



**Faculty of Engineering & Technology
Electrical & Computer Engineering Department
ENCS3320: Computer Networks
Second Semester (2024/2025)
Project 1: Socket Programming**

Prepared by:

Taleed Hamadneh	1220006	sec.4
Qasim Batrawi	1220204	sec.2

Instructors:

Dr. Ibrahim Nemer
Dr. Alhareth Zyoud

Date: May 10, 2025

Abstract

This report explains a project created to help us learn more about computer networks and how to use socket programming. The project is divided into three parts. First, we explored and practiced common network commands and used Wireshark to understand how data moves across the internet. Next, we built a basic web server using sockets that can show web pages in both English and Arabic, manage redirections, and handle missing files or wrong requests. Finally, we created a multiplayer guessing game where players connect to a server and try to guess a secret number. The game uses TCP to set up the connection and UDP for fast communication during the game. Through this project, we gained practical experience in networking, programming, and working as a team.

Table of Contents

Abstract	I
Theory & Procedure.....	1
Socket Programming.....	1
Socket Programming with UDP	1
Socket Programming with TCP	2
Network Commands	3
Wireshark	4
Web Server.....	4
Methods and Components Used in Task 2.....	5
Methods and Components Used in Task 3.....	6
Results and Discussion	7
Task 1: Network Commands and Wireshark	7
Executing Network Commands	7
Wireshark DNS Capture for gaia.cs.umass.edu	16
Task 2: Simple Web Server Using Socket Programming	17
Webpages	17
HTML & CSS	27
Python Server Code	35
Server Functionality Testing	44
TASK 3: TCP/UDP Client-Server Game Using Socket Programming	62
Game Workflow	62
Python Server Code	63
Python Client Code	68
Test Cases.....	72
Alternative Solutions, Issues, and Limitation	76
Teamwork.....	77
References	78

Table of Figures

Figure 1 Socket Programming [1].....	1
Figure 2 UDP VS TCP [1]	2
Figure 3 Web Server [4].....	5
Figure 4 ipconfig/all command	7
Figure 5 ping device.....	8
Figure 6 Ping gaia.cs.umass.edu	8
Figure 7 Location	9
Figure 8 Tracert gaia.cs.umass.edu	10
Figure 9 nslookup gaia.cs.umass.edu.....	11
Figure 10 telnet gaia.cs.umass.edu	12
Figure 11 AS Information for gaia.cs.umass.edu.....	13
Figure 12 Information about gaia.cs.umass.edu	14
Figure 13 Routing relationships between AS1249 and other networks in the internet.....	15
Figure 14 Capturing DNS Query.	16
Figure 15 Main English Webpage 1	18
Figure 16 Main English Webpage 2.....	18
Figure 17 Main English Webpage 3.....	19
Figure 18 Main English Webpage 4.....	20
Figure 19 Main English Webpage 5.....	20
Figure 20 Main English Webpage 6.....	21
Figure 21 Main Arabic Webpage 1	22
Figure 22 Main Arabic Webpage 2	23
Figure 23 Main Arabic Webpage 3	23
Figure 24 Main Arabic Webpage 4	24
Figure 25 Main Arabic Webpage 5	24
Figure 26 Main Arabic Webpage 6	25
Figure 27 English Local Webpage	26
Figure 28 Arabic Local Webpage.....	26
Figure 29 Main English Webpage HTML 1	27
Figure 30 Main English Webpage HTML 2	27
Figure 31 Main English Webpage HTML 3	28
Figure 32 Main English Webpage HTML 4	28
Figure 33 Main English Webpage CSS 1.....	29
Figure 34 Main English Webpage CSS 2.....	29
Figure 35 Main English Webpage CSS 3.....	30

Figure 36 Main English Webpage CSS 4.....	30
Figure 37 Main English Webpage CSS 5.....	31
Figure 38 Main English Webpage CSS 6.....	31
Figure 39 Main English Webpage CSS 7.....	32
Figure 40 English Local Webpage HTML.....	32
Figure 41 English Local Webpage CSS 1	33
Figure 42 English Local Webpage CSS 2	33
Figure 43 English Local Webpage CSS 3	34
Figure 44 Python Server Code 1	35
Figure 45 Python Server Code 2	36
Figure 46 Python Server Code 3	37
Figure 47 Python Server Code 4	38
Figure 48 Python Server Code 5	38
Figure 49 Python Server Code 6.....	39
Figure 50 Python Server Code 7.....	40
Figure 51 Python Server Code 8.....	41
Figure 52Python Server Code 9.....	42
Figure 53 Python Server Code 10.....	43
Figure 54 Python Server Code 11	43
Figure 55 Browser Request for main English webpage	44
Figure 56 index.html request	45
Figure 57 main_en.css request.....	45
Figure 58 members photos requests.....	46
Figure 59 Switching photos requests	46
Figure 60 Background photos requests.....	47
Figure 61 Response for the main English webpage.....	47
Figure 62 Browser Request for main Arabic webpage	48
Figure 63 Part of browser requests	48
Figure 64 Response for the main Arabic webpage	49
Figure 65 invalid URL request.....	49
Figure 66 Request for invalid file	50
Figure 67 Response for invalid Request	50
Figure 68 Birzeit Website.....	51
Figure 69 Textbook Website	51
Figure 70 mySite_1220006_en.html request	52
Figure 71 Local Webpage objects' requests	52
Figure 72 Response for the local Webpage.....	53

Figure 73 Search for image in local Webpage	53
Figure 74 Request for image from local Webpage	54
Figure 75 Response for the image	54
Figure 76 Search for unavailable image	55
Figure 77 Request for unavailable image	55
Figure 78 Response for unavailable image.....	56
Figure 79 Search for video in local Webpage	56
Figure 80 Request for video from local Webpage	57
Figure 81 Response for the Video.....	57
Figure 82 Search for unavailable video	58
Figure 83 Request for unavailable video	58
Figure 84 Response for unavailable video.....	59
Figure 85 Response for css file request	59
Figure 86 Request for a webpage from another device	60
Figure 87 Server Response to Iphone device request.....	60
Figure 88 Server Response to IPAD device request	61
Figure 89 Server Code 1	63
Figure 90 Server Code 2	64
Figure 91 Server Code 3	64
Figure 92 Server Code 4	65
Figure 93 Server Code 5	66
Figure 94 Server Code 6	66
Figure 95 Server Code 7	67
Figure 96 Client Code 1	68
Figure 97 Client Code 2	69
Figure 98 Client Code 3	69
Figure 99 Client Code 4.....	70
Figure 100 Client Code 5	70
Figure 101 Client Code 6.....	71
Figure 102 Client Code 7	71
Figure 103 Test Case 1 – Correct Guess & Invalid Name	72
Figure 104 Test Case 2 – Time Out & Invalid Input	73
Figure 105 Test Case 3 - Client Disconnection.	74
Figure 106 Test Case 3 - Terminal of Disconnected Client	74
Figure 107 Test Case 4 - Client Disconnection	75
Figure 108 Test Case 4 - Terminal of Disconnected Client	75
Figure 109 Teamwork chart	77

Theory & Procedure

Socket Programming

A typical network application consists of a pair of programs—a client program and a server program—residing in two different end systems. When these two programs are executed, a client process and a server process are created, and these processes communicate with each other by reading from, and writing to, sockets. When creating a network application, the developer's main task is therefore to write the code for both the client and server programs [1].

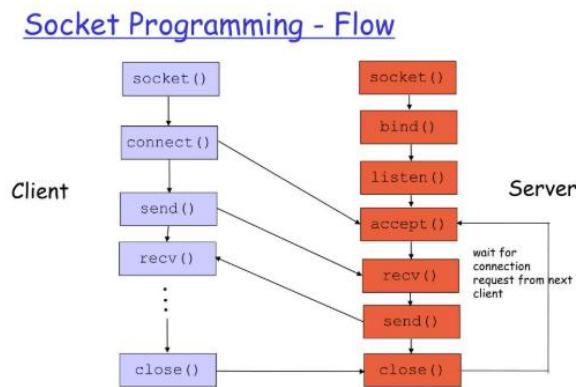


Figure 1 Socket Programming [1]

Socket Programming with UDP

The User Datagram Protocol (UDP) runs on top of the Internet Protocol (IP) and was developed for applications that do not require reliability, acknowledgment, or flow control features at the transport layer. This simple protocol provides transport layer addressing in the form of UDP ports and an optional checksum capability. UDP is a very simple protocol. Messages, so called datagrams, are sent to other hosts on an IP network without the need to set up special transmission channels or data paths beforehand. The UDP socket only needs to be opened for

communication. It listens for incoming messages and sends outgoing messages on request [2].

Socket Programming with TCP

Unlike UDP, TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection. One end of the TCP connection is attached to the client socket and the other end is attached to a server socket. When creating the TCP connection, we associate with it the client socket address (IP address and port number) and the server socket address (IP address and port number). With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket. This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket [1].

TCP	UDP
Transmission Control Protocol	User Datagram Protocol
Connection-Oriented	Connectionless
Slower	Faster
20 Bytes Header	8 Bytes Header
Packet Reorder Mechanism	No Reorder Mechanism
Uses Acknowledgement	No Acknowledgement
Error Checking and Recovery	Basic Error Checking
Guaranteed delivery, reliable	No guarantee, unreliable
Uses SSL/TLS for security	Uses DTLS for security
HTTP, HTTPS, FTP etc.	DHCP, TFTP, SNMP etc.

Figure 2 UDP VS TCP [1]

Network Commands

Network commands are essential tools that help users interact with and control network settings. They are used to troubleshoot problems, check connectivity, gather details about network devices, and monitor how data moves across a network. With these commands, users can check IP settings, trace the path of data packets, translate domain names to IP addresses, and manage different network protocols. These tools are particularly useful for network administrators and others working to improve or repair network performance. Below are the specific commands we will use in this project:

I. ipconfig

Shows your computer's IP address and network details. It's useful for checking your connection or troubleshooting network issues.

II. ping

Sends a small message to another computer or website to see if it's reachable and how long it takes to respond. It's great for checking if your internet or a specific site is working.

III. tracert/traceroute

Shows the path your data takes to get to another computer or website, step by step. It helps to find where delays or problems happen along the way.

IV. telnet

Let's you connect to another computer over the internet and control it using text commands. It's mostly used for testing or remote access to servers, but isn't very secured.

V. nslookup

Finds the IP address of a website name (or the name from an IP address). It helps check if domain name systems (DNS) are working properly.

Wireshark

Wireshark is a free and powerful tool used to check and understand what's happening on a network. It lets users see network traffic in real time or look at saved data to find problems. Many IT and cybersecurity professionals use it to help fix issues and keep networks secure. With its easy-to-use design, Wireshark supports many types of network systems and offers helpful features to filter, highlight, and study the data—making it easier to spot things like slow connections, lost data, or unusual activity.

Its applications include identifying performance bottlenecks, analyzing VoIP call quality, studying protocol behavior, and detecting security threats such as malware communications. However, using Wireshark requires a good understanding of network protocols, as it can only display the data it captures without decrypting encrypted traffic unless the proper keys are provided. Legal and ethical considerations are also essential, as capturing data without proper authorization may violate privacy laws and regulations [\[3\]](#).

Web Server

Web server is a program which processes the network requests of the users and serves them with files that create web pages. This exchange takes place using Hypertext Transfer Protocol (HTTP). Basically, web servers are computers used to store HTTP files which makes a website and when a client requests a certain website, it delivers the requested website to the client. For example, you want to open Facebook on your laptop and enter the URL in the search bar of google. Now, the laptop will send an HTTP request to view the facebook webpage to another computer known as the webserver. This computer (webserver) contains all the files (usually in HTTP format) which make up the website like text, images, gif files, etc. After processing the request, the webserver will send the requested website-

related files to your computer and then you can reach the website. Different websites can be stored on the same or different web servers but that doesn't affect the actual website that you are seeing in your computer. The web server can be any software or hardware but is usually a software running on a computer. One web server can handle multiple users at any given time which is a necessity otherwise there had to be a web server for each user and considering the current world population, is nearly close to impossible. A web server is never disconnected from the internet because if it was, then it won't be able to receive any requests, and therefore cannot process them [4].

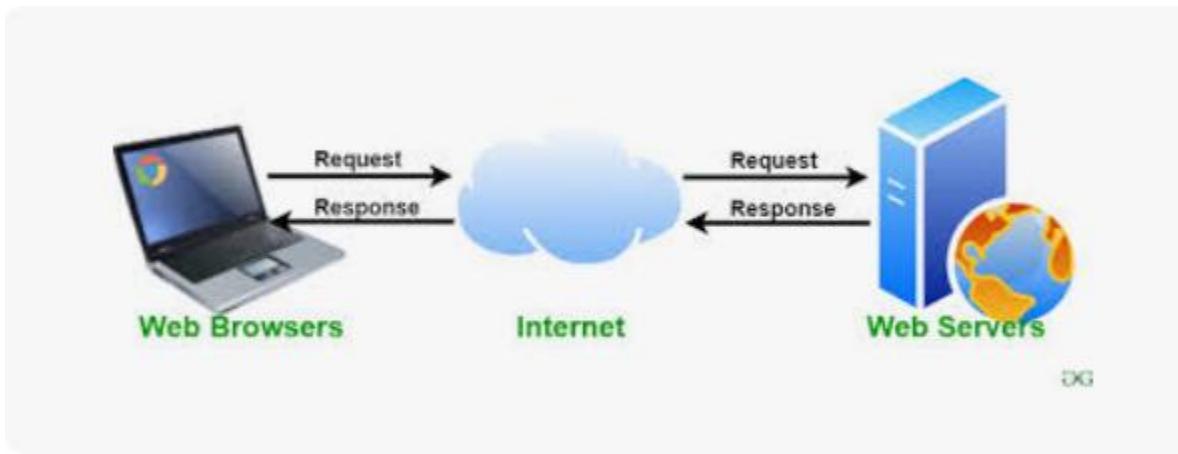


Figure 3 Web Server [4]

Methods and Components Used in Task 2

For Task 2, we developed a simple HTTP server using the socket library that listens on port 9906 to handle requests for static resources such as HTML, CSS, images, and videos. It begins by creating a TCP socket using **socket()**, binds it to the specified IP address and port using **bind()**, and prepares to accept incoming client connections with **listen()**. The server listens for requests, and once a connection is made, it accepts the client's connection with **accept()**.

The server processes incoming HTTP requests by first identifying the requested file. It checks for the existence of HTML, CSS, image, and video files. If the file is found, the server sends a 200 OK response along with the requested content using the `send_response()` method. If the file is not found, the server responds with a 404 Not Found error page designed using the `Error()` method. The error page includes the client's IP address and port number for transparency.

Additionally, for image and video requests, if the file is missing, the server issues a 307 Temporary Redirect response. In such cases, the client is redirected to Google for images or YouTube for videos for alternative search options. This approach provides a flexible way for the server to respond to various file requests, ensuring a smooth user experience even in cases of missing media files, by managing both errors and redirections effectively.

Methods and Components Used in Task 3

For Task 3, we developed a hybrid TCP/UDP server for a multiplayer guessing game, using TCP for reliable connection setup and UDP for fast-paced, real-time gameplay. The server listens on two different ports, one for TCP (port 6000) and one for UDP (port 6001). Key methods used in the code include: `bind()` to assign the server IP and ports to the sockets, `listen()` to listen for incoming TCP connections, and `accept()` to handle client connections via TCP. Once connected, the server checks for unique usernames and adds players to the game. The game starts once the minimum number of players is reached, and players send their guesses via UDP. `recvfrom()` is used to receive guesses from clients, while `sendto()` sends real-time feedback to the clients, such as "Higher," "Lower," or "Correct." The server also manages game flow by continuously checking the number of players and monitoring the guessing phase, and uses `Thread()` to handle multiple clients simultaneously. Additionally, methods like `settimeout()` ensure the

server manages time limits, and `close()` properly closes the connections after the game ends. This approach efficiently combines TCP for control messages and UDP for fast gameplay updates, creating an interactive and responsive game environment.

Results and Discussion

Task 1: Network Commands and Wireshark

Executing Network Commands

I. ipconfig /all command

```
Command Prompt
Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Local Area Connection* 12:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Description . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #4
Physical Address . . . . . : B6-8C-9D-2C-9A-B9
DHCP Enabled. . . . . : No
Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Wi-Fi:
Description-specific DNS Suffix . . . . . : asaltech.com
Description . . . . . : MediaTek Wi-Fi 6 MT7921 Wireless LAN Card
Physical Address . . . . . : B4-8C-9D-2C-9A-99
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::ab39:1ae8:6011:fe3d%3(Preferred)
IPv4 Address . . . . . : 172.20.258.26(Preferred)
Subnet Mask . . . . . : 255.255.254.0
Lease Obtained. . . . . : Sunday, May 4, 2025 8:27:25 AM
Lease Expires . . . . . : Sunday, May 4, 2025 6:27:25 PM
Default Gateway . . . . . : 172.20.258.1
DHCP Server . . . . . : 172.22.1.30
DHCPv6 IID . . . . . : 95718551
DHCPv6 Client DUID. . . . . : 00-01-00-2A-26-87-23-50-EF-FE-E6-A1-FA
DNS Servers . . . . . : 8.8.8.8
NetBIOS over Tcpip. . . . . : Enabled

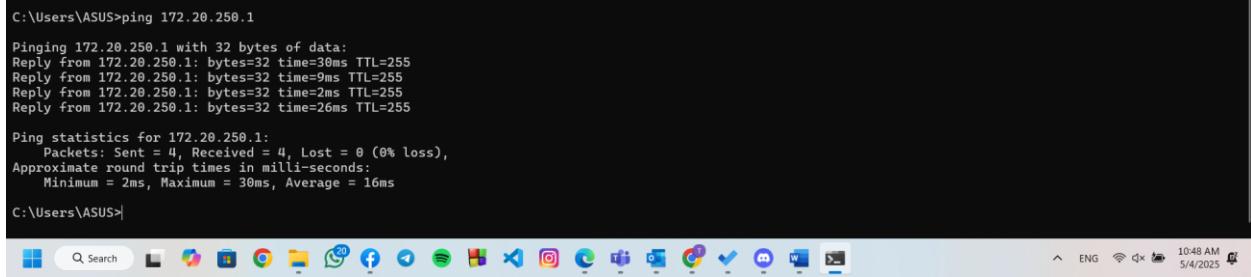
Ethernet adapter Bluetooth Network Connection:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . . . . : Bluetooth Device (Personal Area Network)
Description . . . . . : Bluetooth Device (Personal Area Network)
Physical Address . . . . . : B4-8C-9D-2C-9A-98
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes

Ethernet adapter Ethernet:
```

Figure 4 ipconfig/all command

- IPv4 Address: 172.20.250.26
 - Subnet Mask: 255.255.254.0
 - Default Gateway: 172.20.250.1
 - DNS Server: 8.8.8.8

II. Ping command



```
C:\Users\ASUS>ping 172.20.250.1

Pinging 172.20.250.1 with 32 bytes of data:
Reply from 172.20.250.1: bytes=32 time=30ms TTL=255
Reply from 172.20.250.1: bytes=32 time=9ms TTL=255
Reply from 172.20.250.1: bytes=32 time=2ms TTL=255
Reply from 172.20.250.1: bytes=32 time=26ms TTL=255

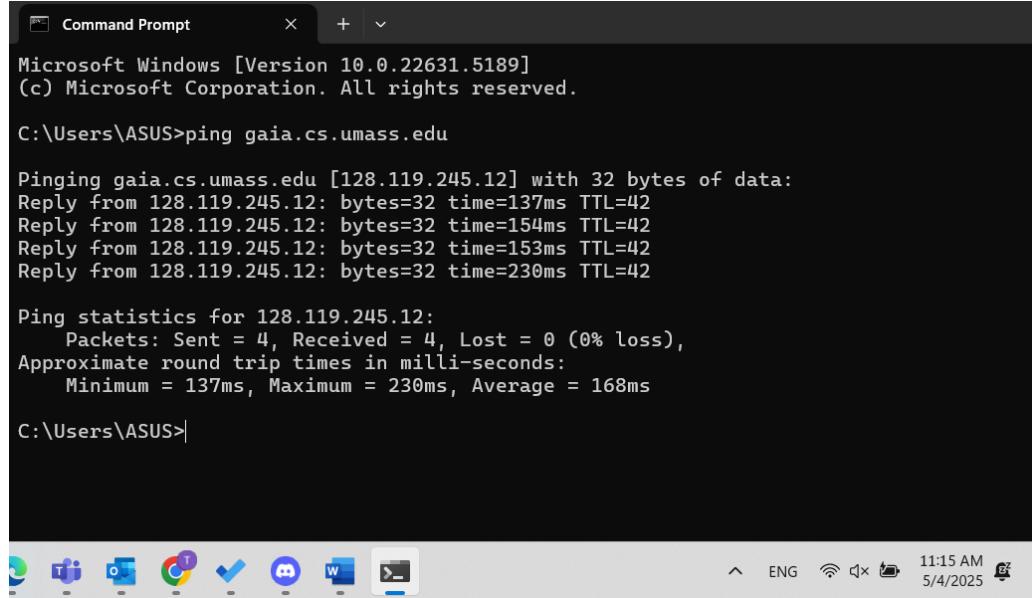
Ping statistics for 172.20.250.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2ms, Maximum = 30ms, Average = 16ms

C:\Users\ASUS>
```

Figure 5 ping device

The ping command was used to check the connection to 172.20.250.1. All 4 packets were sent and received with no loss, showing a stable connection. The response times ranged from 2ms to 30ms, averaging 16ms. The TTL value was 255. This confirms the device is reachable and the network is working well.

III. Ping gaia.cs.umass.edu command



```
Command Prompt      + | - 
Microsoft Windows [Version 10.0.22631.5189]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS>ping gaia.cs.umass.edu

Pinging gaia.cs.umass.edu [128.119.245.12] with 32 bytes of data:
Reply from 128.119.245.12: bytes=32 time=137ms TTL=42
Reply from 128.119.245.12: bytes=32 time=154ms TTL=42
Reply from 128.119.245.12: bytes=32 time=153ms TTL=42
Reply from 128.119.245.12: bytes=32 time=230ms TTL=42

Ping statistics for 128.119.245.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 137ms, Maximum = 230ms, Average = 168ms

C:\Users\ASUS>
```

Figure 6 Ping gaia.cs.umass.edu

The server **gaia.cs.umass.edu** resolves to the IP address **128.119.245.12**. A ping test shows that four packets were sent and received successfully with **no packet loss**. The round-trip time for the packets ranged between **137 and 230ms**,

indicating stable network performance. These results confirm that the server is reachable and properly connected to the network.

Based on the results shown in the figure below, the IP address **128.119.245.12** is located in **Amherst, Massachusetts, USA**. Although the domain is associated with an educational institution, the **University of Massachusetts - Amherst**, the server is hosted in the United States. This indicates that the service is operated locally by the university.

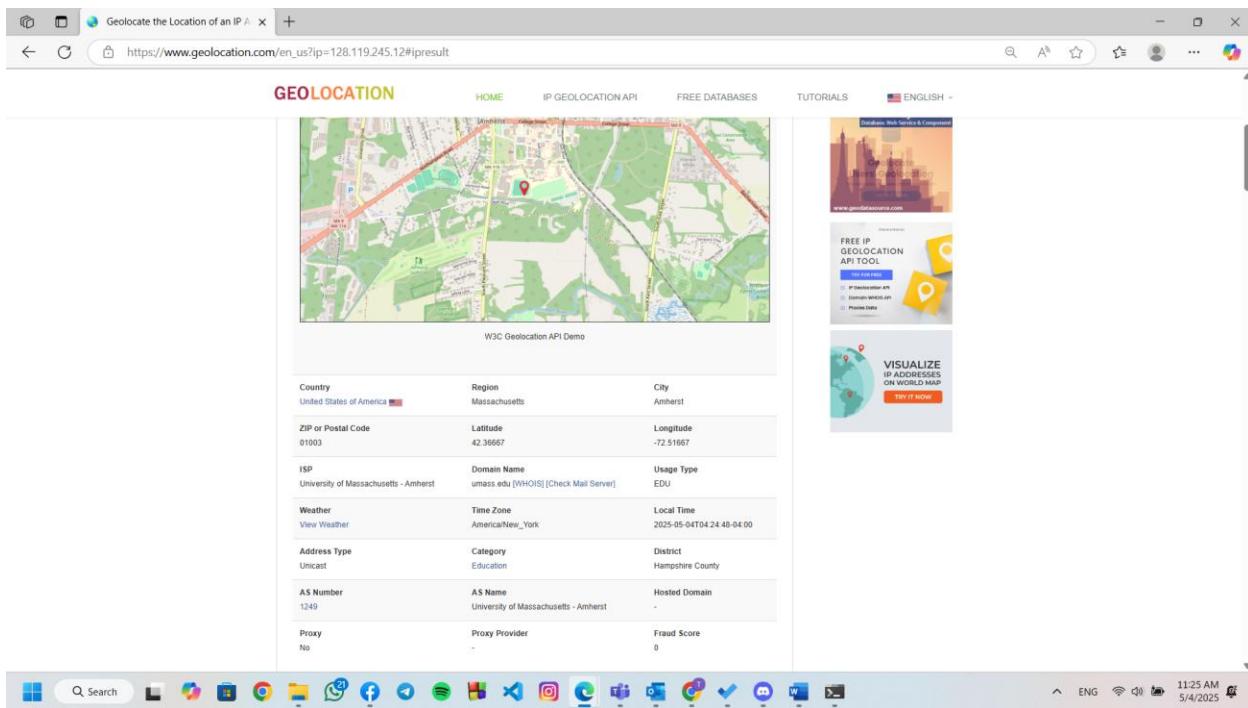


Figure 7 Location

IV. Tracert gaia.cs.umass.edu command

```
Microsoft Windows [Version 10.0.22631.5189]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS>tracert gaia.cs.umass.edu

Tracing route to gaia.cs.umass.edu [128.119.245.12]
over a maximum of 30 hops:

 1   3 ms    38 ms    2 ms  172.20.250.1
 2   12 ms    3 ms    2 ms  82.213.2.185
 3   58 ms   73 ms   61 ms  10.74.19.110
 4   62 ms   52 ms   48 ms  10.74.19.158
 5   87 ms   45 ms   54 ms  10.74.59.238
 6   93 ms   61 ms  108 ms  et-4-0-13.edge1.Marseille3.Level3.net [213.242.111.121]
 7   58 ms   59 ms   68 ms  ae2.3605.edge4.mrs1.neo.colt.net [171.75.8.225]
 8   *       *       * Request timed out.
 9   66 ms   62 ms   83 ms  be2779.ccr41.par01.atlas.cogentco.com [154.54.72.109]
10   68 ms   81 ms   69 ms  be3684.ccr51.lhr01.atlas.cogentco.com [154.54.60.170]
11   140 ms  140 ms  139 ms  be3393.ccr31.bos01.atlas.cogentco.com [154.54.47.141]
12   147 ms  156 ms  141 ms  be8038.rcr71.orh02.atlas.cogentco.com [154.54.169.254]
13   143 ms  141 ms  142 ms  be8628.rcr51.orh01.atlas.cogentco.com [154.54.164.126]
14   155 ms  134 ms  171 ms  38.184.218.14
15   146 ms  139 ms  147 ms  69.16.0.8
16   188 ms  170 ms  140 ms  69.16.1.0
17   158 ms  147 ms  152 ms  core2-rt-et-8-3-0.gw.umass.edu [192.80.83.113]
18   169 ms  147 ms  148 ms  n1-pt-1-1-pt-18-0-0.gw.umass.edu [128.119.0.120]
19   144 ms  164 ms  188 ms  n1-fnt-fw-1-1-1-31-vl1092.gw.umass.edu [128.119.77.233]
20   *       *       * Request timed out.
21   189 ms  144 ms  141 ms  core2-rt-et-7-2-1.gw.umass.edu [128.119.0.121]
22   202 ms  147 ms  168 ms  n5-rt-1-1-xe-2-1-0.gw.umass.edu [128.119.3.33]
23   181 ms  141 ms  141 ms  rics-rt-xe-0-0-0.gw.umass.edu [128.119.3.32]
24   *       *       * Request timed out.
25   171 ms  137 ms  151 ms  gala.cs.umass.edu [128.119.245.12]

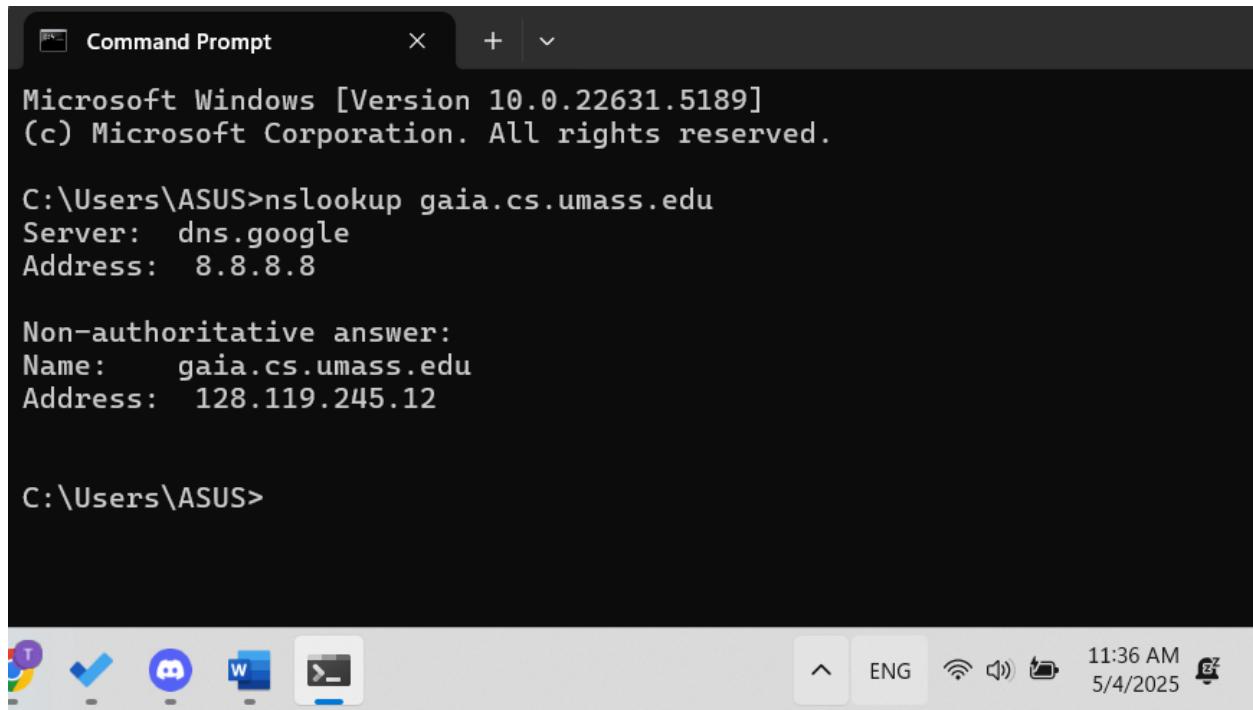
Trace complete.

C:\Users\ASUS>
```

Figure 8 Tracert gaia.cs.umass.edu

The figure above displays the result of the "tracert" command, which tracks the route a packet follows from the user's device to its destination. Here, the target IP address is **128.119.245.12**. The command reveals each step the packet takes, showing the IP addresses of the routers it passes through. The final IP address shown is that of the destination, confirming the packet has successfully arrived.

V. nslookup gaia.cs.umass.edu command



```
Command Prompt      + | - Microsoft Windows [Version 10.0.22631.5189]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS>nslookup gaia.cs.umass.edu
Server:  dns.google
Address: 8.8.8.8

Non-authoritative answer:
Name:    gaia.cs.umass.edu
Address: 128.119.245.12

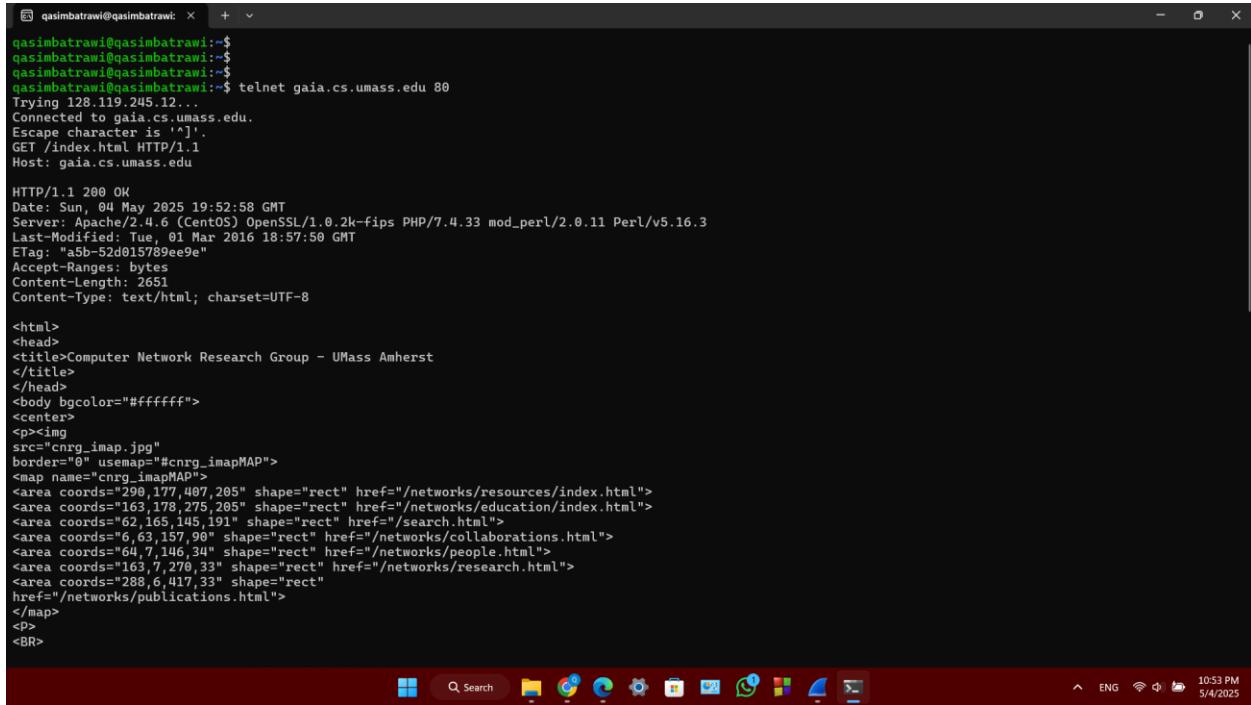
C:\Users\ASUS>
```

The screenshot shows a Microsoft Windows Command Prompt window. The title bar says "Command Prompt". The window displays the output of the "nslookup gaia.cs.umass.edu" command. It shows the server used is "dns.google" with address "8.8.8.8". A "Non-authoritative answer" is provided, showing the name "gaia.cs.umass.edu" and its address "128.119.245.12". The system tray at the bottom right shows the date and time as "5/4/2025 11:36 AM".

Figure 9 nslookup gaia.cs.umass.edu

The figure above shows the result of the "nslookup" command, which translates a domain name into its IP address. It also displays the DNS server used, helping verify domain resolution and DNS settings.

VI. telnet gaia.cs.umass.edu command



```
qasimbatrawi@qasimbatrawi:~$ telnet gaia.cs.umass.edu 80
Trying 128.119.245.12...
Connected to gaia.cs.umass.edu.
Escape character is ']'.
GET /index.html HTTP/1.1
Host: gaia.cs.umass.edu

HTTP/1.1 200 OK
Date: Sun, 04 May 2025 19:52:58 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.33 mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Tue, 01 Mar 2016 18:57:50 GMT
ETag: "a5b-52d015789ee9e"
Accept-Ranges: bytes
Content-Length: 2651
Content-Type: text/html; charset=UTF-8

<html>
<head>
<title>Computer Network Research Group - UMass Amherst
</title>
</head>
<body bgcolor="#ffffff">
<center>
<p>
<map name="cnrg_imapMAP">
<area coords="290,177,407,205" shape="rect" href="/networks/resources/index.html">
<area coords="163,178,275,205" shape="rect" href="/networks/education/index.html">
<area coords="62,165,145,191" shape="rect" href="/search.html">
<area coords="6,63,157,90" shape="rect" href="/networks/collaborations.html">
<area coords="64,7,146,34" shape="rect" href="/networks/people.html">
<area coords="163,7,278,33" shape="rect" href="/networks/research.html">
<area coords="288,6,417,33" shape="rect"
href="/networks/publications.html">
</map>
</p>
<br>
```

Figure 10 telnet gaia.cs.umass.edu

The result shows a successful connection to gaia.cs.umass.edu on port 80, which is the default port for HTTP communication. Using telnet command, we have sent an HTTP GET request for the /index.html page and the server responded with 200 OK status, followed by the HTML content page.

VII. BGP and AS Information Lookup for gaia.cs.umass.edu

The screenshot shows a web browser window for 'bgp.tools' with the URL 'bgp.tools/dns/gaia.cs.umass.edu'. The page displays DNS results for the domain 'gaia.cs.umass.edu'. Under the 'A' records section, it lists an IP address (128.119.245.12) with a prefix (128.119.0.0/16) and an ASN (AS1249 - University of Massachusetts - AMHERST). Below this, the 'MX Records' section shows an MX record for barramail.cs.umass.edu with an IP of 128.119.240.3 and a prefix of 128.119.0.0/16, also associated with AS1249. A note at the bottom states 'Queried from the local bgp.tools unbound DNS instance. (lookup bgp tools for source IPs)'. The browser's status bar at the bottom right shows '10:28 PM 5/4/2025'.

Figure 11 AS Information for gaia.cs.umass.edu

Using <http://bgp.tools/>, the figure above shows that gaia.cs.umass.edu has an autonomous system number of **AS1429**, which is registered to the University of Massachusetts Amherst. Using www.bgpview.io, the following information was obtained:

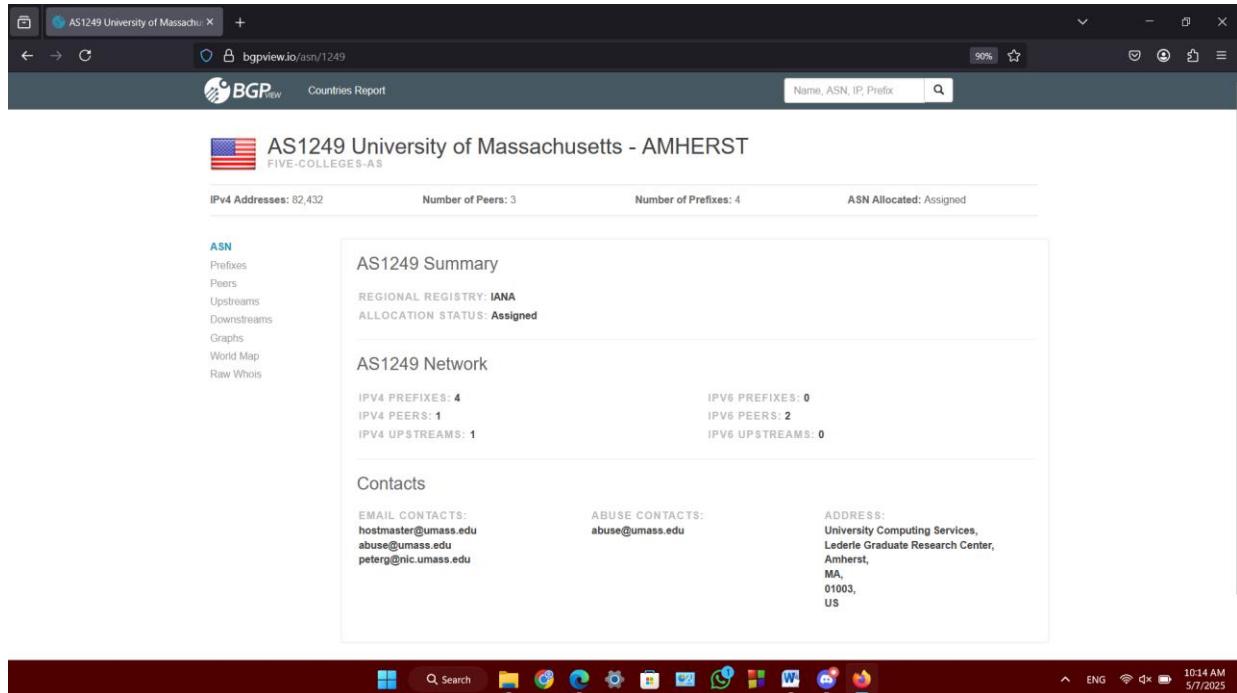


Figure 12 Information about gaia.cs.umass.edu

- Number of IP addresses: 82,432.
- Number of prefixes: 4 prefixes for IPv4, and 0 for IPv6.
- Number of peers: 1 for IPv4, and 2 for IPv6.

Results shows that the autonomous system AS1249 has a total of 82,432 unique IPv4 addresses, 4 blocks of IPv4 addresses and 0 for IPv6, 1 direct peer for IPv4 and 2 for IPv6. The following figure shows the routing relationships between AS1249 and other networks in the internet:

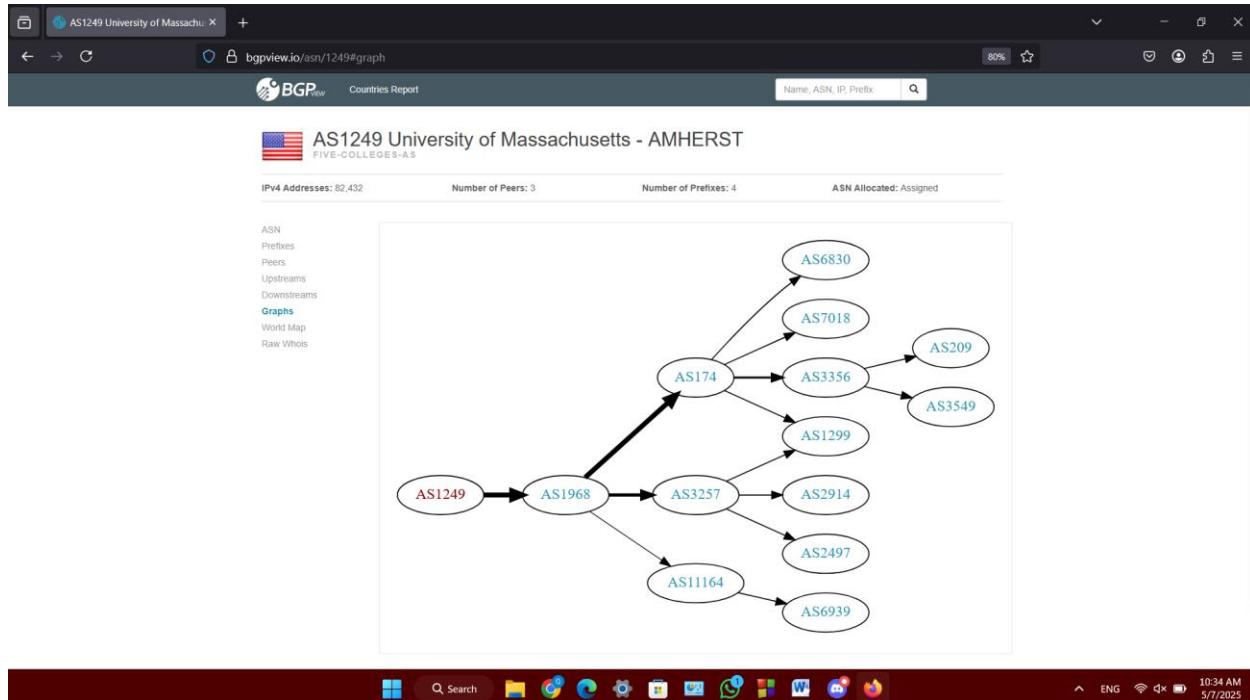


Figure 13 Routing relationships between AS1249 and other networks in the internet

Searching the list of Tier 1 ISPs on [Wikipedia](#) and looking at the autonomous systems in the figure above, we have reached that the following are considered as Tier 1 ISPs:

- AS7018 – AT&T
- AS3356 – Lumen Technologies
- AS1299 – Arelion
- AS6830 – Liberty Global
- AS3257 – GTT Communications
- AS2914 – NTT Communications

Wireshark DNS Capture for gaia.cs.umass.edu

To capture a DNS query and response for gaia.cs.umass.edu, the cache has been cleared and the following result is obtained using Wireshark:

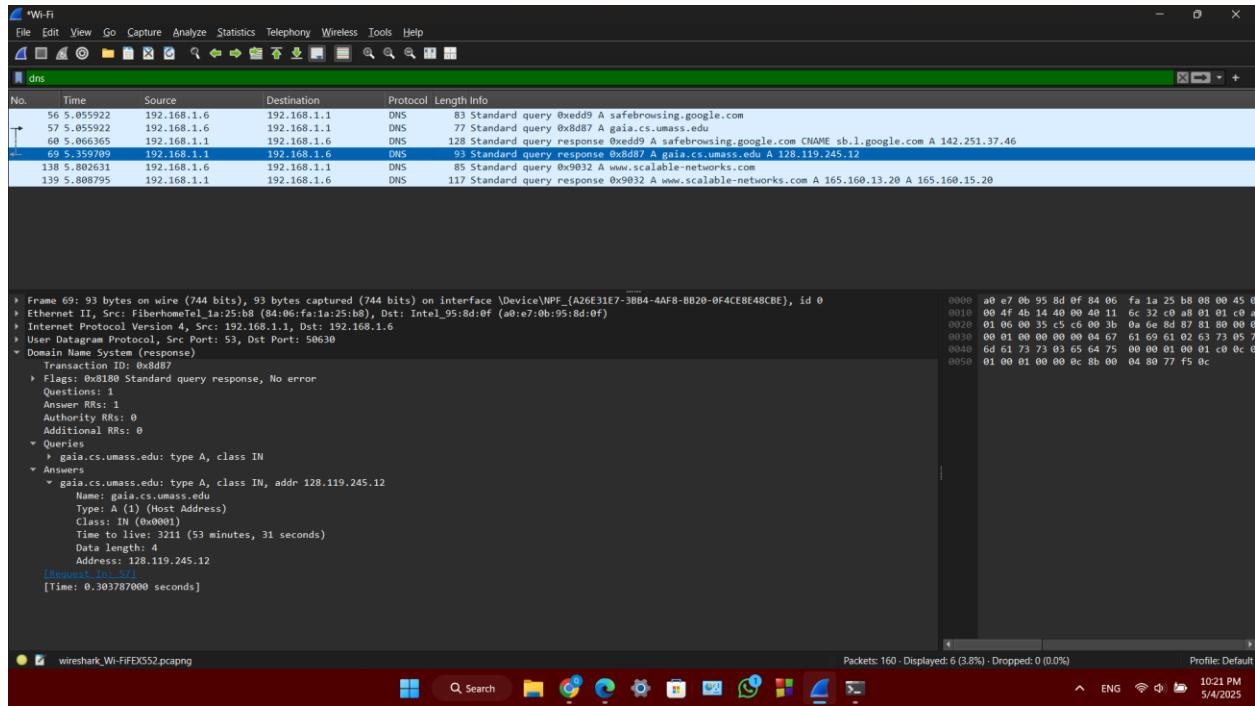


Figure 14 Capturing DNS Query.

As shown in the figure, in packet 57, the client device with IP address 192.168.1.1 sends a DNS query to the DNS server with IP 192.168.1.6. The client is requesting for the IP address of gaia.cs.umass.edu, which uses UDP on port 53. The DNS server responds in packet 69 showing the IP address for gaia.cs.umass.edu, which is 128.119.245.12.

Task 2: Simple Web Server Using Socket Programming

In this task, we were asked to build a web server using socket programming. The server uses the TCP protocol to handle HTTP requests from clients and listens on port **9906**.

First, we created several web pages using HTML and CSS: the main English page, the main Arabic page, the Local Web English page, and the Local Web Arabic page. Our server is designed to support all of these pages.

After that, we developed the web server using Python socket programming. It listens for client connections on port **9906** and uses TCP sockets to receive and respond to HTTP requests. The server replies with a **200 OK** status when the requested file is available, sends a **307 Temporary Redirect** to Google (for images) or YouTube (for videos) if the file is not on the server, and responds with **404 Not Found** if the requested item doesn't exist at all. The functionality and implementation details of the server will be explained in depth throughout this report.

Webpages

Main English Webpage (`main_en.html`):

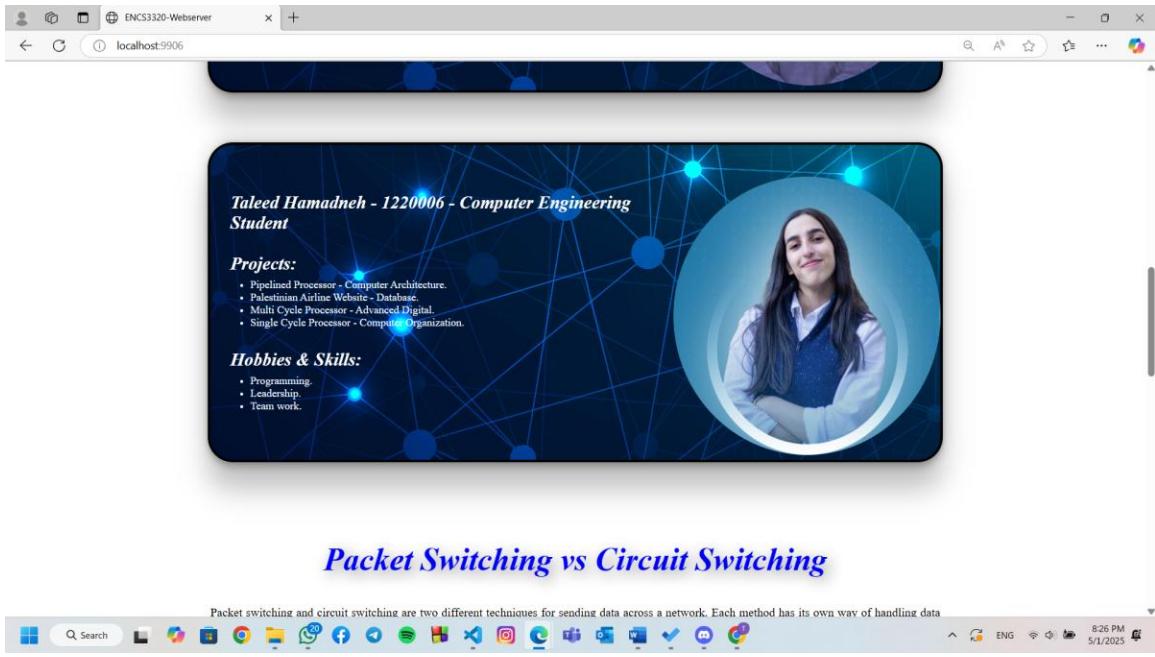
This webpage serves as the main page of the English version. It includes the page title, detailed information about the team members, and a link to the Local Webpage. As shown in the figure below, the browser tab displays the title "**ENCS3320-Webserver**" in red, and the main heading on the page is "**Welcome to ENCS3320 - Computer Networks Webserver**", presented with a background image. The page also features three navigation buttons that link to the Birzeit University website, the textbook website, and the Local Webpage.



Figure 15 Main English Webpage 1

A screenshot of a web browser window showing a team member profile. The title "Team Members" is at the top in blue. Below it, a section for "Qasim Batrawi - 1220204 - Computer Engineering Student" is shown. It includes a circular profile picture of a young man in a purple hoodie, a list of "Projects" (Pipelined Processor - Computer Architecture, Palestinian Airline Website - Database, Multi Cycle Processor - Advanced Digital, Single Cycle Processor - Computer Organization), and a list of "Hobbies & Skills" (Programming, Leadership, Team work). The background features a network of glowing blue dots and lines against a dark blue gradient. The browser's address bar shows "localhost:9906". The taskbar at the bottom includes icons for various applications like File Explorer, Google Chrome, and Microsoft Word.

Figure 16 Main English Webpage 2



Packet Switching vs Circuit Switching



Figure 17 Main English Webpage 3

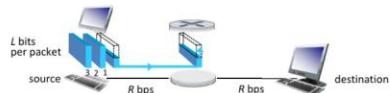
The team members' section presents individual profiles, including each member's name, student ID, academic major, completed projects, skills, and hobbies. Each profile is accompanied by a personal photo and is organized in a clear, structured layout, as shown in the figures above.



Packet Switching vs Circuit Switching

Packet switching and circuit switching are two different techniques for sending data across a network. Each method has its own way of handling data transmission, with unique advantages and disadvantages.

Packet Switching



In **packet switching**, data is broken into small packets that are transmitted independently across the network. Each packet may follow a different path depending on network conditions and is reassembled at the destination. This technique is commonly used in the Internet and other modern data networks.

Advantages:

1. Scalable and cost-effective.
2. Efficient use of bandwidth.
3. Better overall network efficiency.

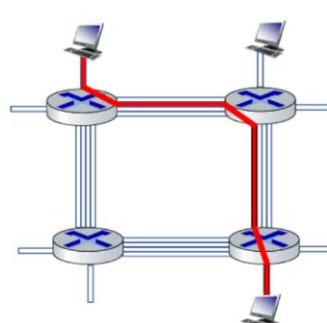
Disadvantages:



Figure 18 Main English Webpage 4



Circuit Switching



In **circuit switching**, a communication path is established between the sender and receiver before data transfer starts. This path stays active for the entire duration of the communication session. Circuit switching is mainly used in traditional telephone networks.

Advantages:



Figure 19 Main English Webpage 5

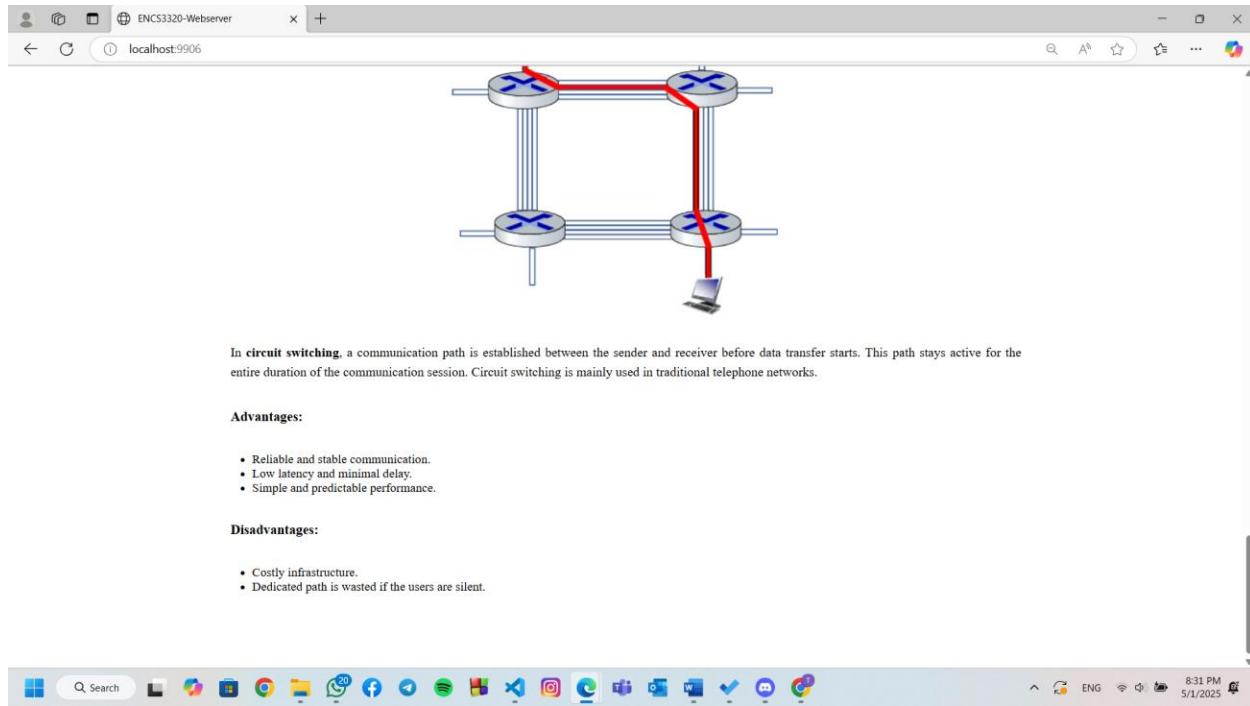


Figure 20 Main English Webpage 6

In the three figures above, a well-structured comparison between packet switching and circuit switching is presented as part of the first chapter topic from the textbook. Each technique is explained through clear headings, descriptive paragraphs, and relevant images in .jpg format. The packet switching section includes an ordered list highlighting its advantages and disadvantages, such as efficient bandwidth usage and potential packet loss. Similarly, the circuit switching section is supported by an unordered list, outlining its stable performance and higher cost. This section effectively uses headings, formatted text, and color styling to enhance readability and support the explanation visually and textually.

Main Arabic Webpage (main_ar.html):

This webpage serves as the main page of the Arabic version and contains the same content as the English main page, but presented in the Arabic language.

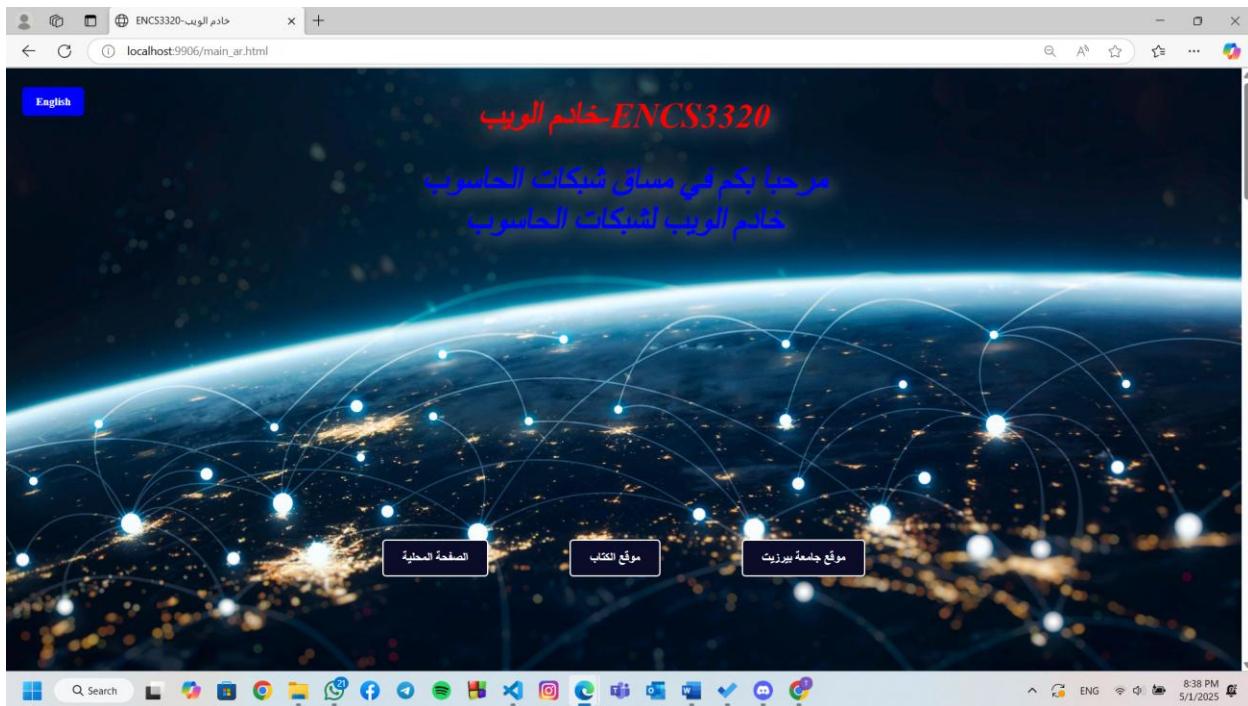
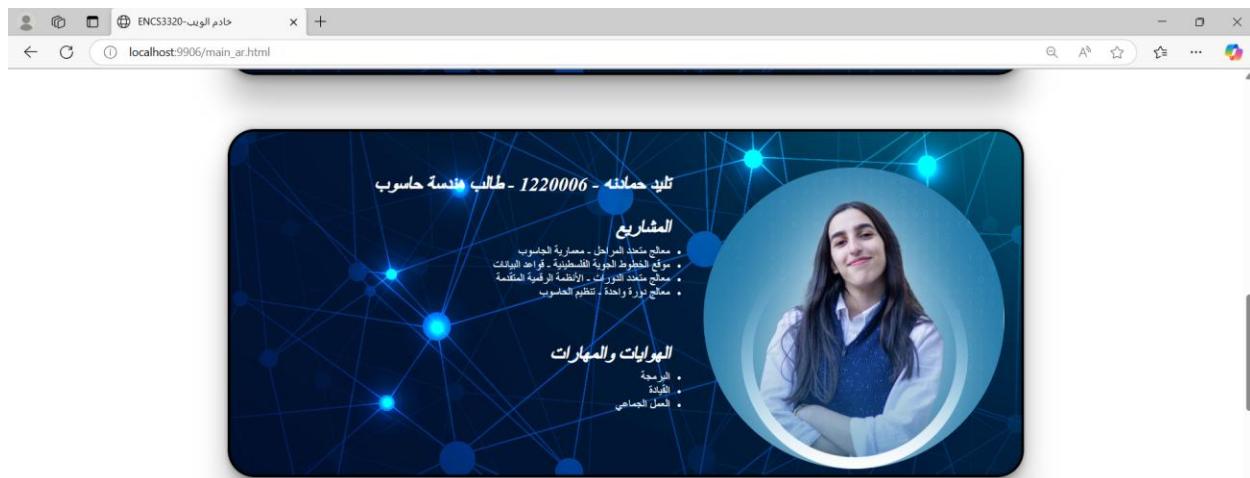


Figure 21 Main Arabic Webpage 1



Figure 22 Main Arabic Webpage 2



التحويل بالحزم مقابل التحويل الدائري

التحويل بالحزم والتحويل الدائري هما نتائج مختلفان لإرسال البيانات عبر الشبكة. كل طريقة لها طريقتها الخاصة في التعامل مع نقل البيانات، ولكن منها مزايا وعيوب مختلفة.



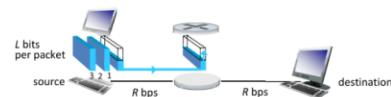
Figure 23 Main Arabic Webpage 3



التحويل بالحزم مقابل التحويل الدائري

التحويل بالحزم والتحويل الدائري هما تقنيتان مختلفتان لإرسال البيانات عبر الشبكة. كل طريقة لها طرقها الخاصة في التعامل مع نقل البيانات، وكل منها مزايا وعيوب مختلفة.

التحويل بالحزم



في التحويل بالحزم، يتم تقسيم البيانات إلى حزم صغيرة ترسل بشكل منفصل عبر الشبكة. قد تسلك كل حزمة مساراً مختلفاً بناءً على حالة الشبكة، وتعاد تجميعها عند الوصول إلى الوجهة. تستخدم هذه التقنية بشكل شائع في الانترنэт وشبكات البيانات الحديثة.

المزايا:

1. قليلة التوسيع وفعالة من حيث الكلفة.
2. استخدام قنال عرض واسع للطاقة الترددية.
3. كفاءة عالية للشبكة.

العيوب:

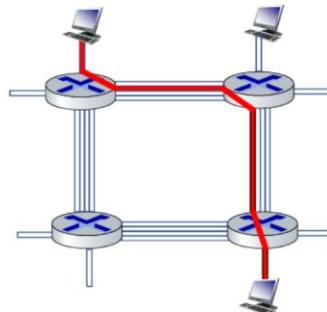
1. إمكانية فقدان الحزم.
2. وجود حمل إضافي بسبب رؤوس الحزم.



Figure 24 Main Arabic Webpage 4



التحويل الدائري



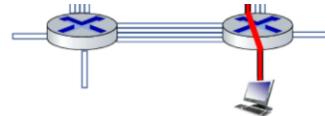
في التحويل الدائري، يتم إنشاء مسار اتصال بين المرسل والمسلك قبل بدء نقل البيانات، ويظل هذا المسار شططاً طوال فترة الاتصال. يستخدم التحويل الدائري بشكل أساسي في شبكات الهواتف التلفزيونية.

المزايا:

- اتصال متزامن (مستقر).
- زمن تأثير ملحوظ.
- إنارة سريعة ويمكن التحكم بها.



Figure 25 Main Arabic Webpage 5



في التحويل الداخلي، يتم إنشاء مسار اتصال بين المرسل والمستقبل قبل بدء نقل البيانات، ويظل هذا المسار نشطا طوال فترة الاتصال. يستخدم التحويل الداخلي بشكل أساسي في شبكات الهواتف التقليدية.

المزايا:

- اتصال متوقف ومستمر.
- زمن تأثير مختصر.
- نظام بسيط ويمكن تنفيذه.

العيوب:

- عملية تحويلية مكلفة.
- المسار المحجوز يهدى إذا لم يكن هناك نقل بيانات.



Figure 26 Main Arabic Webpage 6

English Local Webpage (mySite_1220006_en.html):

The Local Webpage, as shown in the figure below, is designed to provide users with supporting material related to the computer networking topic introduced in the main English page. It features a form where users can enter the name of an image or video file they wish to request. If the requested file is available on the server, it is returned and displayed accordingly. However, if the file is not found, the server responds with a "307 Temporary Redirect" status code. Based on the file type, the user is then redirected either to a Google Images search (for images) or to a video search (for videos), allowing them to find the content externally.

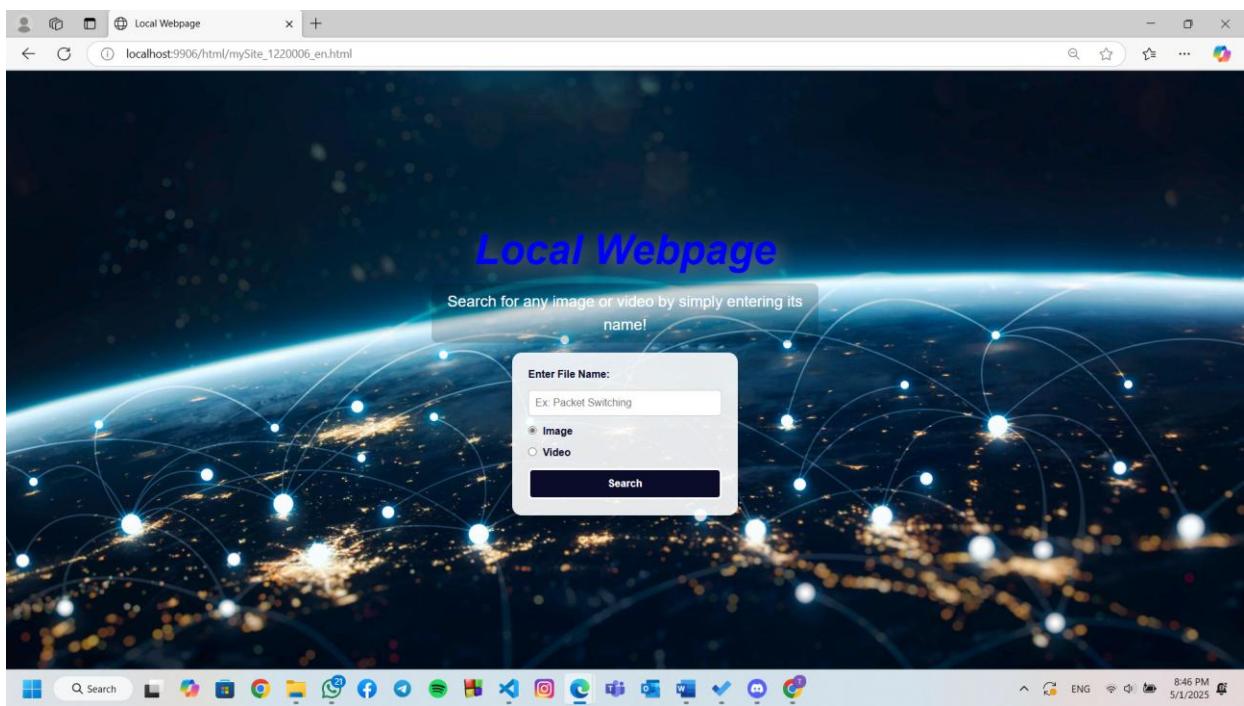


Figure 27 English Local Webpage

Arabic Local Webpage (mySite_1220006_ar.html):

The Arabic version of this page contains the same content and functionality, but is fully translated into Arabic.

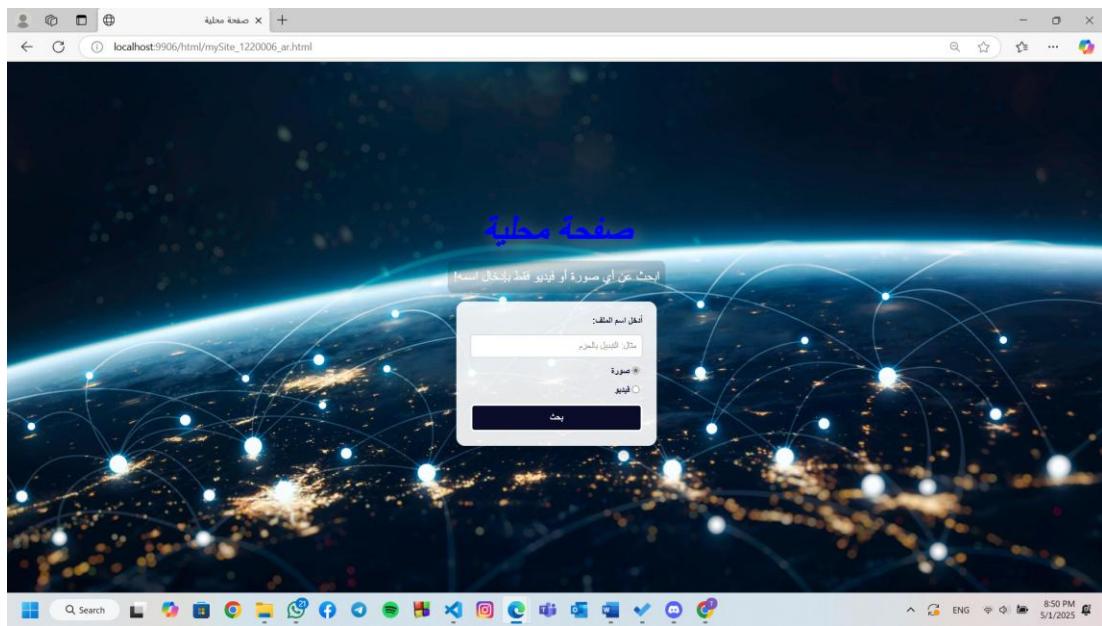
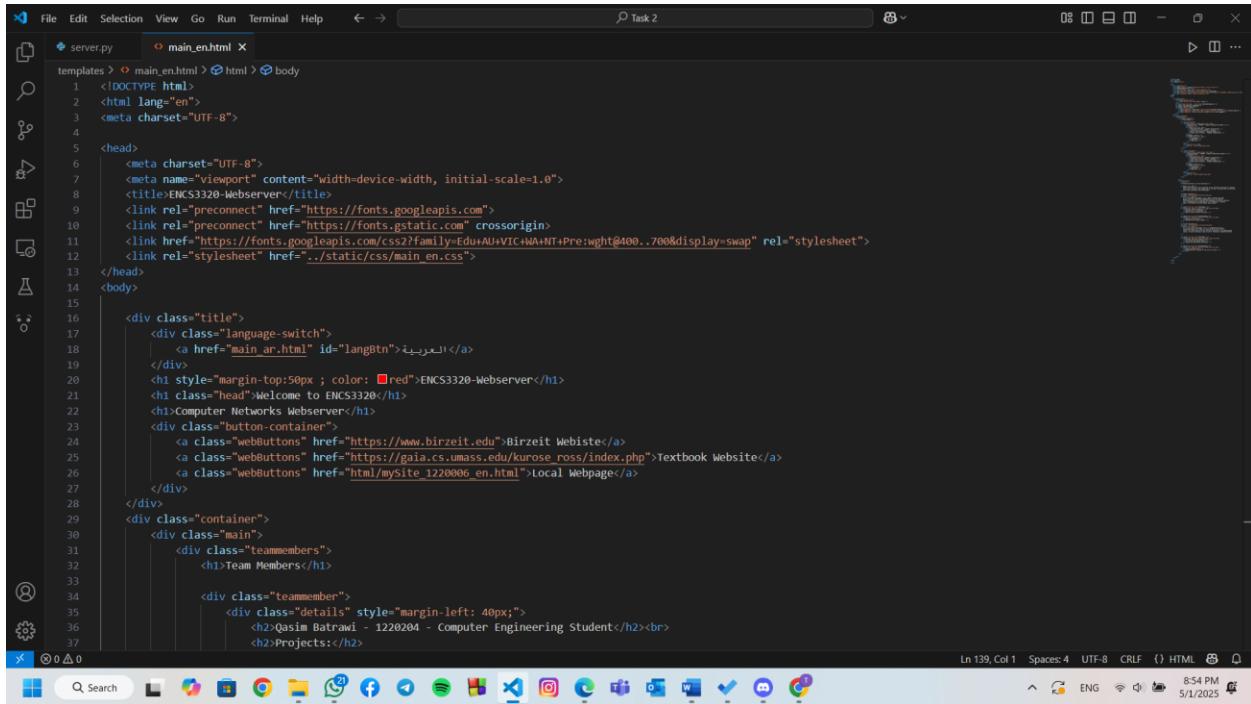


Figure 28 Arabic Local Webpage

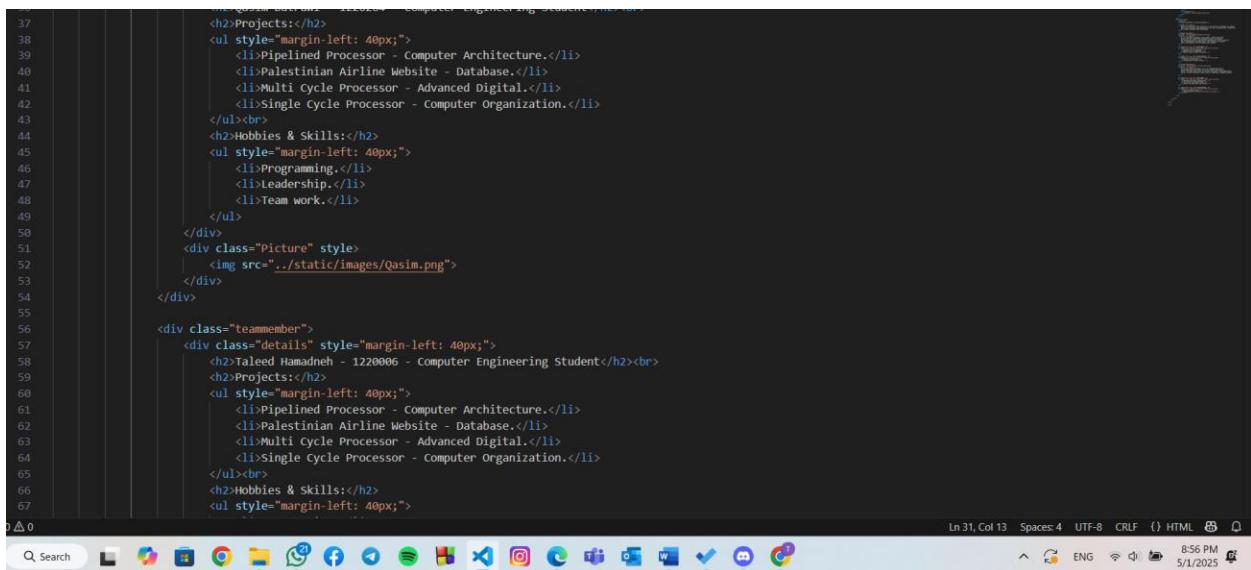
HTML & CSS

Main English Webpage HTML



```
File Edit Selection View Go Run Terminal Help Task 2
server.py main_en.html
templates > main_en.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="UTF-8">
5  <head>
6  <meta charset="UTF-8">
7  <meta name="viewport" content="width=device-width, initial-scale=1.0">
8  <title>ENCS3320-Webserver</title>
9  <link rel="preconnect" href="https://fonts.googleapis.com">
10 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
11 <link href="https://fonts.googleapis.com/css2?family=Edu+AU+VIC+WA+NT+Pre:wght@400..700&display=swap" rel="stylesheet">
12 <link rel="stylesheet" href="../static/css/main_en.css">
13 </head>
14 <body>
15
16 <div class="title">
17   <div class="language-switch">
18     <a href="#" id="langBtn" style="color: red;">English</a>
19   </div>
20   <h1 style="margin-top:50px ; color: red;">ENCS3320-Webserver</h1>
21   <h1 class="head">Welcome to ENCS3320</h1>
22   <h2>Computer Networks Webserver:</h2>
23   <div class="button-container">
24     <a class="webbuttons" href="http://www.birzeit.edu">Birthzeit Website</a>
25     <a class="webbuttons" href="https://gaia.cs.umass.edu/kurose_ross/index.php">Textbook Website</a>
26     <a class="webbuttons" href="http://mySite.1220006.en.html">Local Webpage</a>
27   </div>
28 </div>
29 <div class="container">
30   <div class="main">
31     <div class="teammembers">
32       <h3>Team Members:</h3>
33       <div class="teammember">
34         <div class="details" style="margin-left: 40px;">
35           <h2>Qasim Batrawi - 1220204 - Computer Engineering Student</h2><br>
36           <h2>Projects:</h2>
37           <ul style="margin-left: 40px;">
38             <li>Pipelined Processor - Computer Architecture.</li>
39             <li>Palestinian Airline Website - Database.</li>
40             <li>Multi Cycle Processor - Advanced Digital.</li>
41             <li>Single Cycle Processor - Computer Organization.</li>
42           </ul><br>
43           <h2>Hobbies & Skills:</h2>
44           <ul style="margin-left: 40px;">
45             <li>Programming.</li>
46             <li>Leadership.</li>
47             <li>Team work.</li>
48           </ul>
49         </div>
50       <div class="picture" style="text-align: center;">
51         
52       </div>
53     </div>
54   </div>
55 </div>
56 <div class="teammember">
57   <div class="details" style="margin-left: 40px;">
58     <h2>Taled Hamadneh - 1220006 - Computer Engineering Student</h2><br>
59     <h2>Projects:</h2>
60     <ul style="margin-left: 40px;">
61       <li>Pipelined Processor - Computer Architecture.</li>
62       <li>Palestinian Airline Website - Database.</li>
63       <li>Multi Cycle Processor - Advanced Digital.</li>
64       <li>Single Cycle Processor - Computer Organization.</li>
65     </ul><br>
66     <h2>Hobbies & Skills:</h2>
67     <ul style="margin-left: 40px;">
```

Figure 29 Main English Webpage HTML 1



```
File Edit Selection View Go Run Terminal Help Task 2
server.py main_en.html
templates > main_en.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="UTF-8">
5  <head>
6  <meta charset="UTF-8">
7  <meta name="viewport" content="width=device-width, initial-scale=1.0">
8  <title>ENCS3320-Webserver</title>
9  <link rel="preconnect" href="https://fonts.googleapis.com">
10 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
11 <link href="https://fonts.googleapis.com/css2?family=Edu+AU+VIC+WA+NT+Pre:wght@400..700&display=swap" rel="stylesheet">
12 <link rel="stylesheet" href="../static/css/main_en.css">
13 </head>
14 <body>
15
16 <div class="title">
17   <div class="language-switch">
18     <a href="#" id="langBtn" style="color: red;">English</a>
19   </div>
20   <h1 style="margin-top:50px ; color: red;">ENCS3320-Webserver</h1>
21   <h1 class="head">Welcome to ENCS3320</h1>
22   <h2>Computer Networks Webserver:</h2>
23   <div class="button-container">
24     <a class="webbuttons" href="http://www.birzeit.edu">Birthzeit Website</a>
25     <a class="webbuttons" href="https://gaia.cs.umass.edu/kurose_ross/index.php">Textbook Website</a>
26     <a class="webbuttons" href="http://mySite.1220006.en.html">Local Webpage</a>
27   </div>
28 </div>
29 <div class="container">
30   <div class="main">
31     <div class="teammembers">
32       <h3>Team Members:</h3>
33       <div class="teammember">
34         <div class="details" style="margin-left: 40px;">
35           <h2>Qasim Batrawi - 1220204 - Computer Engineering Student</h2><br>
36           <h2>Projects:</h2>
37           <ul style="margin-left: 40px;">
38             <li>Pipelined Processor - Computer Architecture.</li>
39             <li>Palestinian Airline Website - Database.</li>
40             <li>Multi Cycle Processor - Advanced Digital.</li>
41             <li>Single Cycle Processor - Computer Organization.</li>
42           </ul><br>
43           <h2>Hobbies & Skills:</h2>
44           <ul style="margin-left: 40px;">
45             <li>Programming.</li>
46             <li>Leadership.</li>
47             <li>Team work.</li>
48           </ul>
49         </div>
50       <div class="picture" style="text-align: center;">
51         
52       </div>
53     </div>
54   </div>
55 </div>
56 <div class="teammember">
57   <div class="details" style="margin-left: 40px;">
58     <h2>Taled Hamadneh - 1220006 - Computer Engineering Student</h2><br>
59     <h2>Projects:</h2>
60     <ul style="margin-left: 40px;">
61       <li>Pipelined Processor - Computer Architecture.</li>
62       <li>Palestinian Airline Website - Database.</li>
63       <li>Multi Cycle Processor - Advanced Digital.</li>
64       <li>Single Cycle Processor - Computer Organization.</li>
65     </ul><br>
66     <h2>Hobbies & Skills:</h2>
67     <ul style="margin-left: 40px;">
```

Figure 30 Main English Webpage HTML 2

```

68     </li>Programming.</li>
69     <li>Leadership.</li>
70     <li>Team work.</li>
71   </ul>
72 </div>
73 <div class="Picture">
74   
75 </div>
76 </div>
77 </div>
78 <div class="topic">
79   <br><br><br>
80   <h1>Packet Switching vs Circuit Switching</h1>
81
82   <p style="color: #black;">
83     Packet switching and circuit switching are two different techniques for sending
84     data across a network. Each method has its own way of handling data transmission,
85     with unique advantages and disadvantages.
86   </p>
87
88   <h2>Packet Switching</h2>
89   
90   <p style="color: #black;">
91     In <strong>packet switching</strong>, data is broken into small
92     packets that are transmitted independently across the network.
93     Each packet may follow a different path depending on network conditions
94     and is reassembled at the destination. This technique is commonly
95     used in the Internet and other modern data networks.
96   </p>
97
98 <h3 style="font-size: 25px">Advantages</h3>
99
100 <ol style="color: #black; margin-left: 40px; font-size:23px">
101   <li>Scalable and cost-effective.</li>
102   <li>Efficient use of bandwidth.</li>
103   <li>Better overall network efficiency.</li>
104 </ol>
105 <h3 style="font-size: 25px">Disadvantages:</h3>
106 <ol style="color: #black; margin-left: 40px; font-size:23px">
107   <li>Packet loss possible.</li>
108   <li>Overhead due to packet headers.</li>
109 </ol>
110 <h2>Circuit Switching</h2>
111 
112 <p style="color: #black;">
113     In <strong>circuit switching</strong>, a communication path is
114     established between the sender and receiver before data transfer
115     starts. This path stays active for the entire duration of the communication
116     session. Circuit switching is mainly used in traditional telephone networks.
117 </p>
118 <h3 style="font-size: 25px">Advantages:</h3>
119 <ul style="color: #black; margin-left: 40px; font-size:23px">
120   <li>Reliable and stable communication.</li>
121   <li>Low latency and minimal delay.</li>
122   <li>Simple and predictable performance.</li>
123 </ul>
124 <h3 style="font-size: 25px">Disadvantages:</h3>
125 <ul style="color: #black; margin-left: 40px; font-size:23px">
126   <li>Costly infrastructure.</li>
127   <li>Dedicated path is wasted if the users are silent.</li>
128 </ul><br><br>
129 </div>
130 </div>

```

Figure 31 Main English Webpage HTML 3

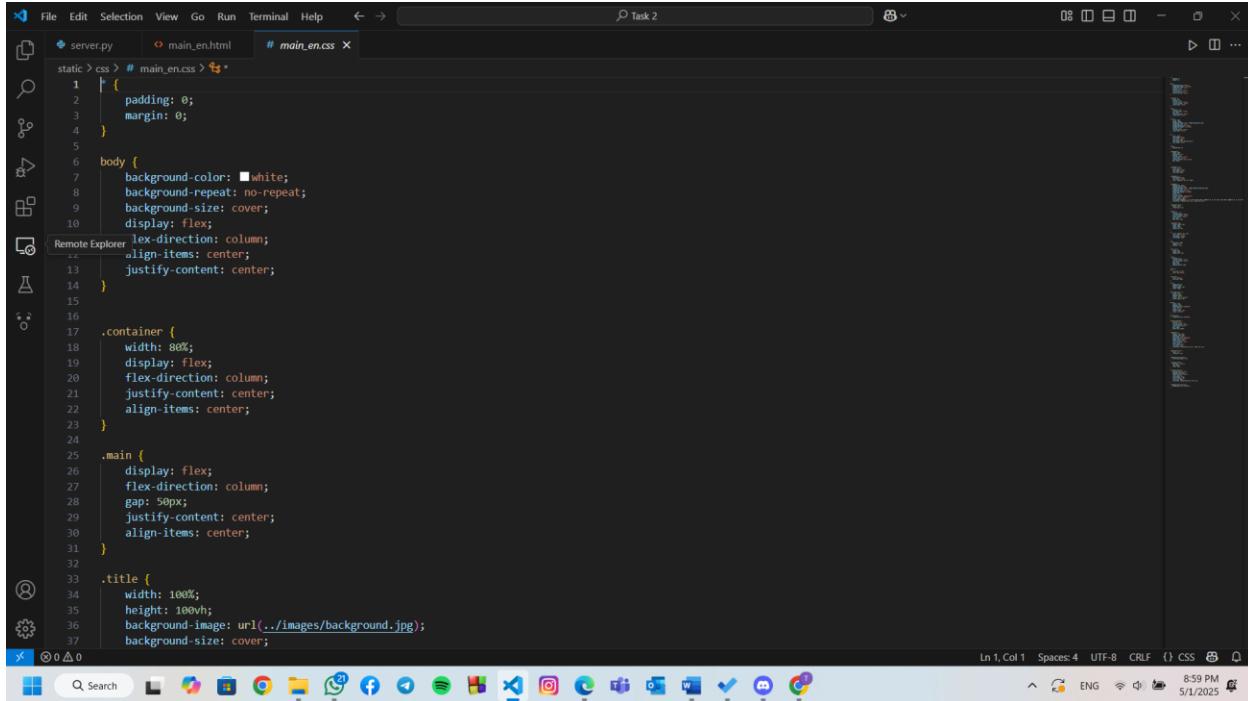
```

99 <h3 style="font-size: 25px">Advantages:</h3>
100 <ol style="color: #black; margin-left: 40px; font-size:23px">
101   <li>Scalable and cost-effective.</li>
102   <li>Efficient use of bandwidth.</li>
103   <li>Better overall network efficiency.</li>
104 </ol>
105 <h3 style="font-size: 25px">Disadvantages:</h3>
106 <ol style="color: #black; margin-left: 40px; font-size:23px">
107   <li>Packet loss possible.</li>
108   <li>Overhead due to packet headers.</li>
109 </ol>
110 <h2>Circuit Switching</h2>
111 
112 <p style="color: #black;">
113     In <strong>circuit switching</strong>, a communication path is
114     established between the sender and receiver before data transfer
115     starts. This path stays active for the entire duration of the communication
116     session. Circuit switching is mainly used in traditional telephone networks.
117 </p>
118 <h3 style="font-size: 25px">Advantages:</h3>
119 <ul style="color: #black; margin-left: 40px; font-size:23px">
120   <li>Reliable and stable communication.</li>
121   <li>Low latency and minimal delay.</li>
122   <li>Simple and predictable performance.</li>
123 </ul>
124 <h3 style="font-size: 25px">Disadvantages:</h3>
125 <ul style="color: #black; margin-left: 40px; font-size:23px">
126   <li>Costly infrastructure.</li>
127   <li>Dedicated path is wasted if the users are silent.</li>
128 </ul><br><br>
129 </div>
130 </div>

```

Figure 32 Main English Webpage HTML 4

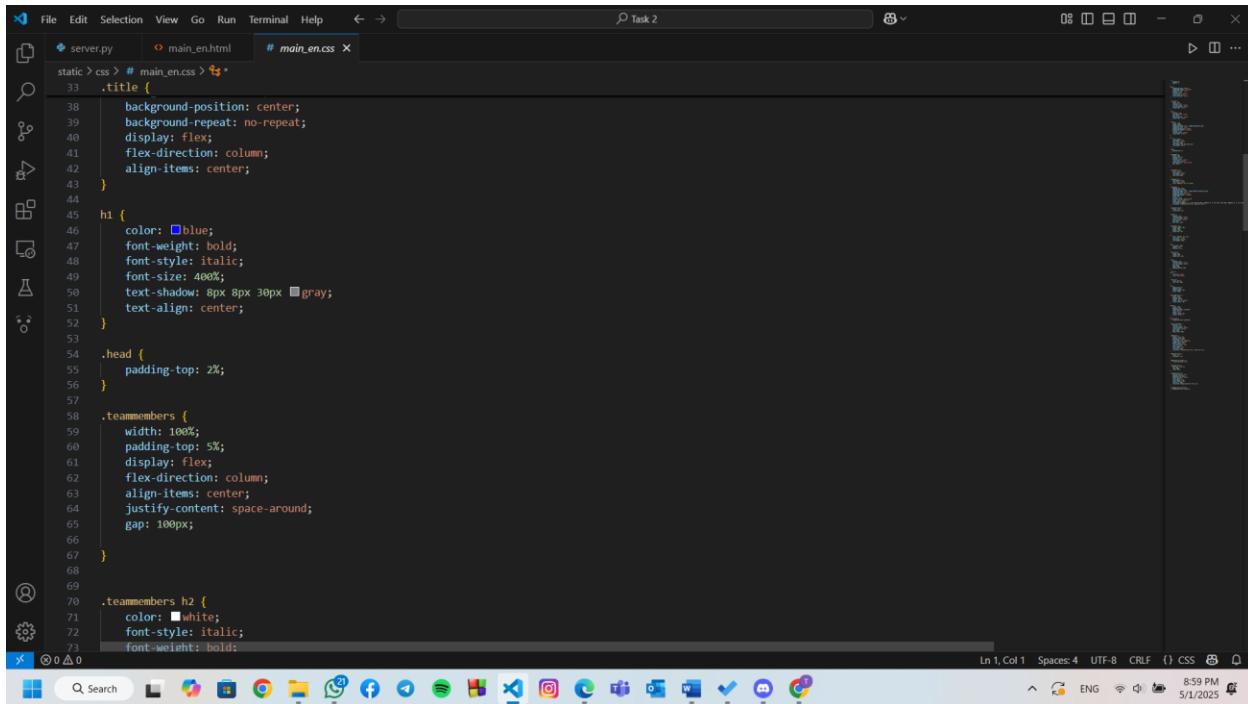
Main English Webpage CSS



A screenshot of a code editor window titled "main_en.css". The code is as follows:

```
static > css > # main_en.css > # main_en.css *
1  {
2      padding: 0;
3      margin: 0;
4  }
5
6  body {
7      background-color: white;
8      background-repeat: no-repeat;
9      background-size: cover;
10     display: flex;
11     flex-direction: column;
12     align-items: center;
13     justify-content: center;
14 }
15
16 .container {
17     width: 80%;
18     display: flex;
19     flex-direction: column;
20     justify-content: center;
21     align-items: center;
22 }
23
24 .main {
25     display: flex;
26     flex-direction: column;
27     gap: 50px;
28     justify-content: center;
29     align-items: center;
30 }
31
32
33 .title {
34     width: 100%;
35     height: 100vh;
36     background-image: url(..../images/background.jpg);
37     background-size: cover;
38 }
```

Figure 33 Main English Webpage CSS I



A screenshot of a code editor window titled "main_en.css". The code continues from the previous snippet:

```
33 .title {
34     background-position: center;
35     background-repeat: no-repeat;
36     display: flex;
37     flex-direction: column;
38     align-items: center;
39 }
40
41 h1 {
42     color: blue;
43     font-weight: bold;
44     font-style: italic;
45     font-size: 400%;
46     text-shadow: 8px 8px 30px gray;
47     text-align: center;
48 }
49
50 .head {
51     padding-top: 2%;
52 }
53
54 .teammembers {
55     width: 100%;
56     padding-top: 5%;
57     display: flex;
58     flex-direction: column;
59     align-items: center;
60     justify-content: space-around;
61     gap: 100px;
62 }
63
64
65 .teammembers h2 {
66     color: white;
67     font-style: italic;
68     font-weight: bold;
69 }
70
71
72 }
```

Figure 34 Main English Webpage CSS 2

A screenshot of a code editor window titled "Task 2". The editor shows two tabs: "server.py" and "# main_en.css". The "# main_en.css" tab is active and displays the following CSS code:

```
static > css > # main_en.css > # main_en.css *
70 .teammembers h2 {
71     font-size: 200%;
72 }
73 .teammembers a {
74     font-weight: bold;
75     font-size: x-large;
76     color: #rgba(14, 43, 97, 0.848);
77 }
78 .teammember {
79     padding-top: 10px;
80     padding-bottom: 10px;
81     background-image: url(..../images/member_background.jpg);
82     background-size: cover;
83     background-position: center;
84     background-repeat: no-repeat;
85     width: 80%;
86     display: flex;
87     justify-content: space-around;
88     border: 5px solid black;
89     border-radius: 50px;
90     box-shadow: #rgba(0, 0, 0.25) 0px 54px 55px, #rgba(0, 0, 0.12) 0px -12px 30px, #rgba(0, 0, 0.12) 0px 4px 6px, #rgba(0, 0, 0, 0.17) 0px 12px 13px, #rgba(0, 0, 0.17) 0px 24px 25px;
91     transition: transform 0.5s ease, scale 0.5s ease;
92 }
93 .teammember:hover {
94     scale: 1.05;
95     transition: 0.5s;
96 }
97 .details {
98     display: flex;
99     flex-direction: column;
100    justify-content: center;
101    align-content: center;
102    gap: 10px;
103 }
104 .teammember img {
105     border-radius: 50%;
106     width: 120px;
107     height: 50vh;
108     margin-top: 10px;
109 }
110 p {
111     color: #rgb(37, 83, 124);
112     font-size: x-large;
113     text-align: justify;
114     line-height: 150%;
115 }
116 img {
117     max-width: 100%;
118     height: auto;
119     display: block;
120 }
121 .topic img {
122     width: 50%;
123     height: 50%;
124     margin-left: 25%;
125 }
126 .topic {
127     display: flex;
128     flex-direction: column;
129     justify-content: center;
130     gap: 50px;
131     width: 80%;
```

Figure 35 Main English Webpage CSS 3

A screenshot of a code editor window titled "Task 2". The editor shows two tabs: "server.py" and "# main_en.css". The "# main_en.css" tab is active and displays the following CSS code:

```
static > css > # main_en.css > # main_en.css *
104 .details {
105     gap: 10px;
106     font-size: 18px;
107 }
108 .teammember img {
109     border-radius: 50%;
110     width: 120px;
111     height: 50vh;
112     margin-top: 10px;
113 }
114 p {
115     color: #rgb(37, 83, 124);
116     font-size: x-large;
117     text-align: justify;
118     line-height: 150%;
119 }
120 img {
121     max-width: 100%;
122     height: auto;
123     display: block;
124 }
125 .topic img {
126     width: 50%;
127     height: 50%;
128     margin-left: 25%;
129 }
130 .topic {
131     display: flex;
132     flex-direction: column;
133     justify-content: center;
134     gap: 50px;
135     width: 80%;
```

Figure 36 Main English Webpage CSS 4

A screenshot of a code editor window titled "Task 2". The editor shows two files: "server.py" and "# main_en.css". The "# main_en.css" file contains the following CSS code:

```
static > css > # main_en.css > # main_en.css
139 .topic {
140     padding-bottom: 50px;
141 }
142 .topic ol,
143     ul {
144     font-size: larger;
145     color: #aliceblue;
146 }
147 .topic h2 {
148     color: black;
149     font-size: 35px;
150 }
151 form {
152     margin-top: 5%;
153     padding: 20px;
154     border-radius: 8px;
155     width: 300px;
156 }
157 input[type="text"] {
158     width: 100%;
159     padding: 10px;
160     margin: 10px 0;
161     border: 1px solid #ccc;
162     border-radius: 5px;
163 }
164 button {
165     width: 100%;
166     padding: 10px;
167     background-color: #4CAF50;
168     color: white;
169     border: none;
170     border-radius: 5px;
171 }
172 
```

Figure 37 Main English Webpage CSS 5

A screenshot of a code editor window titled "Task 2". The editor shows two files: "server.py" and "# main_en.css". The "# main_en.css" file contains the following CSS code:

```
static > css > # main_en.css > # main_en.css
174 button {
175     border-radius: 5px;
176     cursor: pointer;
177 }
178 button:hover {
179     background-color: #45a049;
180 }
181 .button-container {
182     display: flex;
183     flex-direction: row;
184     justify-content: center;
185     align-items: center;
186     gap: 150px;
187     margin-top: 550px;
188 }
189 .webButtons {
190     border: 3px solid;
191     padding: 18px 35px;
192     text-align: center;
193     background-color: #00bcd4;
194     border-color: white;
195     border-radius: 7px;
196     text-decoration: none;
197     color: white;
198     font-weight: bold;
199     font-size: 24px;
200     transition: transform 0.5s ease, scale 0.5s ease;
201 }
202 .webButtons:hover {
203     scale: 1.1;
204     transform: none;
205 }
```

Figure 38 Main English Webpage CSS 6

A screenshot of a code editor window titled "Task 2". The editor displays a block of CSS code. The code includes various styles for buttons, a language switcher, and other UI elements. The interface shows a sidebar with icons for file operations, a top bar with tabs like "File", "Edit", "Selection", etc., and a status bar at the bottom showing "Ln 1, Col 1" and "Spaces: 4".

```
210 .webButtons:hover {  
211     scale: 1.1;  
212     transform: none;  
213 }  
214  
215 .webButtons:not(:hover) {  
216     transition-delay: 0.1s;  
217 }  
218  
219 .language-switch {  
220     position: absolute;  
221     top: 50px;  
222     right: 30px;  
223 }  
224  
225 .language-switch a {  
226     text-decoration: none;  
227     padding: 15px 25px;  
228     background-color: #blue;  
229     color: white;  
230     font-weight: bold;  
231     border-radius: 8px;  
232     font-size: 20px;  
233     transition: background-color 0.3s ease;  
234 }  
235  
236 .language-switch a:hover {  
237     background-color: ##00c2a;  
238 }  
239
```

Figure 39 Main English Webpage CSS 7

English Local Webpage HTML

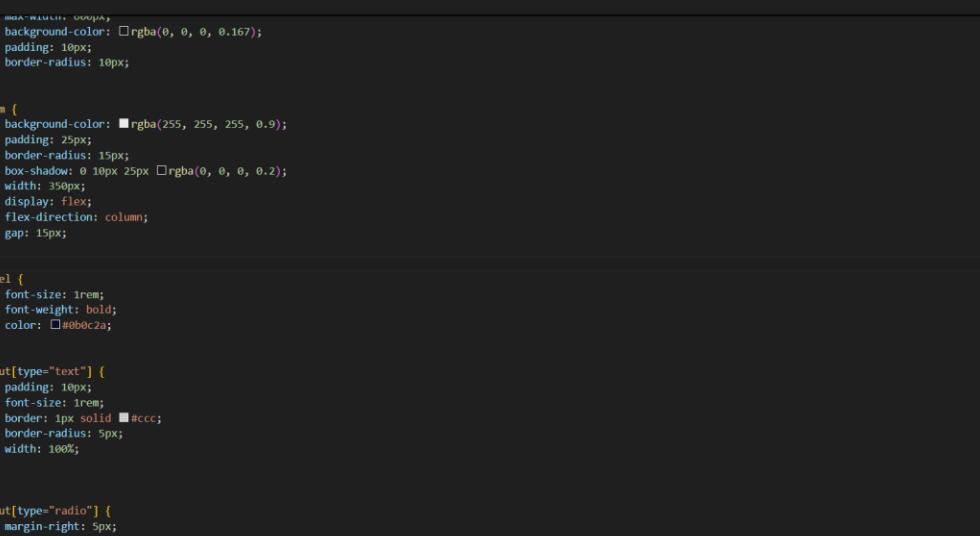
A screenshot of a code editor window titled "Task 2". The editor displays a block of HTML code. The code includes a server script ("server.py") and two HTML files ("main_en.html" and "mySite_1220006_en.html"). The "mySite_1220006_en.html" file contains a search form for files. The interface shows a sidebar with icons for file operations, a top bar with tabs like "File", "Edit", "Selection", etc., and a status bar at the bottom showing "Ln 1, Col 1" and "Spaces: 4".

```
1 <!DOCTYPE html>  
2 <html lang="en">  
3  
4     <head>  
5         <meta charset="UTF-8">  
6         <meta name="viewport" content="width=device-width, initial-scale=1.0">  
7         <title>Local Webpage</title>  
8         <link rel="stylesheet" href="../static/css/mySite_1220006_en.css">  
9     </head>  
10  
11    <body class="body">  
12        <h1>Local Webpage</h1>  
13        <p>Search for any image or video by simply entering its name!</p>  
14  
15        <form method="GET" action="request_handler">  
16            <label for="material">Enter File Name:</label>  
17            <input type="text" id="material" name="material" placeholder="Ex: Packet Switching" required>  
18            <label>  
19                <input type="radio" name="type" value="image" checked> Image  
20            </label>  
21            <label>  
22                <input type="radio" name="type" value="video"> Video  
23            </label>  
24            <button type="submit">Search</button>  
25        </form>  
26    </body>  
27  
28 </html>
```

Figure 40 English Local Webpage HTML

English Local Webpage CSS

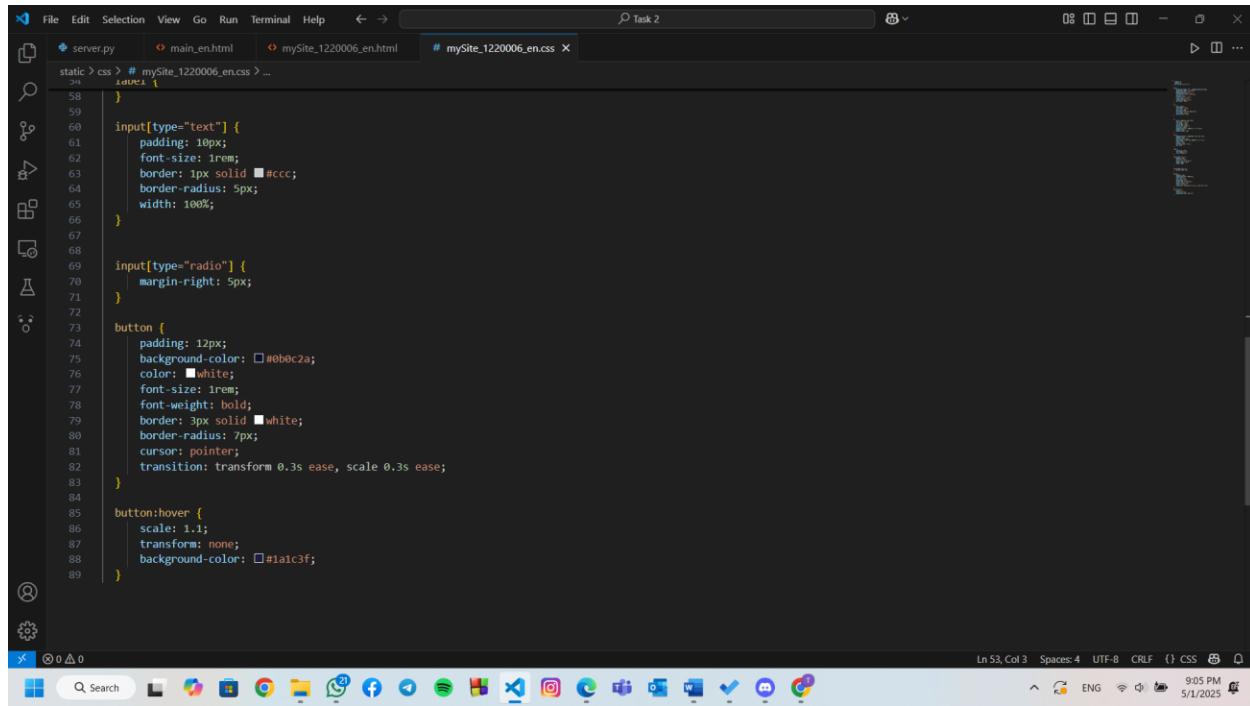
Figure 41 English Local Webpage CSS 1



A screenshot of a Windows desktop environment. The main window is a code editor displaying a CSS file named 'mySite_1220006_en.css'. The code contains styles for a form, its label, and input fields. The taskbar at the bottom shows icons for File Explorer, Edge browser, FileZilla, and other system tools. The system tray indicates the date as 5/1/2025.

```
File Edit Selection View Go Run Terminal Help ← → Task 2  
server.py main_en.html mySite_1220006_en.html # mySite_1220006_en.css  
static > css > # mySite_1220006_en.css ...  
31 p {  
32     background-color: #fff;  
33     background-color: #fff;  
34     background-color: #fff;  
35     background-color: #fff;  
36     background-color: #fff;  
37     background-color: #fff;  
38     background-color: #fff;  
39     background-color: #fff;  
40     background-color: #fff;  
41 }  
42  
43 form {  
44     background-color: #fff;  
45     padding: 25px;  
46     border-radius: 15px;  
47     box-shadow: 0 10px 25px #000;  
48     width: 350px;  
49     display: flex;  
50     flex-direction: column;  
51     gap: 15px;  
52 }  
53  
54 label {  
55     font-size: 1rem;  
56     font-weight: bold;  
57     color: #000;  
58 }  
59  
60 input[type="text"] {  
61     padding: 10px;  
62     font-size: 1rem;  
63     border: 1px solid #ccc;  
64     border-radius: 5px;  
65     width: 100%;  
66 }  
67  
68  
69 input[type="radio"] {  
70     margin-right: 5px;  
71 }  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
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  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2049
```

Figure 42 English Local Webpage CSS 2

A screenshot of a code editor window titled "Task 2". The window shows a file named "mySite_1220006_en.css" with the following CSS code:

```
static > css > # mySite_1220006_en.css > ...
58 }
59
60 input[type="text"] {
61   padding: 10px;
62   font-size: 1rem;
63   border: 1px solid #ccc;
64   border-radius: 5px;
65   width: 100%;
66 }
67
68 input[type="radio"] {
69   margin-right: 5px;
70 }
71
72 button {
73   padding: 12px;
74   background-color: #00b0c2a;
75   color: white;
76   font-size: 1rem;
77   font-weight: bold;
78   border: 3px solid white;
79   border-radius: 7px;
80   cursor: pointer;
81   transition: transform 0.3s ease, scale 0.3s ease;
82 }
83
84 button:hover {
85   scale: 1.1;
86   transform: none;
87   background-color: #1a1c3f;
88 }
```

The status bar at the bottom indicates "Ln 53, Col 3" and "Spaces: 4".

Ln 53, Col 3 Spaces: 4 UTF-8 CRLF (CSS 9:05 PM 5/1/2025

Figure 43 English Local Webpage CSS 3

The HTML and CSS structures illustrated in the figures above represent the design and functionality of all the English pages. Each corresponding Arabic page mirrors the same layout, styling, and features, with the content accurately translated into Arabic to ensure consistency across both language versions.

Python Server Code

```
❶ server.py > ...
1  #Taleed Hamadneh 1220006
2  #Qasim Batrawi 1220204
3  from socket import *
4  import os
5  serverPort = 9906
6  #TCP socket
7  serverSocket = socket(AF_INET, SOCK_STREAM)
8  serverSocket.bind(('', serverPort))
9  # listen to the HTTP request on port 9906
10 serverSocket.listen(30) # up to 30 clients
11 print('Web Server is listening on port 9906 ...')
12
13 def send_response(response_code, response_type):
14     status_line = ""
15     if int(response_code) == 200:
16         status_line = "HTTP/1.1 200 OK\r\n"
17     elif int(response_code) == 404:
18         status_line = "HTTP/1.1 404 Not Found\r\n"
19     elif int(response_code) == 307:
20         status_line = "HTTP/1.1 307 Temporary Redirect\r\n"
21     else:
22         status_line = f"HTTP/1.1 {response_code} Unknown\r\n"
23
24     connectionSocket.send(status_line.encode())
25     res = "Content-Type: " + response_type + "; charset=utf-8\r\n"
26     connectionSocket.send(res.encode())
27     connectionSocket.send("\r\n".encode())
28
29     print(f"Response Sent: {status_line.strip()} | Client IP: {client_ip} | Client Port: {client_port} | Server Port: {serverPort}")
30
```

Figure 44 Python Server Code 1

As shown in the segment of the code above, a TCP socket is created using Python's socket module and is bound to port 9906. The server listens for incoming HTTP requests, allowing up to 30 clients at once. The function `send_response` is designed to handle the initial part of the HTTP response. Depending on the status code passed (200, 404, or 307), it sends the appropriate HTTP response line to the client. It then adds the Content-Type header based on the specified resource type (such as `text/html` or `image/jpeg`) and sends a blank line to indicate the end of the header section. This ensures proper formatting of the HTTP response so that browsers can correctly interpret the server's replies.

```

#when the requested file is unavailable
def Error(ip,port):
    send_response(404,'text/html')
    error = ('<!DOCTYPE html>'
             '<html lang="en">'
             '<style>'
             '*{text-align: center;padding:10px;}'
             'h1{font-size:50px;}'
             'p{font-size:20px;}'
             '</style>'
             '<head>'
             '<meta charset="UTF-8">'
             '<meta name="viewport" content="width=device-width, initial-scale=1.0">'
             '<title>Error 404</title>'
             '</head>'
             '<body>'
             '<h1 style="color: red;">The file is not found!</h1>'
             '<p>IP Address: ' + str(ip) + ', Port Number: ' + str(port) + '</p>'
             '</body></html>')
    connectionSocket.send(error.encode())

```

Figure 45 Python Server Code 2

As shown in the code segment above, the Error function is called when the requested file is not found on the server. In this case, the server sends a custom error webpage as a response. The page includes the title "**Error 404**" on the browser tab and shows the message "**The file is not found!**" in red on the page itself. It also displays the IP address and port number of the client who made the request. The HTML content of this error page is written directly inside the Python code as a string and sent to the client through the socket connection.

As shown in the code segment below, this part of the server is responsible for continuously listening and handling incoming client requests. It begins by organizing the available server files into categories such as HTML, CSS, images, and videos, based on their respective folders. Within an infinite loop, the server accepts a connection from a client, capturing their IP address and port number. It then receives the HTTP request sent by the client and attempts to extract the requested file name from the first line of the request message. This information is

also printed to the console for monitoring purposes. In case the request is malformed or an unexpected error occurs during this process, the server handles it gracefully by printing "BAD REQUEST" and waiting for the next client, ensuring that the server remains operational without interruption.

```
55 while True: #ALWAYS ON SERVER
56     # Show all the files (that are available in the server) and separate them based on their types
57     files = os.listdir('.')
58     temp_files = []
59     for f in files:
60         if os.path.isfile(f):
61             temp_files.append(f)
62     files = temp_files
63
64     html = os.listdir('./html')
65     temp_html = []
66     for f in html:
67         if os.path.isfile(os.path.join('./html', f)):
68             temp_html.append(f)
69     html = temp_html
70
71     css = os.listdir('./css')
72     temp_css = []
73     for f in css:
74         if os.path.isfile(os.path.join('./css', f)):
75             temp_css.append(f)
76     css = temp_css
77
78     images = os.listdir('./imgs')
79     normal_images = []
80     for f in images:
81         if os.path.isfile(os.path.join('./imgs', f)):
82             normal_images.append(f)
83     images = normal_images
84
85     videos = os.listdir('./videos')
```

Figure 46 Python Server Code 3

```

80     #connection is established
81     connectionSocket, addr = serverSocket.accept()
82     client_ip = addr[0]
83     client_port = addr[1]
84     print('Got connection from', "IP: " + client_ip + ", Port: " + str(client_port) + ", Server Port: 9906")
85
86     sentence = connectionSocket.recv(4096).decode() #4096 is the maximum number of bytes to receive from the socket at once
87
88     try:
89         #get the request line from the request (This line has the requested object)
90         sen = sentence.split('\n')[0]
91         request = sen.split(' ')[1]
92         print("The Request is : " + request) # print http request
93         #print(sentence)
94     except :
95         print("BAD REQUEST")
96         continue
97

```

Figure 47 Python Server Code 4

```

105    # Main_English request
106    if request == "/" or request == "/index.html" or request == "/main_en.html" or request == "/en":
107
108        if 'main_en.html' in html: # check if the file exists
109            with open('../html/main_en.html','r', encoding='utf-8') as Main_English: # encoding is needed to show arabic words
110                Main_English = Main_English.read()
111                send_response(200,'text/html') # 200 -> OK , the content is html
112                connectionSocket.send(Main_English.encode()) # send the file to the client through the socket (encoding from strings to byte)
113            else:
114                Error(client_ip, client_port) # 404 ERROR
115
116    # Main Arabic request
117    elif request == "/ar" or request == "/main_ar.html":
118
119        if 'main_ar.html' in html: # check if the file exists
120            with open('../html/main_ar.html','rb') as Main_Arabic:
121                Main_Arabic = Main_Arabic.read()
122                send_response(200,'text/html')
123                connectionSocket.send(Main_Arabic)
124            else:
125                Error(client_ip, client_port)

```

Figure 48 Python Server Code 5

This part of the code handles HTTP requests for the main English and Arabic pages. As shown in the segment above, if the request matches common paths like /, /index.html, /main_en.html, or /en, the server interprets it as a request for the English main page. It first checks if the main_en.html file exists. If so, it opens the file using UTF-8 encoding (to correctly display Arabic content if present), reads its content, and sends it back to the client with an HTTP 200 OK response and a content type of HTML. If the file is not found, the server responds with a 404 error using the custom Error function. Similarly, if the request is for the Arabic version (e.g., /ar or /main_ar.html), the server attempts to open and return main_ar.html.

The file is read in binary mode and sent directly to the client, also with a 200 OK response. If this file does not exist either, the same error handling procedure is applied.

```

128 # if this statement (/request_handler?material) is in the url, then the user submitted an image/video name in the html form
129 # request: /html/request_handler?material=example&type=image
130 elif request.startswith("/html/request_handler?material"):
131     input = request.split('=')[1].split('&')[0] # to get the input from the user
132     type = request.split('=')[2] # to get the type image/video
133
134     if type == 'image':
135         object1 = input + '.jpg' # to search for the image with jpg extension
136         object2 = input + '.png' # to search for the image with png extension
137
138         if object1 in images:
139             path = './imgs/' + object1
140             with open(path, 'rb') as image:
141                 image = image.read()
142             send_response(200, 'image/jpg')
143             connectionSocket.send(image)
144         elif object2 in images:
145             path = './imgs/' + object2
146             with open(path, 'rb') as image:
147                 image = image.read()
148             send_response(200, 'image/png')
149             connectionSocket.send(image)
150     else:
151         connectionSocket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
152         connectionSocket.send('Content-Type: text/html; charset=utf-8\r\n'.encode())
153         type = type.replace(" ", "+")
154         location = "Location: http://www.google.com/search?q=" + input + "&udm=2\r\n"
155         connectionSocket.send(location.encode())
156         connectionSocket.send('\r\n'.encode())
157         print(f"Response Sent: HTTP/1.1 307 Temporary Redirect | Client IP: {client_ip} | Client Port: {client_port} | Server Port: {serverPort}\n")
158         print("Redirect to Google\r\n")

```

Figure 49 Python Server Code 6

This part of the code handles user requests for images or videos that are submitted through the form on the Local webpage. When the URL contains /request_handler?material, the server understands that the user is asking for a specific file. The URL is typically in the format /request_handler?material=FileName&type=image (or type=video). The server extracts the file name and its type by splitting the request at the (=) signs and (&). If the requested type is an image, the server checks whether the image file (with a .jpg or .png extension) exists in the images folder. If it finds the image, it sends it back with an HTTP 200 OK response and the appropriate content type (like image/png or image/jpg). However, if the image is not found on the server, the server sends a 307 Temporary Redirect response, guiding the browser to automatically search for the image on Google Images. This is done by

sending a redirect location like Location: http://www.google.com/search?q=FileName&udm=2, where FileName is the user's input, and udm=2 is used to display image results. This ensures that even if the file isn't available on the server, the user is still directed to helpful results.

```
159     else:
160         object = input + '.mp4'
161         if object in videos:
162             path = './videos/' + object
163             with open(path,'rb') as video:
164                 video = video.read()
165             send_response(200, 'video/mp4')
166             connectionSocket.send(video)
167
168     else:
169         connectionSocket.send("HTTP/1.1 307 Temporary Redirect\r\n".encode())
170         connectionSocket.send('Content-Type: text/html; charset=utf-8\r\n'.encode())
171         location = "Location:https://www.youtube.com/results?search_query=" + input +"\r\n"
172         connectionSocket.send(location.encode())
173         connectionSocket.send('\r\n'.encode())
174         print(f"Response Sent: HTTP/1.1 307 Temporary Redirect | Client IP: {client_ip} | Client Port: {client_port} | Server Port: {serverPort}")
175         print("Redirect to YouTube\r\n")
176
```

Figure 50 Python Server Code 7

This part of the code handles video requests. When a user asks for a video, the server checks if the video exists in the videos folder with a .mp4 extension. If it finds the video, it sends the video back to the user with a "200 OK" response.

If the video isn't found, the server sends a "307 Temporary Redirect" response, telling the browser to go to YouTube and search for the video. The server creates a YouTube search link based on what the user entered and sends it to the browser, redirecting the user to YouTube.

```

178     # other html file request
179     # request: /html/mySite_1220006_ar.html
180     elif request.startswith("/html/"):
181         html_file_name = request.split('/')[2] # get the html file name (for example: mySite_1220006_ar.html)
182         if html_file_name in html:
183             path = "./html/" + html_file_name
184             with open(path,'rb') as Local_Webpage_Arabic:
185                 Local_Webpage_Arabic = Local_Webpage_Arabic.read()
186                 send_response(200,'text/html')
187                 connectionSocket.send(Local_Webpage_Arabic)
188             else:
189                 Error(client_ip, client_port)
190
191     # css file request
192     # request: /css/example.css
193     elif request.startswith("/css/"):
194         css_file_name = request.split('/')[2] # get the image name (for example: main.css)
195         if css_file_name in css: # check if the file exists
196             path = './css/' + css_file_name # path of the css file
197             with open(path,'r') as cssFile:
198                 cssFile = cssFile.read()
199                 send_response(200,'text/css') # 200 -> OK , the content is css
200                 connectionSocket.send(cssFile.encode()) # send the file to the client through the socket (encoding from strings to byte)
201             else:
202                 Error(client_ip, client_port)

```

Figure 51 Python Server Code 8

This part of the code handles requests for other HTML and CSS files. For HTML file requests, the server checks if the requested file exists in the html directory. If the file is found, it is read and sent back to the client with a "200 OK" response. If the file is not found, a "404 Not Found" error page is sent instead. Similarly, for CSS file requests, the server checks if the requested CSS file is available in the css directory. If the file exists, it is read and sent to the client with a "200 OK" response and a content type of "text/css". If the CSS file is not found, the server responds with a "404 Not Found" error page.

```

204     # images request
205     # request: /imgs/example.png
206     elif request.startswith("/imgs/"):
207         image_file_name = request.split('/')[2] # get the image name (for example: background.png)
208         if image_file_name in images:
209             path = './imgs/' + image_file_name # path of the image
210             response_type = 'image/' + path.split('.')[1] # path.split('.')[1] returns the extension (for example: png)
211             with open(path,'rb') as image:
212                 image = image.read()
213                 send_response(200, response_type)
214                 connectionSocket.send(image) # send the image to the client through the socket
215             else:
216                 Error(client_ip, client_port)
217
218     # videos requests (similar to image request)
219     # request: request: /videos/example.mp4
220     elif request.startswith("/videos/"):
221         video_file_name = request.split('/')[2] # get the video file name
222         if video_file_name in videos:
223             type = 'video/' + video_file_name.split('.')[1] # get the video extension
224             path = './videos/' + video_file_name
225             with open(path, 'rb') as video:
226                 video = video.read()
227                 send_response(200, type)
228                 connectionSocket.send(video)
229             else:
230                 Error(client_ip, client_port)

```

Figure 52 Python Server Code 9

This section of the code handles requests for images and videos. For image requests, it checks if the requested image exists in the images directory. If the file is found, the server determines the appropriate response type based on the image file's extension (e.g., PNG or JPEG). The image is then read from the server and sent back to the client with a "200 OK" response. If the image is not found, an error page is returned instead. Similarly, for video requests, the server checks if the requested video file exists in the videos directory. If the file is available, it reads and sends the video to the client with the corresponding content type based on the video file's extension. If the video is not found, an error page is returned.

```

233     #if the client made any request that does not exist
234     else:
235         try:
236             object = request.split('/')[1]
237             if object in files or object in html or object in css or object in images or object in videos:
238                 type = object.split('.')[1]
239                 if object in html:
240                     object = './html/' + object
241                 elif object in css:
242                     object = './css/' + object
243                 elif object in images:
244                     object = './imgs/' + object
245                 elif object in videos:
246                     object = './videos/' + object
247
248                 if type == 'html':
249                     send_response(200, 'text/html')
250                     with open(object, 'r') as file:
251                         file = file.read()
252                         connectionSocket.send(file.encode())
253                 elif type == 'css':
254                     send_response(200, 'text/css')
255                     with open(object, 'rb') as file:
256                         file = file.read()
257                         connectionSocket.send(file)
258                 elif type == 'jpg':
259                     send_response(200, 'image/jpg')
260                     with open(object, 'rb') as file:
261                         file = file.read()
262                         connectionSocket.send(file)
263                 elif type == 'png':
264                     send_response(200, 'image/png')
265                     with open(object, 'rb') as file:
266                         file = file.read()
267                         connectionSocket.send(file)

```

Figure 53 Python Server Code 10

```

262             connectionSocket.send(file)
263         elif type == 'png':
264             send_response(200, 'image/png')
265             with open(object, 'rb') as file:
266                 file = file.read()
267                 connectionSocket.send(file)
268         elif type == 'mp4':
269             send_response(200, 'video/mp4')
270             with open(object, 'rb') as file:
271                 file = file.read()
272                 connectionSocket.send(file)
273         else:
274             send_response(200, 'text/html') # default
275             with open(object, 'rb') as file:
276                 file = file.read()
277                 connectionSocket.send(file)
278         else:
279             Error(client_ip,client_port)
280     except IndexError:
281         print('Bad Request')
282
283     connectionSocket.close()

```

Figure 54 Python Server Code 11

This above of the code handles requests for any files that do not match the predefined categories of HTML, CSS, images, or videos. When the client makes a request for a file, the server checks if the requested file exists in any of these categories (files, html, css, images, or videos). If the file is found, the server determines the appropriate file path based on the type of file and reads it from the appropriate directory (./html/ for HTML files, ./css/ for CSS files, etc.). Depending on the file type (HTML, CSS, image, or video), the server sends the file back with the correct content type (e.g., text/html for HTML, image/jpg for images, video/mp4 for videos). If the file is not found in any of these categories, the server sends an error page (Error 404) to the client. If there's an issue with the request, such as an invalid file path, it prints "Bad Request" in the console. Finally, the connection to the client is closed.

Server Functionality Testing

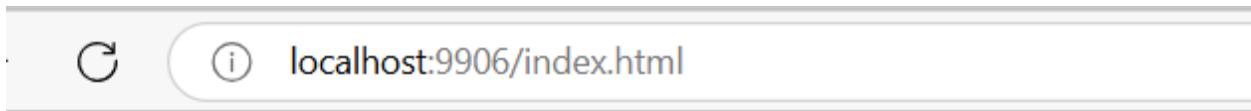


Figure 55 Browser Request for main English webpage

For the client (browser) request: ‘/’, ‘/index.html’, ‘/main_en.html’ or ‘/en’, The server receives a request for the main English webpage, and then receives a request for each object in this page: (main_en.css, Qasim.png, taleed.png, PacketSwitching.jpg, CircuitSwitching.jpg, background.jpg, member_background.jpg).

```
Web Server is listening on port 9906 ...
Got connection from IP: 127.0.0.1, Port: 63121, Server Port: 9906
The Request is : /index.html
GET /index.html HTTP/1.1
Host: localhost:9906
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "chromium";v="135"
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
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-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63121 | Server Port: 9906
```

Figure 56 index.html request

```
Got connection from IP: 127.0.0.1, Port: 63129, Server Port: 9906
The Request is : /static/css/main_en.css
GET /static/css/main_en.css HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "chromium";v="135"
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:9906/index.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63129 | Server Port: 9906
```

Figure 57 main_en.css request

```

Got connection from IP: 127.0.0.1, Port: 63130, Server Port: 9906
The Request is : /static/images/Qasim.png
GET /static/images/Qasim.png HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/index.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63130 | Server Port: 9906
Got connection from IP: 127.0.0.1, Port: 63131, Server Port: 9906
The Request is : /static/images/taleed.png
GET /static/images/taleed.png HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/index.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63131 | Server Port: 9906

```

Figure 58 members photos requests

```

Got connection from IP: 127.0.0.1, Port: 63134, Server Port: 9906
The Request is : /static/images/PacketSwitching.jpg
GET /static/images/PacketSwitching.jpg HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/index.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63134 | Server Port: 9906
Got connection from IP: 127.0.0.1, Port: 63135, Server Port: 9906
The Request is : /static/images/CircuitSwitching.jpg
GET /static/images/CircuitSwitching.jpg HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/index.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63135 | Server Port: 9906

```

Figure 59 Switching photos requests

```

Got connection from IP: 127.0.0.1, Port: 63136, Server Port: 9906
The Request is : /static/images/background.jpg
GET /static/images/background.jpg HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A-Brand";v="8", "Chromium";v="135"
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:9906/static/css/main_en.css
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63136 | Server Port: 9906
Got connection from IP: 127.0.0.1, Port: 63137, Server Port: 9906
The Request is : /static/images/member_background.jpg
GET /static/images/member_background.jpg HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A-Brand";v="8", "Chromium";v="135"
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:9906/static/css/main_en.css
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63137 | Server Port: 9906

```

Figure 60 Background photos requests

The server response for these requests is the html file for the main English webpage, and all objects in this page. The browser (client) displays these objects to the user (the Main English Webpage) as shown in the figure below:



Figure 61 Response for the main English webpage

Requests to /ar or /main_ar.html work the same as English page requests, the server sends the Arabic main page, and the browser loads its linked files like CSS and images.

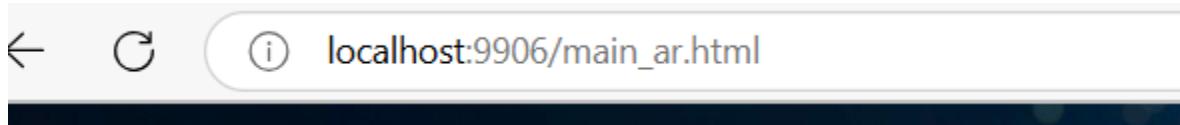


Figure 62 Browser Request for main Arabic webpage

```
Got connection from IP: 127.0.0.1, Port: 63226, Server Port: 9906
The Request is : /static/css/main_ar.css
GET /static/css/main_ar.css HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/main_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63226 | Server Port: 9906
Got connection from IP: 127.0.0.1, Port: 63230, Server Port: 9906
The Request is : /static/images/Qasim.png
GET /static/images/Qasim.png HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/main_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63230 | Server Port: 9906
```

Figure 63 Part of browser requests

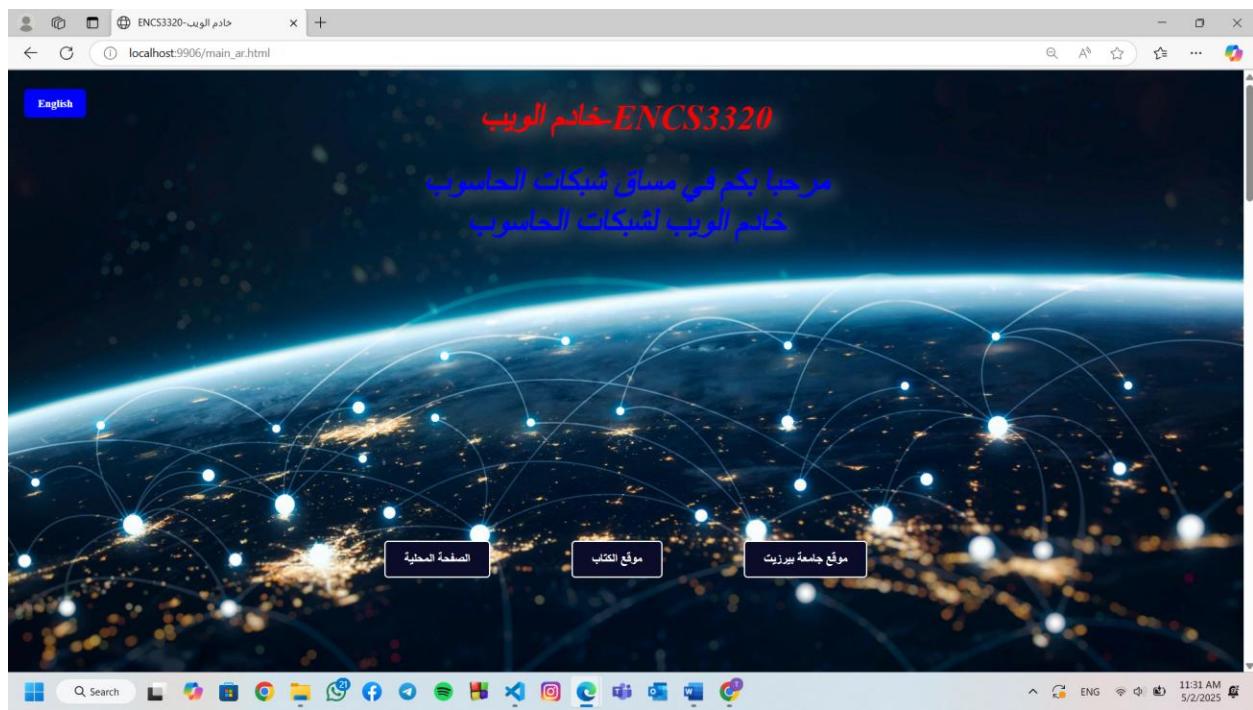


Figure 64 Response for the main Arabic webpage

When the client (browser) requests a file that doesn't exist on the server, the server responds with a 404 Not Found status and sends an error webpage (as shown in the figure below). The browser tab shows "Error 404," and the page displays "The File Not Found!" along with the client's IP address and port number.



Figure 65 invalid URL request

```

Got connection from IP: 127.0.0.1, Port: 63241, Server Port: 9906
The Request is : /testmyserver
GET /testmyserver HTTP/1.1
Host: localhost:9906
Connection: keep-alive
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
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-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 404 Not Found | Client IP: 127.0.0.1 | Client Port: 63241 | Server Port: 9906

```

Figure 66 Request for invalid file

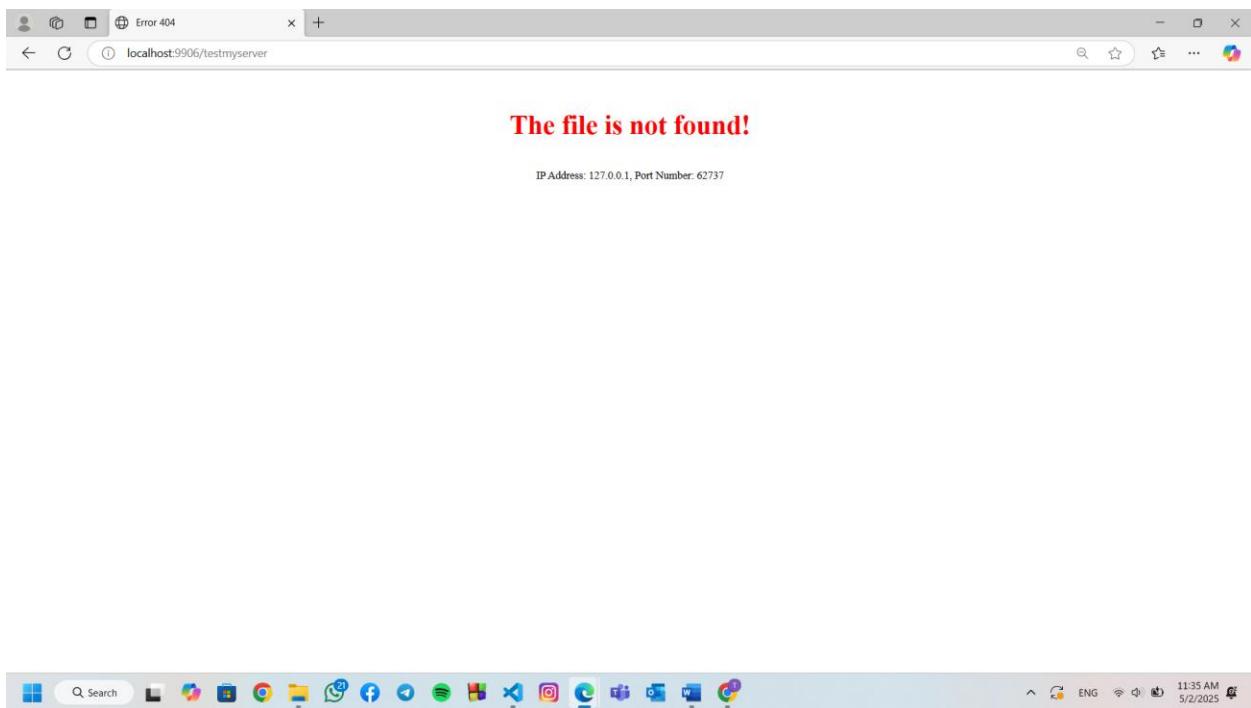


Figure 67 Response for invalid Request

When the client clicks on the **Birzeit** button, the browser opens the official Birzeit University website (<https://www.birzeit.edu/>) in a new tab. Similarly, clicking the **Textbook** button directs the user to the textbook's official website (https://gaia.cs.umass.edu/kurose_ross/index.php). Both links are handled using standard HTML hyperlinks, allowing users to access external resources easily.

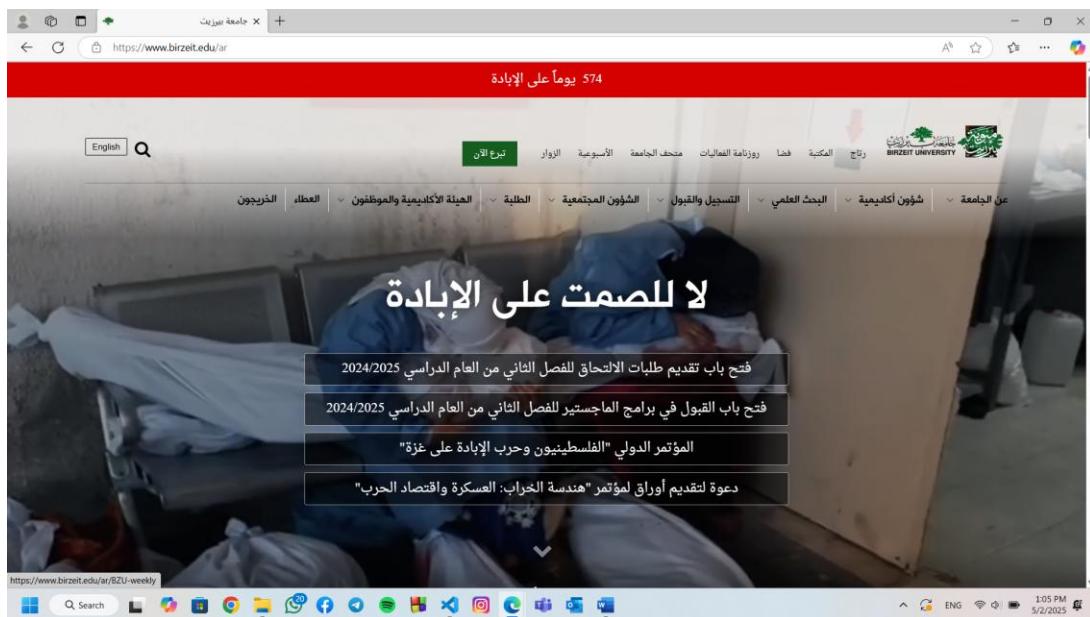


Figure 68 Birzeit Website

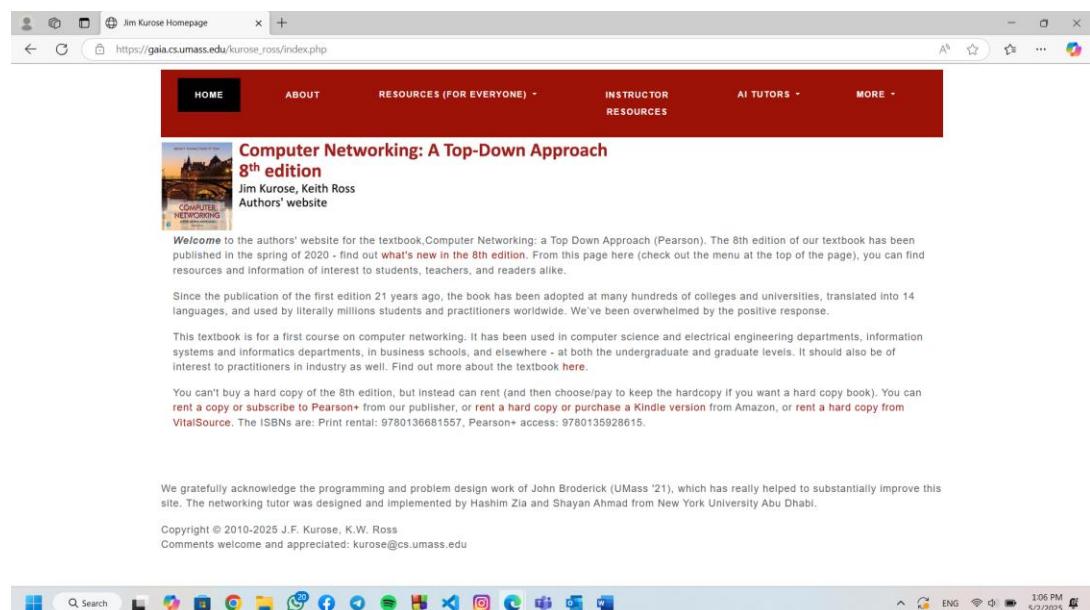


Figure 69 Textbook Website

When the user clicks on the **Local Webpage** button, the server receives requests for the HTML file of the page and all the associated objects it includes.

```

Got connection from IP: 127.0.0.1, Port: 63661, Server Port: 9906
BAD REQUEST
Got connection from IP: 127.0.0.1, Port: 63746, Server Port: 9906
The Request is : /html/mySite_1220006_en.html
GET /html/mySite_1220006_en.html HTTP/1.1
Host: localhost:9906
Connection: keep-alive
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
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-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:9906/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63746 | Server Port: 9906

```

Figure 70 mySite_1220006_en.html request

```

Got connection from IP: 127.0.0.1, Port: 63748, Server Port: 9906
The Request is : /static/css/mySite_1220006_en.css
GET /static/css/mySite_1220006_en.css HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/html/mySite_1220006_en.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63748 | Server Port: 9906

Got connection from IP: 127.0.0.1, Port: 63751, Server Port: 9906
The Request is : /static/images/background.jpg
GET /static/images/background.jpg HTTP/1.1
Host: localhost:9906
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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:9906/static/css/mySite_1220006_en.css
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63751 | Server Port: 9906

```

Figure 71 Local Webpage objects' requests

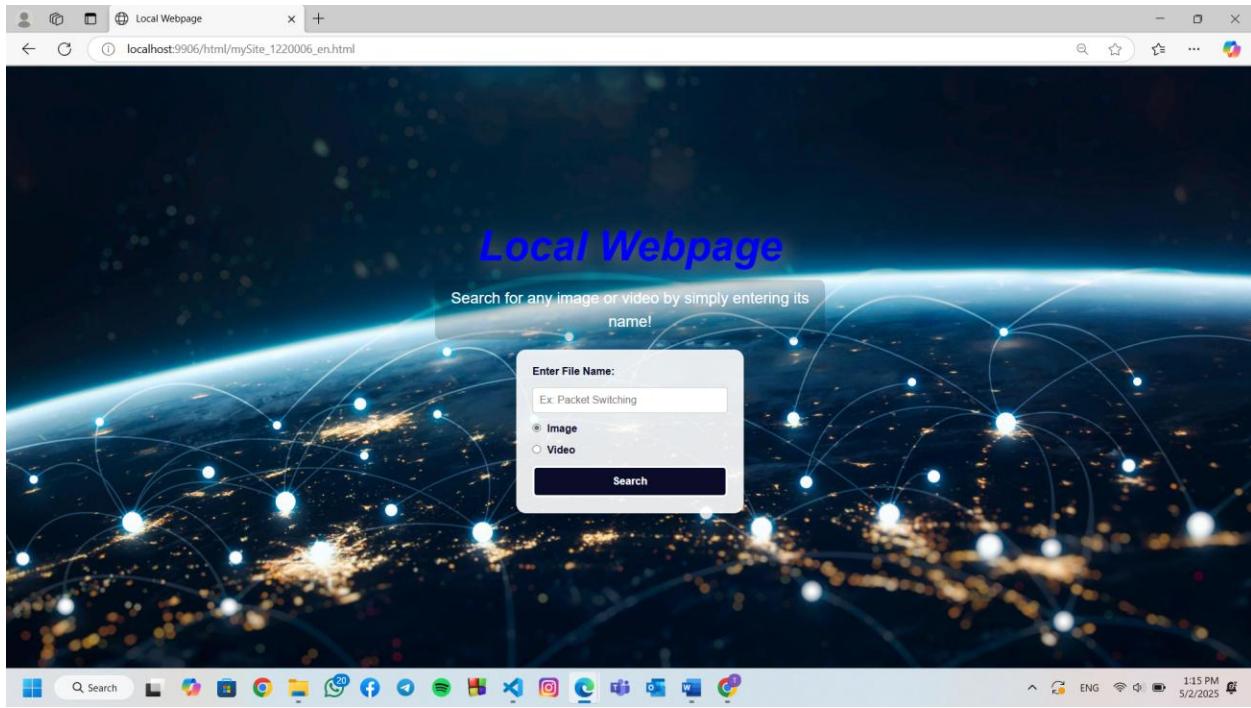


Figure 72 Response for the local Webpage

The user types the name of the image they want from the server and clicks submit. The server then processes the request and either returns the image if it exists or redirects the user to Google if it doesn't.



Figure 73 Search for image in local Webpage

```

Got connection from IP: 127.0.0.1, Port: 63790, Server Port: 9906
The Request is : /html/request_handler?material=CircuitSwitching&type=image
GET /html/request_handler?material=CircuitSwitching&type=image HTTP/1.1
Host: localhost:9906
Connection: keep-alive
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "chromium";v="135"
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:9906/html/mysite_1220006_en.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63790 | Server Port: 9906

```

Figure 74 Request for image from local Webpage

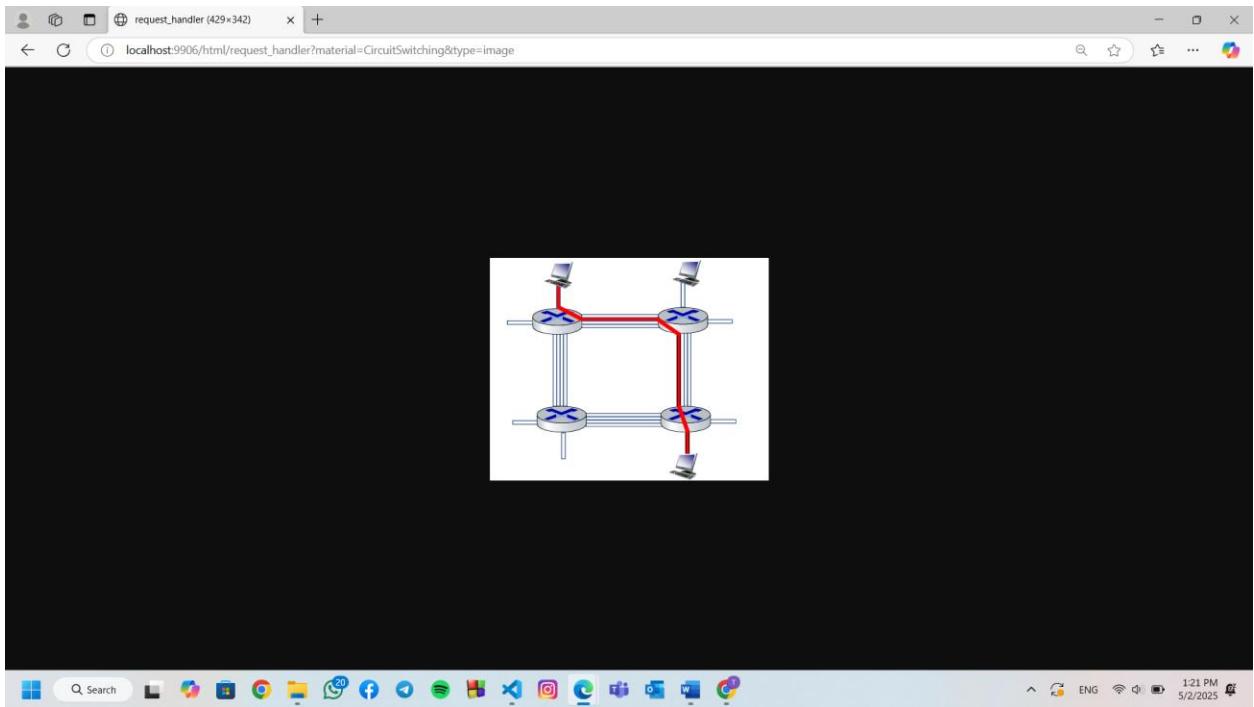


Figure 75 Response for the image

The Response of the server includes status code of (200 OK) as shown in request figure above with content type (image/jpg), and the requested image.

If the requested image, such as one named "Networks", is not found on the server, the server responds by redirecting the client to Google to view image search results related to "Networks". Along with this redirection, the server sends a response with the status code **307 Temporary Redirect**, indicating that the client should look elsewhere for the requested content.

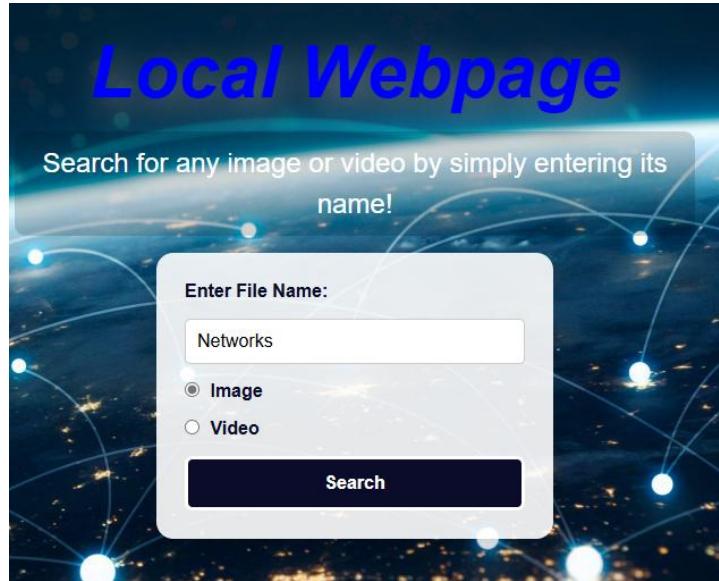


Figure 76 Search for unavailable image

```

Got connection from IP: 127.0.0.1, Port: 63813, Server Port: 9906
The Request is : /html/request_handler?material=Networks&type=image
GET /html/request_handler?material=Networks&type=image HTTP/1.1
Host: localhost:9906
Connection: keep-alive
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
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-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:9906/html/mysite_1220006_en.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 307 Temporary Redirect | Client IP: 127.0.0.1 | Client Port: 63813 | Server Port: 9906
Redirect to Google

```

Figure 77 Request for unavailable image

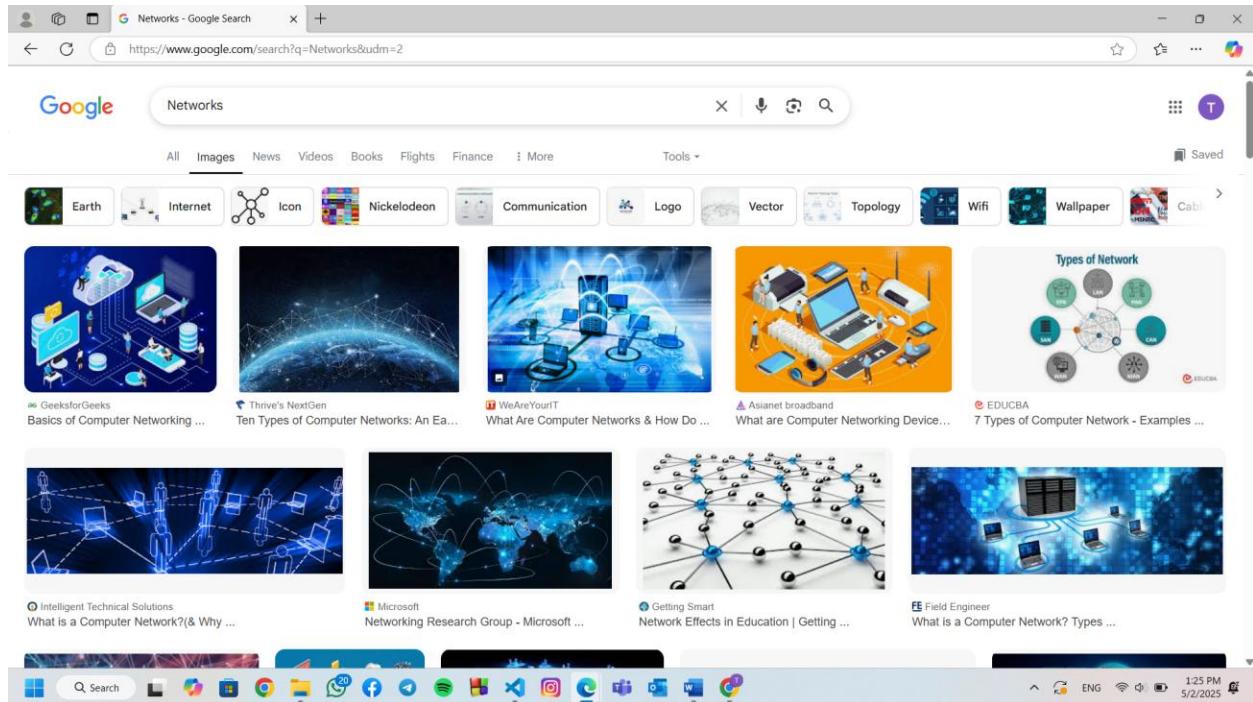


Figure 78 Response for unavailable image

Same for Videos:

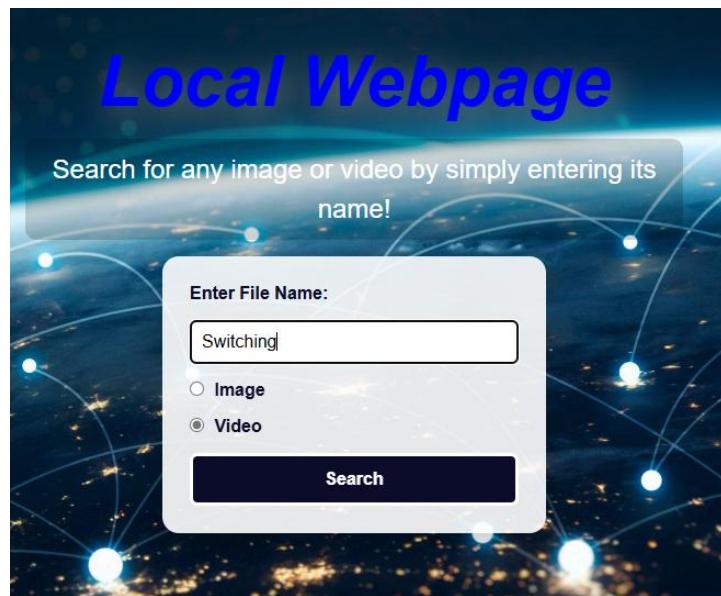


Figure 79 Search for video in local Webpage

```

Got connection from IP: 127.0.0.1, Port: 63845, Server Port: 9906
The Request is : /html/request_handler?material=Switching&type=video
GET /html/request_handler?material=Switching&type=video HTTP/1.1
Host: localhost:9906
Connection: keep-alive
sec-ch-ua-platform: "Windows"
Accept-Encoding: identity;q=1, *;q=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36 Edg/135.0.0.0
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
sec-ch-ua-mobile: ?0
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: video
Referer: http://localhost:9906/html/request_handler?material=Switching&type=video
Accept-Language: en-US,en;q=0.9,ar;q=0.8
Range: bytes=0-

```

Response Sent: HTTP/1.1 200 OK | Client IP: 127.0.0.1 | Client Port: 63845 | Server Port: 9906

Figure 80 Request for video from local Webpage



Figure 81 Response for the Video

If the requested video is not available on the server, the server redirects the client to YouTube to view search results related to that video.

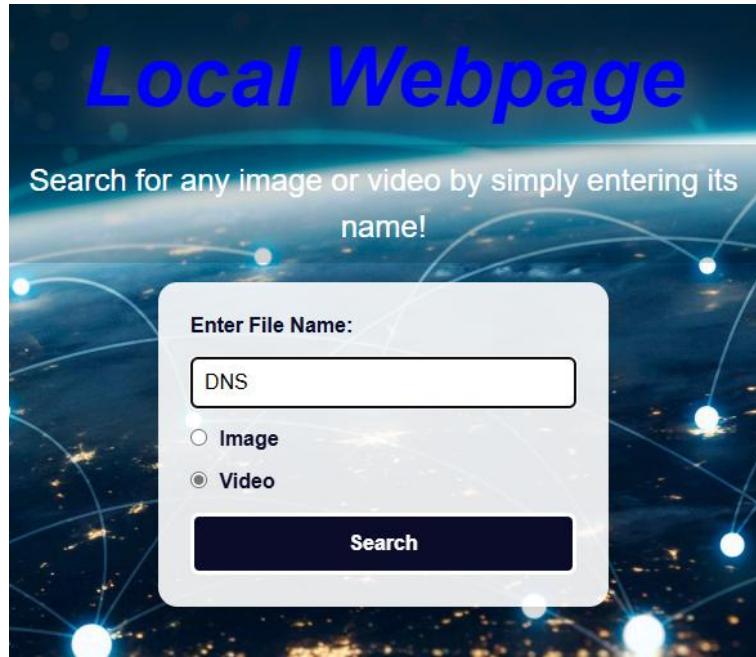


Figure 82 Search for unavailable video

```
Got connection from IP: 127.0.0.1, Port: 63846, Server Port: 9906
BAD REQUEST
Got connection from IP: 127.0.0.1, Port: 63876, Server Port: 9906
The Request is : /html/request_handler?material=DNS&type=video
GET /html/request_handler?material=DNS&type=video HTTP/1.1
Host: localhost:9906
Connection: keep-alive
sec-ch-ua: "Microsoft Edge";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
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/135.0.0.0 Safari/537.36 Edg/135.0.0.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:9906/html/mySite_1220006_en.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Response Sent: HTTP/1.1 307 Temporary Redirect | Client IP: 127.0.0.1 | Client Port: 63876 | Server Port: 9906
Redirect to Youtube
```

Figure 83 Request for unavailable video

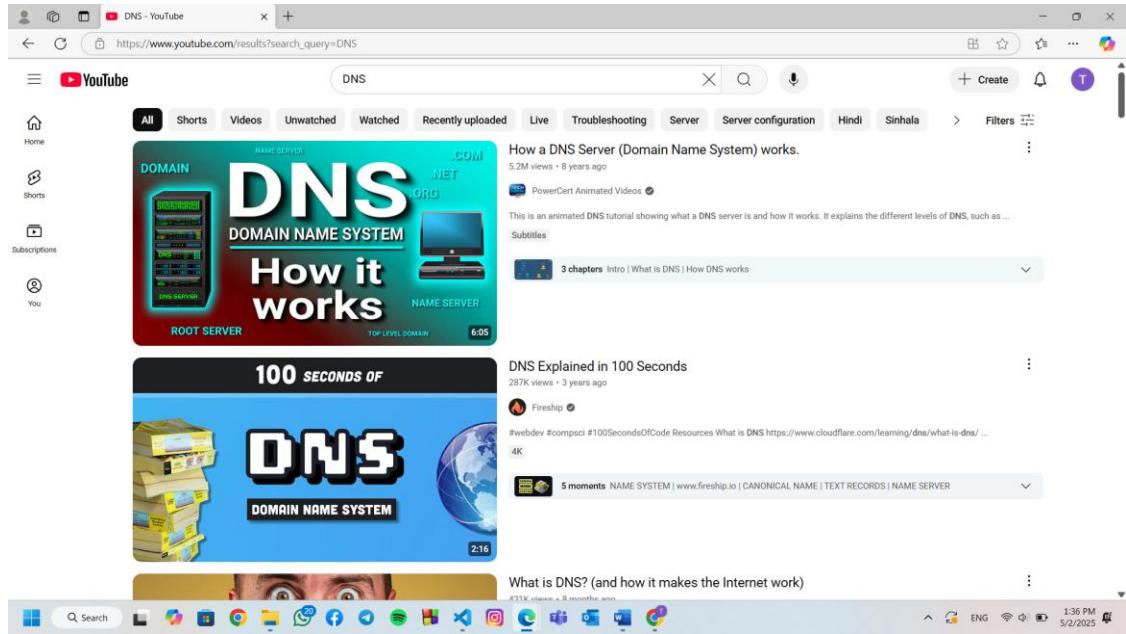


Figure 84 Response for unavailable video

When the client (browser) requests a file like *mySite_1220006_en.css*, the server checks if the file exists in the CSS directory. If it exists, it will respond with the file and a status code of **200 OK**; otherwise, it returns an error response.

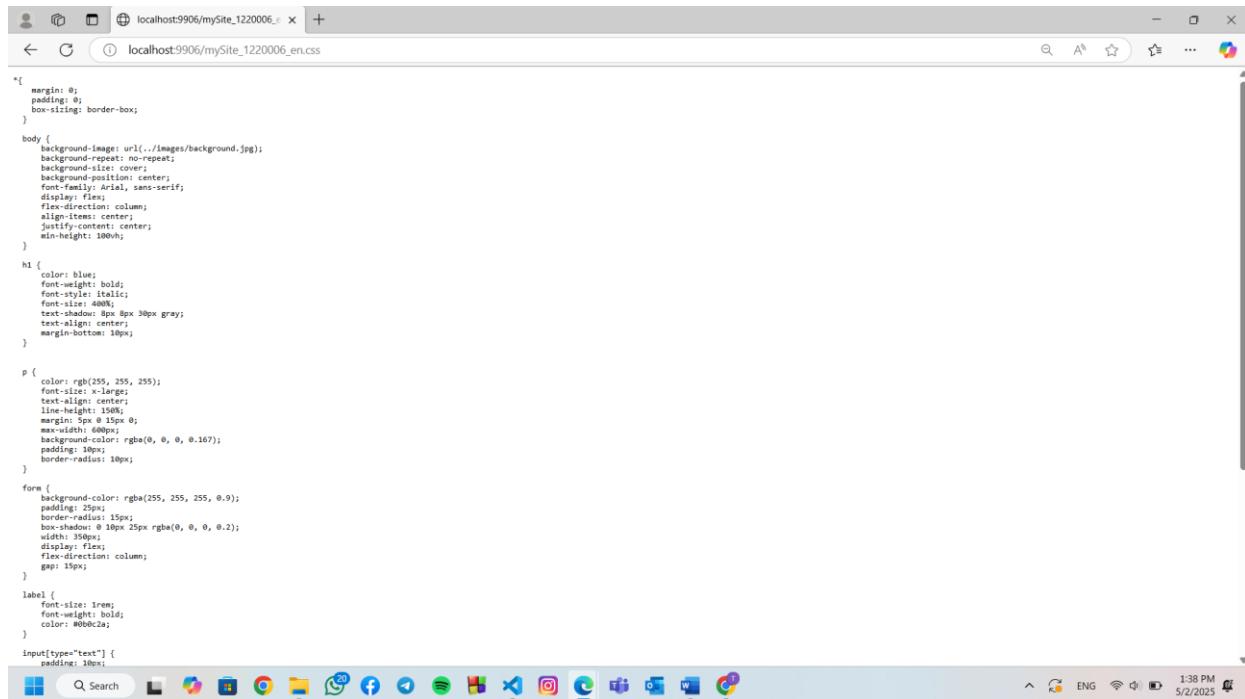


Figure 85 Response for css file request

Another device on the same local network requests a webpage from the server:

```
Got connection from IP: 192.168.1.101, Port: 59905, Server Port: 9906
The Request is : /
GET / HTTP/1.1
Host: 192.168.1.126:9906
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 18_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.3 Mobile/15E148 Safari/604.1
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Priority: u=0, i
Accept-Encoding: gzip, deflate
Connection: keep-alive

Response Sent: HTTP/1.1 200 OK | Client IP: 192.168.1.101 | Client Port: 59905 | Server Port: 9906

Got connection from IP: 192.168.1.101, Port: 59906, Server Port: 9906
The Request is : /static/css/main_en.css
GET /static/css/main_en.css HTTP/1.1
Host: 192.168.1.126:9906
Referer: http://192.168.1.126:9906/
Accept: text/css,*/*;q=0.1
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 18_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.3 Mobile/15E148 Safari/604.1
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Priority: u=1, i
Accept-Encoding: gzip, deflate
Connection: keep-alive

Response Sent: HTTP/1.1 200 OK | Client IP: 192.168.1.101 | Client Port: 59906 | Server Port: 9906
```

Figure 86 Request for a webpage from another device



Figure 87 Server Response to Iphone device request

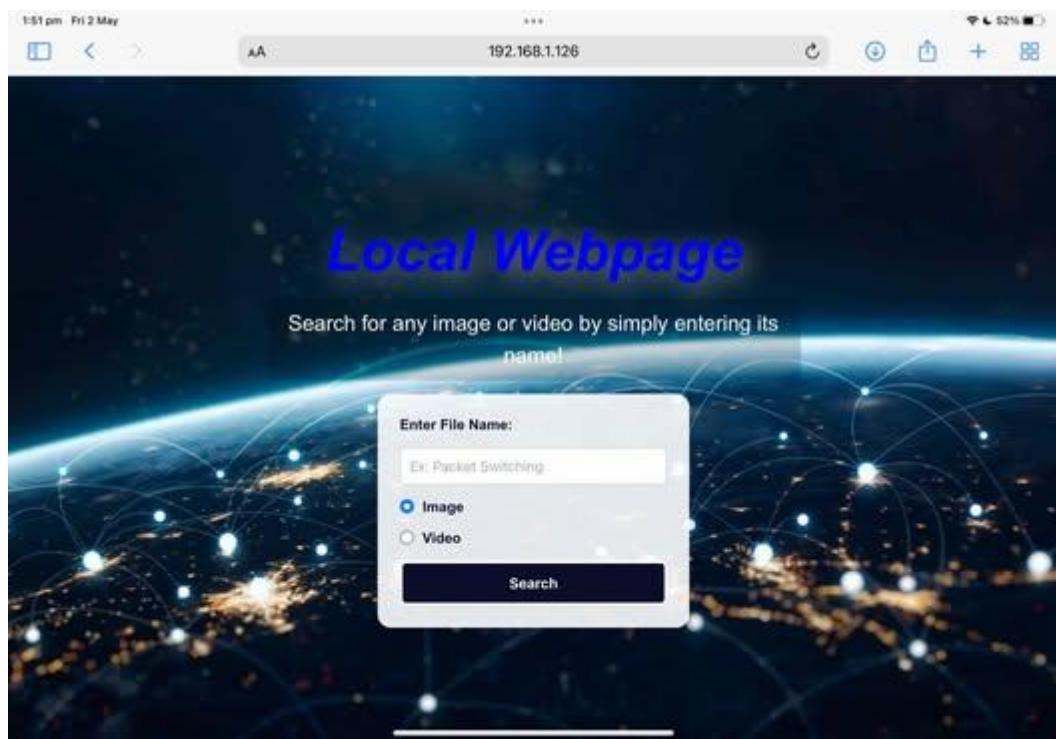


Figure 88 Server Response to IPAD device request

TASK 3: TCP/UDP Client-Server Game Using Socket Programming

In this task, we have implemented a hybrid network guessing game where the server uses TCP for reliable control messages in the game setup and final results, and uses UDP for fast, real-time feedbacks during the gameplay. The game supports multiple players, each one trying to guess a randomly number generated by the server before the others.

Game Workflow

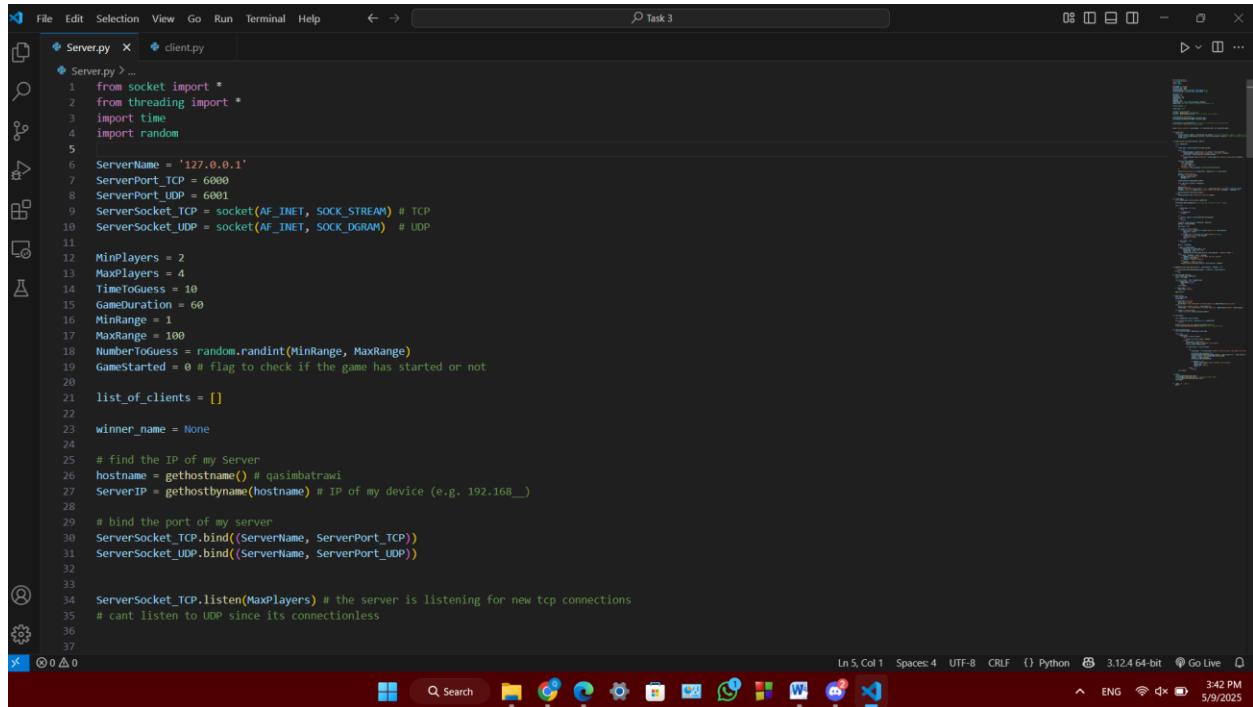
The server opens TCP and UDP sockets with port numbers 6000 and 6001, respectively. The server waits for at least two clients (players) to connect to the TCP socket by entering a unique name. Once two players connected, the game begins, and all players simultaneously enter their guess to the server via UDP. For each guess, the server will respond with a feedback indicating whether the guessed number that the player entered is higher, lower, or equal to the target number. The first player that guesses the target number correctly wins the game. The server finally sends the final results for each client via TCP.

The total gameplay duration is 60 seconds and players must try to guess the number within this time or the game will end with no winner. Each player has a time of 10 seconds to send his guessed number, and numbers must be within the range defined by the server.

In case a player disconnects from the server during the gameplay, other players will be notified and asked if they want to continue guessing or end the game.

Players will play the game in rounds, each round is as described above, and after the results are announced, a new game begins.

Python Server Code



```
File Edit Selection View Go Run Terminal Help Task 3

Server.py > ...
1  from socket import *
2  from threading import *
3  import time
4  import random
5
6  ServerName = '127.0.0.1'
7  ServerPort_TCP = 6000
8  ServerPort_UDP = 6001
9  ServerSocket_TCP = socket(AF_INET, SOCK_STREAM) # TCP
10 ServerSocket_UDP = socket(AF_INET, SOCK_DGRAM) # UDP
11
12 MinPlayers = 2
13 MaxPlayers = 4
14 TimeToGuess = 10
15 GameDuration = 60
16 MinRange = 1
17 MaxRange = 100
18 NumberToGuess = random.randint(MinRange, MaxRange)
19 GameStarted = 0 # flag to check if the game has started or not
20
21 list_of_clients = []
22
23 winner_name = None
24
25 # find the IP of my Server
26 hostname = gethostname() # qasimbatrawi
27 ServerIP = gethostbyname(hostname) # IP of my device (e.g. 192.168...)
28
29 # bind the port of my server
30 ServerSocket_TCP.bind((ServerName, ServerPort_TCP))
31 ServerSocket_UDP.bind((ServerName, ServerPort_UDP))
32
33
34 ServerSocket_TCP.listen(MaxPlayers) # the server is listening for new tcp connections
35 # can't listen to UDP since its connectionless
36
37

Ln 5, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live 3:42 PM 5/9/2025
```

Figure 89 Server Code 1

The above figure shows the main parameters that we have used for our server, which includes the server name (localhost), the port number of TCP (6000) and UDP (6001), the server sockets TCP (for reliable connection) and UDP (for fast messaging), where we have used the `socket()` function to create them. The target number which is randomly generated using `random()` function. Each socket (TCP and UDP) is bound to a specific IP address using `bind()` method. The TCP socket then starts listening to any client connection, where it can listen to at max four players.

```

224
225
226     def main():
227         Thread(target=listen_tcp).start()
228         Thread(target=listen_udp).start()
229         Thread(target=check_disconnections).start()
230         start_game()
231
232     if __name__ == '__main__':
233         main()

```

Figure 90 Server Code 2

Figure 88 shows the server's main function, where three threads are created: one for TCP connections, another for UDP fast messaging, and thread to check if some client has disconnected during the gameplay. The main thread will enter strat_game() function. Each will be discussed later.

```

43     def listen_tcp():
44         while True:
45             connectionSocket, address = ServerSocket.TCP.accept() # new client connection. address = (client ip, client tcp port)
46             thread = Thread(target=handle_client_tcp, args=(connectionSocket, address)) # apply new thread for the new client
47             thread.start()
48
49     def handle_client_tcp(connectionSocket, address):
50         global GameStarted
51         try:
52             client_name = connectionSocket.recv(2048).decode()
53             while True:
54                 if any(client['name'] == client_name for client in list_of_clients):
55                     connectionSocket.send("Username already taken, try another.".encode())
56                     client_name = connectionSocket.recv(2048).decode()
57                 else:
58                     connectionSocket.send(f"Connected as {client_name}\nTCP connection established\n".encode())
59                     break
60
61             list_of_clients.append({
62                 'name': client_name,
63                 'IP': address[0],
64                 'tcp_port': address[1],
65                 'udp_port': None,
66                 'tcp_socket': connectionSocket
67             })
68
69             print(f"New Connection from ({address[0]}, {address[1]}) as {client_name}")
70             message = "Waiting Room:\n"
71             for client in list_of_clients:
72                 message+=client['name']
73                 message+='\n'
74
75             connectionSocket.send(message.encode())
76
77             while len(list_of_clients) < MinPlayers:
78                 continue
79
80             GameStarted = True
81             message = "Game started with players: " + ", ".join(client['name'] for client in list_of_clients)
82             message+=f"\nYou have {GameDuration} seconds to guess the number ({MinRange} - {MaxRange})!\n"
83
84             connectionSocket.send(message.encode())
85         except:
86             connectionSocket.send("\nConnection Failed!".encode())

```

Figure 91 Server Code 3

Figure 89 shows the function that handles clients TCP connection. The server creates a socket of the connected client and receives its address, and then asks the client to enter a player name. The server will ensure that the name is unique; if not, it will send a message to the client via the created socket to enter a new name. After guaranteeing that the name is unique, the server adds the information of the client to the list of clients and sends the waiting room to the client. The server will wait until another client connects to start the game.

The screenshot shows a Python code editor with a file named `Server.py` open. The code implements a UDP-based guessing game server. It uses a global variable `winner_name` to store the name of the winning player and a list `list_of_clients` to store client information (IP and UDP port). The server listens on port 2048 for incoming UDP messages. When a message is received, it checks if the client has already participated. If so, it compares the guess with the target number (`NumberToGuess`). If correct, it declares the client as the winner. If incorrect, it provides feedback ('Higher' or 'Lower'). If the client has not participated yet, it adds them to the list. The server also handles cases where the game has ended or not started.

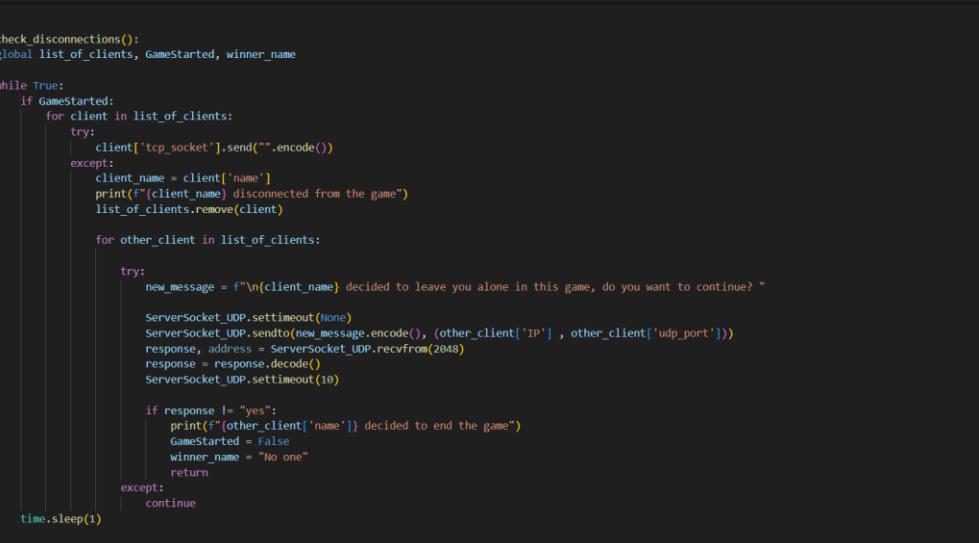
```
File Edit Selection View Go Run Terminal Help < > Task 3
Server.py x client.py
Server.py > listen_udp
88
89 def listen_udp():
90     global winner_name, list_of_clients, GameStarted
91     ServerSocket_UDP.settimeout(TimeToGuess) # the client has 10 seconds to send a response
92
93     while True:
94         if winner_name is not None: # the game has ended
95             break
96         if not GameStarted: # the game has not started yet
97             continue
98         try:
99             message, address = ServerSocket_UDP.recvfrom(2048)
100        except:
101            continue
102
103        client_IP, client_udp_port = address[0], address[1]
104        message = message.decode()
105        the_client = None
106
107        for client in list_of_clients:
108            if client['IP'] == client_IP and client['udp_port'] == client_udp_port:
109                the_client = client
110                break
111            elif client['IP'] == client_IP and client['udp_port'] is None:
112                client['udp_port'] = client_udp_port
113                the_client = client
114                break
115
116        if the_client is None:
117            continue
118
119        guess = int(message)
120        if guess == NumberToGuess:
121            if GameStarted and winner_name is None:
122                winner_name = the_client['name']
123                GameStarted = False
124                response_to_one_client_udp(client_IP, client_udp_port, "Feedback: CORRECT.")
125            else:
126                if guess < MinRange or guess > MaxRange:
127                    feedback = "Warning: Out of the range, miss your chance\n"
128                elif guess < NumberToGuess:
129                    feedback = "Feedback: Higher\n"
130                else:
131                    feedback = "Feedback: Lower\n"
132                response_to_one_client_udp(client_IP, client_udp_port, feedback)
```

Figure 92 Server Code 4

Once two clients connect to the server, the game will start and the UDP socket will start receive guessing from the clients. As shown in the figure above, UDP socket will receive from each client its guess and it will send a feedback if the client is higher, lower, or equal to the target number.

Figure 93 Server Code 5

The figure above shows the function used to send a response to the client. The server sends a feedback to the client using its UDP socket, as we need to specify the client's IP and UDP port number, unlike TCP, where the connection already handles this information.



The screenshot shows a Windows desktop environment with a terminal window open. The terminal window has tabs for 'Server.py' and 'client.py'. The code in 'Server.py' is a Python script for a game server. It defines a function 'check_disconnections()' which loops until a condition is met. Inside the loop, it checks if a client has disconnected by sending an empty message and checking for a response. If a client disconnects, it prints a message and removes the client from the list. It then tries to send a message to another client asking if they want to continue. If the response is 'no', it sets game variables to indicate the game is over and no winner. A 'time.sleep(1)' call is at the end of the loop. The code ends with a 'def main()' function. The terminal window also shows file navigation and a taskbar with various icons.

```
File Edit Selection View Go Run Terminal Help ← → Task 3

Server.py x client.py
Server.py > ⚡ wait_and_send_results

181
182
183 def check_disconnections():
184     global list_of_clients, GameStarted, winner_name
185
186     while True:
187         if GameStarted:
188             for client in list_of_clients:
189                 try:
190                     client['tcp_socket'].send("".encode())
191                 except:
192                     client_name = client['name']
193                     print(f"{client_name} disconnected from the game")
194                     list_of_clients.remove(client)
195
196             for other_client in list_of_clients:
197
198                 try:
199                     new_message = f"\n{client_name} decided to leave you alone in this game, do you want to continue?"
200
201                     ServerSocket_UDP.settimeout(None)
202                     ServerSocket_UDP.sendto(new_message.encode(), (other_client['IP'], other_client['udp_port']))
203                     response, address = ServerSocket_UDP.recvfrom(2048)
204                     response = response.decode()
205                     ServerSocket_UDP.settimeout(10)
206
207                     if response != "yes":
208                         print(f"{other_client['name']} decided to end the game")
209                         GameStarted = False
210                         winner_name = "No one"
211                         return
212
213                 except:
214                     continue
215
216             time.sleep(1)
217
218     def main():

Ln 143, Col 24 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live
ENG Wi-Fi 5/9/2025
```

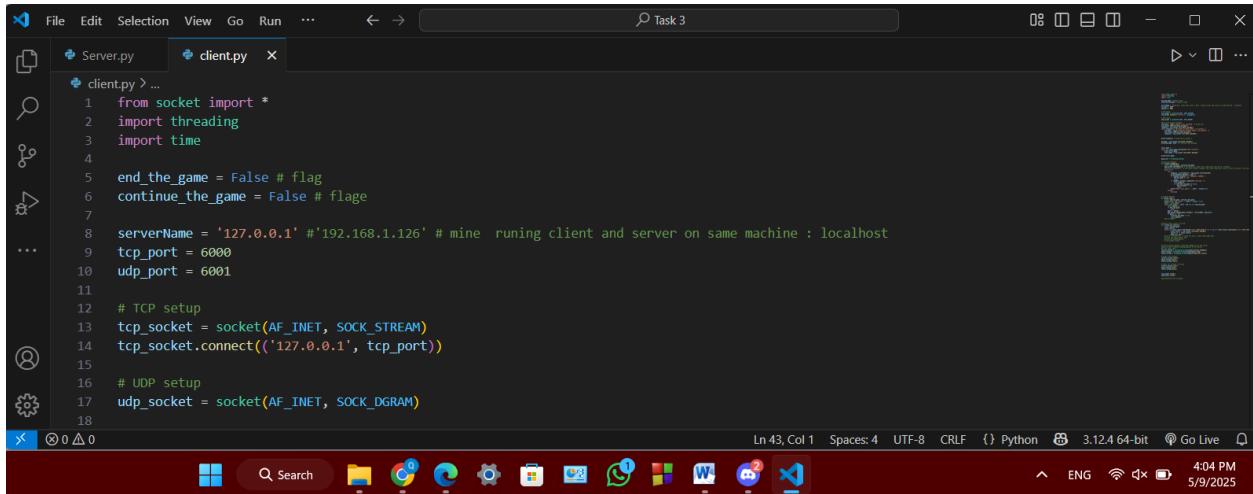
Figure 94 Server Code 6

The figure above shows how the server checks if a client has disconnected. Every second during the game, the server will send an empty message to the client through the TCP socket. If a client did not receive the message, it means that the client has disconnected. The server then asks the remaining client if he wants to continue playing; if not, the game will end with no winner.

Figure 95 Server Code 7

As we have mentioned before, the main thread runs the `start_game()` function. As shown in the figure above, the function creates a new thread when the game starts. This thread will run until the game duration ends, one of the clients wins, or all clients disconnects. Game results will be sent to each client through the TCP socket, indicating the winner and the target number.

Python Client Code



A screenshot of a Windows desktop environment showing a code editor window titled "Task 3". The window displays two files: "Server.py" and "client.py". The "client.py" file is open and contains the following Python code:

```
from socket import *
import threading
import time

end_the_game = False # flag
continue_the_game = False # flag

serverName = '127.0.0.1' #'192.168.1.126' # mine runing client and server on same machine : localhost
tcp_port = 6000
udp_port = 6001

# TCP setup
tcp_socket = socket(AF_INET, SOCK_STREAM)
tcp_socket.connect(('127.0.0.1', tcp_port))

# UDP setup
udp_socket = socket(AF_INET, SOCK_DGRAM)
```

The code imports the socket module and threading module. It defines two flags: "end_the_game" and "continue_the_game", both initially set to False. It sets the server name to "127.0.0.1" or "#192.168.1.126" (commented out) and specifies TCP and UDP ports (6000 and 6001 respectively). It then creates a TCP socket and connects it to the server at the specified address and port. Finally, it creates a UDP socket.

Figure 96 Client Code 1

The figure shows the parameters that we have used for the clients. We have defined two flags: one to determine if the game has ended or not, and the other to determine whether the client want to continue the game or not after another client disconnects. The server name is set to the localhost number to allow connection to the server on the same machine. TCP and UDP ports match those used by the server. TCP and UDP sockets are created for the client, and the TCP socket sends a connection request to the server.

A screenshot of a code editor window titled "client.py". The code is a Python script for a client application. It starts by prompting the user to enter a unique username. If the username is already taken, it asks for another. Once a unique username is chosen, the client sends it to the server via a TCP socket. The client then waits for a response from the server indicating the start of the game. The code uses standard Python libraries like `socket` and `threading`.

```
File Edit Selection View Go Run ... ← → Task 3
Server.py client.py
client.py > ...
18
19 #join with unique username
20 username = input("\nEnter your username: ") #from user
21 tcp_socket.send(username.encode())
22 response = tcp_socket.recv(1024).decode()
23 while response == "Username already taken, try another.":
24     username = input("Username already taken, try another: ")
25     tcp_socket.send(username.encode())
26 response = tcp_socket.recv(1024).decode()
27
28
29 print(response) # Connected as player 1
30
31 message = tcp_socket.recv(1024).decode()
32 print(message, end="") # waiting room message
33
34
35 start_game = ''
36 while not start_game.startswith('Game started'):
37     print(start_game)
38     start_game = tcp_socket.recv(1024).decode()
39
40 print(start_game)
Ln 33, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live
4:56 PM 5/9/2025
```

Figure 97 Client Code 2

As shown in the figure above, the client here enters and sends his name to the server using the TCP socket. If the server responds that the name is not unique, the client will keep re-entering and sending a new name until it is unique. Client will then wait for another message from the server indicating the start of game.

A screenshot of a code editor window titled "client.py". The code has been modified to handle multiple threads. It creates three threads: one for receiving feedback from the server, one for sending guesses to the server, and one for listening for results. The `receive_thread` handles the server's responses, `guess_thread` handles sending guesses, and `result_thread` handles receiving the final results. The code uses the `threading` module to manage these concurrent operations.

```
File Edit Selection View Go Run ... ← → Task 3
Server.py client.py
client.py > ⚡ receive_feedback
client.py > 
91
92
93 receive_thread = threading.Thread(target=receive_feedback) # receive_thread: Handles receiving feedback from the server
94 guess_thread = threading.Thread(target=guess_loop) # guess_thread: Handles sending guesses to the server
95 result_thread = threading.Thread(target=listen_for_result)
96
97 # start the threads
98 receive_thread.start()
99 guess_thread.start()
100 result_thread.start()
101
Ln 63, Col 16 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live
7:02 PM 5/9/2025
```

Figure 98 Client Code 3

Once the game starts, three threads will be created: one to handle the feedbacks from the server, another to handle the guess response to the server, and one to receive the final results. Each will be discussed later.

```
File Edit Selection View Go Run ... ↵ → Task 3
Server.py client.py x
client.py > ⚡ receive_feedback
45 def receive_feedback():
46     global end_the_game, continue_the_game
47
48     # settimeout(2) makes the client wait only up to 2 seconds #Without settimeout(), if the server doesn't respond,
49     # the client would get stuck on that recvfrom() line forever
50     udp_socket.settimeout(2)
51     while True:
52         try:
53             feedback, serverAddress = udp_socket.recvfrom(1024)
54             print(feedback.decode(), end="")
55             if feedback.decode() == "Feedback: CORRECT.":
56                 end_the_game = True
57                 break
58             if feedback.decode().endswith("continue? "):
59                 time.sleep(5)
60                 if continue_the_game == False:
61                     end_the_game = True
62                     break
63             print("Enter your guess: ", end="", flush=True)
64         except:
65             continue
Ln 49, Col 65 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live 7:09 PM 5/9/2025
```

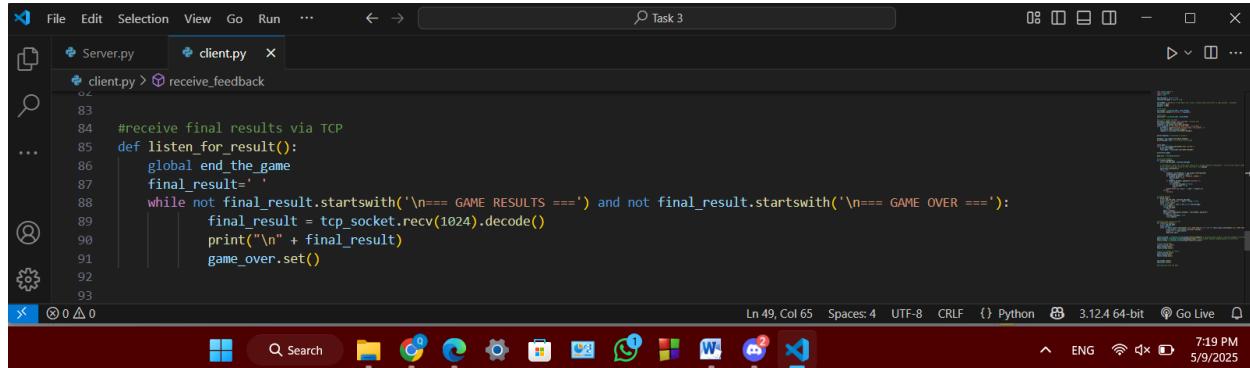
Figure 99 Client Code 4

The figure above shows the how the thread handles the feedbacks from the server. It runs in an infinite loop until it receives a feedback from the server that the last guess was correct, another client has won, or the client decided to leave the game after the other client disconnected.

```
File Edit Selection View Go Run ... ↵ → Task 3
Server.py client.py x
client.py > ⚡ receive_feedback
67
68     # sending guesses
69     def guess_loop():
70         global end_the_game, continue_the_game
71         print("Enter your guess: ", end="", flush=True)
72         start = time.time()
73         while time.time() - start < 60 and not end_the_game:
74             time.sleep(0.1)
75             if end_the_game:
76                 break
77             guess = input()
78             udp_socket.sendto(guess.encode(), (serverName, udp_port))
79             if guess == "yes":
80                 continue_the_game = True
81             time.sleep(5)
82
Ln 49, Col 65 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live 7:13 PM 5/9/2025
```

Figure 100 Client Code 5

The thread running the function shown in the figure above handles sending the client's guess to the server using the UDP socket. It also sends the response indicating if the client wants to continue or not after the other client disconnects. A small time delay is added to avoid communication issues.



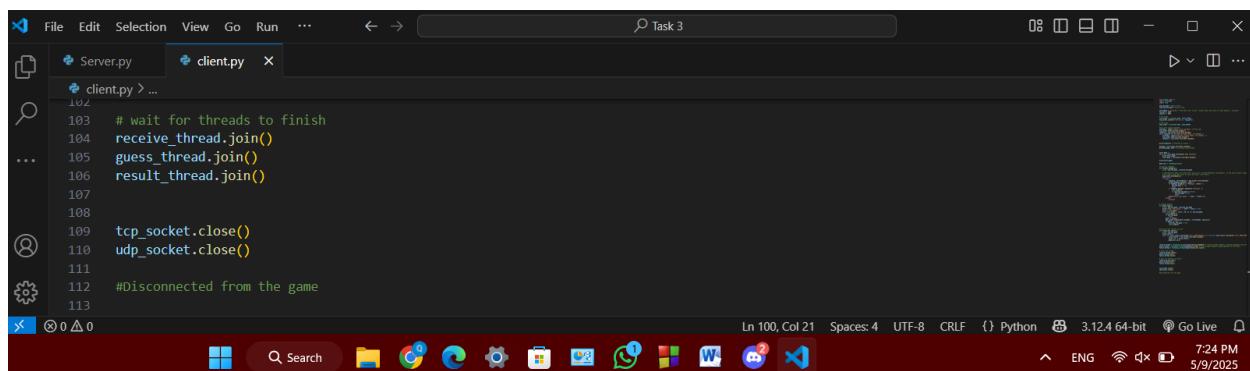
```

File Edit Selection View Go Run ... Task 3
Server.py client.py > receive_feedback
...
83
84     #receive final results via TCP
85     def listen_for_result():
86         global end_the_game
87         final_result=' '
88         while not final_result.startswith('\n==== GAME RESULTS ===') and not final_result.startswith('\n==== GAME OVER ==='):
89             final_result = tcp_socket.recv(1024).decode()
90             print("\n"+final_result)
91             game_over.set()
92
93 Ln 49, Col 65 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live 7:19 PM 5/9/2025

```

Figure 101 Client Code 6

The final thread, running in the function shown in the figure above, waits for a response from the server on its TCP socket indicating that the game has ended and receives the results of the game.



```

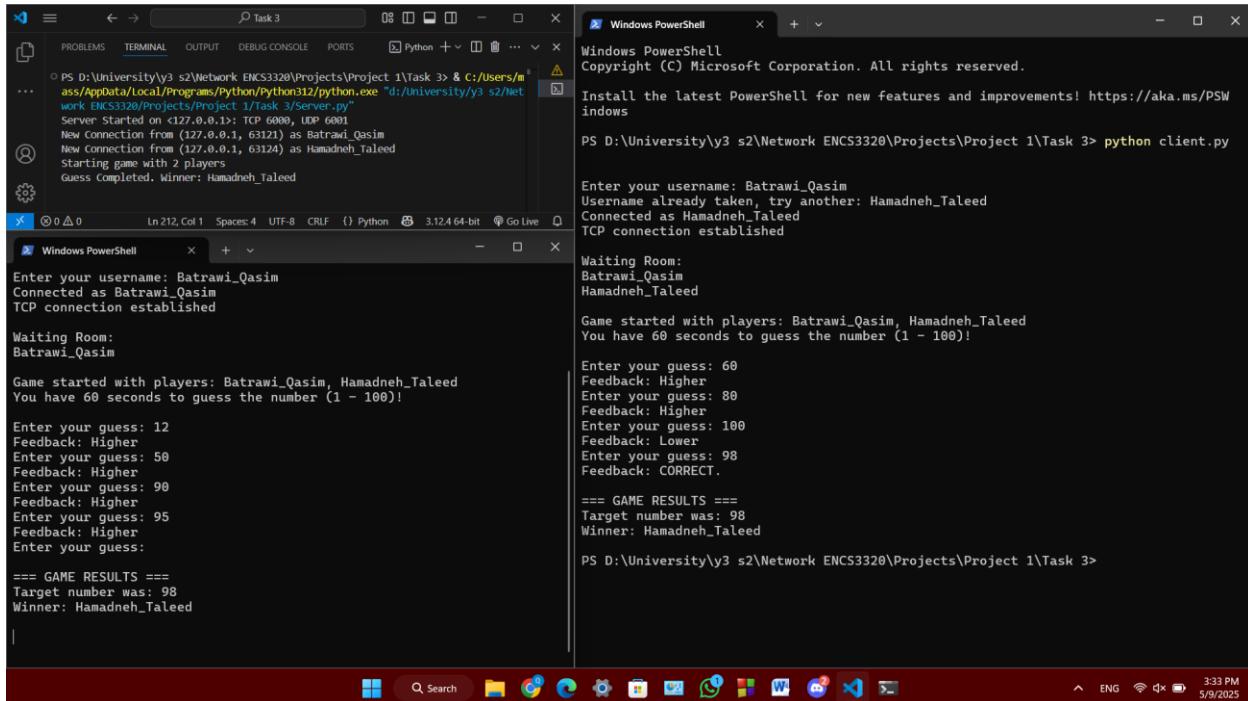
File Edit Selection View Go Run ... Task 3
Server.py client.py ...
...
102
103     # wait for threads to finish
104     receive_thread.join()
105     guess_thread.join()
106     result_thread.join()
107
108
109     tcp_socket.close()
110     udp_socket.close()
111
112     #Disconnected from the game
113 Ln 100, Col 21 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit Go Live 7:24 PM 5/9/2025

```

Figure 102 Client Code 7

Finally, the figure above shows the end of the client code, where the main thread waits for other threads to finish and closes the TCP and UDP sockets.

Test Cases



The screenshot shows a Windows desktop environment with two open PowerShell windows and a taskbar at the bottom.

The left PowerShell window (Task 3) displays the output of a Python server script. It shows the server starting, accepting connections from 'Batrawi_Qasim' and 'Hamadneh_Taleed', and then starting a game with two players. The server waits for another player. It then receives a guess from 'Batrawi_Qasim' (60), which is higher than the target (98). It receives another guess from 'Batrawi_Qasim' (80), which is also higher. The server then receives a correct guess from 'Hamadneh_Taleed' (98), which triggers a 'CORRECT!' feedback. The game results show that 'Hamadneh_Taleed' won.

```
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3> & C:/Users/m...  
ass/AppData/Local/Programs/Python/Python312/python.exe "d:/University/y3 s2/Net  
work ENCS3320/Projects/Project 1/Task 3/Server.py"  
Server Started on <127.0.0.1>; TCP 6000, UDP 6001  
New Connection from (127.0.0.1, 63121) as Batrawi_Qasim  
New Connection from (127.0.0.1, 63124) as Hamadneh_Taleed  
Starting game with 2 players  
Guess Completed. Winner: Hamadneh_Taleed  
Waiting Room:  
Batrawi_Qasim  
Hamadneh_Taleed  
Game started with players: Batrawi_Qasim, Hamadneh_Taleed  
You have 60 seconds to guess the number (1 - 100)!  
Enter your guess: 60  
Feedback: Higher  
Enter your guess: 80  
Feedback: Higher  
Enter your guess: 98  
Feedback: Higher  
Enter your guess: 95  
Feedback: Higher  
Enter your guess:  
==== GAME RESULTS ====  
Target number was: 98  
Winner: Hamadneh_Taleed
```

The right PowerShell window shows the output of a client script. It connects to the server, enters 'Batrawi_Qasim' as the username, and receives a rejection message stating the name is already taken. It then connects as 'Hamadneh_Taleed' and receives a 'TCP connection established' message. The client then sends a guess of 60, receives a 'Higher' feedback, and then a correct guess of 98, which triggers a 'CORRECT!' feedback. The game results show that 'Hamadneh_Taleed' won.

```
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
Install the latest PowerShell for new features and improvements! https://aka.ms/PSW  
indows  
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3> python client.py  
  
Enter your username: Batrawi_Qasim  
Username already taken, try another: Hamadneh_Taleed  
Connected as Hamadneh_Taleed  
TCP connection established  
  
Waiting Room:  
Batrawi_Qasim  
Hamadneh_Taleed  
  
Game started with players: Batrawi_Qasim, Hamadneh_Taleed  
You have 60 seconds to guess the number (1 - 100)!  
Enter your guess: 60  
Feedback: Higher  
Enter your guess: 80  
Feedback: Higher  
Enter your guess: 100  
Feedback: Lower  
Enter your guess: 98  
Feedback: CORRECT.  
  
==== GAME RESULTS ====  
Target number was: 98  
Winner: Hamadneh_Taleed
```

The taskbar at the bottom shows various pinned icons, including File Explorer, Edge, and other system icons.

Figure 103 Test Case 1 – Correct Guess & Invalid Name

The above figure shows a gameplay of two clients. The server opens his TCP and UDP sockets, and waits for clients to connect. The first client has connected with a name **Batravi_Qasim** and waits for another client to connect. The second client connected and tried to enter the same name as client one, but the server rejects it, so she entered the name as **Hamadneh_Taleed**. The game starts and the two clients were trying to guess the number, Hamadneh_Taleed has guessed the number first and wins the game while Batravi_Qasim was still trying, so the game ends immediately and the server sent the results to both clients.

```

Task 3
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE PORTS Python + v ... x
PS D:\University\y3 s2\Network ENC53320\Projects\Project 1\Task 3> & c:/Users/hamadne... /mas.../AppData/Local/Programs/Python/Python312/python.exe "d:/University/y3 s2 /Network ENC53320/Projects/Project 1/Task 3/Server.py"
Server Started on <127.0.0.1>; TCP 6000, UDP 6001
New Connection From (127.0.0.1, 62557) as Batrawi_Qasim
New Connection From (127.0.0.1, 62560) as Hamadneh_Taleed
Starting game with 2 players
No One Wins

Windows PowerShell
Connected as Batrawi_Qasim
TCP connection established
Waiting Room:
Batrawi_Qasim

Game started with players: Batrawi_Qasim, Hamadneh_Taleed
You have 60 seconds to guess the number (1 - 100)

Enter your guess: 10
Feedback: Higher
Enter your guess: 78
Feedback: Lower
Enter your guess: 118
Warning: Out of the range, miss your chance
Enter your guess: 58
Feedback: Higher
Enter your guess: 65
Feedback: Lower
Enter your guess:

==== GAME OVER ====
Target number was: 59
No one wins

|_

```

Figure 104 Test Case 2 – Time Out & Invalid Input

The above figure shows another test case where the clients **Batrawi_Qasim** and **Hamadneh_Taleed** were trying to guess the number. However, the 60 seconds game duration ran out before anyone guessed correctly, so the server ended the game for both clients with no winner. During the gameplay, both clients entered numbers that were out of range, and the server responded with a warning.

```

Task 3
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE PORTS Python + v ... x
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3> & C:/Users/ham/AppData/Local/Programs/Python/Python312/python.exe "d:/University/y3 s2/Network ENCS3320/Projects/Project 1/Task 3/Server.py"
Server Started on <127.0.0.1>; TCP 6000, UDP 6001
New Connection from (127.0.0.1, 62845) as Batrawi_Qasim
New Connection from (127.0.0.1, 62847) as Hamadneh_Taleed
Starting game with 2 players
Batrawi_Qasim disconnected from the game
Guess Completed. Winner: Hamadneh_Taleed

Windows PowerShell
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3> python client.py
Enter your username: Hamadneh_Taleed
Connected as Hamadneh_Taleed
TCP connection established

Waiting Room:
Batrawi_Qasim
Hamadneh_Taleed

Game started with players: Batrawi_Qasim, Hamadneh_Taleed
You have 60 seconds to guess the number (1 - 100)!

Enter your guess: 50
Feedback: Lower
Enter your guess: 30
Feedback: Lower
Enter your guess:
Batrawi.Qasim decided to leave you alone in this game, do you want to continue? yes
Enter your guess: 20
Feedback: Higher
Enter your guess: 25
Feedback: Lower
Enter your guess: 23
Feedback: Lower
Enter your guess: 22
Feedback: Lower
Enter your guess: 21
Feedback: CORRECT.

*** GAME RESULTS ***
Target number was: 21
Winner: Hamadneh_Taleed

PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3>

```

Figure 105 Test Case 3 - Client Disconnection.

```

Windows PowerShell
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3>p
thon client.py
Enter your username: Batrawi_Qasim
Connected as Batrawi_Qasim
TCP connection established

Waiting Room:
Batrawi_Qasim

Game started with players: Batrawi_Qasim, Hamadneh_Taleed
You have 60 seconds to guess the number (1 - 100)!

Enter your guess: 12
Feedback: Higher
Enter your guess: 20
Feedback: Higher
Enter your guess:

```

Figure 106 Test Case 3 - Terminal of Disconnected Client

The figure above shows a gameplay of two clients. One of the clients, Batrawi_Qasim, decided to leave the game after his second guess while the game was running. When he disconnected, the server notified the other client, Hamadneh_Taleed, and asked her if she want to continue the game or not, and she decided to continue guessing and she won the game after that.

```

Task 3
PROBLEMS TERMINAL OUTPUT ...
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3> python client.py
C:\Users\mash\AppData\Local\Programs\Python\Python312\python.exe "d:/University/y3 s2\Network ENCS3320\Projects\Project 1\Task 3\server.py"
Server Started on <127.0.0.1>; TCP 6000, UDP 6001
New Connection from (127.0.0.1, 63072) as Hamadneh_Taleed
Starting game with 2 players
Batrawi_Qasim disconnected from the game
Hamadneh_Taleed decided to end the game
No One Wins

Windows PowerShell
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3> python client.py
Enter your username: Hamadneh_Taleed
Connected as Hamadneh_Taleed
TCP connection established
Waiting Room:
Hamadneh_Taleed

Game started with players: Hamadneh_Taleed, Batrawi_Qasim
You have 60 seconds to guess the number (1 - 100)!

Enter your guess: 12
Feedback: Higher
Enter your guess: 20
Feedback: Higher
Enter your guess:
Batrawi_Qasim decided to leave you alone in this game, do you want to continue? no

== GAME OVER ==
Target number was: 51
No one wins
|
```

Figure 107 Test Case 4 - Client Disconnection

```

Windows PowerShell
PS D:\University\y3 s2\Network ENCS3320\Projects\Project 1\Task 3> python client.py
Enter your username: Batrawi_Qasim
Connected as Batrawi_Qasim
TCP connection established

Waiting Room:
Hamadneh_Taleed
Batrawi_Qasim

Game started with players: Hamadneh_Taleed, Batrawi_Qasim
You have 60 seconds to guess the number (1 - 100)!

Enter your guess: 30
Feedback: Higher
Enter your guess: 40
Feedback: Higher
Enter your guess: |
```

Figure 108 Test Case 4 - Terminal of Disconnected Client

The figure above shows a gameplay of two clients. Similar to the previous game, one of the clients, Batrawi_Qasim, decided to leave the game after his second guess while the game was running. When he disconnected, the server notified the other client, Hamadneh_Taleed, and asked her if he wanted to continue the game or not, and she decided to end the game with no winner.

Alternative Solutions, Issues, and Limitation

All parts of the project are working as expected. As an alternative approach, we could have used a higher-level framework like Flask to simplify some tasks. During development, we faced some issues with multithreading, especially when multiple threads accessed shared data at the same time. To solve this, we created threads responsible for TCP, and other responsible for UDP. We also used some sleep functions to help with synchronization and timing. Other minor challenges came up along the way, but we were able to fix them through testing and debugging. Overall, everything is running smoothly.

Teamwork

Throughout the project, tasks were divided between Taleed Hamadneh and Qasim Batrawi to ensure balanced collaboration. In Task 1, Taleed handled the first five shell commands, while Qasim completed the remaining commands and conducted Wireshark analysis. For Task 2, Qasim designed the main web page, and Taleed worked on the local web page. Within the server-side implementation of Task 2, Taleed was responsible for developing the connection methods, the function to respond to clients, and for splitting the files into appropriate types such as HTML and images. Qasim focused on retrieving the correct files, managing error responses, and implementing redirection handling. Task 3 was completed together during a Zoom meeting, with both members equally contributing to the planning and implementation. The project report was also divided fairly: Qasim wrote the Task 3 section, while Taleed wrote about Task 2. Task 1, as well as the theory and procedure sections, were completed together. This balanced approach led to efficient teamwork and a well-rounded final submission.

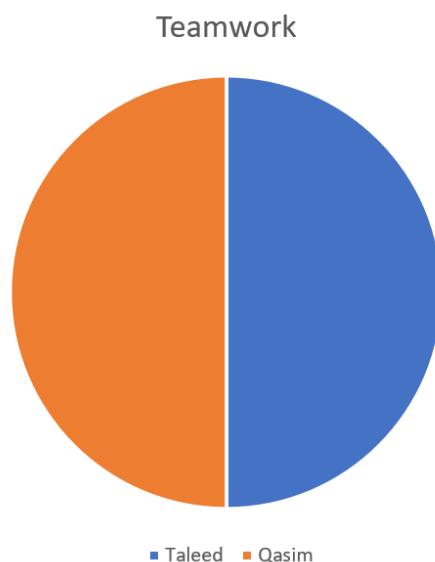


Figure 109 Teamwork chart

References

- [1]: James W. Kurose, Keith W. Ross – Computer Networking_A top-Down Approach Pearson.pdf
- [2]: [Network Component: UDP Socket](#)
- [3]: <https://www.theknowledgeacademy.com/blog/what-is-wireshark/>
- [4]: [GeeksforGeeks | Your All-in-One Learning Portal](#)