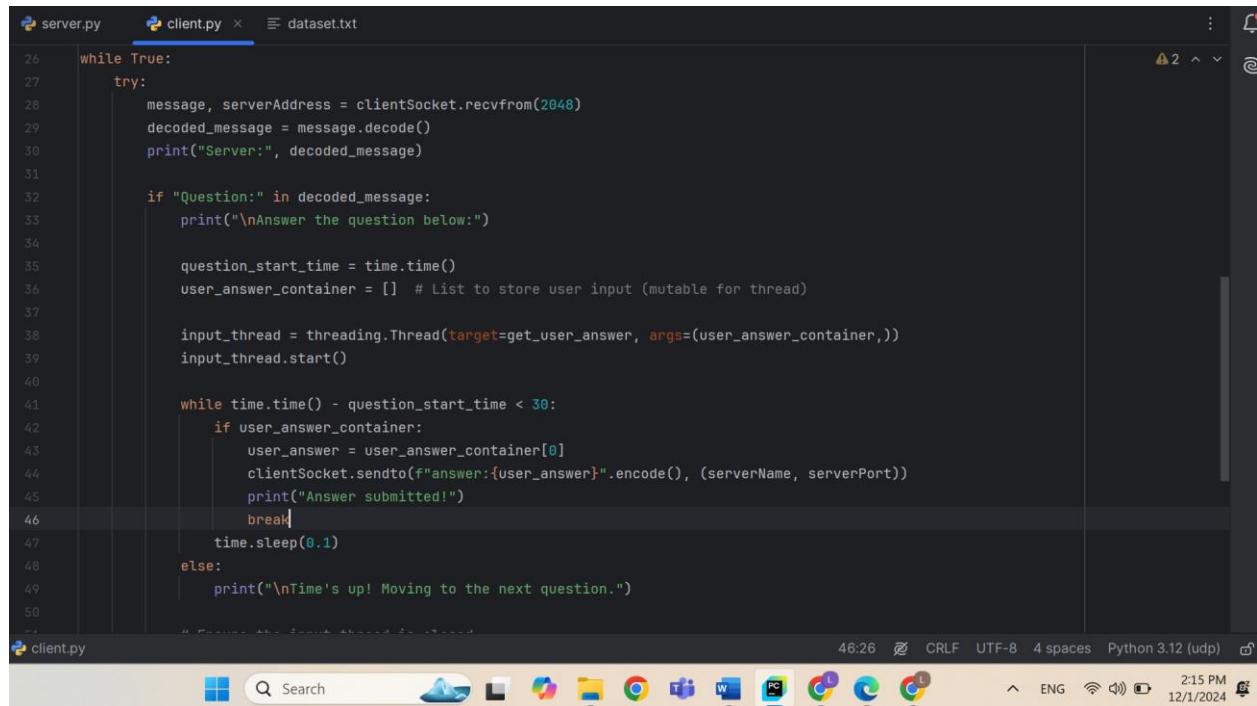
```

Figure 46 client code 1

Table 10 methods used in client code

Component/Method	Purpose
socket(AF_INET, SOCK_DGRAM)	Creates a UDP client socket to communicate with the server.
input("Enter the server IP Address...")	takes the server's IP and port number as input to connect the client to the game
register_username()	Prompts the user to input a username and sends it to the server in the format username:<name>. Ensures the username is not empty.
sendto(f"username:{username}".encode(), (serverName, serverPort))	Sends the username to the server, enabling the user to join the game.

## 2- Listening to Server Messages and Answer Submission



The screenshot shows a Windows desktop environment. In the center is a code editor window displaying the contents of the `client.py` file. The code implements a UDP client that listens for server messages and submits answers. The code uses threads and sockets to handle the interaction. The taskbar at the bottom shows various open applications, including a browser, file explorer, and system icons. The system tray indicates the date and time as 12/1/2024 at 2:15 PM.

```

26 while True:
27     try:
28         message, serverAddress = clientSocket.recvfrom(2048)
29         decoded_message = message.decode()
30         print("Server:", decoded_message)
31
32         if "Question:" in decoded_message:
33             print("\nAnswer the question below:")
34
35             question_start_time = time.time()
36             user_answer_container = [] # List to store user input (mutable for thread)
37
38             input_thread = threading.Thread(target=get_user_answer, args=(user_answer_container,))
39             input_thread.start()
40
41             while time.time() - question_start_time < 30:
42                 if user_answer_container:
43                     user_answer = user_answer_container[0]
44                     clientSocket.sendto(f"answer:{user_answer}".encode(), (serverName, serverPort))
45                     print("Answer submitted!")
46                     break
47                 time.sleep(0.1)
48             else:
49                 print("\nTime's up! Moving to the next question.")
50

```

Figure 47 client code 2

Table 11 methods used in client code 2

Method	Purpose
recvfrom(2048)	Listens for messages from the server (e.g., questions, instructions, or results)
decode()	Decodes the received bytes into a readable string for display to the user.
if "Question:" in decoded_message:	Checks if the server has sent a new question, triggering the question-answer phase.
print("Server:", decoded_message)	Displays messages from the server, such as game updates or questions, to the user.
threading.Thread(target=get_user_answer...)	Starts a thread to allow the user to input their answer while a timer runs concurrently.
time.time() - question_start_time < 30	Sets a 30-second timer for answering the question. If the user doesn't respond in time, it moves to the next question.
clientSocket.sendto(f"answer:{user_answer}".encode(), (serverName, serverPort))	Sends the user's answer to the server.
input_thread.join(timeout=0)	Ensures the input thread terminates after the answer is submitted or the timer expires.

### 3- Error Handling

The screenshot shows a Windows desktop environment. In the center is a code editor window titled "client.py" containing Python code. The code handles socket communication between a client and a server, including sending and receiving messages, handling player counts, and managing connections. Below the code editor is a taskbar with various icons for common applications like File Explorer, Edge, and Google Chrome. The system tray shows the date and time as 12/1/2024 at 2:16 PM.

```
44     clientSocket.sendto(answer.encode(), (serverName, serverPort))
45     print("Answer submitted!")
46     break
47     time.sleep(0.1)
48 else:
49     print("\nTime's up! Moving to the next question.")
50
51 # Ensure the input thread is closed
52 input_thread.join(timeout=0)
53
54 elif "Not enough active players" in decoded_message:
55     print("Not enough active players. The game is ending.")
56     break
57
58 except Exception as e:
59     pass
60
61 clientSocket.sendto("disconnect".encode(), (serverName, serverPort))
62 print("Disconnected from the server.")
63 clientSocket.close()
64 |
```

Figure 48 client code 3

Table 12 methods used in client code3

Method	Purpose
if "Not enough active players" in decoded_message:	Terminates the game if the server ends it due to insufficient players.
sendto("disconnect".encode(), (serverName, serverPort))	Notifies the server when the client disconnects.
except Exception as e:	Handles errors, such as network issues or invalid server responses, ensuring the client does not crash unexpectedly.
clientSocket.close()	Closes the socket connection after the client disconnects, freeing up resources.

## Results and Discussions

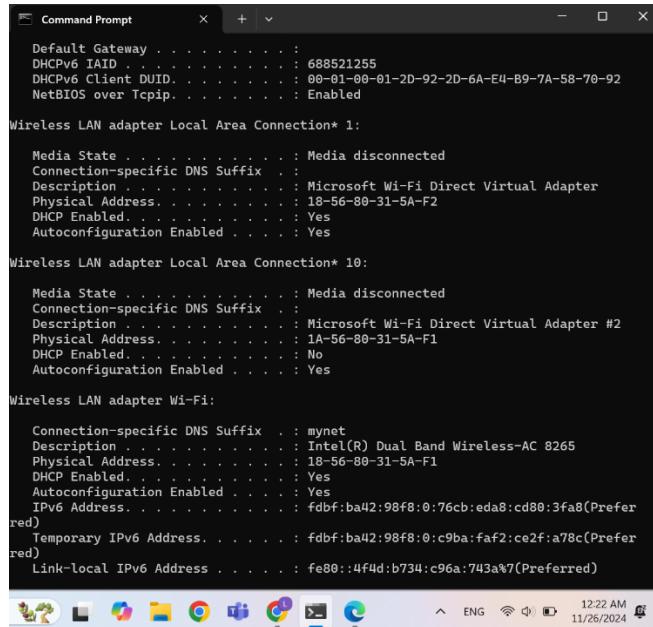
### Task one:

#### Explanation

- ipconfig - this is used to show the network configuration of a computer, including its IP address, subnet mask, and default gateway that are crucial for communication over networks.
- ping: Sends small packets of data to a specific IP address to determine if a device is reachable over the network, while it measures how long it takes for packets to make the round trip.
- tracert: Packet sent to a destination IP, route followed traced showing the IP of every router packets pass through and time taken to reach each hop.
- Nslookup: Queries a DNS server to translate a domain name into its corresponding IP address or to resolve an IP address back to its domain name.
- Telnet: A protocol that enables a user to establish a command-line interface session with a remote server or device, interactively getting connected over a network.

## Trying the commands

1. Run the ipconfig /all



```
Command Prompt
Default Gateway . . . . . : 688521255
DHCPv6 IAID . . . . . : 688521255
DHCPv6 Client DUID . . . . . : 00-01-00-01-2D-92-2D-6A-E4-B9-7A-58-70-92
NetBIOS over Tcpip . . . . . : Enabled

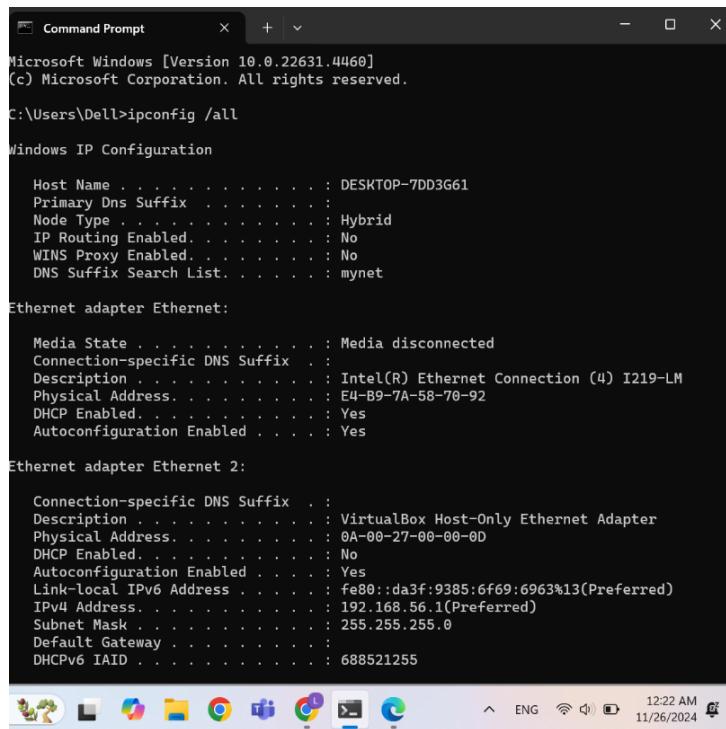
Wireless LAN adapter Local Area Connection* 1:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Description . . . . . : Microsoft Wi-Fi Direct Virtual Adapter
Physical Address. . . . . : 18-56-80-31-5A-F2
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . . : Yes

Wireless LAN adapter Local Area Connection* 10:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Description . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #2
Physical Address. . . . . : 1A-56-80-31-5A-F1
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . . : Yes

Wireless LAN adapter Wi-Fi:
Connection-specific DNS Suffix . . mynet
Description . . . . . : Intel(R) Dual Band Wireless-AC 8265
Physical Address. . . . . : 18-56-80-31-5A-F1
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . . : Yes
IPv6 Address . . . . . : fdbf:ba42:98f8:0:76cb:eda8:cd80:3fa8(Prefered)
Temporary IPv6 Address. . . . . : fdbf:ba42:98f8:0:c9ba:faf2:ce2f:a78c(Preferred)
Link-local IPv6 Address . . . . . : fe80::4f4d:b734:c96a:743a%7(Preferred)

12:22 AM 11/26/2024
```

Figure 49 running ipconfig



```
Command Prompt
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>ipconfig /all

Windows IP Configuration

Host Name . . . . . : DESKTOP-7DD3G61
Primary Dns Suffix . . . . . :
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No
DNS Suffix Search List. . . . . : mynet

Ethernet adapter Ethernet:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Description . . . . . : Intel(R) Ethernet Connection (4) I219-LM
Physical Address. . . . . : E4-B9-7A-58-70-92
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . . : Yes

Ethernet adapter Ethernet 2:

Connection-specific DNS Suffix . . . . . : VirtualBox Host-Only Ethernet Adapter
Description . . . . . : Intel PRO/100 MT Desktop
Physical Address. . . . . : 0A-00-27-00-00-0D
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::da3f:9385:6f69:6963%13(Preferred)
IPv4 Address . . . . . : 192.168.56.1(Preferred)
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :
DHCPv6 IAID . . . . . : 688521255

12:22 AM 11/26/2024
```

Figure 50 running ipconfig 2



```
Temporary IPv6 Address . . . . . : fdbf:ba42:98f8:0:c9ba:faf2:ce2f:a78c(Preferred)
Link-local IPv6 Address . . . . . : fe80::4f4d:b734:c96a:743a%7(Preferred)
IPv4 Address . . . . . : 192.168.1.12(Preferred)
Subnet Mask . . . . . : 255.255.255.0
Lease Obtained . . . . . : Monday, November 25, 2024 11:57:00 PM
Lease Expires . . . . . : Tuesday, November 26, 2024 1:57:00 AM
Default Gateway . . . . . : 192.168.1.1
DHCP Server . . . . . : 192.168.1.1
DHCPv6 IAID . . . . . : 68703872
DHCPv6 Client DUID . . . . . : 00-01-00-01-2D-92-2D-6A-E4-B9-7A-58-70-92
DNS Servers . . . . . : 192.168.1.1
NetBIOS over Tcpip. . . . . : Enabled
Connection-specific DNS Suffix Search List :
    mynet
    mynet

Ethernet adapter Bluetooth Network Connection:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . . .
    Description . . . . . : Bluetooth Device (Personal Area Network)
    Physical Address. . . . . : 18-56-80-31-5A-F5
    DHCP Enabled. . . . . : Yes
    Autoconfiguration Enabled . . . . . : Yes

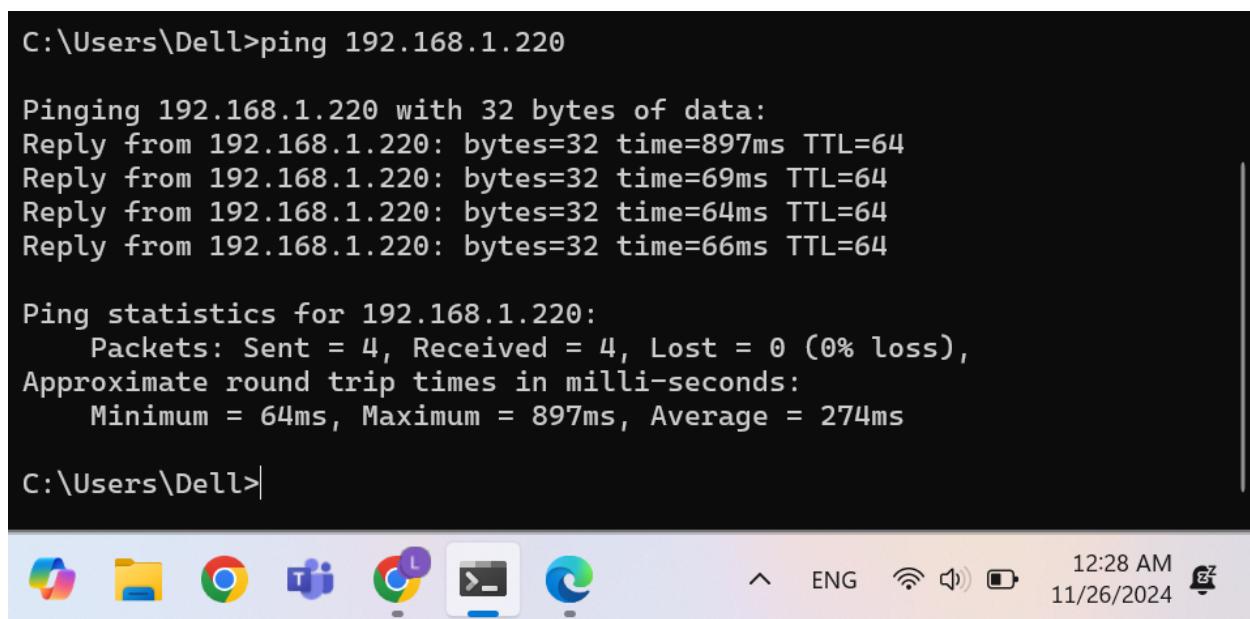
C:\Users\Dell>
```

Figure 51 running ipconfig 3

This output shows the network configuration of the computer, indicating details like:

- IP or (IPv4) Address: 192.168.1.12
- subnet mask: 255.255.255.0
- default gateway: 192.168.1.1(The address of the router, which connects the local network to the internet.)
- DNS server addresses: 192.168.1.1 (Indicates that the router is also serving as the DNS server for name resolution).

## 2. Ping



```
C:\Users\Dell>ping 192.168.1.220

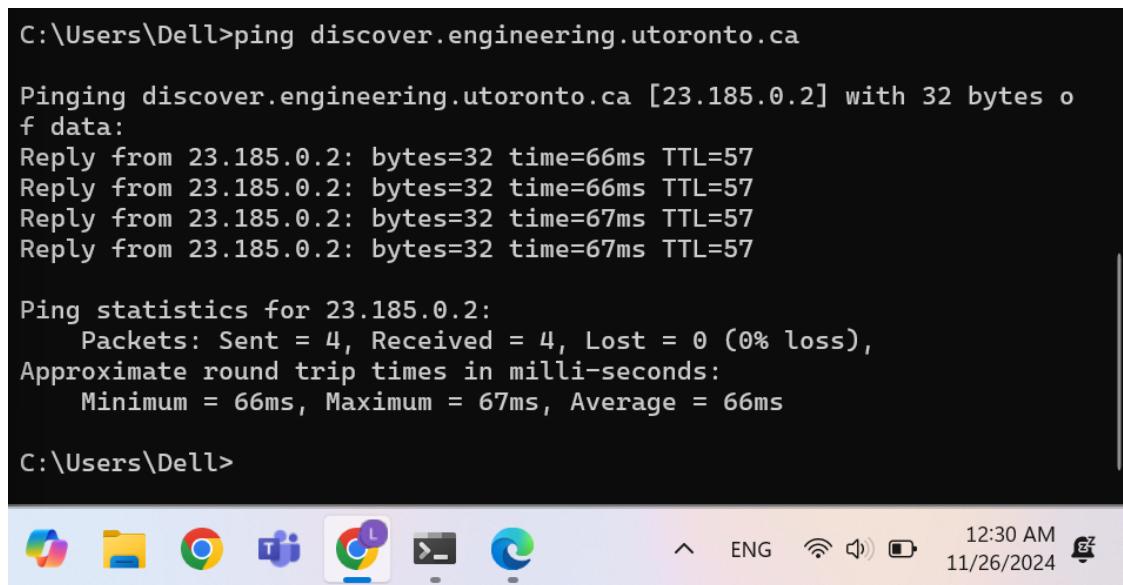
Pinging 192.168.1.220 with 32 bytes of data:
Reply from 192.168.1.220: bytes=32 time=897ms TTL=64
Reply from 192.168.1.220: bytes=32 time=69ms TTL=64
Reply from 192.168.1.220: bytes=32 time=64ms TTL=64
Reply from 192.168.1.220: bytes=32 time=66ms TTL=64

Ping statistics for 192.168.1.220:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 64ms, Maximum = 897ms, Average = 274ms

C:\Users\Dell>
```

Figure 52 testing ping1

To ping a device in the same network, IP address is needed, the device in Fig ----is my smart phone, the IP address:192.168.1.220 was taken from the settings and run the ping command to it. Ping showed path of the packets sent to that device and the times taken.



```
C:\Users\Dell>ping discover.engineering.utoronto.ca

Pinging discover.engineering.utoronto.ca [23.185.0.2] with 32 bytes of data:
Reply from 23.185.0.2: bytes=32 time=66ms TTL=57
Reply from 23.185.0.2: bytes=32 time=66ms TTL=57
Reply from 23.185.0.2: bytes=32 time=67ms TTL=57
Reply from 23.185.0.2: bytes=32 time=67ms TTL=57

Ping statistics for 23.185.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 66ms, Maximum = 67ms, Average = 66ms

C:\Users\Dell>
```

Figure 53 testing ping 2

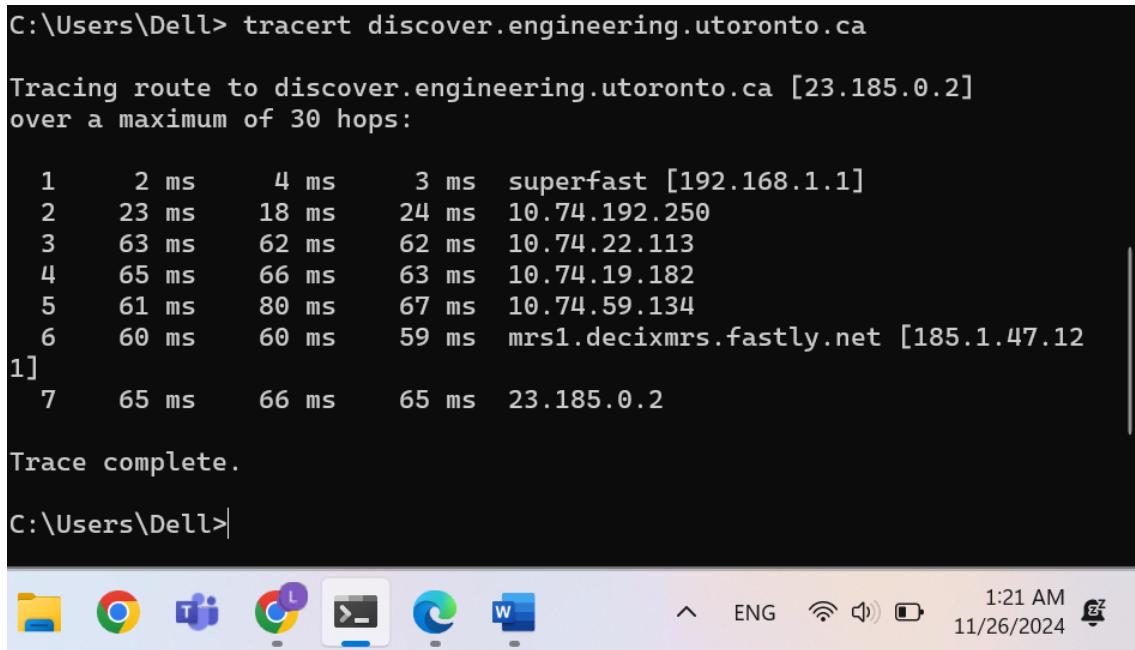
The ping results show a response time of around 65 ms, which suggests the server is not very far. However, the IP address belongs to Fastly, a Content Delivery Network (CDN).

# 23.185.0.2

 San Francisco, California, United States

This means the response might be coming from a server in USA used to cache content, rather than the original server in Canada.

### 3. Tracert



```
C:\Users\Dell> tracert discover.engineering.utoronto.ca

Tracing route to discover.engineering.utoronto.ca [23.185.0.2]
over a maximum of 30 hops:

 1      2 ms      4 ms      3 ms  superfast [192.168.1.1]
 2     23 ms     18 ms     24 ms  10.74.192.250
 3     63 ms     62 ms     62 ms  10.74.22.113
 4     65 ms     66 ms     63 ms  10.74.19.182
 5     61 ms     80 ms     67 ms  10.74.59.134
 6     60 ms     60 ms     59 ms  mrs1.decixmrs.fastly.net [185.1.47.12
1]
 7     65 ms     66 ms     65 ms  23.185.0.2

Trace complete.

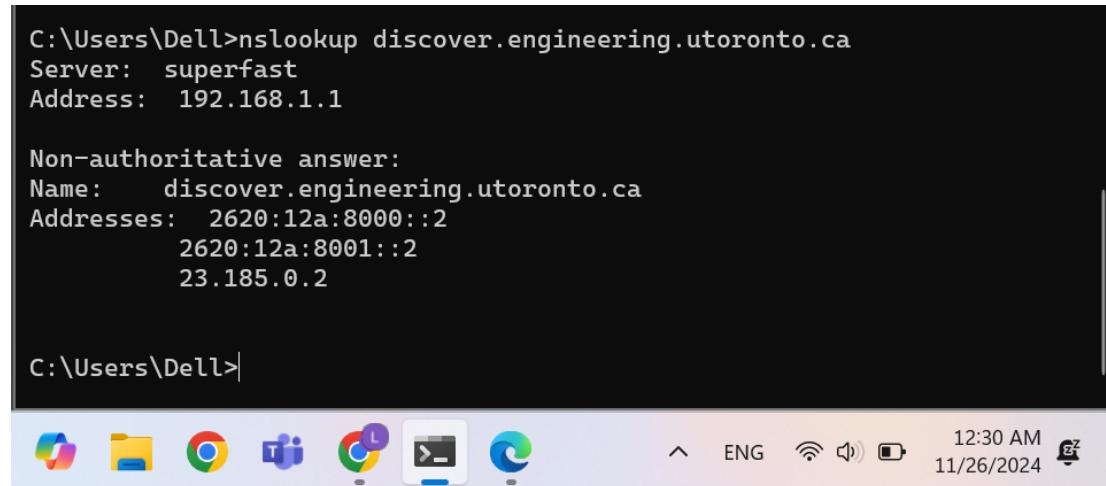
C:\Users\Dell>
```

The screenshot shows a Windows Command Prompt window with the title bar "Windows Terminal". The taskbar at the bottom includes icons for File Explorer, Google Chrome, Microsoft Teams, Google Photos, Task View, File History, File Explorer, Microsoft Edge, and Microsoft Word. The system tray shows the date and time as "11/26/2024 1:21 AM" and battery status.

Figure 54 testing tracert

The results of the tracert show the path taken by packets to reach the server `discover.engineering.utoronto.ca`, listing all intermediate network devices (hops) and their IP addresses. It can be noticed that the final IP is `10.74.59.134`, which is the IP for the website `23.185.0.2`, as it was shown in the previous Figure (when run ping).

#### 4. nslookup



```
C:\Users\Dell>nslookup discover.engineering.utoronto.ca
Server:  superfast
Address:  192.168.1.1

Non-authoritative answer:
Name:    discover.engineering.utoronto.ca
Addresses:  2620:12a:8000::2
           2620:12a:8001::2
           23.185.0.2

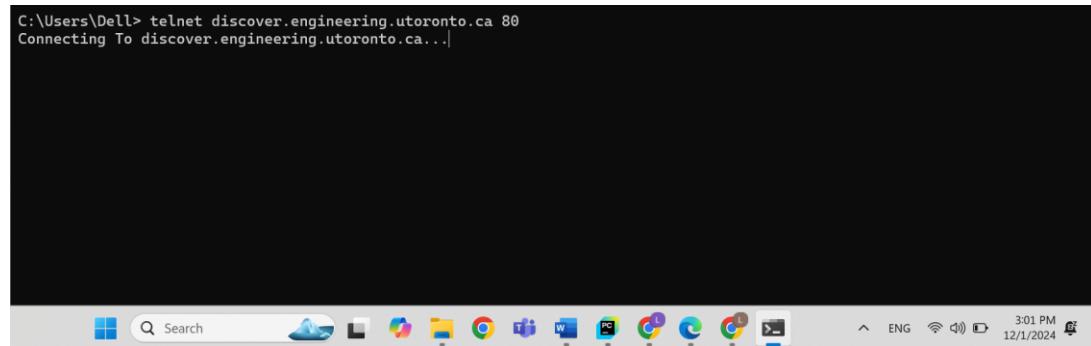
C:\Users\Dell>
```

The screenshot shows a Windows command prompt window. The command `nslookup discover.engineering.utoronto.ca` is entered, and the output shows the server used (superfast at 192.168.1.1) and the non-authoritative answers for the name, which include multiple IP addresses (IPv6 and IPv4). The taskbar at the bottom displays various application icons and the system tray shows the date and time as 12:30 AM on 11/26/2024.

Figure 55 testing nslookup

DNS lookup for `discover.engineering.utoronto.ca`, showing the resolved IP address and the DNS server used to perform the query.

#### 5. Telent



```
C:\Users\Dell> telnet discover.engineering.utoronto.ca 80
Connecting To discover.engineering.utoronto.ca...|
```

The screenshot shows a Windows command prompt window. The command `telnet discover.engineering.utoronto.ca 80` is entered, and the output shows the connection is being established to the port 80 of the host `discover.engineering.utoronto.ca`. The taskbar at the bottom displays various application icons and the system tray shows the date and time as 3:01 PM on 12/1/2024.

Figure 56 testing telnet

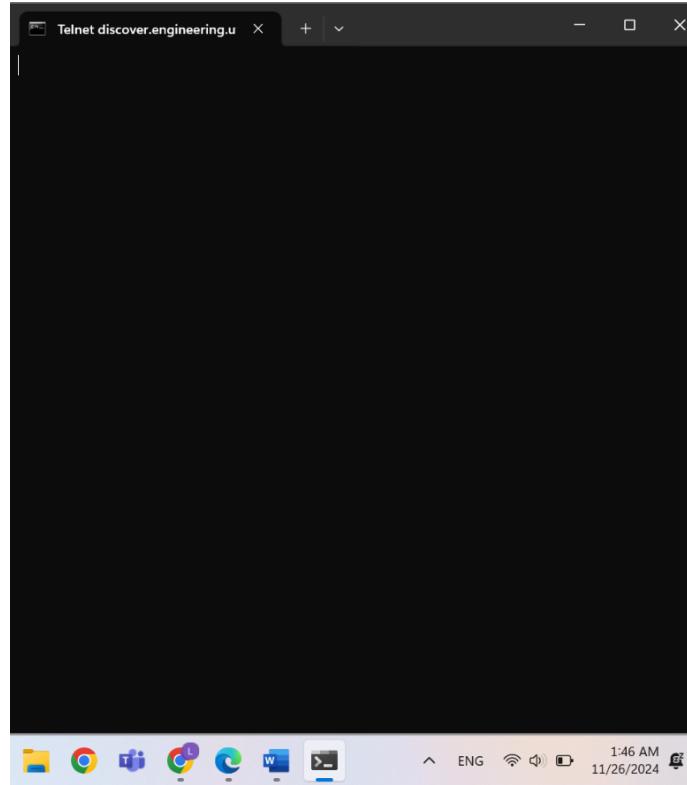


Figure 57 testing telnet 2

The connection for telnet is running, the server is waiting for a message.

## Wireshark

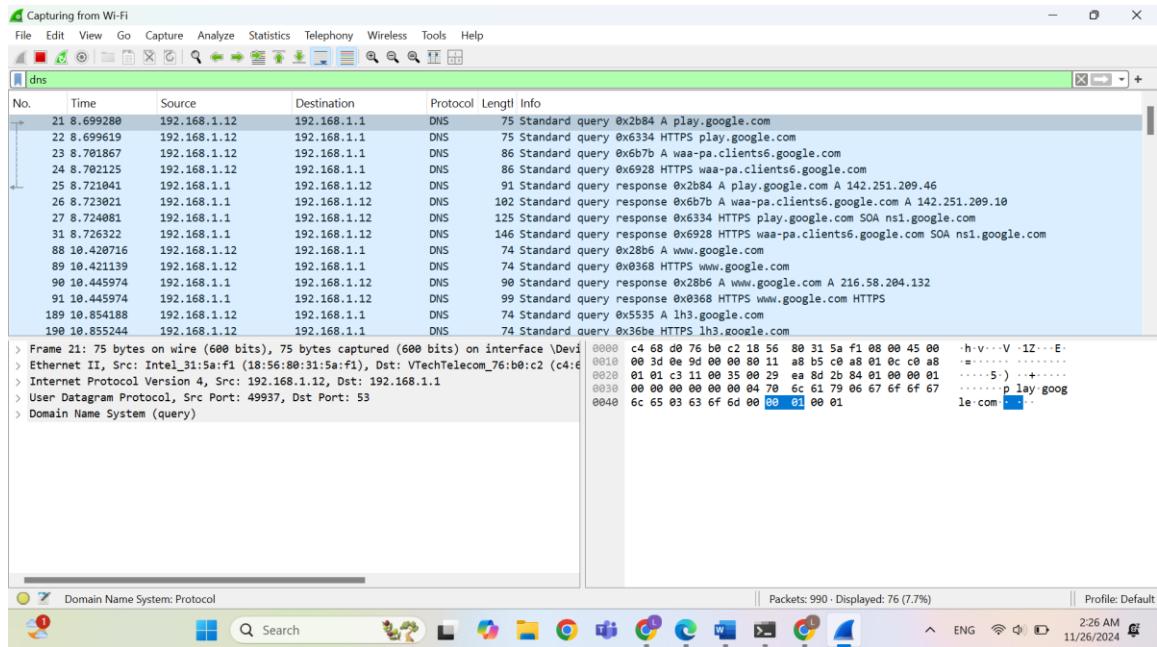


Figure 58 Wireshark test

Here is the output of the Wireshark software when capturing some DNS messages.

## Task two:

The testing process for Task 2 is detailed in the procedure section. It explains the steps taken to verify the functionality, ensuring that the task met the required objectives and operated as expected.

## Task three:

The screenshot shows four separate Command Prompt windows running on a Windows 10 desktop. The top-left window shows the server's continuous message: "Waiting for players to join...". The top-right window shows a client connecting: "Client layan ('192.168.56.1', 62615) connected.". The bottom-left window shows another client connecting: "Client ahmad ('192.168.56.1', 62952) connected.". The bottom-right window shows the server confirming player counts: "Number of players: 1" and "Number of players: 2".

```
Waiting for players to join...
Client layan ('192.168.56.1', 62615) connected.

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: layan
Username sent to the server.
Server: You have joined the game!
Server: layan has joined the game
Number of players: 1

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: layan
Username sent to the server.
Server: You have joined the game!
Server: layan has joined the game
Number of players: 1

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: ahmad
Username sent to the server.
Server: You have joined the game!
Server: ahmad has joined the game
Number of players: 2

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: tuleen
Username sent to the server.
Server: You have joined the game!
Server: tuleen has joined the game
Number of players: 3
```

Figure 59 testing task 3 part 1

Figure 59:

Server keep printing until game started, it starts when at least 2 clients joined.

The screenshot shows four separate Command Prompt windows running on a Windows 10 desktop. The top-left window shows the server's continuous message: "Waiting for players to join...". The top-right window shows a client connecting: "Client ahmad ('192.168.56.1', 62952) connected.". The bottom-left window shows another client connecting: "Client tuleen ('192.168.56.1', 62953) connected.". The bottom-right window shows the server confirming player counts: "Number of players: 1", "Number of players: 2", and "Number of players: 3".

```
Waiting for players to join...
Client ahmad ('192.168.56.1', 62952) connected.
90-second countdown started. Waiting for more players...
Waiting for players to join...
Waiting for players to join...
Client tuleen ('192.168.56.1', 62953) connected.
Waiting for players to join...
Waiting for players to join...
Waiting for players to join...

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: layan
Username sent to the server.
Server: You have joined the game!
Server: layan has joined the game
Number of players: 1

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: ahmad
Username sent to the server.
Server: You have joined the game!
Server: ahmad has joined the game
Number of players: 2

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: tuleen
Username sent to the server.
Server: You have joined the game!
Server: tuleen has joined the game
Number of players: 3

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd PycharmProjects\udp
C:\Users\Dell\PycharmProjects\udp>python client.py
Enter the server IP Address: 192.168.56.1
Enter the port number: 5689
Enter your username: tuleen
Username sent to the server.
Server: You have joined the game!
Server: tuleen has joined the game
Number of players: 3
```

Figure 60 testing task 3 part 2

Figure 60:

Client Side: the client enters the IP address and port number that the server is running on. After that, the username is sent to the server.

Server Side: After two players connect, the server waits for 90 seconds before starting the game, allowing time for additional players to join. The server sends updates to the players regarding the number of active players.

The image shows four Command Prompt windows arranged in a 2x2 grid, illustrating the interaction between a client and a server during a game session. The top-left window shows the client's perspective, where the server is waiting for players to join. The top-right window shows the server's perspective, confirming player connections and sending initial game information. The bottom-left window shows the client responding to the server's questions. The bottom-right window shows the server processing the client's answers and sending feedback. The system tray at the bottom indicates the date and time as 12/1/2024 and 3:08 PM.

```
Waiting for players to join...
20 seconds have passed, starting the game now!
Starting the game...
Round 1 starting...
Broadcasting Question 1: What is 7 x 6?
Received answer from ('192.168.56.1', 62615): 42
Received answer from ('192.168.56.1', 62952): 42
Received answer from ('192.168.56.1', 62953): 42
```

```
Number of players: 2
Server: 90 seconds until the game starts! Invite more players.
Server: tuleen has joined the game
Number of players: 3
Server: The game is starting now!
Server: This game contains 3 rounds and each round with 3 questions , be ready!!!
Server: Round 1 Question: 1: What is 7 x 6?
```

```
Answer the question below:
Your answer is:
42
Answer submitted!
```

```
Server: You have joined the game!
Server: tuleen has joined the game
Number of players: 3
Server: The game will start soon. You are in the game!
Server: The game is starting now!
Server: This game contains 3 rounds and each round with 3 questions , be ready!!!
Server: Round 1 Question: 1: What is 7 x 6?
```

```
Answer the question below:
Your answer is:
42
Answer submitted!
```

Figure 61 testing task 3 part 3

Figure 61:

After the time has passed, the server starts sending questions, and the players submit their answers.

```

Command Prompt - python s
Waiting for players to join...
20 seconds have passed, starting the game now!
Starting the game...
Round 1 starting...
Broadcasting Question 1: What is 7 x 6?
Received answer from ('192.168.56.1', 62615): 42
Received answer from ('192.168.56.1', 62952): 42
Received answer from ('192.168.56.1', 62953): 42
The correct answer was: 42
Broadcasting Question 2: What is the capital of palestine?
Received answer from ('192.168.56.1', 62953): jerusalem
Received answer from ('192.168.56.1', 62952):
Received answer from ('192.168.56.1', 62615): jerusalem

Command Prompt - python c
Answer the question below:
Your answer is:
42
Answer submitted!
Server: Correct! You earned 0.33 points.
Server: The correct answer was: 42
Server: Round 1 Question: 2: What is the capital of palestine?

Answer the question below:
Your answer is:
jerusalem
Answer submitted!

Command Prompt - python c
Answer the question below:
Your answer is:
42
Answer submitted!
Server: Correct! You earned 0.67 points.
Server: The correct answer was: 42
Server: Round 1 Question: 2: What is the capital of palestine?

Answer the question below:
Your answer is:
Answer submitted!

```

Figure 62 testing task 3 part 4

Figure 62:

Now, the server awards points as follows: 0 points for wrong answers, and for correct answers, points are assigned in order (1 - index/number of correct players). For example, Layan gets 1 point, Ahmad gets 0.66, and Tuleen gets 0.33. After the time for answering passes, the next question in the round is sent.

```

The correct answer was: 42
Broadcasting Question 2: What is the capital of palestine?
Received answer from ('192.168.56.1', 62953): jerusalem
Received answer from ('192.168.56.1', 62952):
Received answer from ('192.168.56.1', 62615): jerusalem
The correct answer was: Jerusalem
Broadcasting Question 3: What is the atomic number of hydrogen?
Received answer from ('192.168.56.1', 62953): 1
Received answer from ('192.168.56.1', 62952): 2
Received answer from ('192.168.56.1', 62615): 3
The correct answer was: 1
The current leader is tuleen with 2.33 points.
Waiting 30 seconds before starting the next round...

Command Prompt - python c
Answer the question below:
Your answer is:
3
Answer submitted!
Server: Wrong!
Server: The correct answer was: 1
Server: Round Results:
ahmad: 0.67 points
layan: 1.50 points
tuleen: 2.33 points
Server: The current leader is tuleen with 2.33 points.
Server: Prepare for the next round. Starting in 30 seconds!

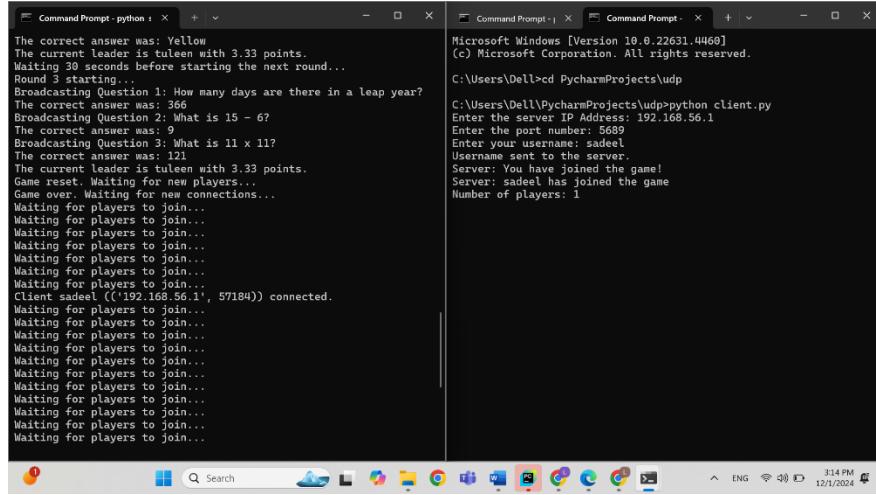
Command Prompt - python c
Answer the question below:
Your answer is:
2
Answer submitted!
Server: Wrong!
Server: The correct answer was: 1
Server: Round Results:
ahmad: 0.67 points
layan: 1.50 points
tuleen: 2.33 points
Server: The current leader is tuleen with 2.33 points.
Server: Prepare for the next round. Starting in 30 seconds!


```

Figure 63 testing task 3 part 4

Figure 63:

After every round, the server sends the results to all players and announces the leader. At the end of the game, the server sends the winner's details to the players.



*Figure 64 testing task 3 part 5*

Figure 64:

After the first game ends, the server continues to run and waits for more players to join, then repeats the game process.

# Alternative Solutions, Issues, and Limitations

## Task 2:

Web Server:

Working on the project without using any library other than the socket library was quite challenging, especially when it came to handling image searches. Normally, parsing a URL would be easy with a library but since we couldn't use it, we had to implement the functionality manually, which was tricky and time-consuming.

Another challenge we faced was figuring out how to automatically redirect to another webpage. It took a lot of research and effort to come up with a working solution.

## Task 3:

### Issues Faced and Solutions

#### 1. Blocked Server Operations

- Issue: The server's message reception was blocked until the message was fully received, preventing the server from continuing its operations. This caused delays and incomplete communication.
- Solution: We set a timeout for message reception and used a try-except block to handle exceptions. This allowed the server to continue its operations even if a message wasn't received within the specified time, ensuring smoother communication and preventing the server from getting stuck.

#### 2. Client Program Stuck on Answer

- Issue: If the client didn't answer the question, the program would get stuck, and the next question wouldn't be sent.
- Solution: We used threading to allow the client to simultaneously handle user input and listen for server messages. This ensured that even if the user didn't provide an answer, the program could continue to receive the next question without delays.

### 3. Handling Unexpected Exceptions

- Issue: Using except timeout alone caused other unexpected exceptions to appear, interrupting the program.
- Solution: We replaced it with a generic except Exception block, which catches all types of exceptions. This ensured the program remained stable and continued running, regardless of the specific error encountered.

### Limitations

In the server code, we didn't use threading, and it worked fine during testing because of the small number of players. However, if a large number of players join, the server could face performance issues. Specifically, the message handling could become sequential, causing delays in processing as the server handles one client at a time. This may lead to slower responses, increased latency, or even unresponsiveness under heavy load. Using threading or asynchronous handling would be necessary to handle a large number of players efficiently.

## Teamwork Chart analysis:

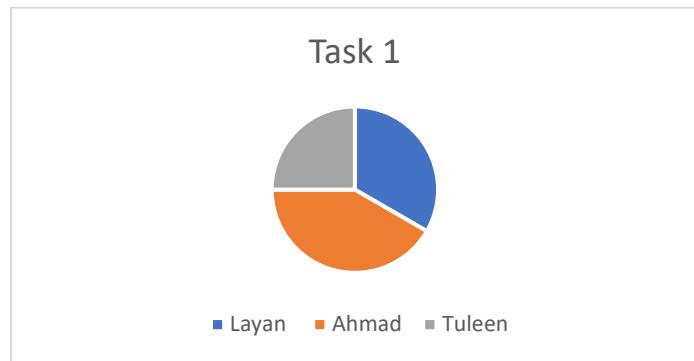


Figure 65 Teamwork Chart analysis

### □ Task One

- In this task, we decided it would be better if each team member tried the commands individually. Ahmad then took responsibility for testing the functionality.

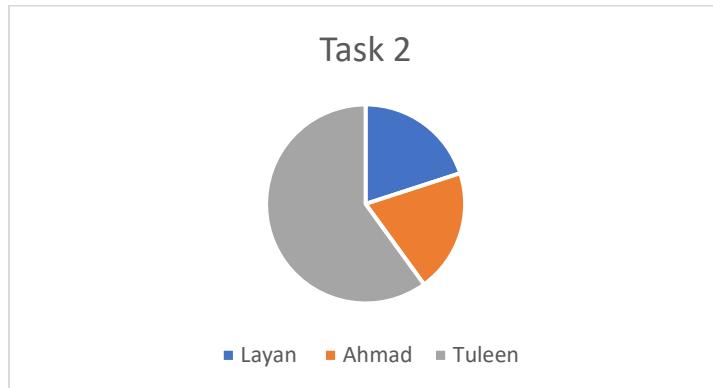
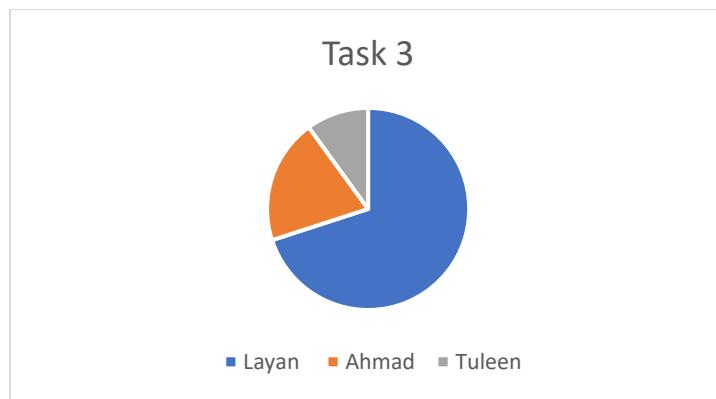


Figure 66 Teamwork Chart analysis

### □ Task Two

- This task required more effort compared to the first one. We divided the work: Tuleen started by designing the pages, and Layan resolved any problems in code encountered during the process.



*Figure 67 Teamwork Chart analysis*

### Task Three

- For this task, Layan wrote the initial code, and Ahmad focused on fixing any issues.

## References

- [1]: Simplilearn. "Understanding Networking Commands." Retrieved from <https://www.simplilearn.com>.
- [2]: N4L Support. "How to Use Telnet to Check the Status of Ports." Retrieved from <https://support.n4l.co.nz>.
- [3]: Wireshark Documentation. "Introduction to Wireshark." Retrieved from <https://www.wireshark.org>.
- [4]: GeeksforGeeks. "Socket Programming in Python." Retrieved from <https://www.geeksforgeeks.org>.
- [5]: Avast. "TCP vs. UDP: Key Differences." Retrieved from <https://www.avast.com>.