Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Networks

ENCS 3320

PNH'S CODE CREATIONS PROJECT

Toqa Abdeen	**1220549**	**section: 5**
Mohammad Ammar	**1212402**	**section: 5**
Omar Diebas	**1210243**	**section: 1**

Submission Date: 1/12/2024

Table of contents:

List of figures

List of tables:

Theory

This project delved into various significant concepts within the realm of computer networks, it is important for several reasons, including:

These essential concepts are used to understand how devices communicate across networks, guarantee secure and accurate data exchange, detect and resolve errors, and make the system secure.

These concepts are as follows:

A. Command:

Use specific instructions or commands on a computer to perform tasks, gather information, or troubleshoot problems.

Some important commandos:

- **Ipconfig**:

This command displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings. This command is mainly used to view the IP addresses on the computers that are configured to obtain their IP address automatically.[1]

The following table lists some important options of the Ipconfig command.

Table 1: The difference between command Ipconfig without parameters and with all parameter

Used without parameters	Displays the IP address, subnet mask, and default gateway for all adapters.
/all	Displays the full TCP/IP configuration for all adapters.

- **Ping:**

used to test connectivity between two hosts. It sends ICMP echo request messages to the destination. The destination host replies with ICMP reply messages. If the ping command gets a reply from the destination host, it displays the reply along with round-trip times.[1]

- **Tracert:**

This command is used to diagnose path-related problems. On an IP network, routers exchange IP packets between the source and the destination. They take IP packets from the source host and forward them in a sequence until they reach the destination host. The sequence of routers between the source and destination is known as the path. A path consists of all routers in a sequence that IP packets sent from the source host traverse to reach the destination host. [1]

- **Nslookup:**

Displays information that you can use to diagnose Domain Name System (DNS) infrastructure. [2]

- **Telnet:**

used for connection and communication with a remote or local host via the Telnet TCP/IP protocol. [3]

B. Socket Programming:

It can be said that it is the basis for network communications in computer networks. Through it, applications are connected across different devices using several protocols, including Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Protocol TCP ensures reliable communication by checking for errors and acknowledging the delivery of data. In order for this protocol to do its work, a connection will be established between the client and the server, unlike protocol UDP, which works to transfer data faster to you without establishing any connection and even without any guarantee of data delivery.

- **TCP:**

  It is a protocol that is primarily responsible for managing the process of exchanging data between devices over the network by ensuring that all data is sent, received, and arranged correctly.

- **UDP:**

  It is a protocol designed for the transmission of data that is sent between devices over the network without verifying delivery or request, with a focus on speed.

Table 2: Key Differences Between TCP and UDP Protocols

Feature	TCP	UDP
Connection Type	Connection-oriented	Connectionless
Reliability	Reliable	Unreliable
Flow Control	Yes	NO
Congestion Control	Yes	NO
Timing	Not provided	Not provided
Throughput Guarantee	Not provided	Not provided
Security	Not provided	Not provided
Speed	Slower	Faster
Use Case	Reliable data transfer (e.g., file transfer, web browsing).	Real-time or lightweight communication (e.g., video streaming, DNS).

C. HTTP:

Stands for **Hypertext Transfer Protocol**, it is an application layer protocol that primarily transfers multimedia documents such as HTML. It is primarily designed for communication between web browsers and a web server to share information over the Internet and can be used for other purposes such as communication between devices.

There are several versions of it, which are:

- **HTTP/0.9**: Basic, single-request protocol.
- **HTTP/1.0**: Introduced headers and multiple request methods.
- **HTTP/1.1**: Improved performance with persistent connections and pipelining.
- **HTTP/2**: Further optimized performance with header compression, multiplexing, and server push.
- **HTTP/3**: Utilizes QUIC protocol for faster, more reliable connections.

Table 3: HTTP version comparisons

Feature	HTTP/0.9	HTTP/1.0	HTTP/1.1	HTTP/2	HTTP/3
Connection	Single request/response	Single request/response	Persistent connections	Persistent connections, multiplexing	QUIC protocol, multiplexing
Headers	No headers	Introduced headers	Improved headers	Header compression	Header compression
Request Methods	Only GET	Multiple methods (GET, POST, HEAD, etc.)	Multiple methods	Multiple methods	Multiple methods
Transport	TCP	TCP	TCP	TCP	UDP (with QUIC)

The references from which we were able to deduce this table will be placed in the References section [4][5][6].

D. DNS:

Domain Name System, it provides the following services:

- hostname to IP address translation.
- host aliasing
    - canonical, alias names.
- mail server aliasing.
- load distribution
    - replicated Web servers: many IP addresses correspond to one name. [7]

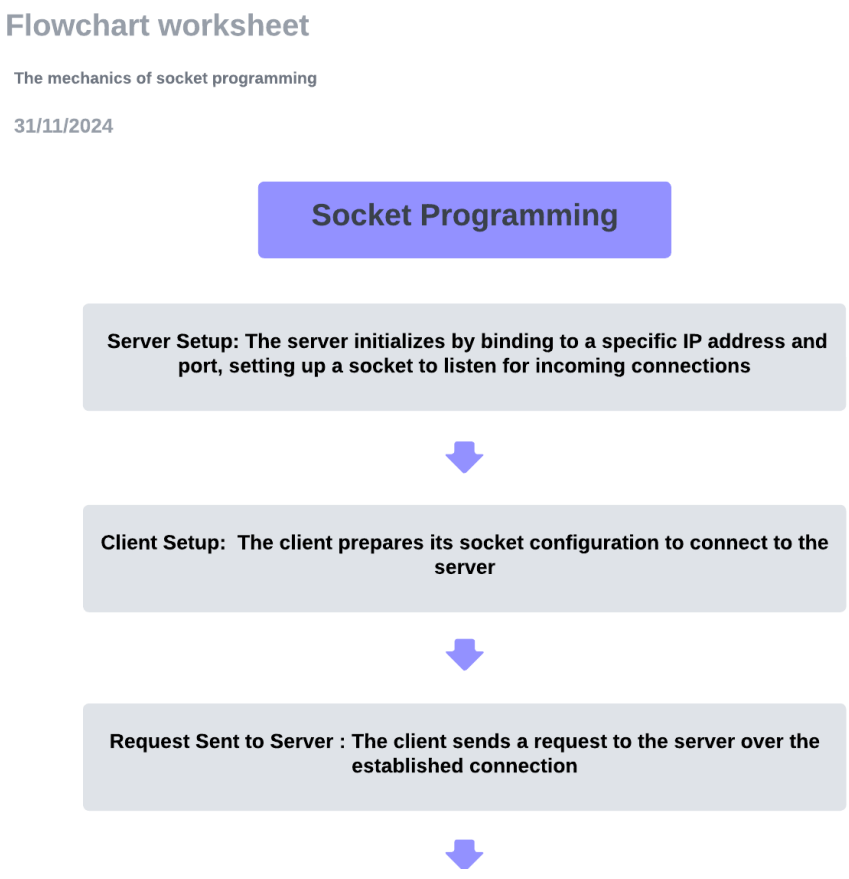
E. Wireshark:

It is a network protocol analyzer that captures and analyzes network packets. It can examine the contents of packets including headers and payloads, providing a clear view of network traffic. It also enables us to detect network problems and understand how network protocols work. It also allows filtering, searching, and displaying packets in different formats, thus providing a deep understanding of network behavior. [8]

Procedure

The project entails multiple key tasks.Each of them requires specific components and ideas to be fully implemented and required. In the first task, several commandos were used, including like Ping, Traceroute, Nslookup, Telnet, and Wireshark, to test the network connection and find out routing problems, verify DNS information, and select remote server connections, and analyze network traffic, respectively. In the socket programming task, client applications were implemented using TCP and UDP to deal with ensuring reliable data transfer, as well as low-latency connections. HTTP was also used to help deliver the web page according to the URL and display it to the client.

The following figure shows how socket programming works:



Client Receives Response: The client accepts the response from the server

Client Processes/Displays Response: The client processes the response and displays it to the user or uses it programmatically

Close Connections: The client or server terminates the connection when no further communication is needed.

Server Receives Request: The server accepts and parses the request from the client

Server Processes Request: The server performs the necessary operations to fulfill the client's request

Server Generates Response: The server creates a structured response containing the requested data or an error message

Response Sent to Client: The server sends the response back to the client via the connection

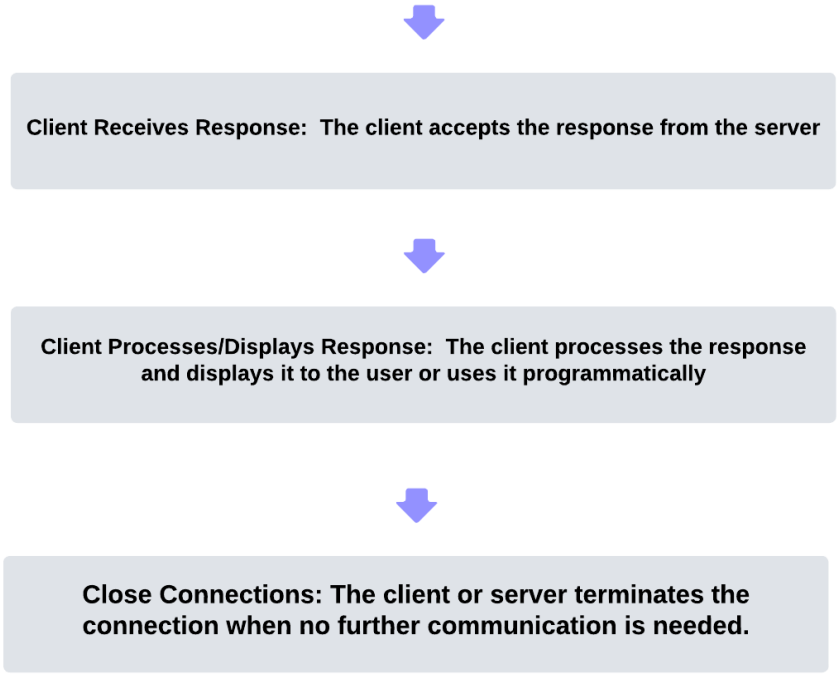


Figure 1: The mechanics of socket programming

Results and Discussions

➢ Task 1:

a. Definition of:

(i) ipconfig, (ii) ping, (iii) tracert, (iv) telnet, and (v) nslookup:

i- Ipconfig /all:

This command displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings.

ii-Ping:

Is a computer network administration software utility used to test the reachability of a host on an Internet Protocol network.

iii- Tracert:

The tracert command prints the path. If all routers on the path are functional, this command prints the full path. If a router is down on the path, this command prints the path up to the last operational router.

iv- Telnet:

The telnet command is used for connection and communication with a remote or local host via the telnet TCP/IP protocol.

v- Nslookup:

The Nslookup command is a DNS lookup utility.

b. Run commands:

1. ipconfig /all :

This command is mainly used to view the IP addresses on the computers that are configured to obtain their IP address automatically.

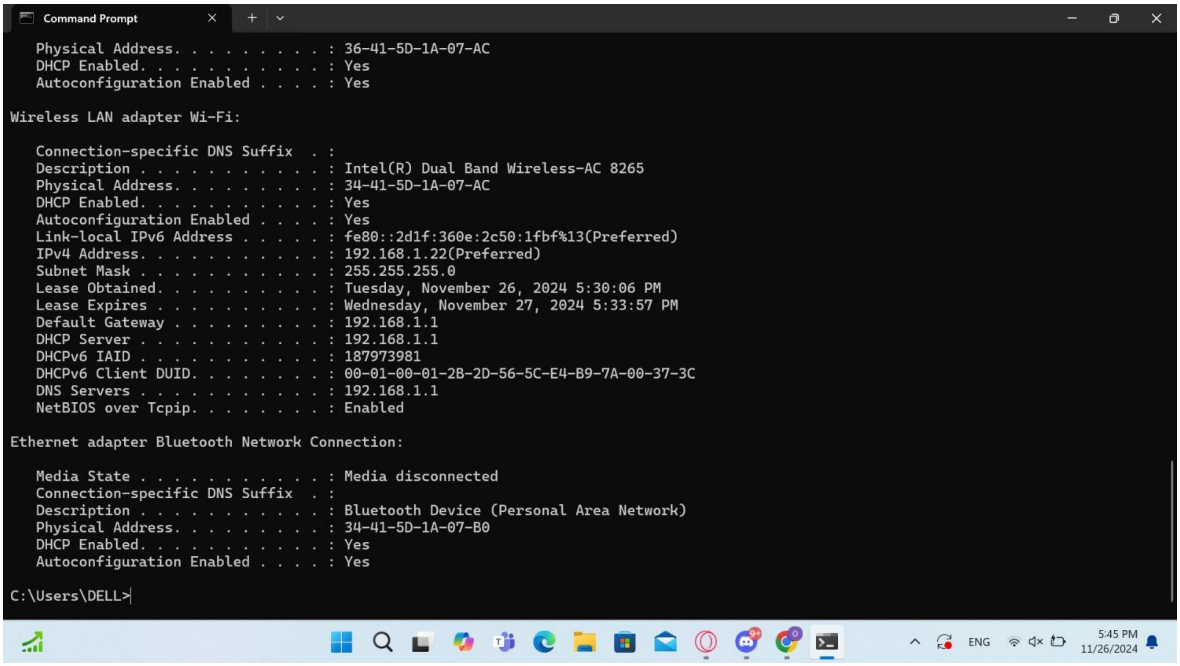


Figure 2: Run ipconfig /all command

2. Ping:

A) Here is the result when we requested from first device to second device by writing ping 192.168.1.22 on the cmd line window.

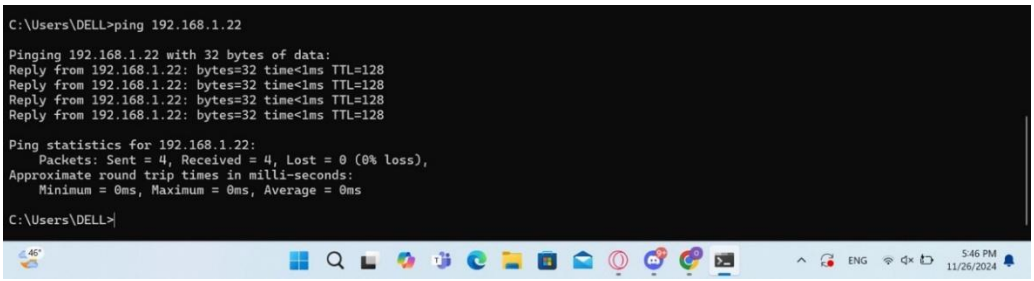


Figure 3: Run ping 192.168.1.22 command

B) ping discover.engineering.utoronto.ca

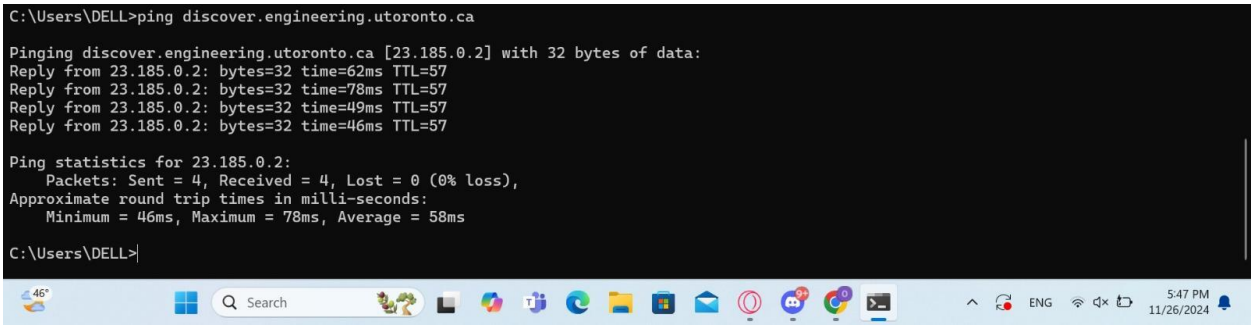


Figure 4: Run ping discover.engineering.utoronto.ca command

The command ping discover.engineering.utoronto.ca sends ICMP echo requests to the server at discover.engineering.utoronto.ca to test its reachability and measure the time it takes for packets to travel to and from the server.

Round trip time (RTT) :

Minimum RTT: 46 ms

Maximum RTT: 78 ms

Average RTT: 58 ms

3. Tracert:

tracert discover.engineering.utoronto.ca

It shows the path of a packet going from your host/computer through each of the individual routes that handle the packet and time required for it to go from one router to another up to the final host/destination.

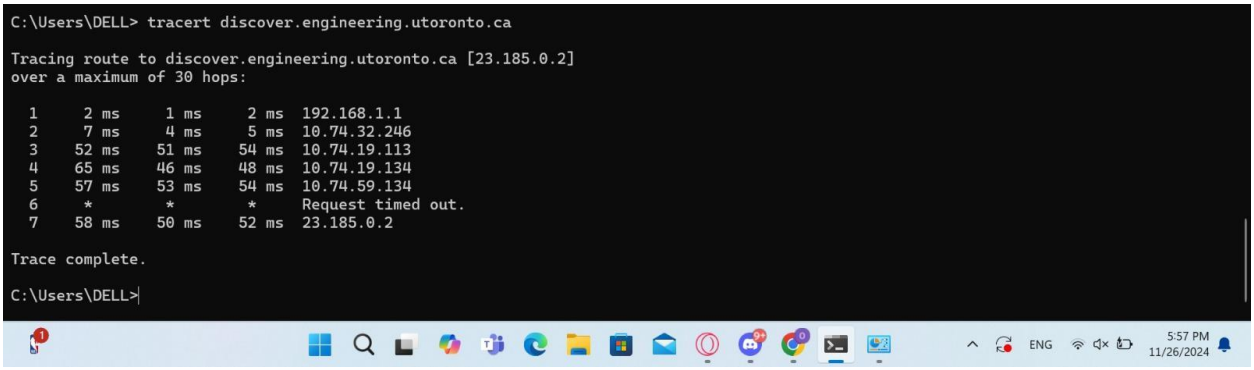


Figure 5: Run tracert command

connection to discover.engineering.utoronto.ca is established successfully. Slight variability in RTT (46–58 ms) is normal, and timeouts mid-route are typically not a concern unless they occur at the final hop.

4. Nslookup:

nslookup discover.engineering.utoronto.ca

The command nslookup discover.engineering.utoronto.ca retrieves the DNS information for the domain. A typical output might look like this:

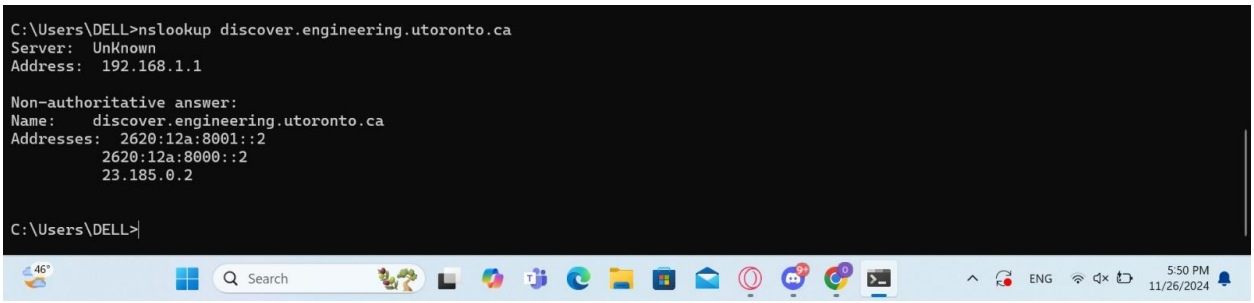


Figure 6: Run nslookup command

It is labeled as UnKnown because it does not have a reverse DNS name associated with it, which is typical for home routers. The DNS server queried is not the authoritative server but is providing a cached response.

5. Telnet:

telnet discover.engineering.utoronto.ca 80

The command telnet discover.engineering.utoronto.ca 80 attempts to establish a connection to the server discover.engineering.utoronto.ca on port 80 (HTTP).

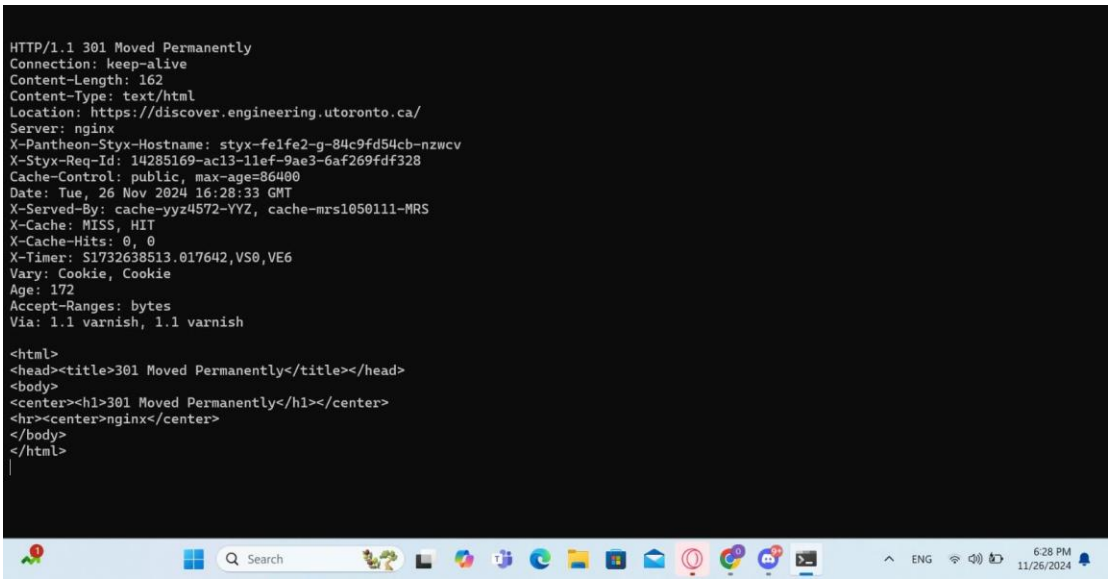


Figure 6: Run telnet command

- The server is redirecting you to a new URL. In this case, it's moving from http:// to https:// (secure connection).
- https://discover.engineering.utoronto.ca/ : This is the new address to which the server is redirecting you.

C. wireshark :

Open Wireshark, then choose the network you're connected to (Wi-Fi or Ethernet). Start Capturing by clicking on the network interface. Open a Browser and type any website (like www.example.com) to make a DNS query. In Wireshark, type dns in the filter bar to only show DNS packets. Look for "Standard query" (your request) and "Standard query response" (the DNS server's reply with the IP address). Stop the capture once you find it .

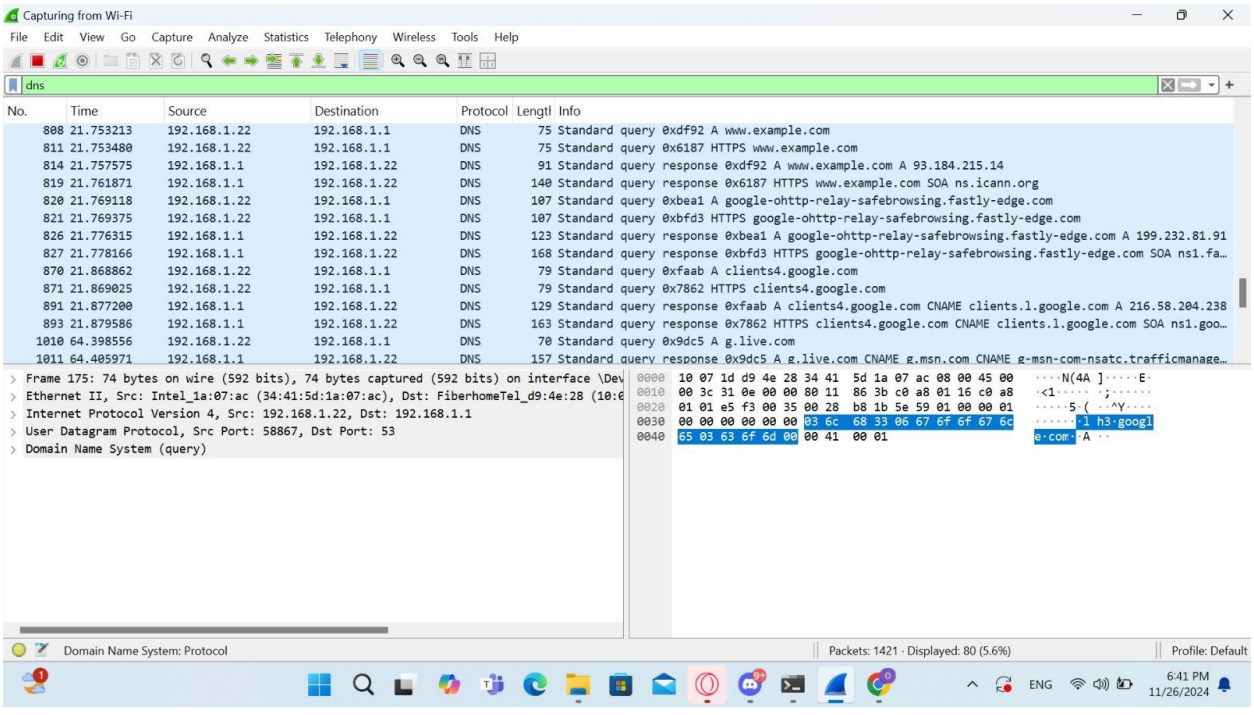


Figure 7: Run Wireshark

➢ Task 2:

In this task, we created a server whose main goal is to listen on port number 5698 using TCP socket programming, through which the mechanism of making requests and responses through the http protocol and working through its methods and also the code status will be clarified.

Also in this task we created two html web pages, each page has its own tasks and elements, each one will be detailed below.

First : SERVER

We created the server for the purposes mentioned previously using the Python programming language. We called two main libraries in it, which are as follows: **socket and os**. First, we used the **socket** library because it allows the server to communicate over the network using transport protocols (**TCP** in this case). Using it, we created and linked the server to a specific port and listened to incoming connections. Also, through it, we can send and receive data, i.e. requests and responses HTTP.

We utilized **sockets** library for socket programming because it provides the tools and functions required to create and manage network connections. Through it, a connection can be created between devices using **TCP/UDP.**

The purpose of using the **os** library is its ability to locate the html files that the server will provide to the client, but its greatest help is in providing assistance to different operating systems.

Imports **urlparse and parse_qs from urllib.parse**,w use function urlparse for its ability to divide the URL into its components, such as the required path or query, while function parse_qs processes the query string resulting from the URL decomposition process. In short, these indications are very important in the process of analyzing the URL to extract its components or parameters from it.

After calling the files, we created several variables as follows:

1. **PORT = 5698**: A variable that stores the port value, which is 5698, as mentioned in the project file.

2. **ADDR = (' ', PORT)**: A variable that contains a set of two parts. The first part is the IP of the server that will be connected to it (it can be any valid IP). The second part contains the port number that was stored in the previous variable (PORT = 5698).

3. **SERVER = socket.socket(socket.AF_INET, socket.SOCK_STREAM)**: A variable through which the usefulness of calling the socket library is demonstrated, which indicates some of the following things:

    - socket.socket: Creating a new socket object that can send and receive data over the network
    - socket.AF_INET: Specifying the address family (IPv4 type)
    - socket.SOCK_STREAM: Specifying the type of socket and connection that should be created

4. **SERVER.bind(ADDR)**: It can be noted here that the server socket (SERVER) will be bound to a specific address for both the IP and port stored in the (ADD) variable.

5. **HEADER = 4096**: Variable to specify the maximum chunk of the data that can be read from the socket.

6. **FORMAT = 'utf-8'**: Character encoding variable used for encryption and decryption (encoding and decoding) for the sender and receiver over the network.

7. **LOCAL_DIRECTORY= os.getcwd()**: A variable we used to set to the current working directory to make the server able to locate and work on files, it was mainly used during image/video search process.

8. **FILES**: Mapping to link URL paths to their contents including both text files (html, css) and image files allows the server to respond correctly to the user when he requests these files stored in it.

9. **notFoundPage**: Variable which stores it in the following way: The content of the notFound.html file is read and stored in this variable.
   The server works as follows:

handle_request:

It is a function whose primary goal is to process client requests on the server from sending and receiving requests. First, it will receive the client request data. If it is present in the files, its response will be as follows: It will respond by displaying the requested content. In this case, the status code will be as follows: HTTP 200 ok. In case of requesting a search page, what happens is the following: After the client enters the name of the file to be searched, the server will check the possibility of the file being in the same directory. If this is confirmed, the server will show the client what was searched for. If it is not found, the server will redirect the client to Google in case of searching for an image. If it is a video, it will redirect it to YouTube. Finally, if the client enters an incorrect extension, page 404 not found will appear to the client. The status code will be 307 Temporary Redirect. When the server has finished responding to each request from the client, a message will appear explaining this. After that, the connection will be closed correctly, that is, after processing the client's request, If any error occurs, a message will appear explaining this, another message will also appear when the connection with the client ends.

handle_404:

A function responsible for dealing with errors 404 not found through which a special page will be created for the client so that the status code will be 404 Not Found. It has been called more than once in the code .

start:

A function that works as follows: It configures the server to listen to connections coming from the client through SERVER. listen(), which will enable the server to accept these connections on the previously defined rescuer. When this process occurs, a message will be printed indicating that it is in the listening phase, and through the loop, the server will continue to wait for new connections. Using SERVER.accept(), a connection can be established with the client that sent a request to the server according to its address stored in addr. After that, the client's request will be processed using the previous function.

Server procedure(ex:request the main_en.html and redirect to the supporting_material_en.html)

I. request the main_en.html:

The main_en.html page can be requested through several links. We will attach all the images that explain this, but we will explain it through one link because it is the same request, meaning the response will be the same.

Links to the main_en.html page:

- **http://localhost:5698/**
- **http://localhost:5698/en**
- **http://localhost:5698/index.html**
- **http://localhost:5698/main_en.html**

A. After starting the server, the following two messages will appear:

- [STARTING] Server is starting...
- [LISTENING] Server is listening on port 5698

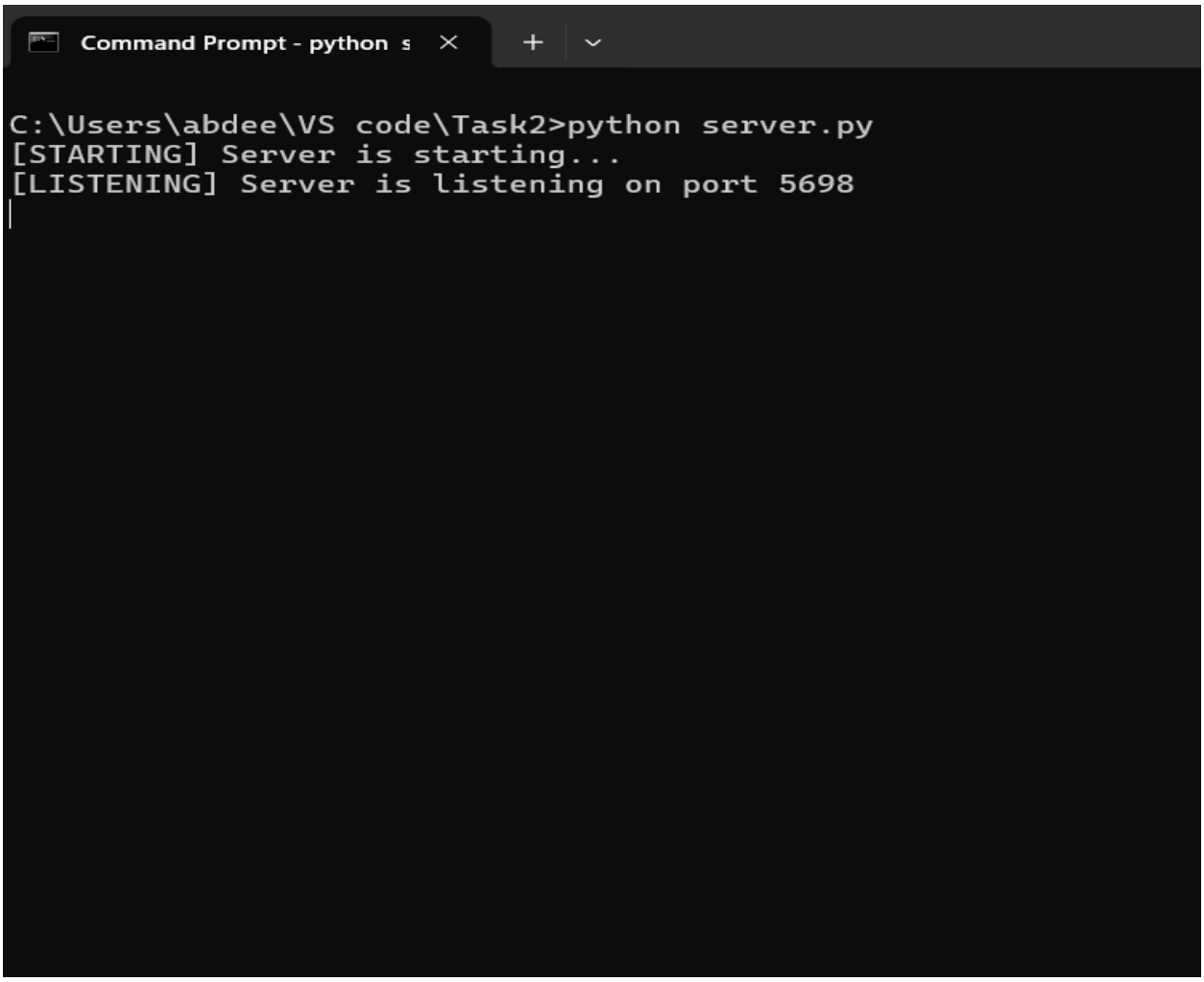


Figure 8: Server Start

The server will remain in this state until a request is sent from the client.

B. When the client orders using one of the previous links, the following will happen:

   The images will be attached first, but the server code has been explained in detail below for easy tracking of events.

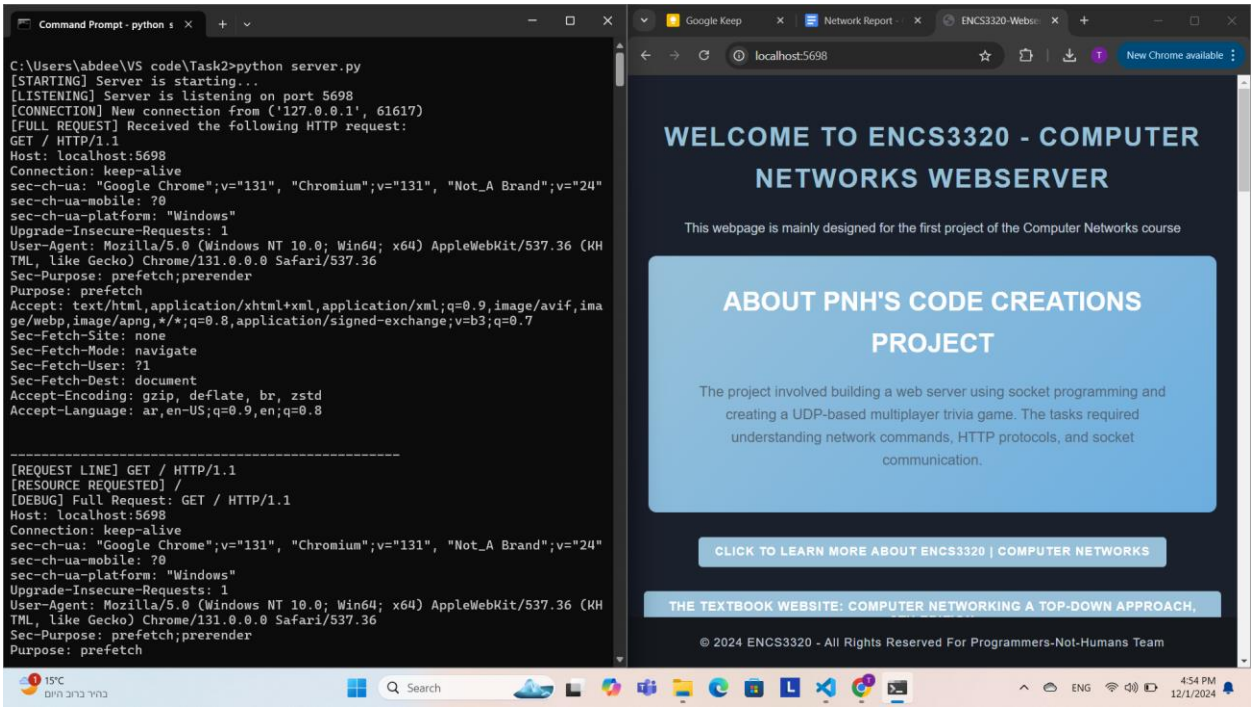


Figure 9: Request main_en web page

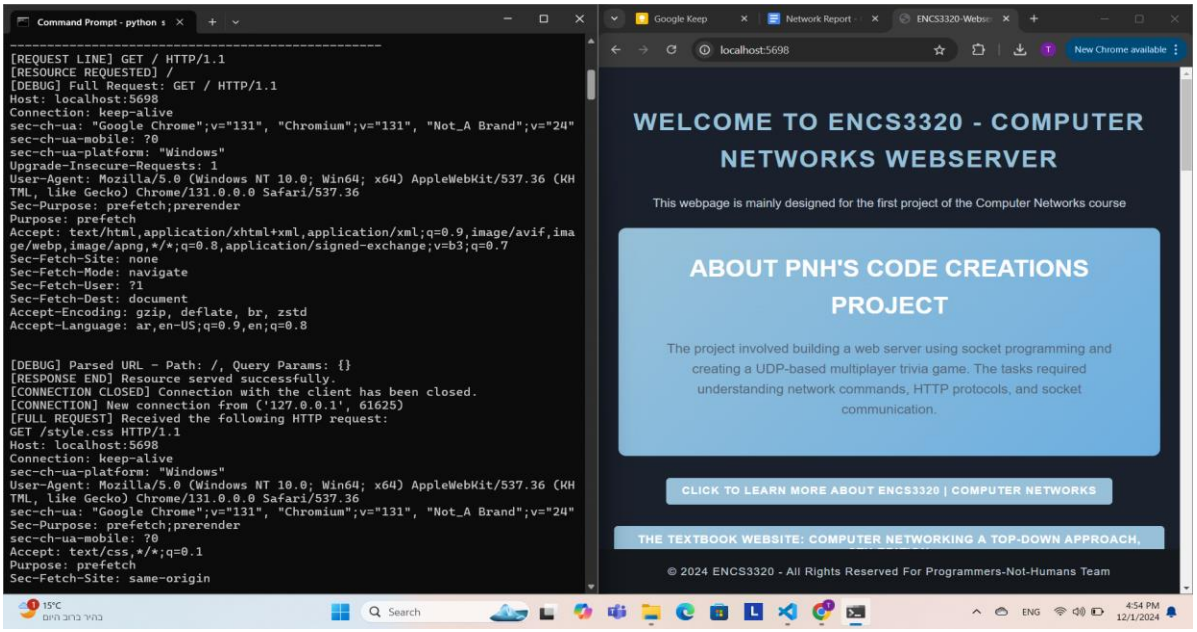


Figure 10: Main_en web page request form 1

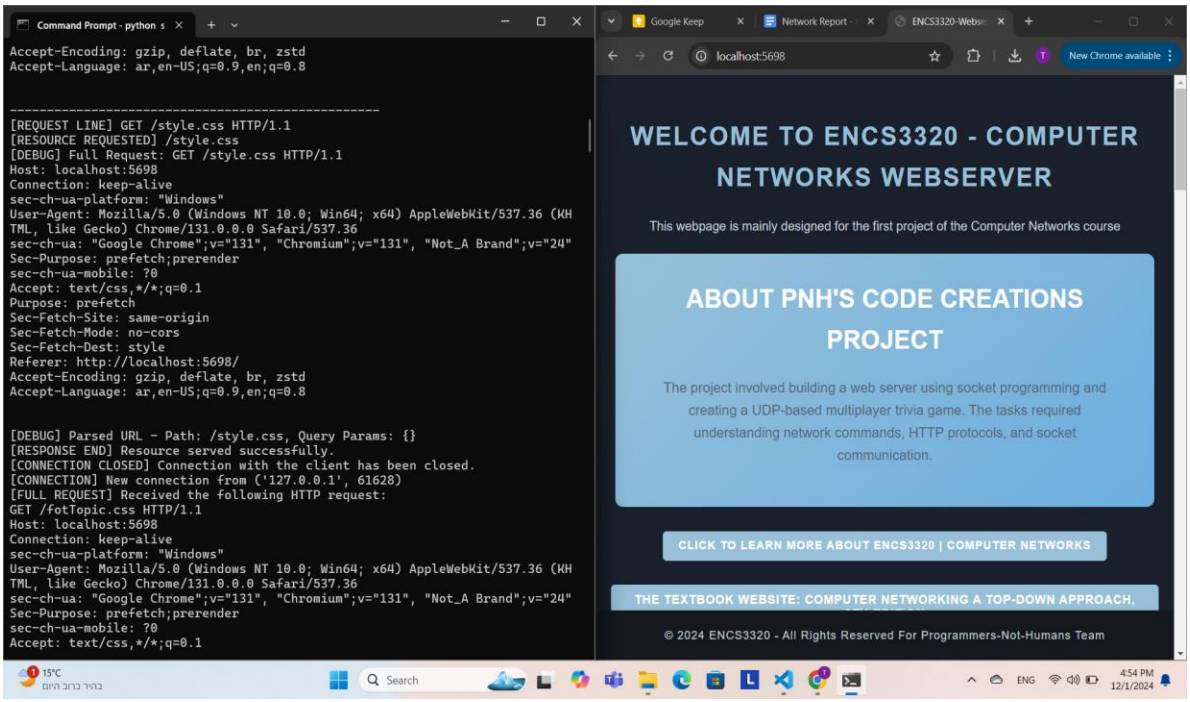


Figure 10: Main_en web page request form 2

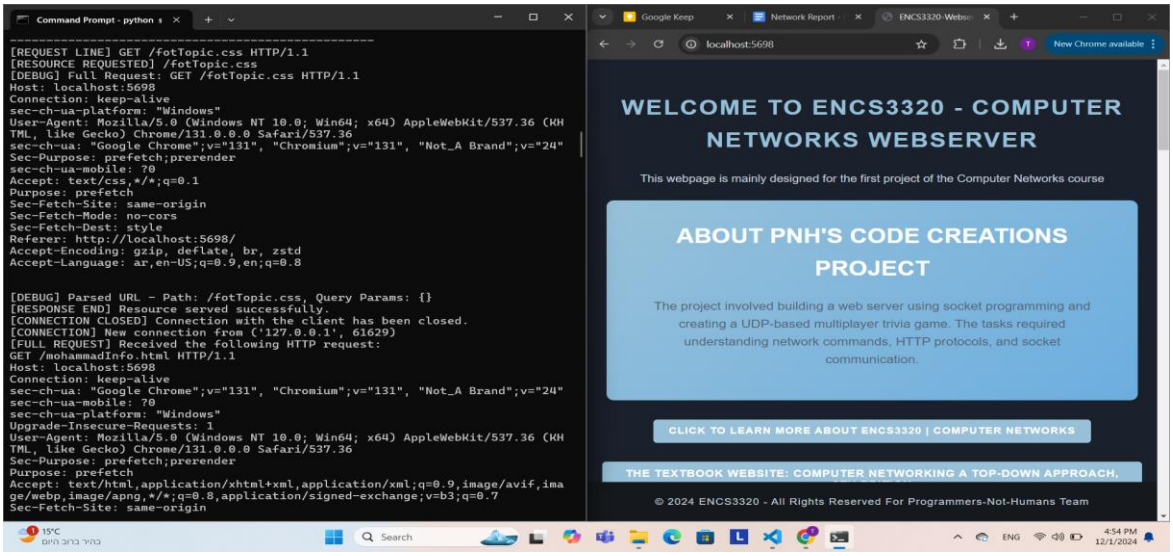


Figure 10: Main_en web page request form 3

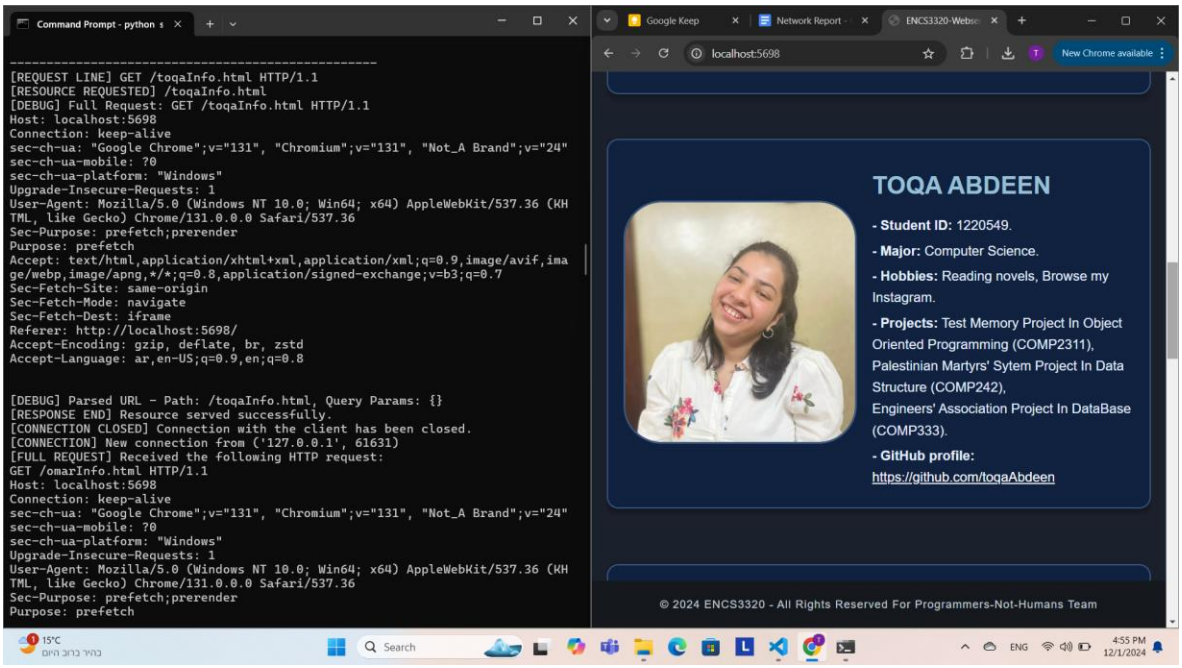


Figure 10: Main_en web page request form 4

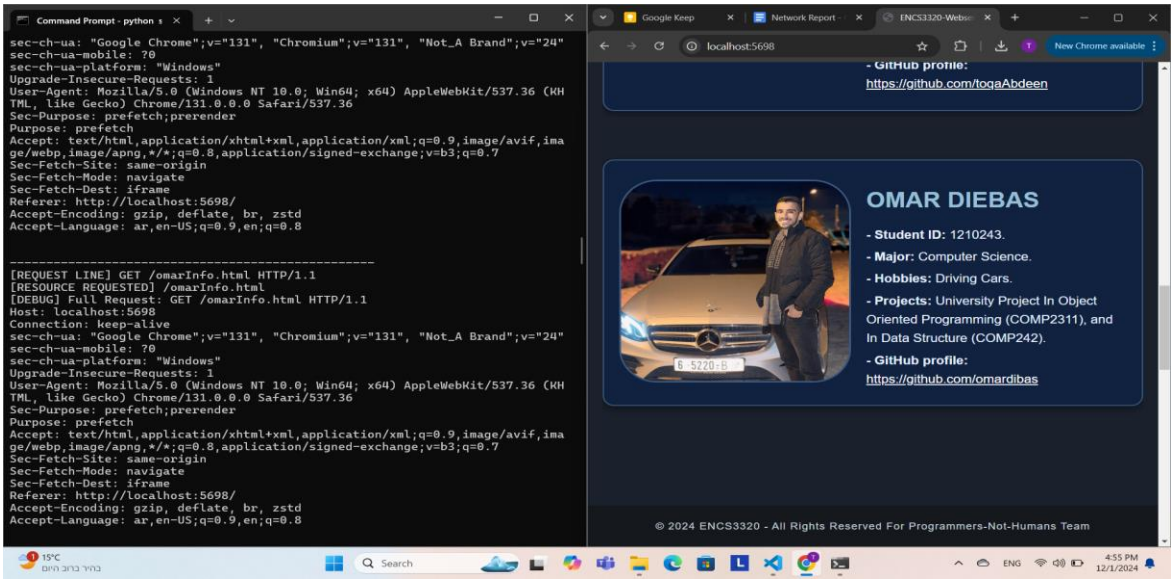


Figure 10: Main_en web page request form 5

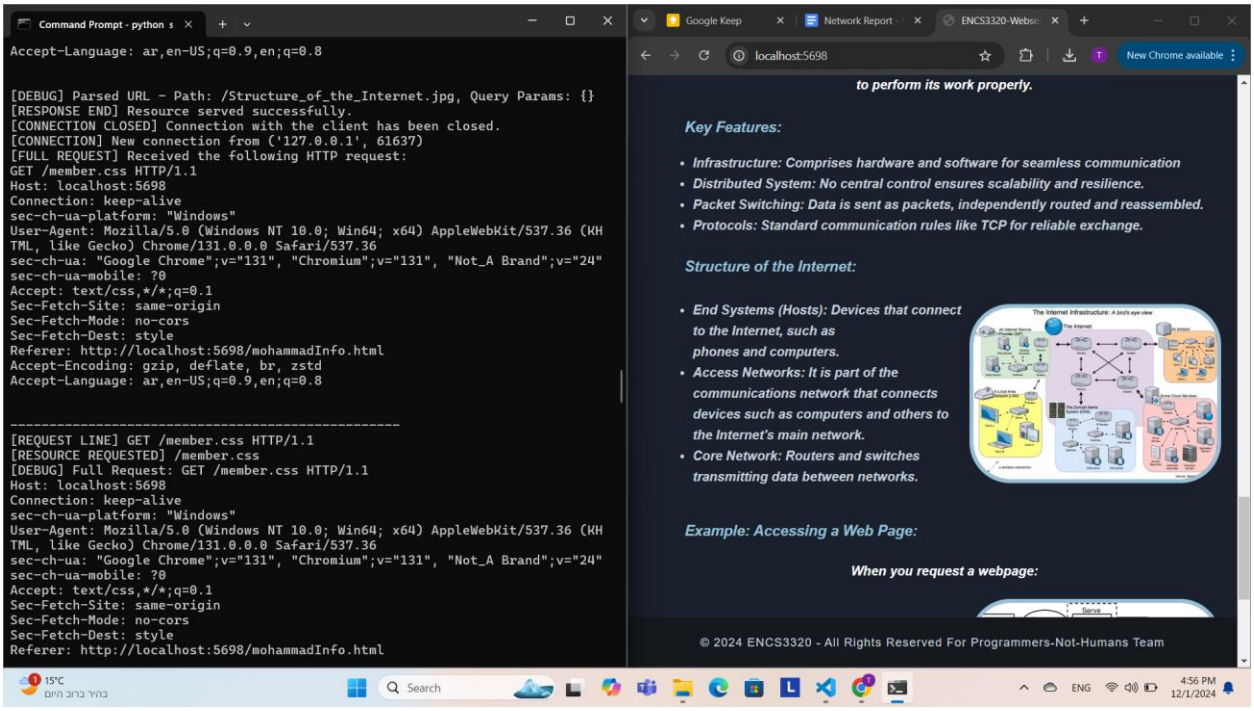


Figure 10: Main_en web page request form 6

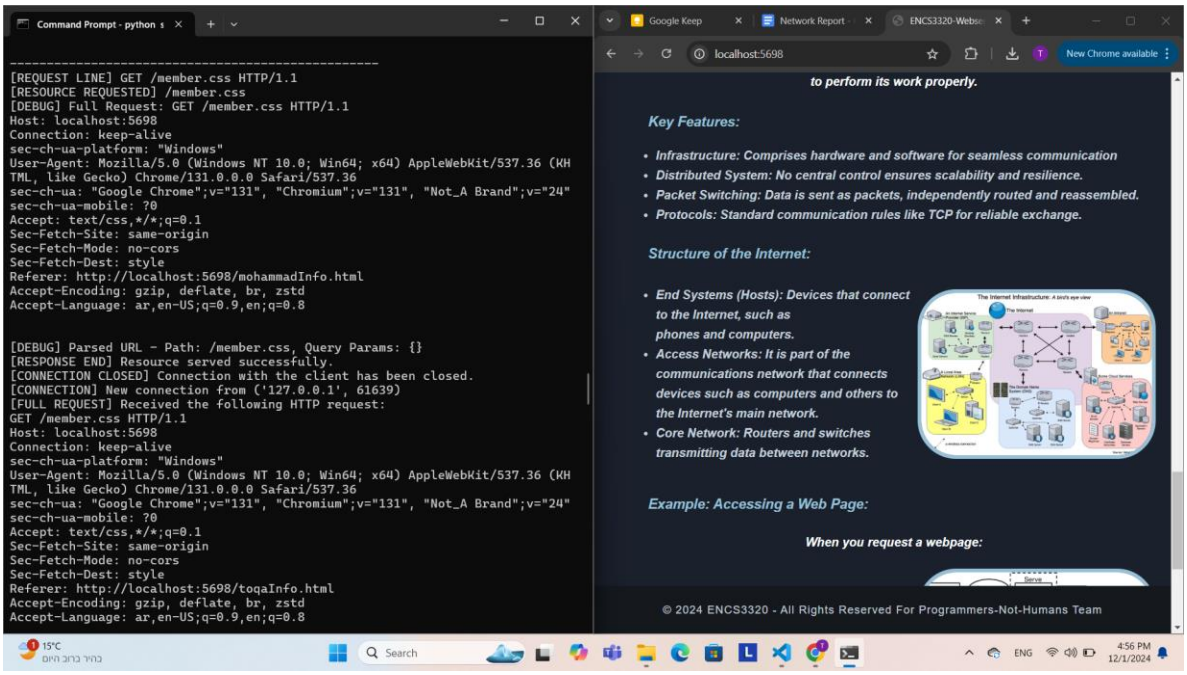


Figure 10: Main_en web page request form 7

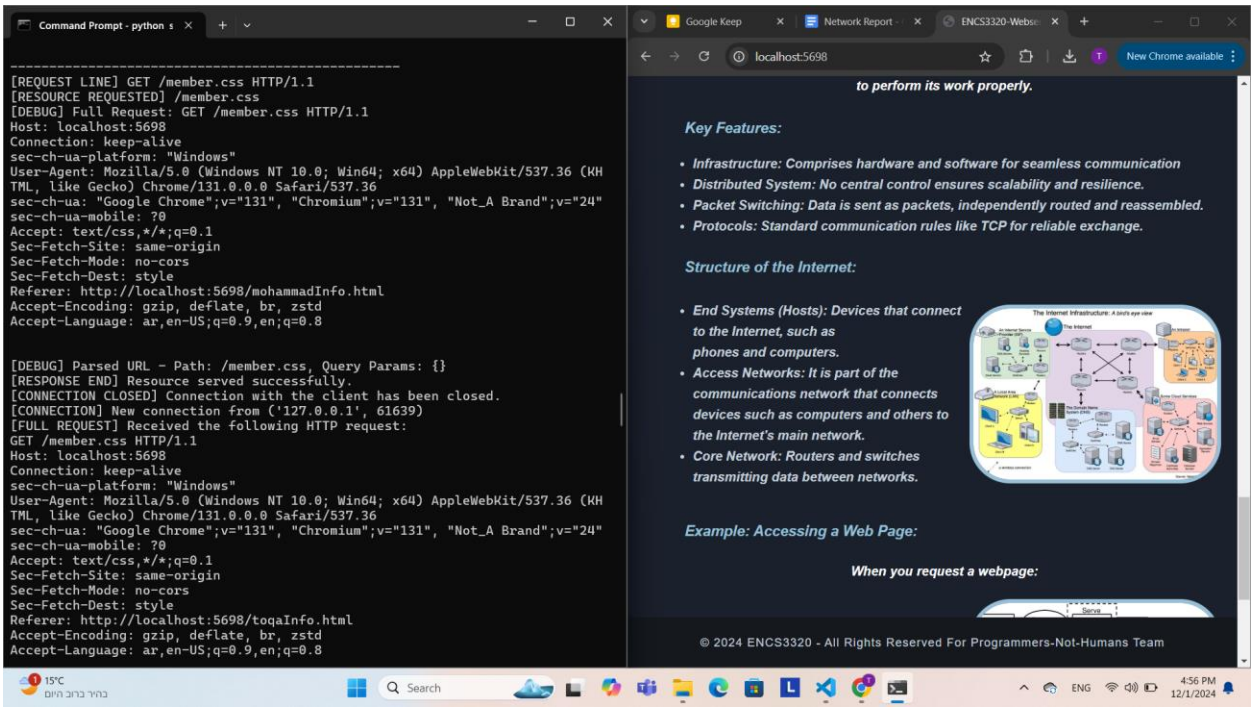


Figure 10: Main_en web page request form 8

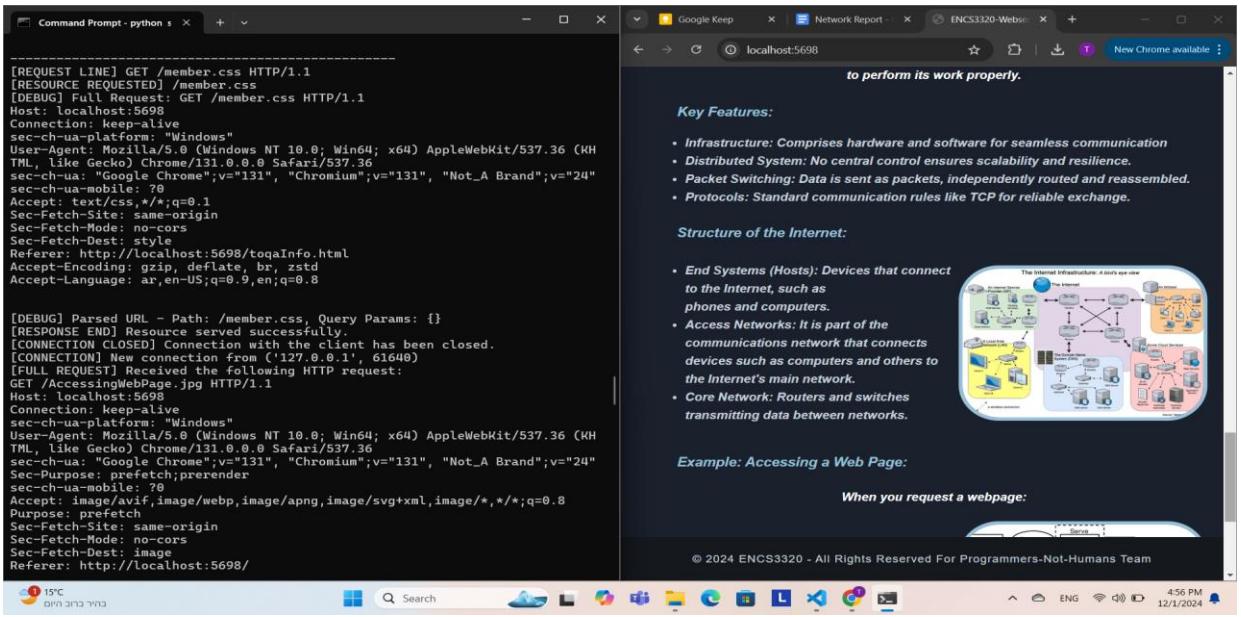


Figure 10: Main_en web page request form 9

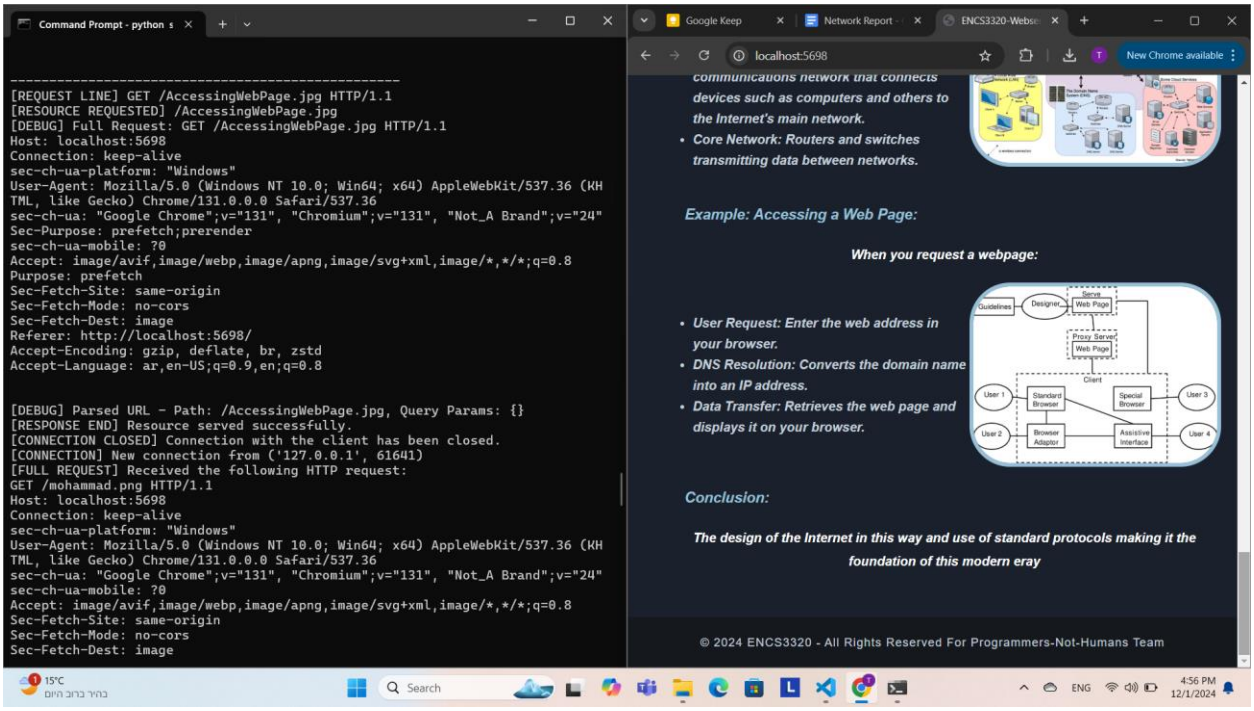


Figure 10: Main_en web page request form 10

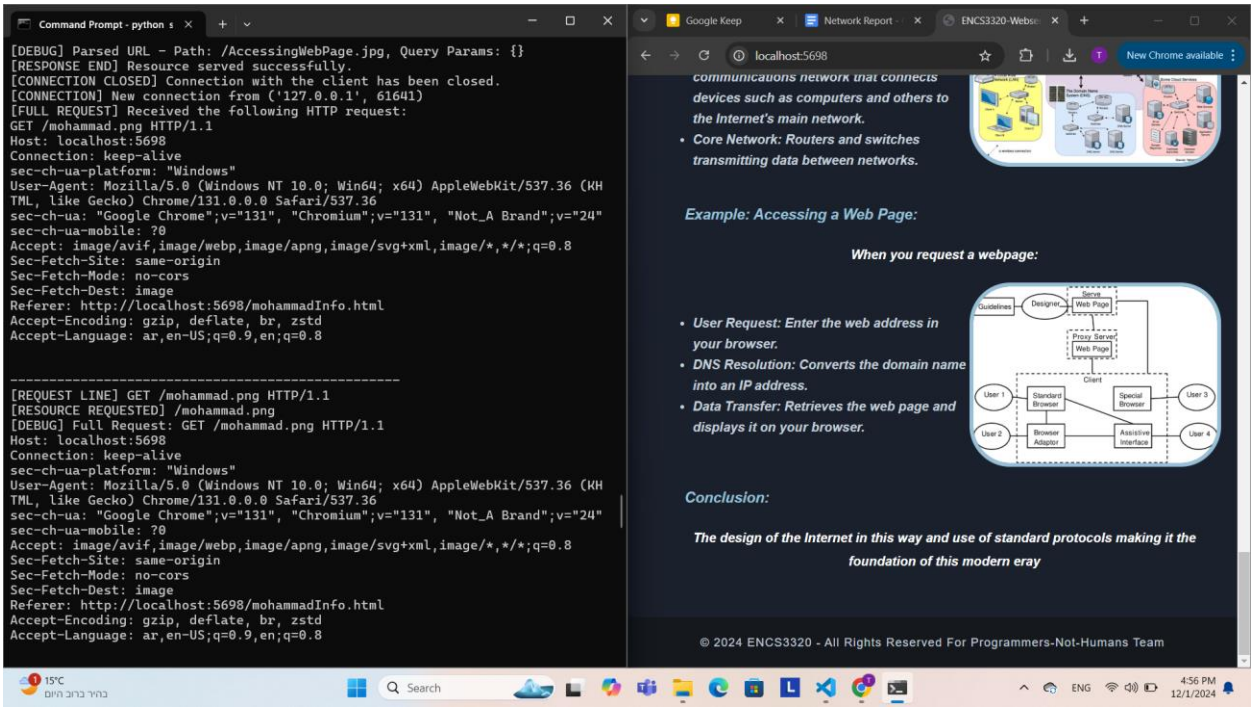


Figure 10: Main_en web page request form 11

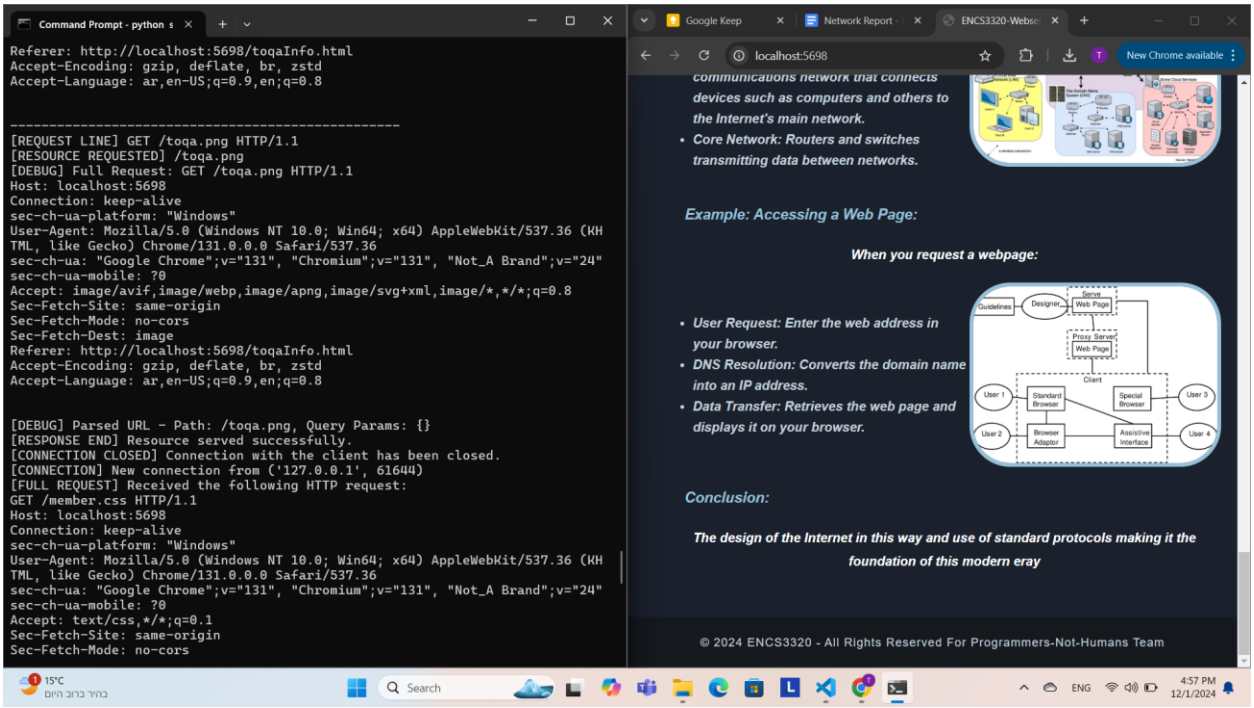


Figure 10: Main_en web page request form 12

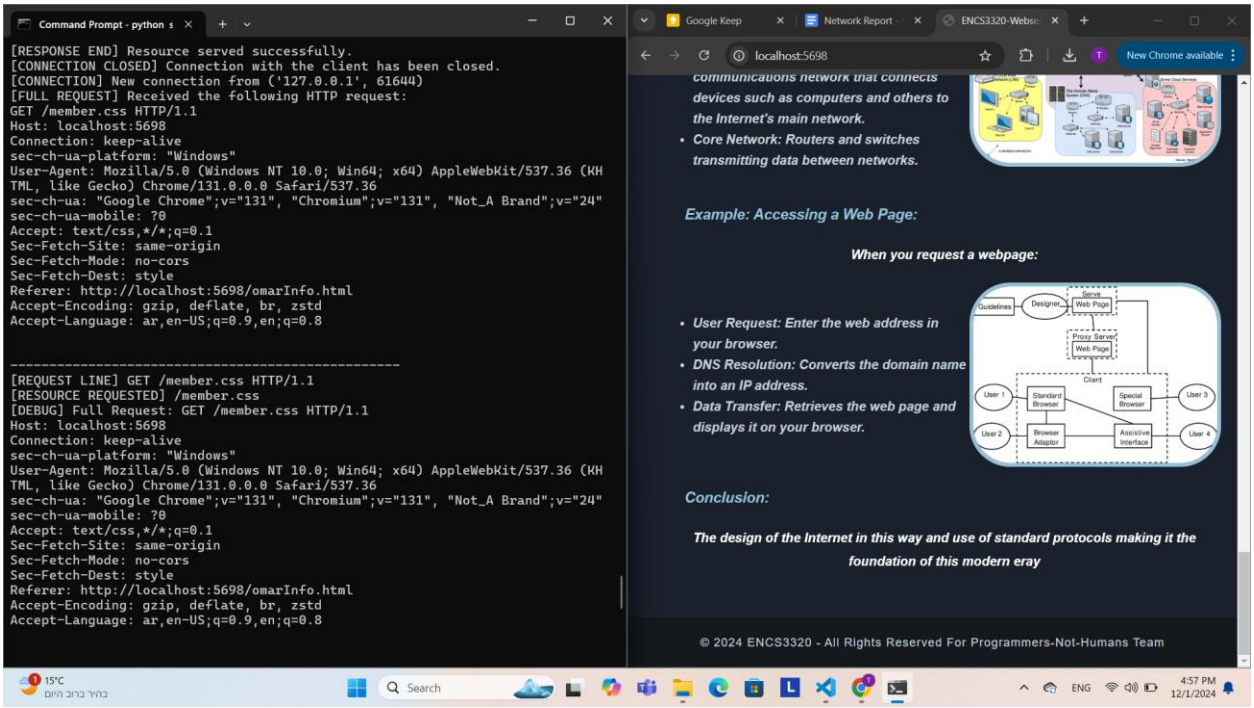


Figure 10: Main_en web page request form 13

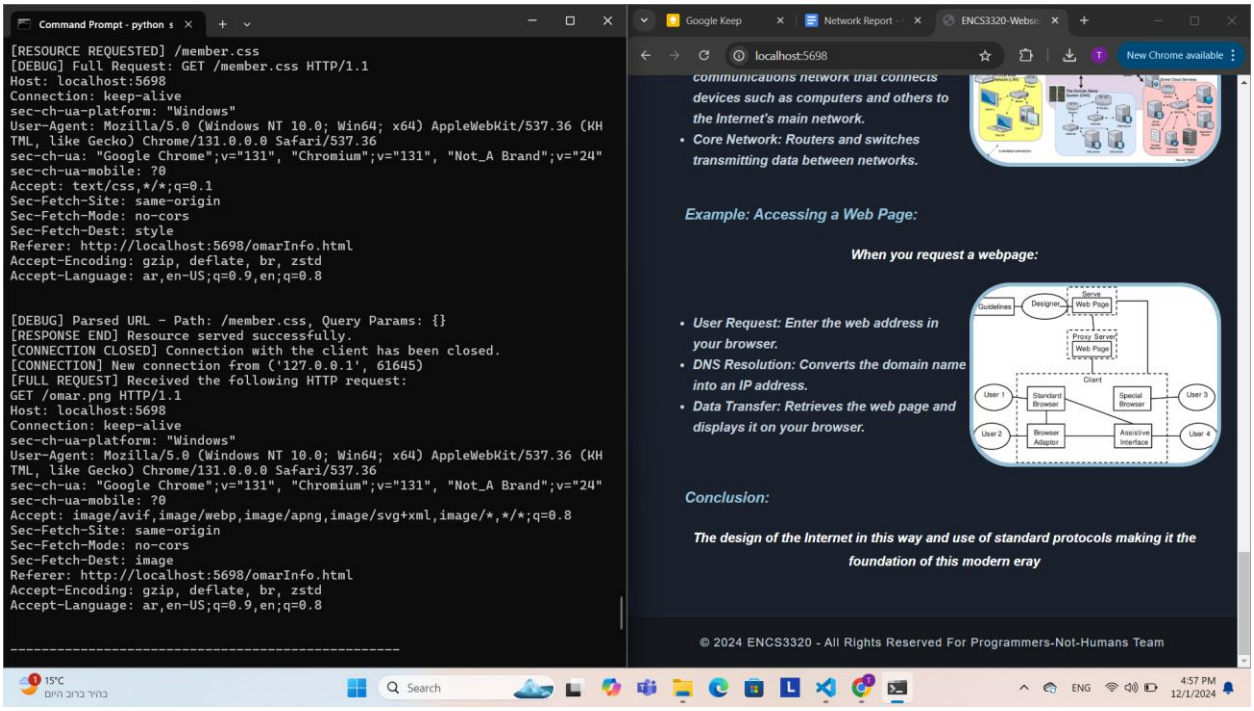


Figure 10: Main_en web page request form 14

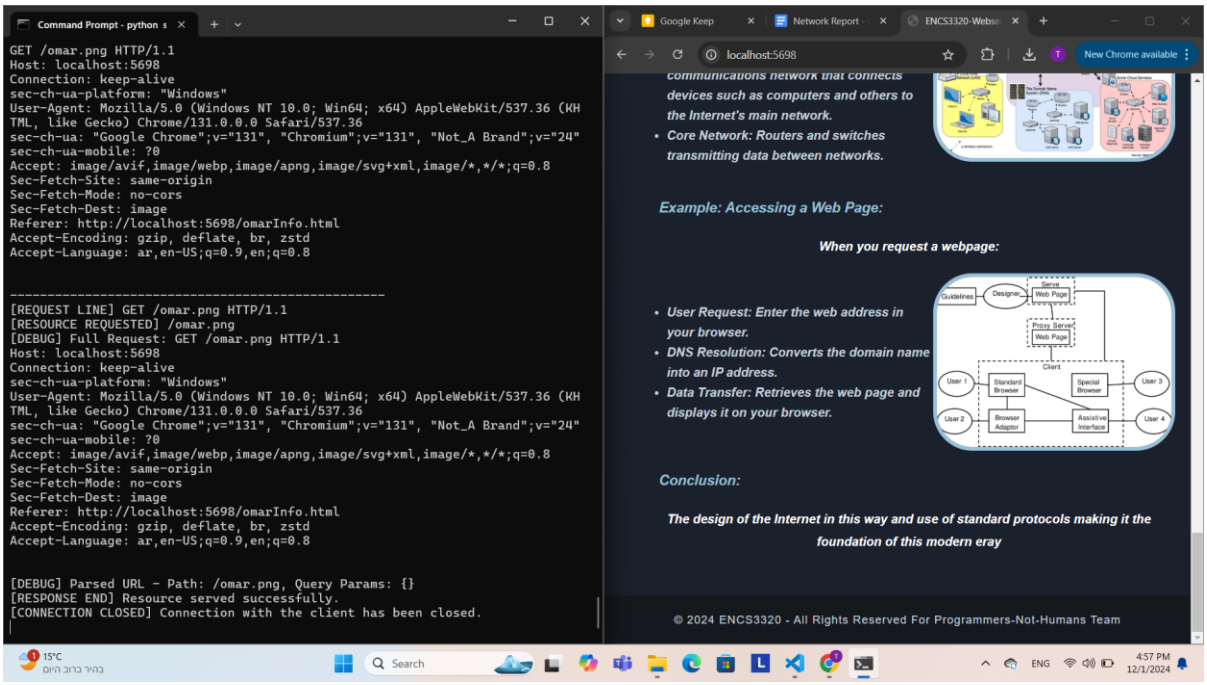


Figure 10: Main_en web page request form 15

1. The start function will be called which will work as explained earlier.
2. As explained earlier, the start function calls a handle_request function inside it, which works as follows: (It will be explained to call the main_en.html page, previously it was explained in general)

- request = conn.recv(HEADER).decode(FORMAT)

➔ The server will receive the client's private data through conn.
➔ This data will be decrypted using the format stored in FORMAT (UTF-8 format).

- print("[FULL REQUEST] Received the following HTTP request:")
- print(request)
- print("-" * 50)

➔ Print sentences whose sole purpose is to print the complete request sent from the customer in an understandable form.

- msg= request.splitlines()[0]
- resource = msg.split(' ')[1]

➔ The server will extract the first line of the HTTP request sent to the server and parse the requested page.

For link: **http://localhost:5698/** (example)

GET / HTTP/1.1

Host: localhost:5698

Connection: keep-alive

…

the first line is extracted. In this case:

msg= "GET / HTTP/1.1"

prase ["GET", "/", "HTTP/1.1"] , this mean

resource = "/"

- print(f"[REQUEST LINE] {msg}")

  - ➔ It will print the first line of HTTP request which includes HTTP method (GET), the requested resource, and HTTP version.
  - ➔ In this case, It will print: [REQUEST LINE] GET / HTTP/1.1

- print(f"[RESOURCE REQUESTED] {resource}")

  - ➔ Here is responsible for printing the requested resource and extracting the first part of the first line that corresponds to the URL.
  - ➔ In this case, It will print: [RESOURCE REQUESTED] /

- parsed_url = urlparse(resource)
- path = parsed_url.path
- query_params = parse_qs(parsed_url.query)

➔ This code was used as follows: urlparse was used to analyze the resource into its components. In our case, the resource is / , urlparse will process the resource and divide it into components. However, in this case, the function

will be for the path / and the query_params will be empty because there are no additional parameters in the request.

(I hope I have conveyed the information correctly.)

- print(f"[DEBUG] Full Request: {request}")
- print(f"[DEBUG] Parsed URL - Path: {path}, Query Params: {query_params}")

➔ Print the request and query_params according to the result of the previous sentences.

- if path in FILES:

➔ In this condition statement it will be checked if the value stored in resources (in this case "/") exists in files and in this case the condition is true then the following will be executed.

- content_type, content = FILES[path]

➔ The value in FILES containing the file type and content is fetched through the content type (e.g., text/html for HTML, text/css for CSS, image/png for PNG, and video/mp4 for MP4), content is the actual content of the file.

- response_headers = (

f"HTTP/1.1 200 OK\r\n"

```
        f"Content-Type: {content_type}\r\n"
        f"Connection: close\r\n\r\n"
    )
```

- ➔ The header for the http response is created in it.
- ➔ HTTP/1.1 200 OK: Through this sentence, we can confirm that the customer's request was completed successfully.
- ➔ Content-Type: {content_type}: Through it, the browser identifies the type of content based on what is in the FILES.
- ➔ Connection: close: The connection will be closed after the response is sent.

- ❖ conn.sendall(response_headers.encode(FORMAT))

- ➔ Sending the response after completion (when the connection is closed according to the previous code, i.e. the response has ended) will be encoded according to the format previously defined in the variable (FORMAT variable)

- ❖ conn.sendall(content if isinstance(content, bytes) else content.encode(FORMAT))

- ➔ Through the above, the client can obtain the file he requested, so that there are two possibilities for the content of this file:

    - ➢ Byte: If it is an image, it will be sent directly without any encoding.
    - ➢ Text: as in an HTML file and a CSS file, it will be encoded into bytes according to the previously defined format before sending.

- print("[RESPONSE END] Resource served successfully.")

➔ When this sentence appears, it can be understood that the resource has been sent to the client successfully.

- if path in ["/supporting_material_en", "/supporting_material__ar"]:

  ➔ Conditional statement, checks if the path (i.e. the client sent a request to page supporting_material_en or supporting_material__ar) contains supporting_material_en or supporting_material_ar and since it isn't applied in this case, it will not be included in the condition.

  (The condition will be discussed in detail in Section (Redirect to the supporting_material_en.html))

- print(f"[ERROR] {e}")

➔ If any error occurs during the process of analyzing the client's request, we will be able to know that by printing it on the screen.

- conn.close()
- print("[CONNECTION CLOSED] Connection with the client has been closed.")

➔ The connection will be closed after the completion of all the process and a sentence will appear explaining this.

- handle_404(conn)

    - ➔ Call this function which will display a 404 not found page when any error occurs.

3. As mentioned earlier, function handle_request calls function handle_404 which works as follows:

    - notFoundForClient= notFoundPage.replace("{{client_ip}}", conn.getpeername()[0])
    - notFoundForClient= notFoundForClient.replace("{{client_port}}", str(conn.getpeername()[1]))

    - ➔ notFoundPage: Through it, page notFound.html will be loaded, which contains the following: client IP address, and client port number, which is considered variable information because it is related to client information.
    - ➔ conn.getpeername(): Through this function we will be able to get the client's IP address and port number.

        - ➢ conn.getpeername()[0] gives the client's IP address.
        - ➢ conn.getpeername()[1] gives the client's port number.

    - ➔ replace(): Through it, information can be shown to the client (client IP address, and client port number) on the notFound page.

    - response_headers =

      ("HTTP/1.1 404 Not Found\r\n"

"Content-Type: text/html\r\n"

"Connection: close\r\n\r\n")

➔ It will function as previously except that the response is different because the resource requested by the client wasn't found.

❖ conn.sendall(response_headers.encode(FORMAT))

➔ The response headers will be sent to the client and encoded according to the defined format to ensure that the headers are sent as bytes.

❖ conn.sendall(notFoundForClient.encode(FORMAT))

➔ The HTML page (notFound.html) and its content will be sent to the client, including the client's IP address and port number.

❖ print(f"[NOT FOUND] Resource {resource} not found")
❖ print("[RESPONSE END] 404 response sent successfully.")

➔ Sentences presented when response is complete.

C. This process will be repeated until all elements of the HTML page are sent and the client will be able to visit the web page (main_en.html) with all its contents.

The subsequent page will be presented to the client.

Through the previous, all the details of the code that will be attached in a file with the project were explained.

II. Redirect to the supporting_material_en.html:

The scenario is as follows: The client sent a request to access the main_en.html page and through the page clicked on the link to access the supporting_material_en.html page (the code was discussed in detail before requesting the supporting_material_en.html page previously), meaning that he sent a new request as follows:

request will be like this:

GET /search.html HTTP/1.1

Host: localhost:5698

Connection: keep-alive

…

msg = GET /search.html HTTP/1.1

resource = /search.html

path = /search.html

query_params = empty value {}

Their values were determined as previously explained.

The following images illustrate the client's search process for the following files:

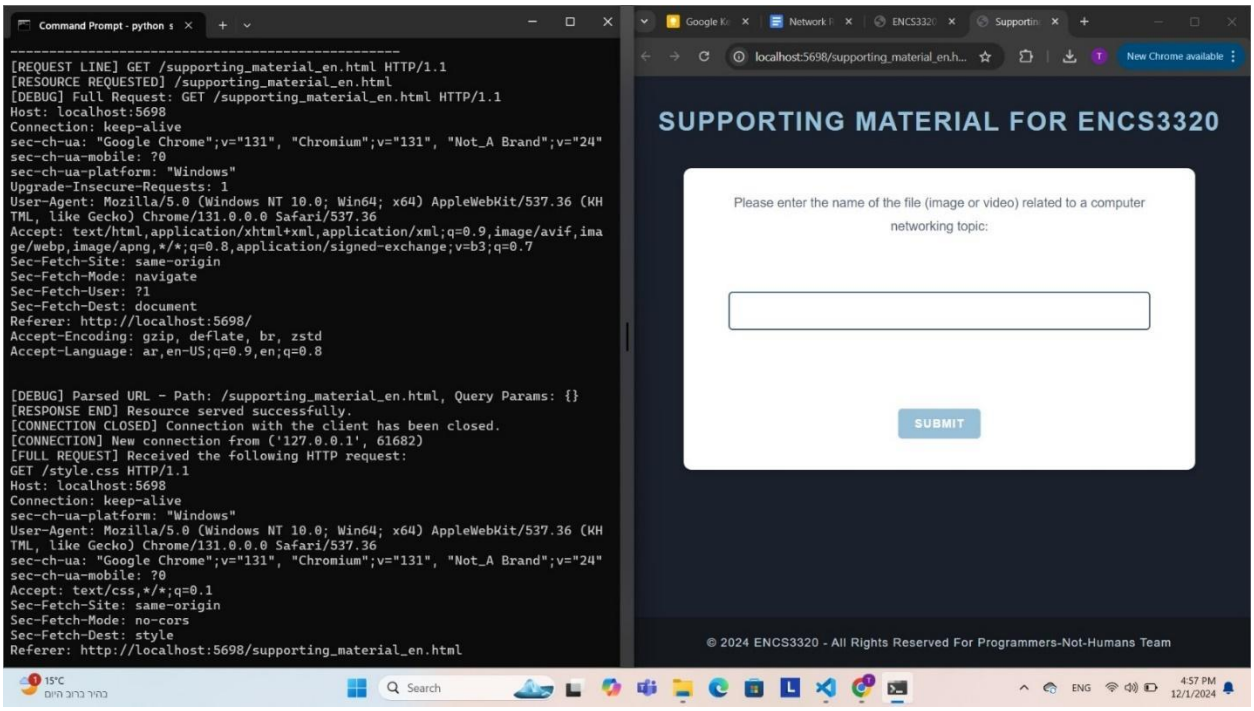


Figure 11: Supporting_matirial_en web page request form 1

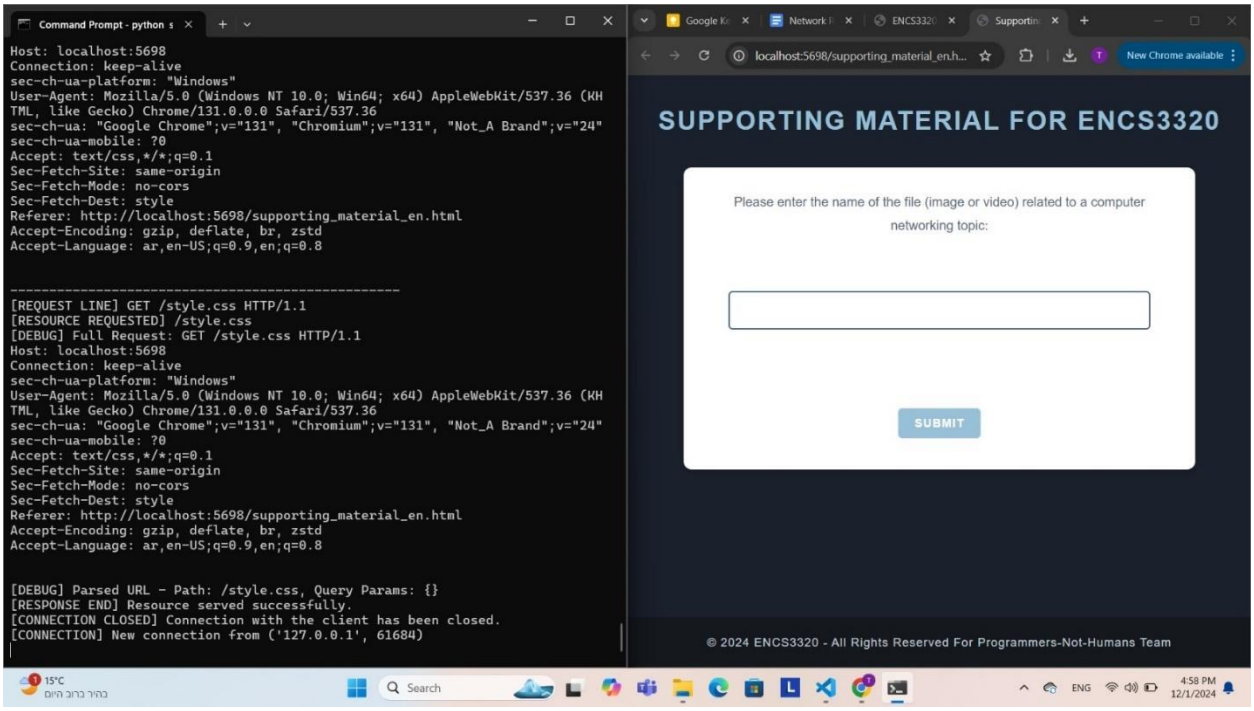


Figure 11: Supporting_matirial_en web page request form 2

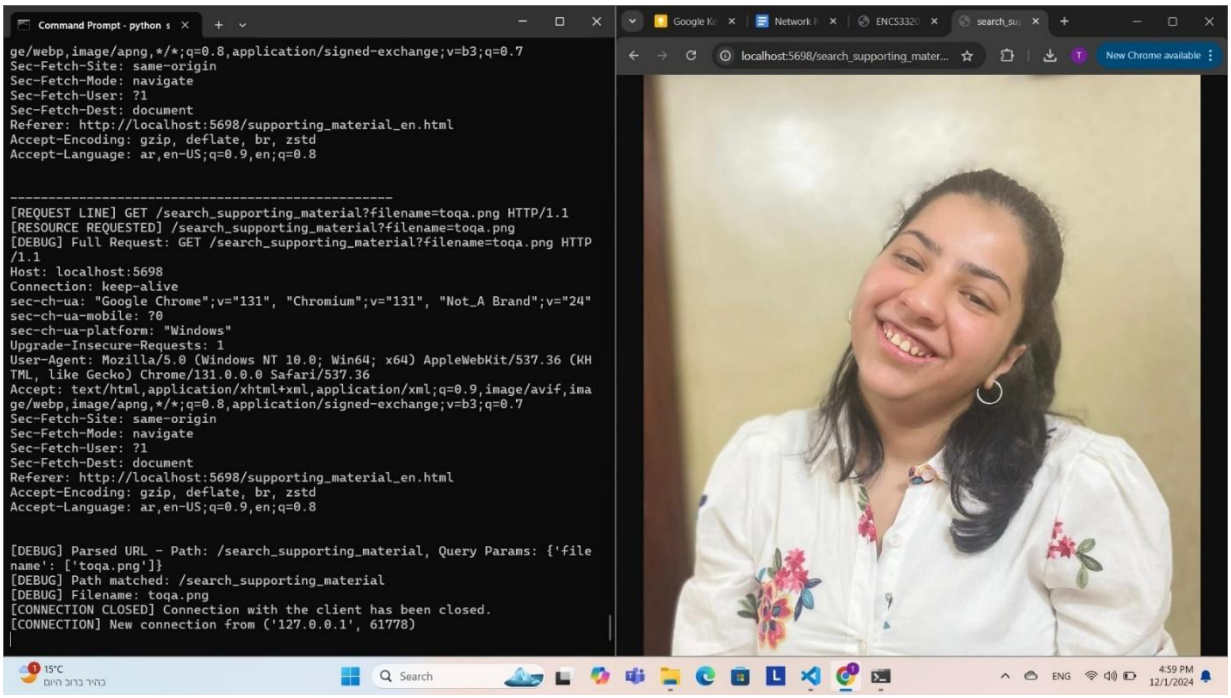


Figure 11: Supporting_matirial_en web page request form 3- request toqa.png

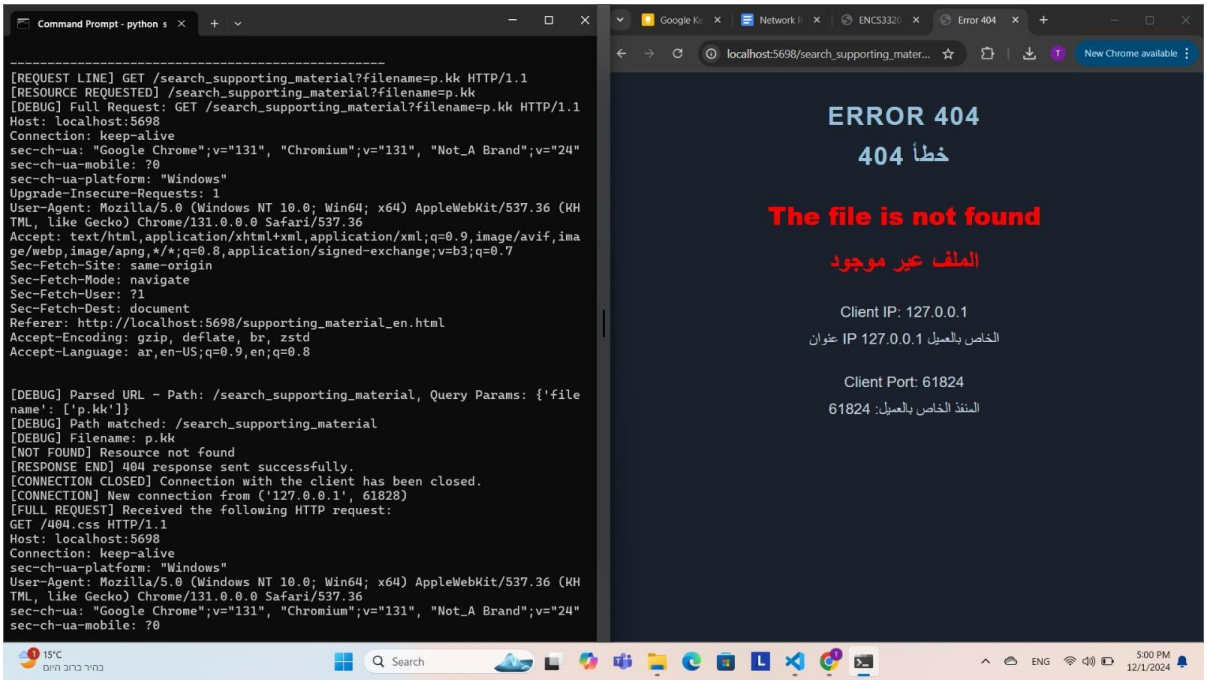


Figure 11: Supporting_matirial_en web page request form 4- request p.kk

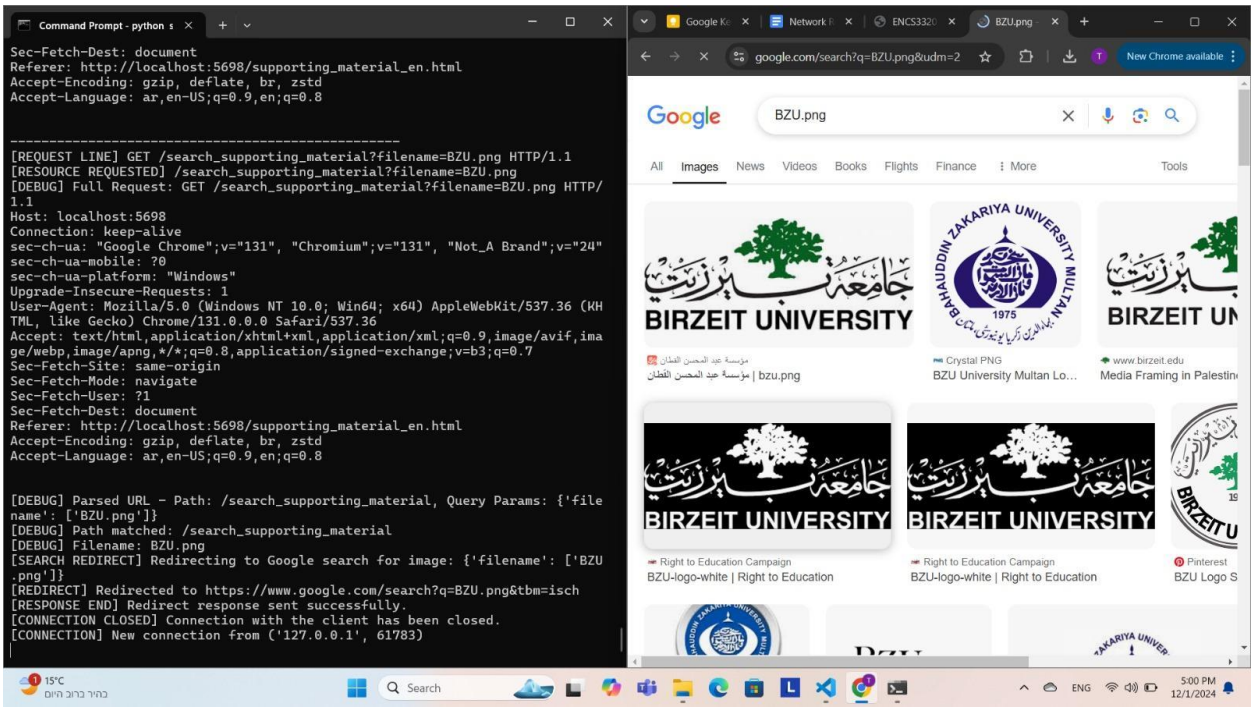


Figure 11: Supporting_matirial_en web page request form 5 – request BZU.png

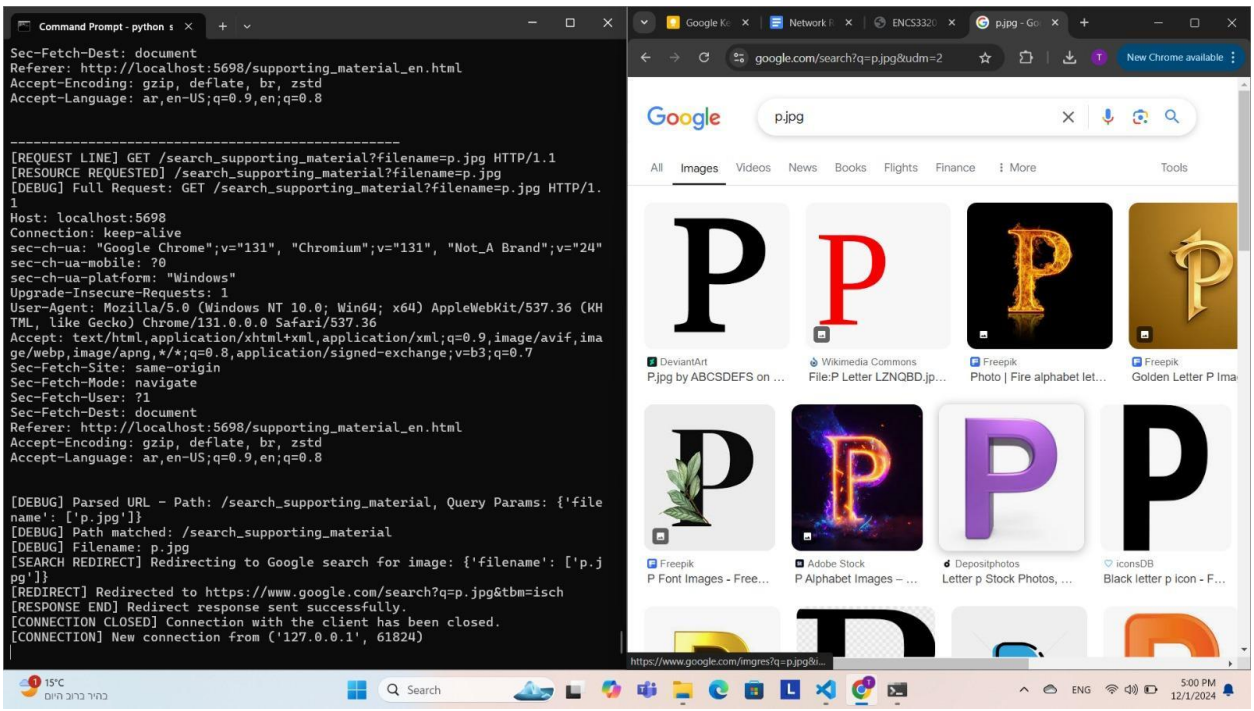


Figure 11: Supporting_matirial_en web page request form 6 -request p.jpg

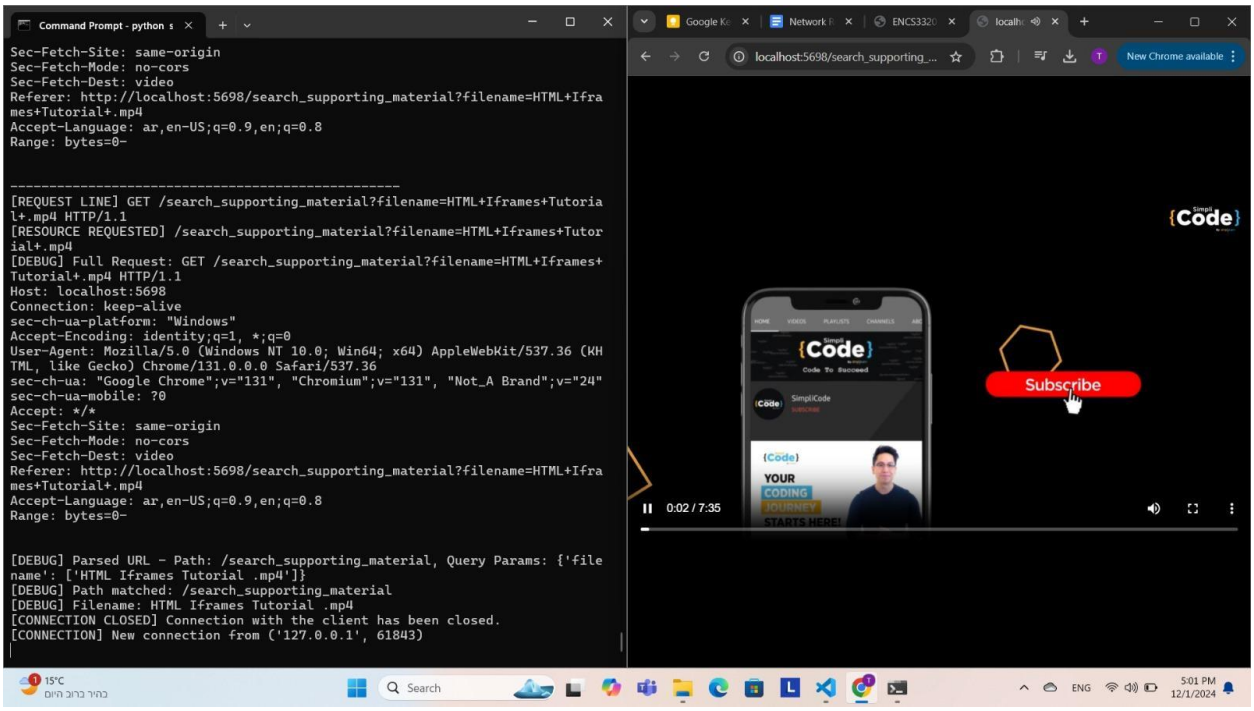


Figure 11: Supporting_matirial_en web page request form 7 – request HTML Iframe Tutorial .mp4

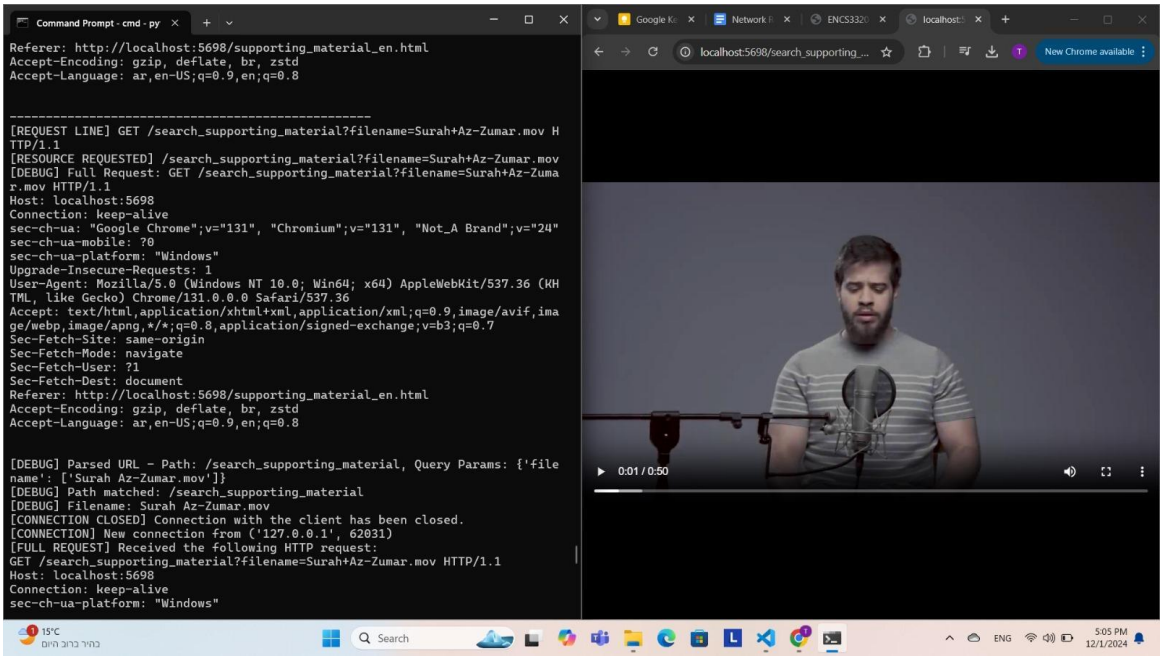


Figure 11: Supporting_matirial_en web page - request form 8 -request Surah Az- Zumar.mov -1

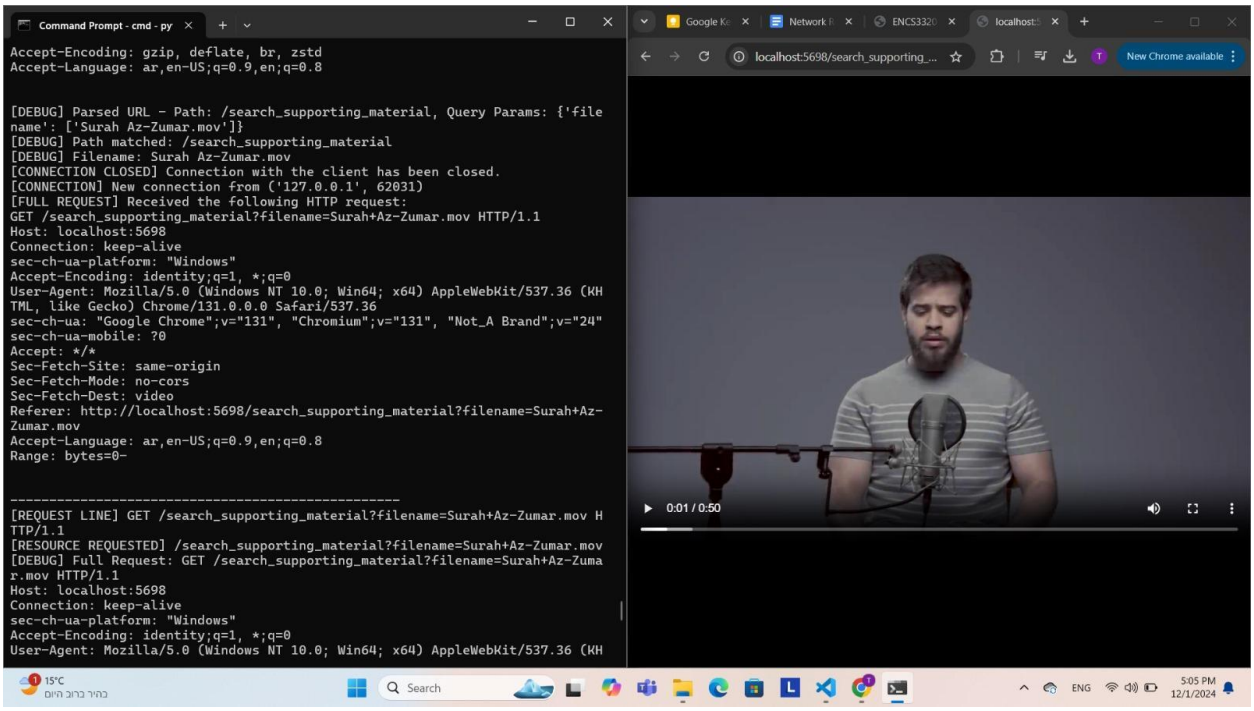


Figure 11: Supporting_matirial_en web page request form 9 - request form 8 -request Surah Az- Zumar.mov -2

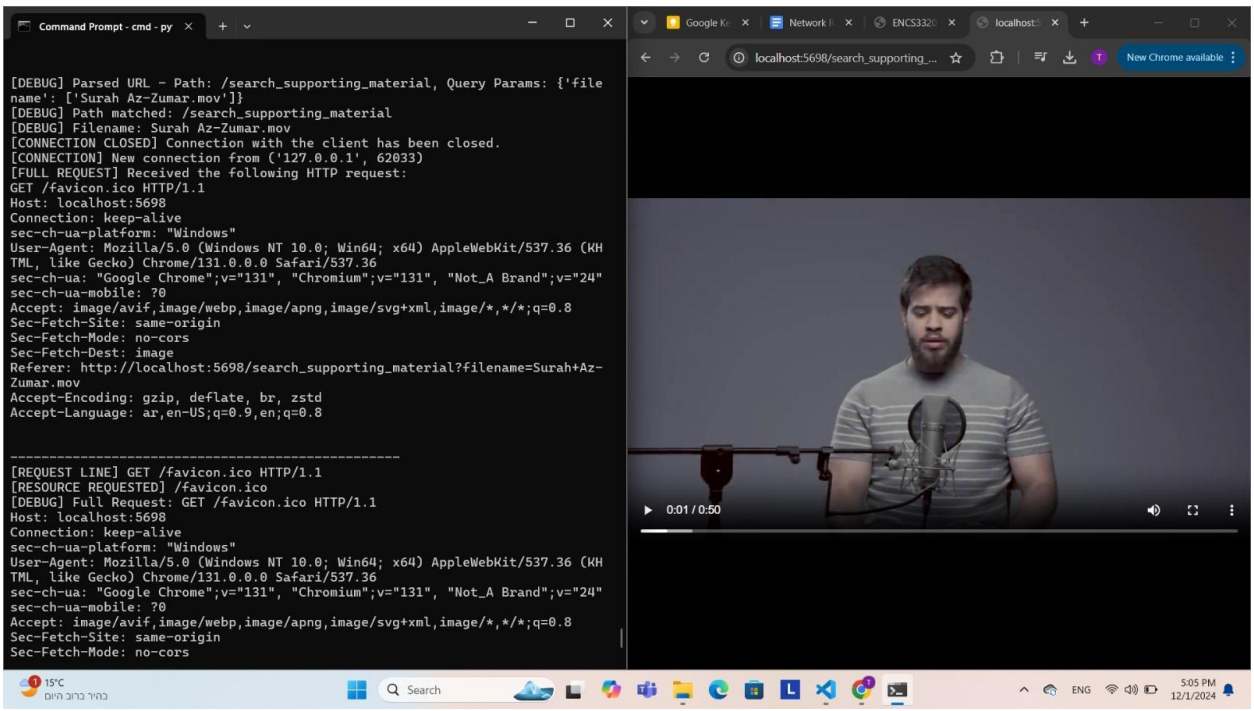


Figure 11: Supporting_matirial_en web page request form 10- request form 8 -request Surah Az- Zumar.mov -3

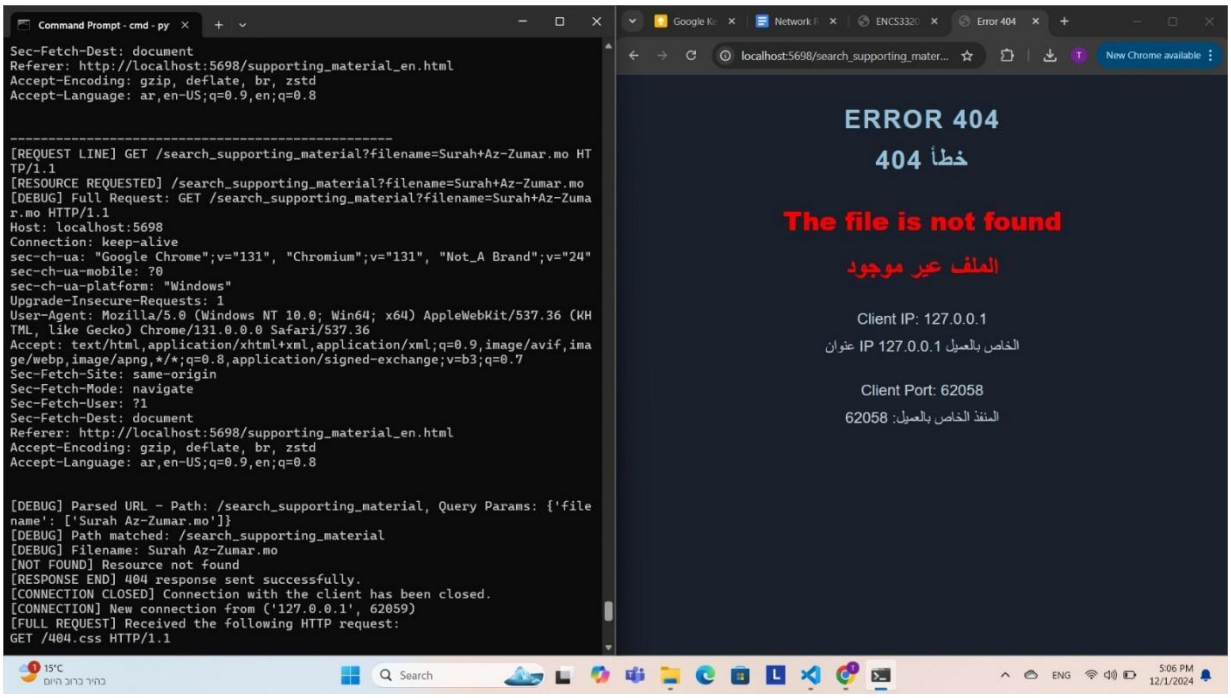


Figure 11: Supporting_matirial_en web page request form 11 - request form 8 -request Surah Az- Zumar.mo

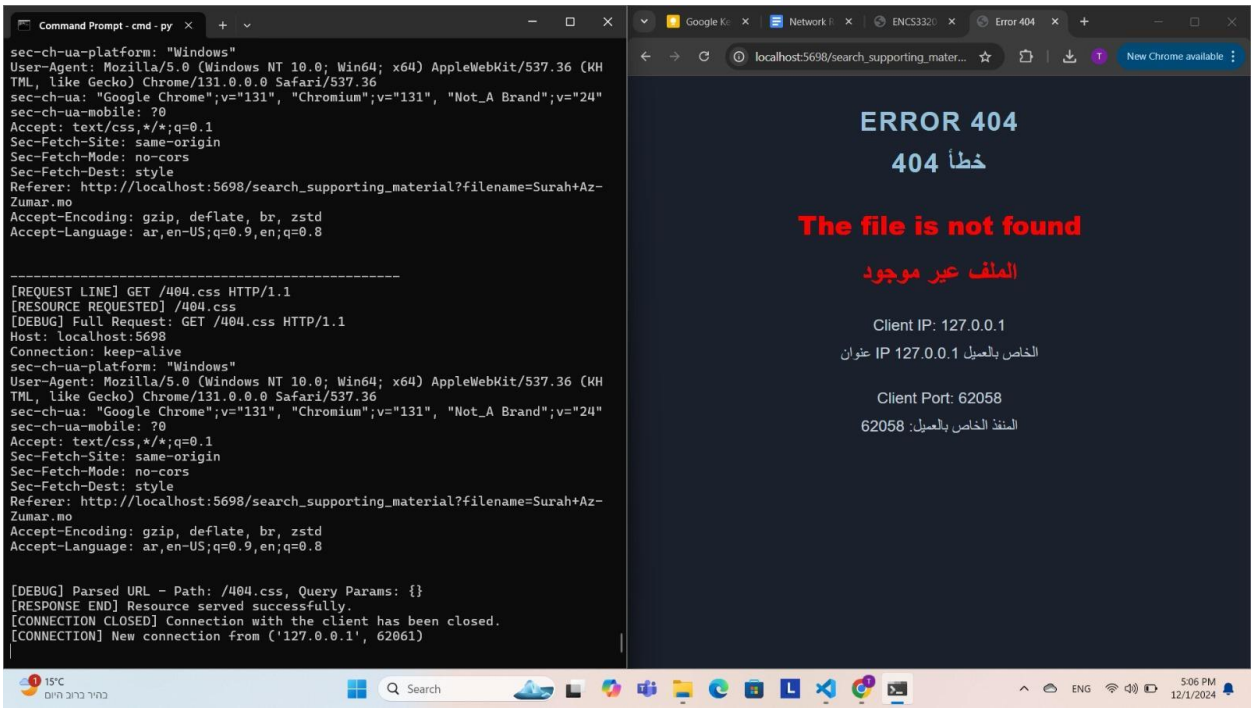


Figure 11: Supporting_matirial_en web page request form 12

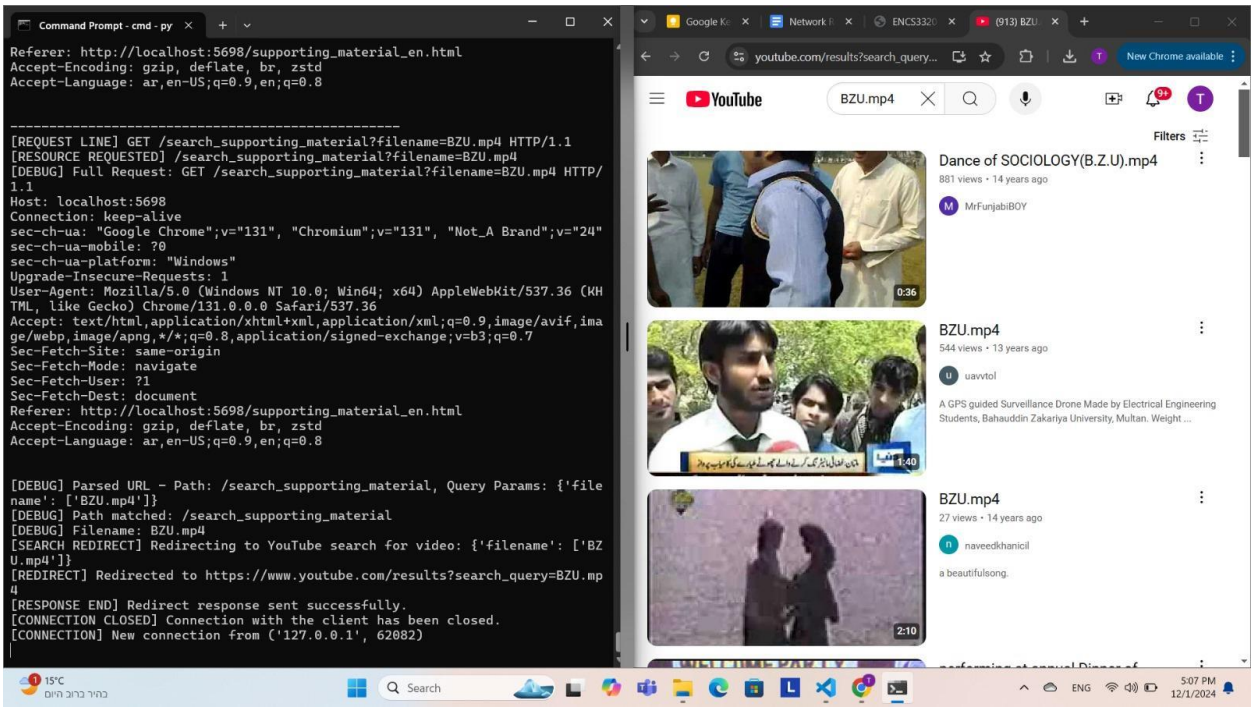


Figure 11: Supporting_matirial_en web page request form 13- request BZU.mp4

Explanation of the conditional statement:

- ❖ if path in ["/search_supporting_material", "/supporting_material__ar"]:

- ➔ Conditional statement, checks if the path (i.e. the client sent a request to page supporting_material_en or supporting_material__ar) contains supporting_material_en or supporting_material__ar and since the condition is met, it will be included in it.

- Example: The client will search for the following files: (The client request form is attached above)
  - ★ toqa.png
  - ★ p.jpg
  - ★ p.kk

- The client entered toqa.png, the request format is:

  GET /search_supporting_material?filename=toqa.png HTTP/1.1

  query_params = {'filename': ['toqa.png']} (according to previous explanation)

  ❖ if "filename" in query_params:

  ➔ It is checked if query_params contains filename and in our case the condition is met, meaning it will be included in the condition.

  ❖ filename = query_params["filename"][0]

  ➔ Get the first value of the filename parameter,i.e. through this sentence we will be able to get what the client is looking for, this means filename = toqa.png.

  ❖ file_path = os.path.join(LOCAL_DIRECTORY, filename)

  ➔ With this line a file path will be created that combines both the base directory (LOCAL_DIRECTORY) and the file name (filename).

  ➢ LOCAL_DIRECTORY: As mentioned before, it is a variable that holds the path to the root directory. In our case, it is (from Toqa Abdeen's laptop, who is preparing this part) C:\Users\abdee\VS code\task2.
  ➢ filename: It is what the client is looking for is toqa.png.
  ➢ os.path.join(): We will be able to collect the components to produce the path through this function.

➔ The result of the file_path will be as follows:

➢ file_path = C:/Users/abdee/VS code/task2/toqa.png

❖ if os.path.exists(file_path):

➔ Simply this line to check if the file_path exists or not using the os function from library os which returns true or false and in our case it will return true.

❖ extension = os.path.splitext(filename)[1].lower()

➔ Through this sentence we will be able to extract the extension of the image/video that the client is looking for.

➔ `lower(): To ensure correct comparison it will be converted to lowercase because pang ≠ PANG.`

➔ extension = .png (toqa.png)

❖ if extension in ['.jpg', '.jpeg']:
❖ content_type = "image/jpeg"
❖ elif extension in ['.png', '.gif']:
❖ content_type = "image/png"
❖ elif extension in ['.mp4', '.avi', '.mov', '.mkv']:
❖ content_type = "video/mp4"
❖ else:
❖ return handle_404(conn)

- ➔ Conditional statements that specify content_type for what the client is searching for. In our case, the second condition will be true and the content_type will be as follows: "image/png".
- ➔ In the last case, if the client requests anything with an extension other than the one above, a 404 page will appear. (example :toqa.kk)

```
with open(file_path, 'rb') as f:
                content = f.read()
            response_headers = (
                f"HTTP/1.1 200 OK\r\n"
                f"Content-Type: {content_type}\r\n"
                f"Connection: close\r\n\r\n"
            )
            conn.sendall(response_headers.encode(FORMAT))
            conn.sendall(content)
            return
```

- ➔ The purpose of these sentences is to present the image/video that the client is searching for i.e. image toqa.png will appear on the client's screen. (The code was explained in detail previously.)

```
else:
        if filename.endswith(('.jpg', '.jpeg', '.png', '.gif')):
            redirect_url = f"https://www.google.com/search?q={filename}&tbm=isch"
            print(f"[SEARCH REDIRECT] Redirecting to Google search for image: {query_params}")

        elif filename.endswith(('.mp4', '.avi', '.mov', '.mkv')):
            redirect_url = f"https://www.youtube.com/results?search_query={filename}"
```

❖ print(f"[SEARCH REDIRECT] Redirecting to YouTube search for video: {query_params}")

➔ Here the else statement will be executed if condition[if os.path.exists(file_path)] is not met, but in our case the condition is met, meaning that the else statement was not entered. However, if the client searches for p.jpg the else statement will achieve , not [if os.path.exists(file_path)] , the following will happen:

➔ There are two possible possibilities:

➢ 1. If the client is searching for an image with one of the following extensions (it was confirmed that the image does not exist in the directory previously) which is '.jpg', '.jpeg', '.png', '.gif', then the server will transfer the client to the Google search page that shows what the client is searching for through the following link:

- **https://www.google.com/search?q={filename}&tbm=isch**

In our example, the link is:

- **https://www.google.com/search?q=p.png&tbm=isch**

if filename.endswith(('.jpg', '.jpeg', '.png', '.gif')):

redirect_url = f"https://www.google.com/search?q={filename}&tbm=isch"

print(f"[SEARCH REDIRECT] Redirecting to Google search for image: {query_params}")

➢ 2. If the client is searching for a video with one of the following extensions (it was confirmed that the video does not exist in the directory previously) which is '.mp4', '.avi', '.mov', '.mkv', then the server will transfer the client to the Youtube search page that shows what the client is searching for through the following link: network.mp4

- **https://www.youtube.com/results?search_query={filename}**

In our example, the link is: (client searched for BZU.mp4)

- **https://www.youtube.com/results?search_query=BZU.mp4**

elif filename.endswith(('.mp4', '.avi', '.mov', '.mkv')):

redirect_url = f"**https://www.youtube.com/results?search_query={filename}**"

print(f"[SEARCH REDIRECT] Redirecting to YouTube search for video: {query_params}")

- else:
- return handle_404(conn)

➔ This else statement is executed if the client enters an extension other than the previous extensions, so that a 404 not found screen will be displayed to him.

- response_headers = (
- f"HTTP/1.1 307 Temporary Redirect\r\n"
- f"Location: {redirect_url}\r\n"
- f"Connection: close\r\n\r\n"
-)
- conn.sendall(response_headers.encode(FORMAT))
- print(f"[REDIRECT] Redirected to {redirect_url}")
- print("[RESPONSE END] Redirect response sent successfully.")
- return

➔ Because the client is being redirected to Google or YouTube, the status code will be 307 Temporary Redirect. Which will be implemented as before in the previous status codes.

- With this function, the client will be able to search for an image or video in the previous cases.
- In the search images the client searched for the following files:

- toqa.png (available in the system)
- BZU.png (not available in the system)
- p.jpg (not available in the system)
- HTML Iframe Tutorial .mp4 (available in the system)
- Surah Az-Zumar.mov (available in the system)
- Surah Az-Zumar.mo (not available in the system)
- BZU.mp4 (not available in the system)

Second: main_en.html and main_ar.html

On this page will show the information in the following order:

Welcoming our clients through the following text: Welcome to ENCS3320 - Computer Networks Web Server .

The following will appear to explain the purpose of the page: This webpage is mainly designed for the first project of the Computer Networks course .

Each page includes a project name and a brief description.

Also, through this web page, the client can access the following links:

1. Both Arabic and English pages:
    - The textbook website: **https://gaia.cs.umass.edu/kurose_ross/index.php**
    - Ritaj website: **https://ritaj.birzeit.edu/**
2. For the English page:

- Birzeit University Computer Networks Course website in English: **https://www.birzeit.edu/en/content/encs3320-computer-networks**
- supporting_material_en.html

3. For the Arabic page:

    - Birzeit University Computer Networks Course website in Arabic: **https://www.birzeit.edu/ar/content/encs3320-shbkt-lhswb**
    - supporting_material_ar.html

Furthermore to a box showing the following information for each team member:

- Member name.
- Student ID.
- Major.
- Hobbies.
- Projects.
- GitHub account link.

As well, through this web page, the client will learn about the first chapter of the book COMPUTER NETWORKING A Top-Down Approach, 8th Edition through the information and pictures [9][10] on the page.

Images for designing main_en.html and main_ar.html

Main_en web page design

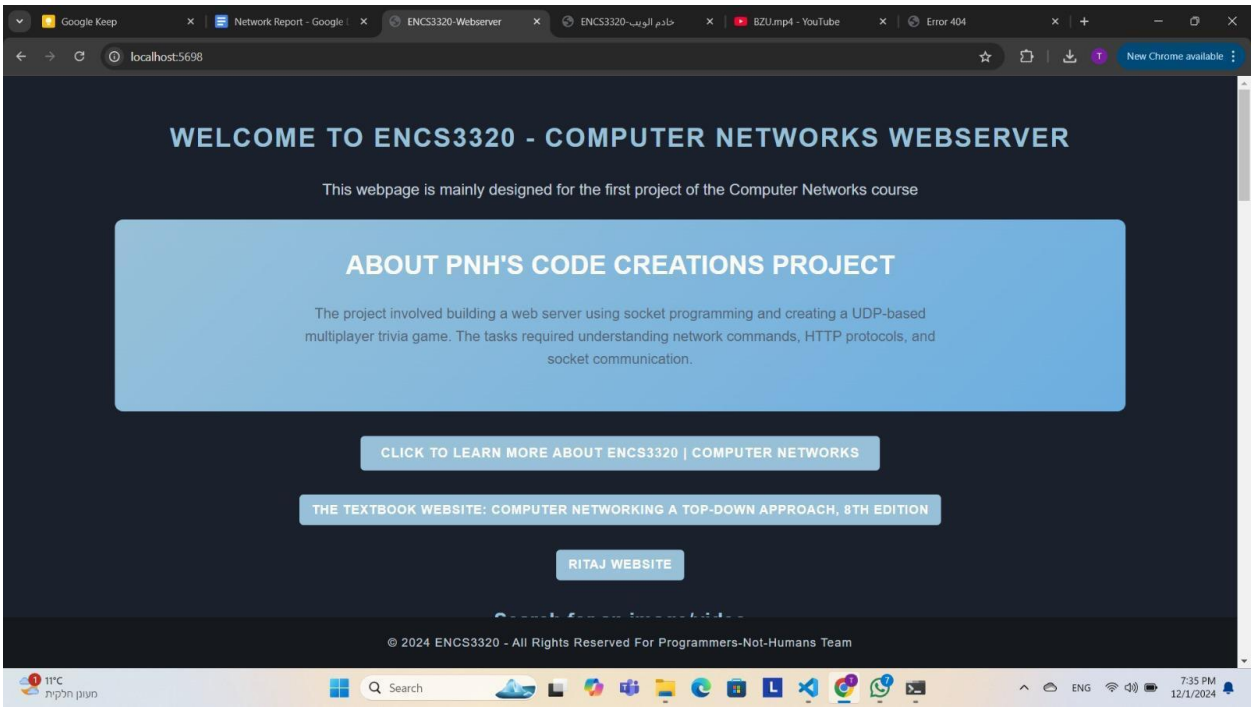


Figure 12: Main_en web page design -1

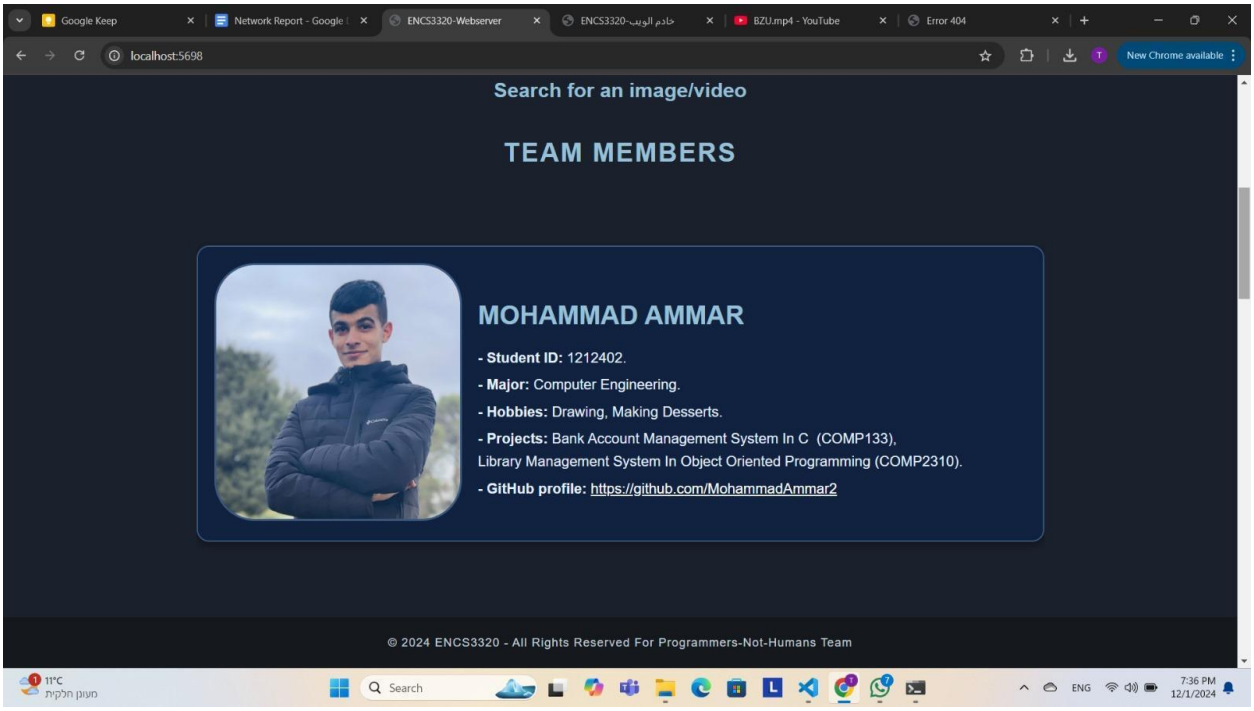


Figure 12: Main_en web page design -2

Figure 12: Main_en web page design – 3

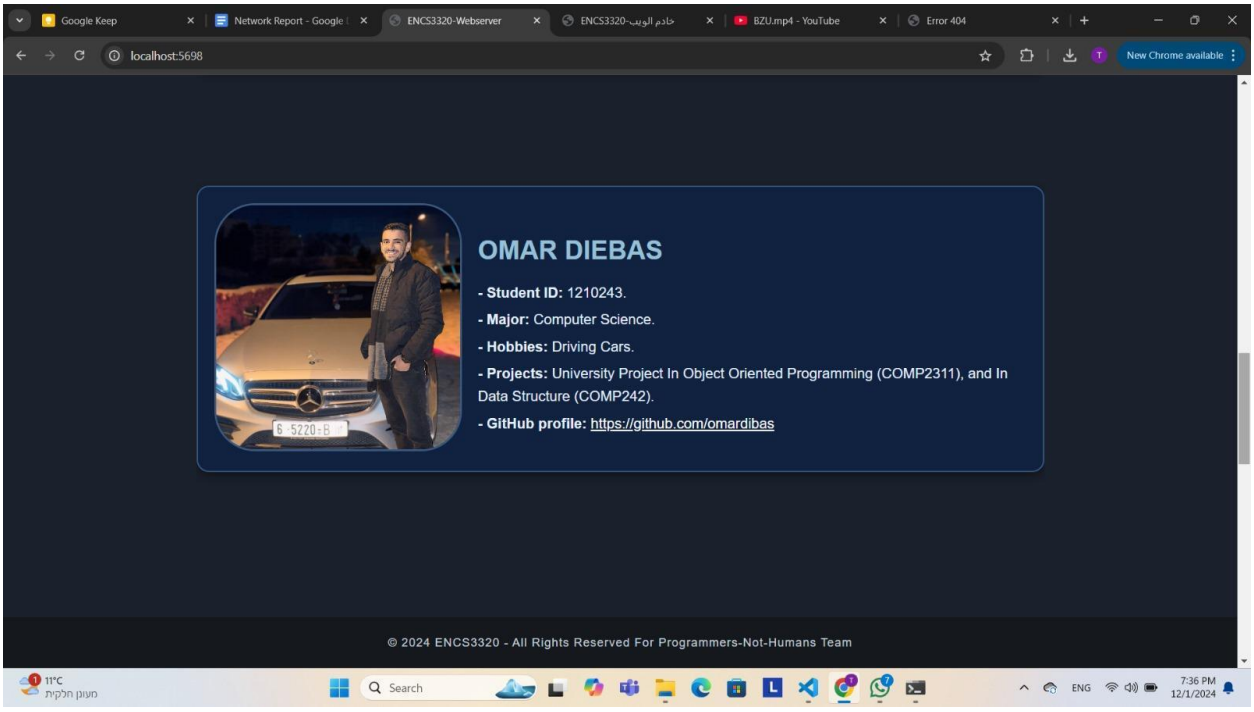


Figure 12: Main_en web page design - 4

Figure 12: Main_en web page design - 5

Figure 12: Main_en web page design - 6

Main_ar web page design

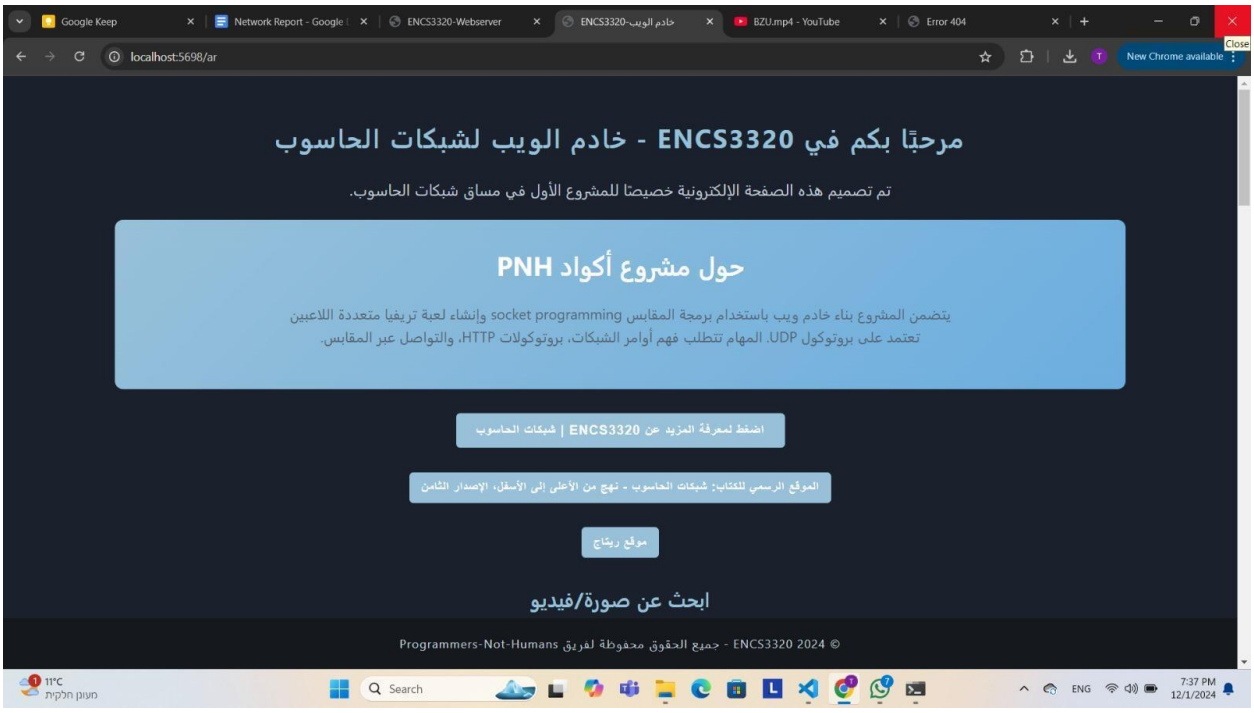


Figure 13: Main_ar web page design – 1

Figure 13: Main_ar web page design – 2

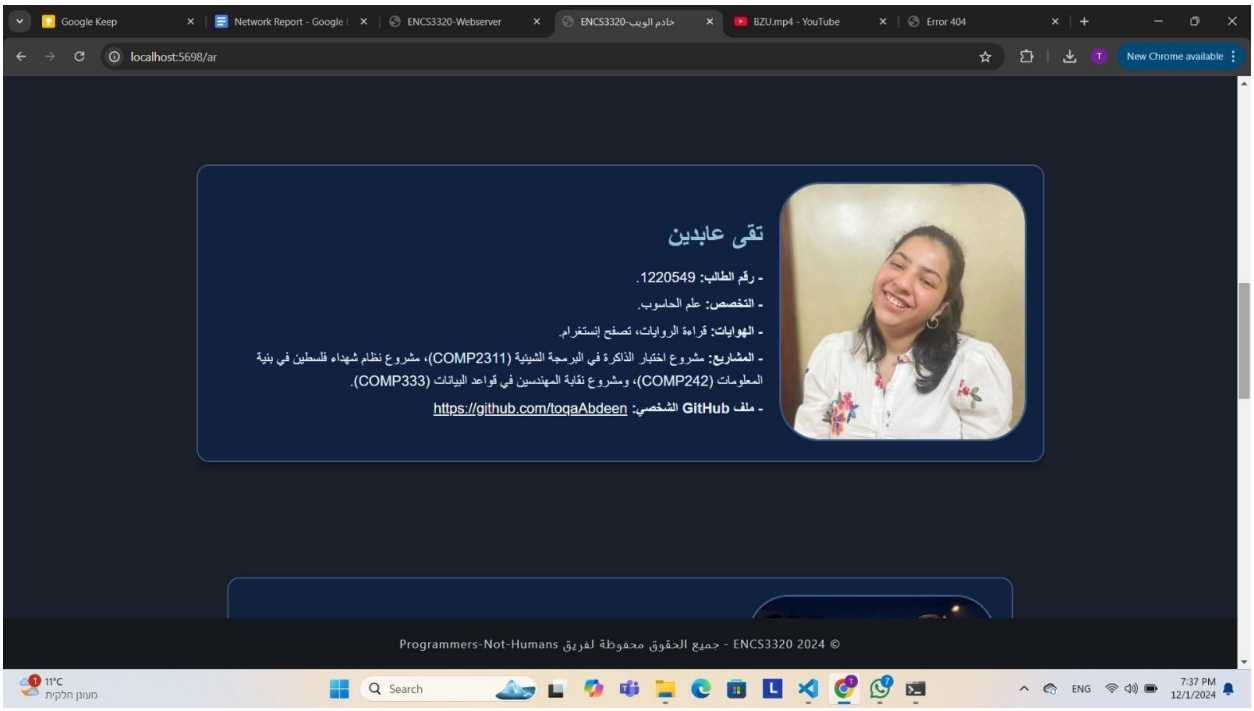


Figure 13: Main_ar web page design – 3

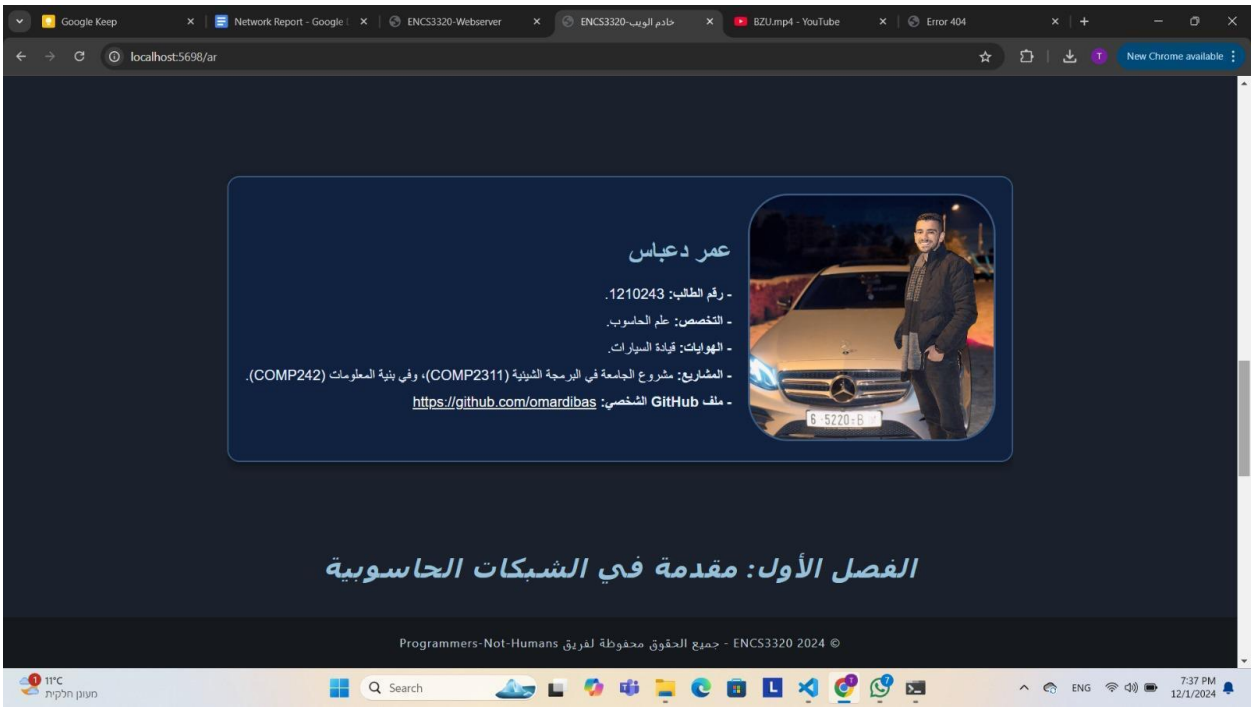


Figure 13: Main_ar web page design – 4

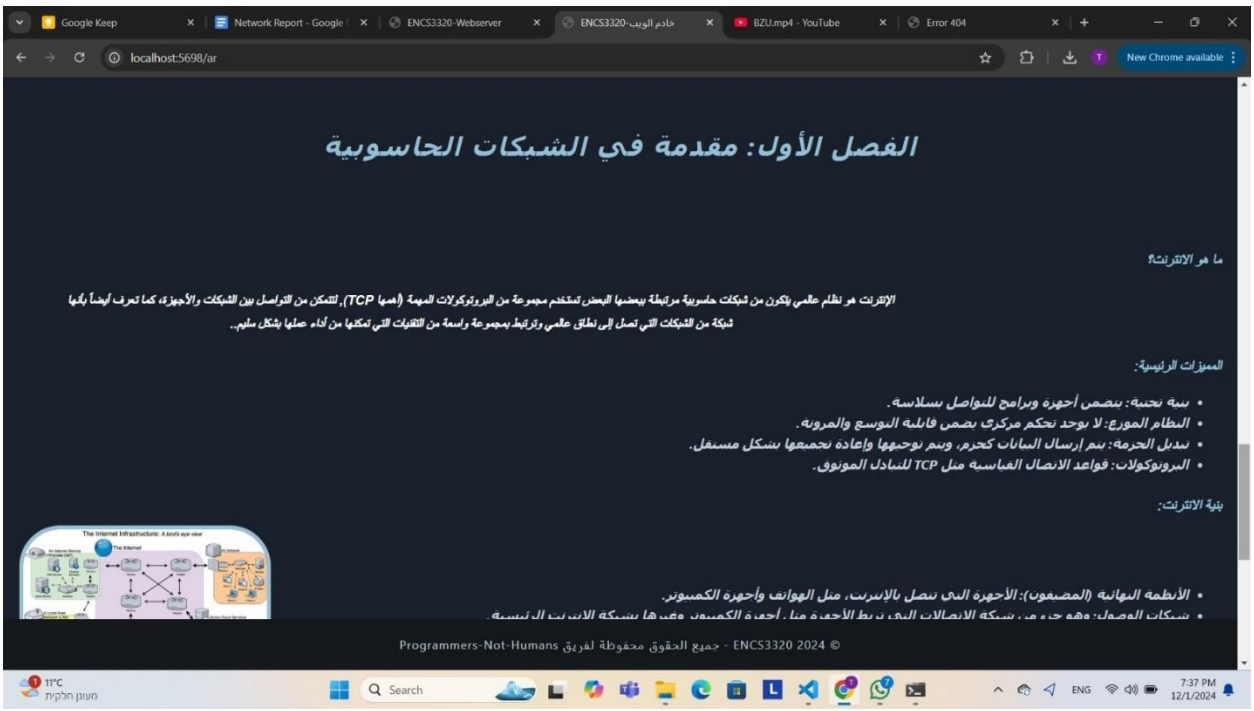


Figure 13: Main_ar web page design – 5

Figure 13: Main_ar web page design – 6

We have also prepared a special page that appears when errors occur, which explains the following information: status code, client IP address, and client port number.

We were keen to display the information in both English and Arabic To consider the client's language preference, as this project is aimed at both Arabic and English speakers.

Files for this web page:

- 404.css
- notFound.html

notFound web page design:

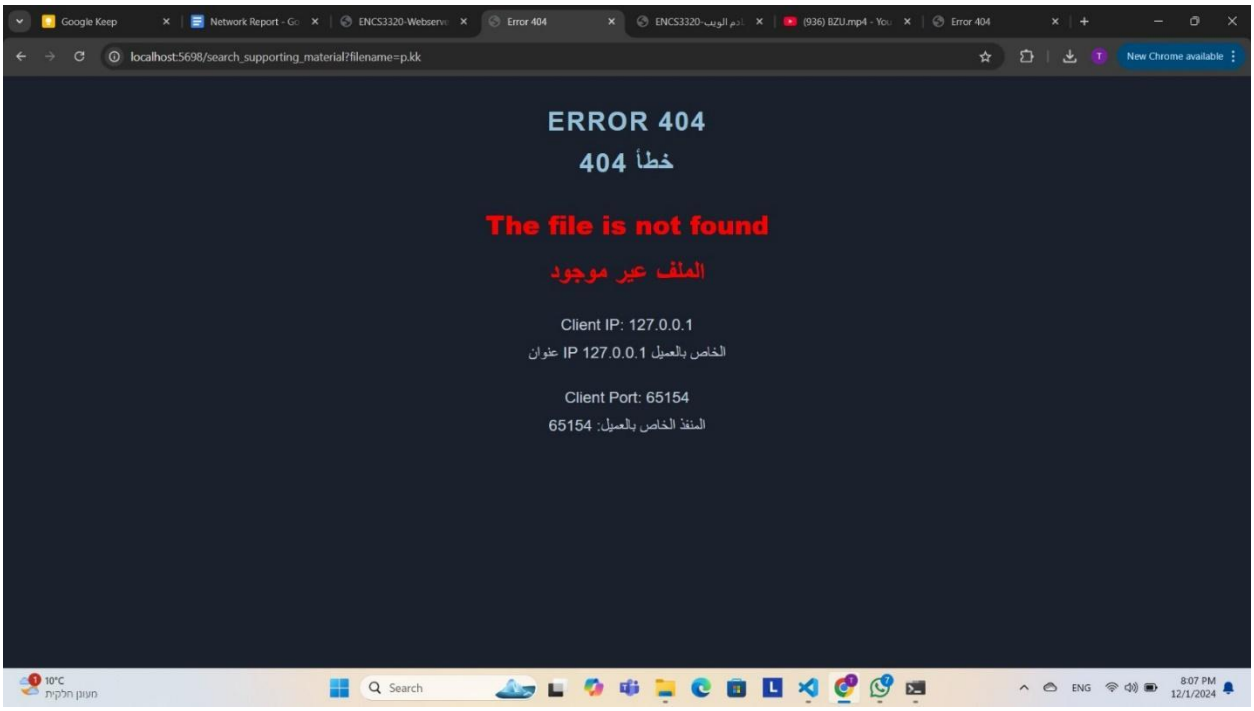


Figure 14: notFound web page design

On all pages, we were keen to create a theme using distinctive colors and design that attracts the attention of clients to visit and use our web pages. We were also keen to introduce them to our team members through information in addition to a link to each member's page on the gitHub to see our other work, hoping that we have met our clients' requests.

GitHub account links for each team member:

- Toqa Adeen account: **https://github.com/toqaAbdeen**
- Mohammad Ammar account: **https://github.com/MohammadAmmar2**
- Omar Diebas account: **https://github.com/omardibas**

Third: supporting_material_en.html and supporting_material_ar.html

The primary function of this page is to search for images and videos, resulting in three scenarios:

1. Searching for an image/video that is already in the task folder. In this case, what was searched for will appear to the client, with .png, .jpg, and .mp4 files being the only searchable formats.
2. Searching for an image/video that is not in the task folder. In this case, the following will occur: If the client searches for an image, the server will redirect him to a Google page that shows what is related to the image name. This is also what will happen in the case of a video, but the search will be on YouTube.
3. Searching for an image/video with the wrong extension, in this case page 404 notFound will appear to the client.

The purpose of this page is to facilitate the client access to his request according to the previous cases.

Supporting_material_en web page design:

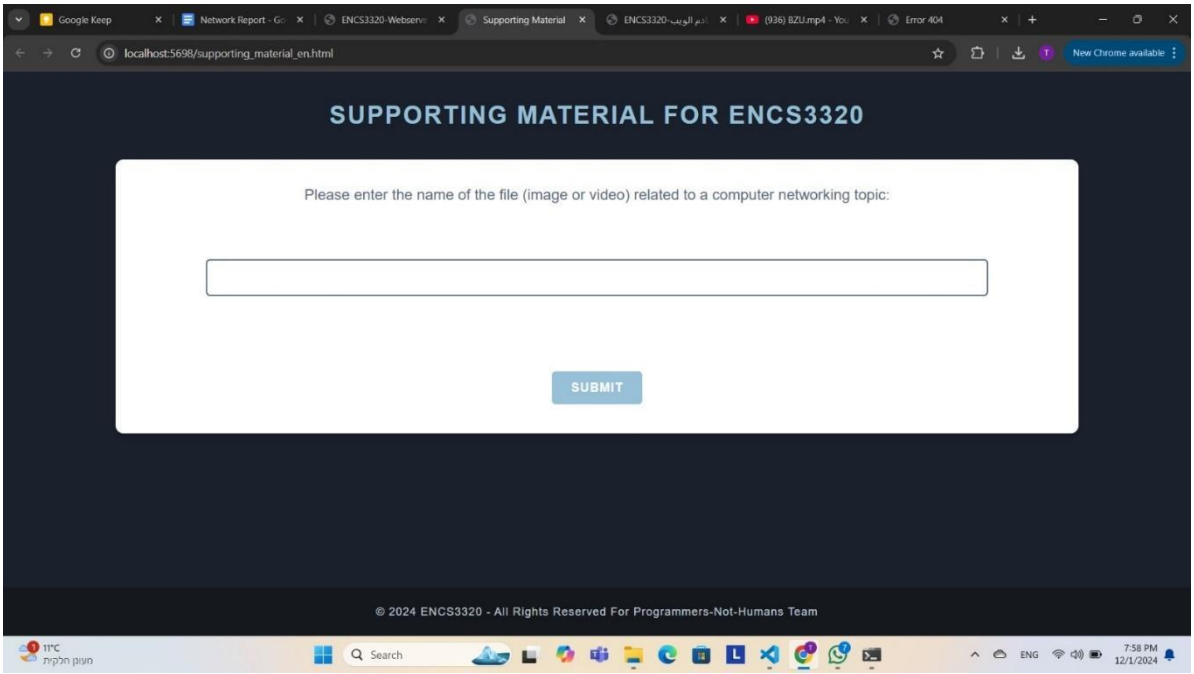


Figure 15: Supporting_material_en web page design

Supporting_material_ar web page design:

Figure 16: Supporting_material_ar web page design

Fourth: Open pages from another laptop

Browser page images for main_en:

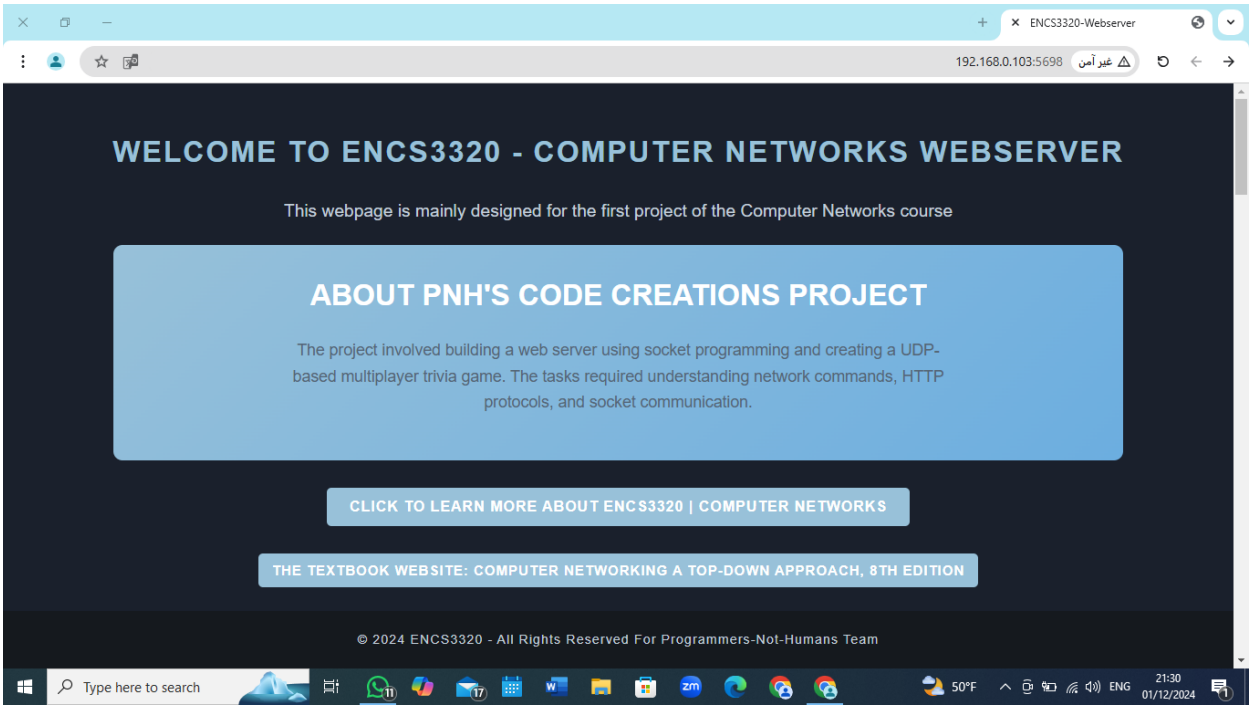


Figure 17: Open main_en web page from another laptop - 1

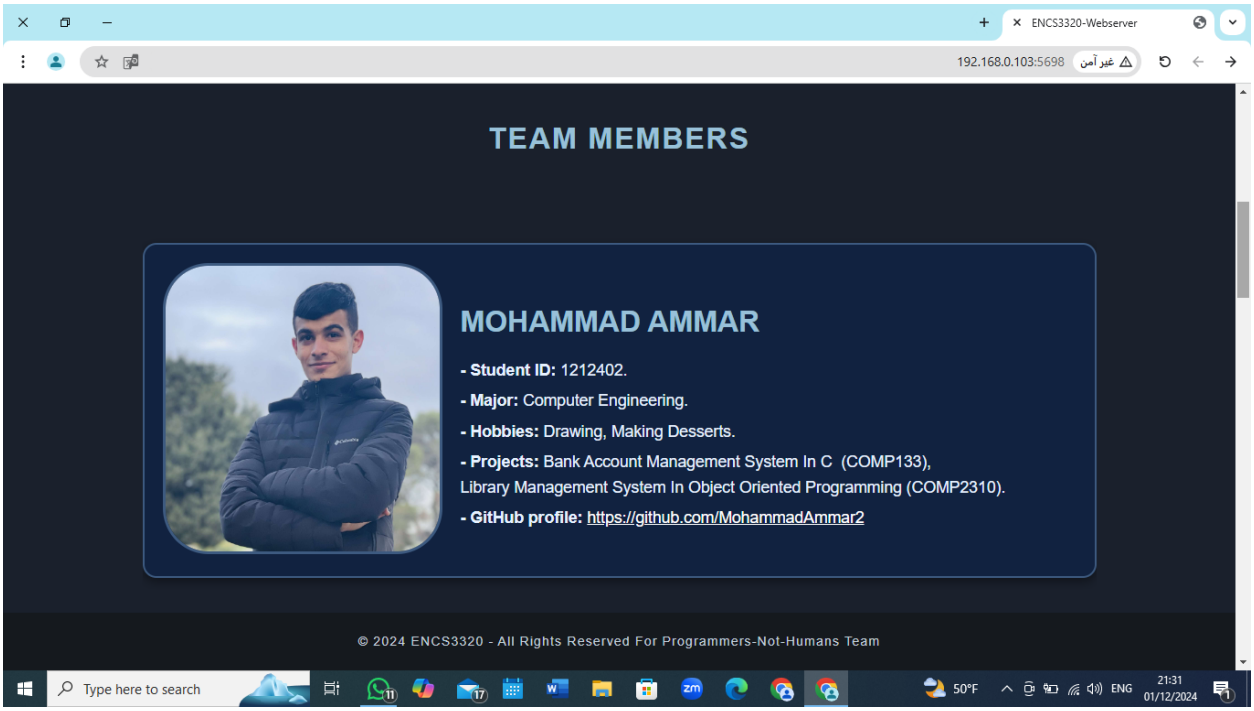


Figure 17: Open main_en web page from another laptop - 2

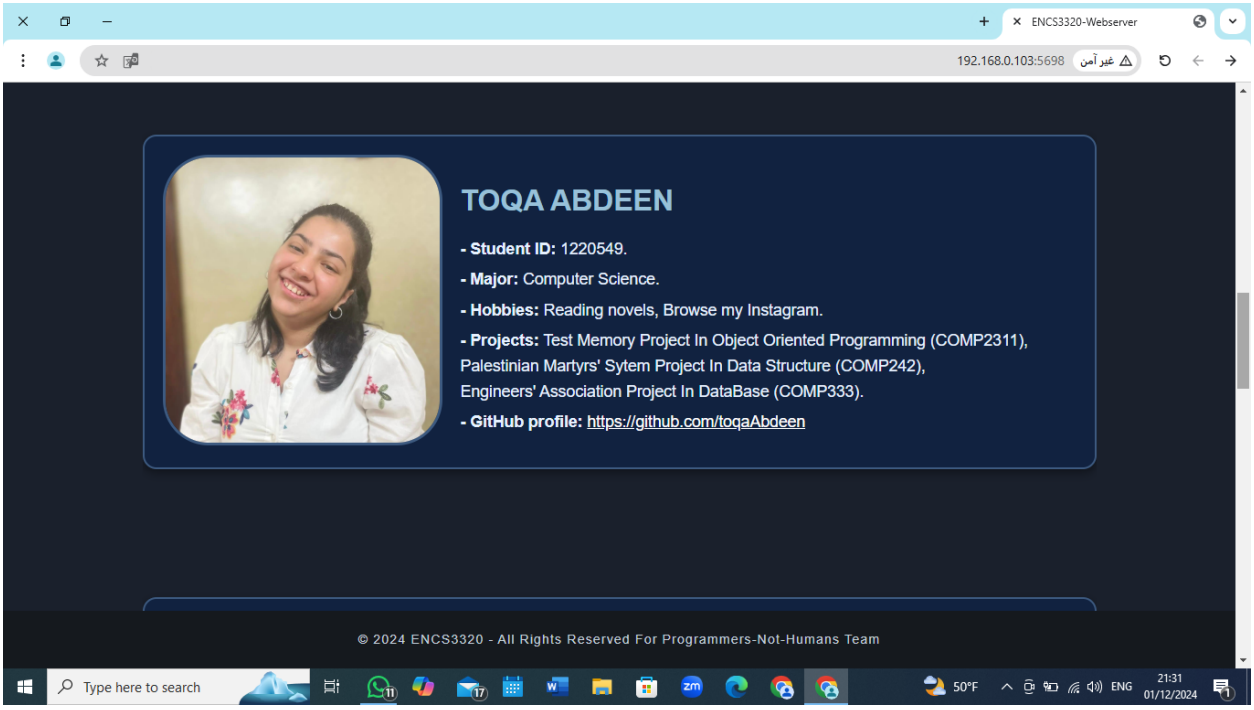


Figure 17: Open main_en web page from another laptop - 3

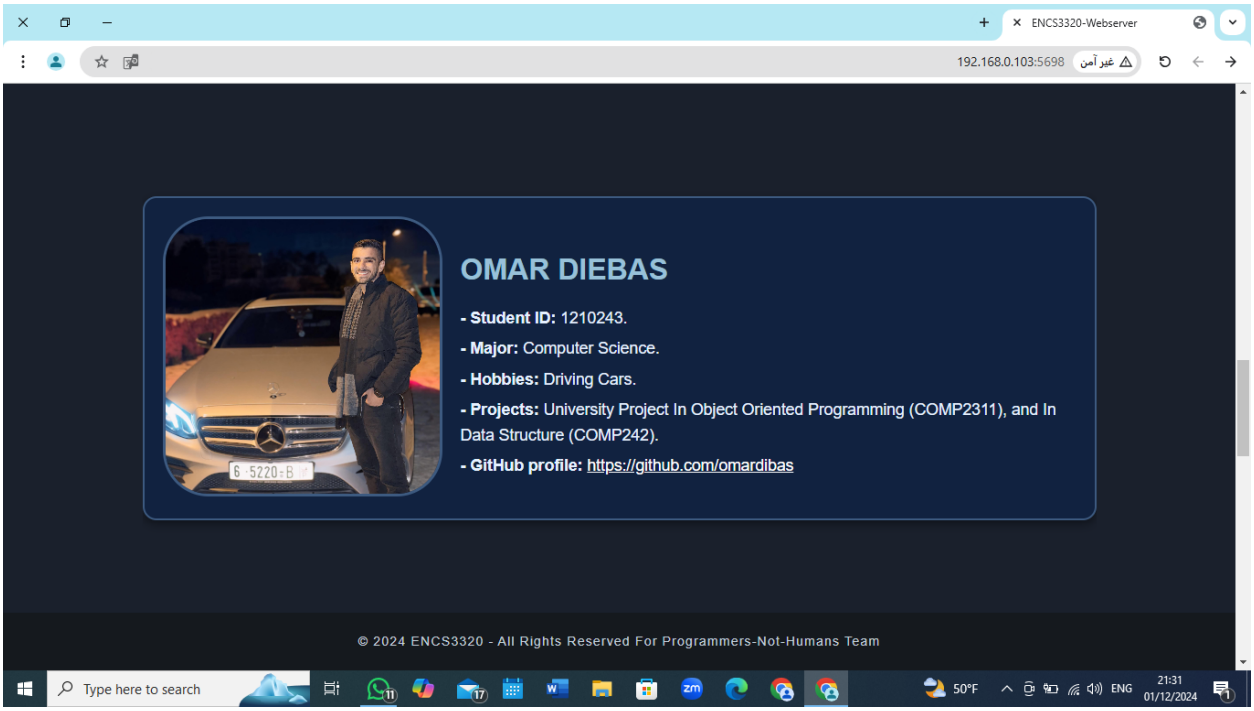


Figure 17: Open main_en web page from another laptop - 4

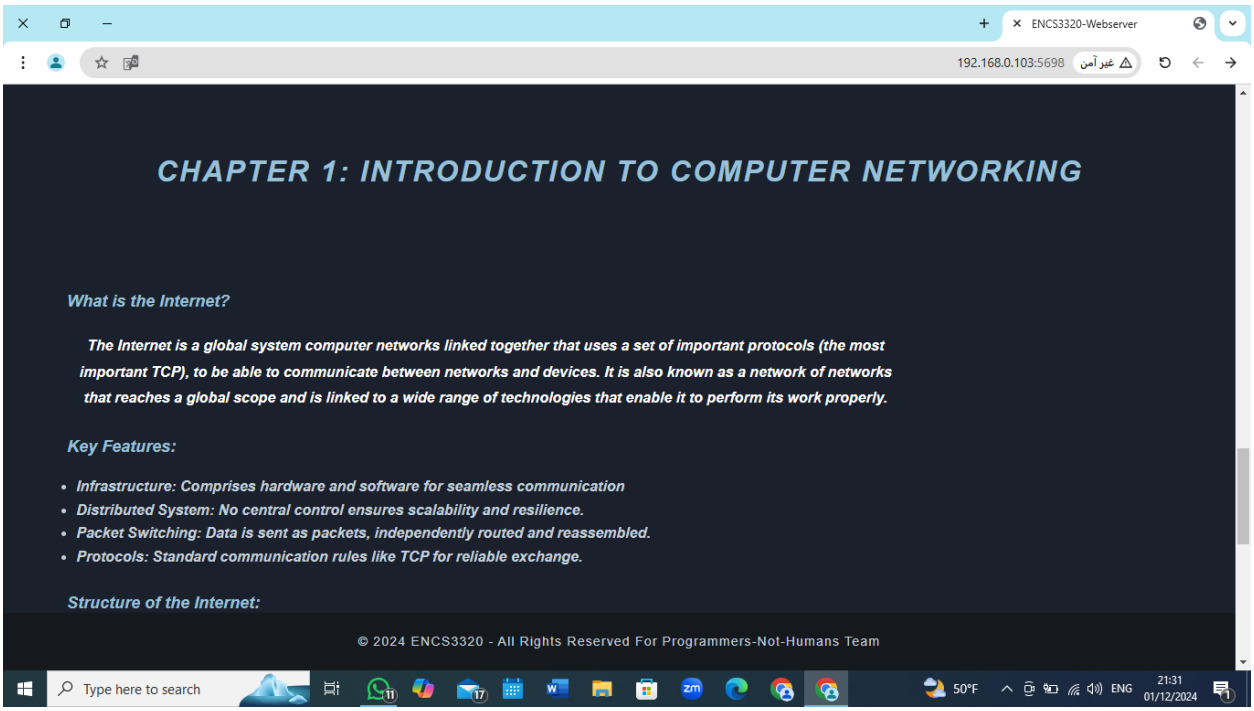


Figure 17: Open main_en web page from another laptop – 5

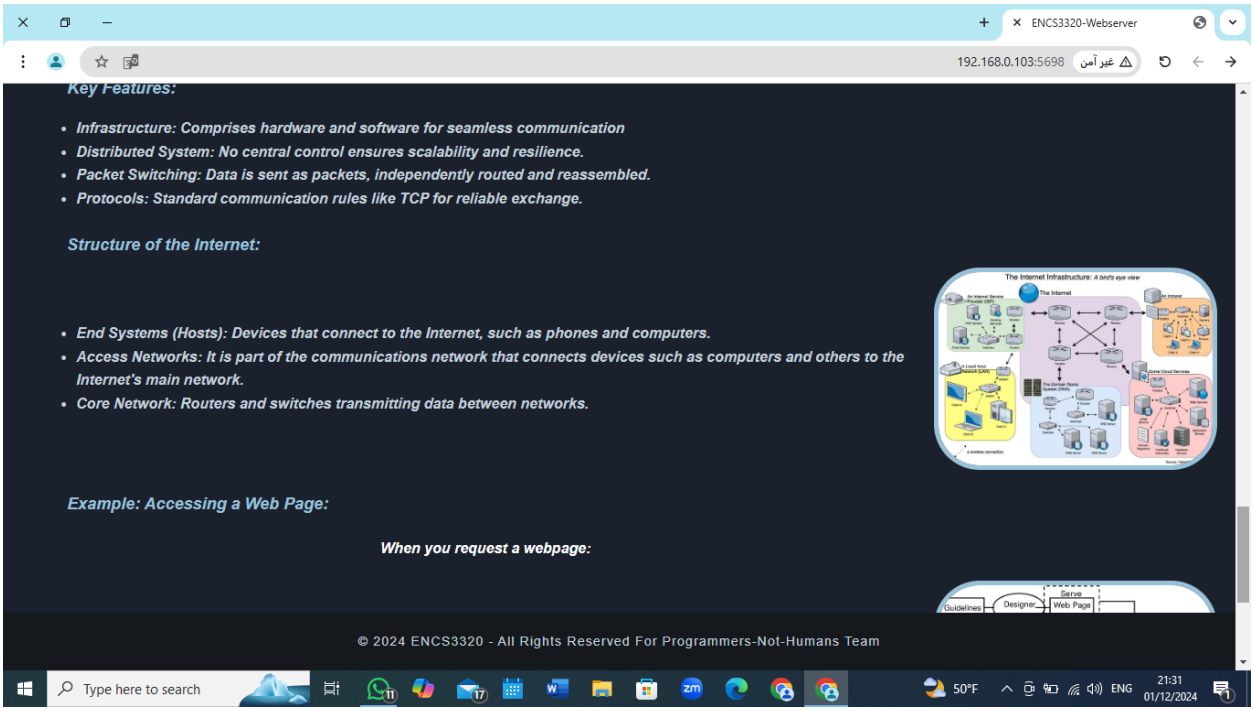


Figure 17: Open main_en web page from another laptop – 6

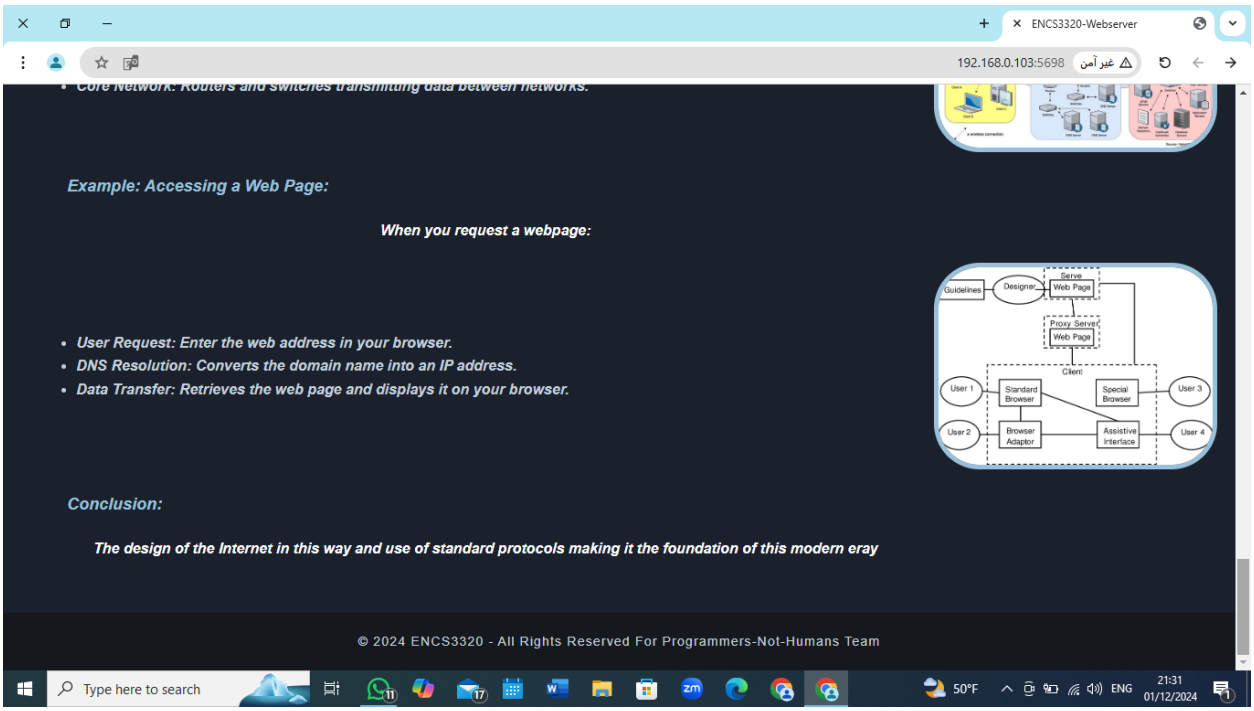


Figure 17: Open main_en web page from another laptop – 7

CMD images for main_en request:

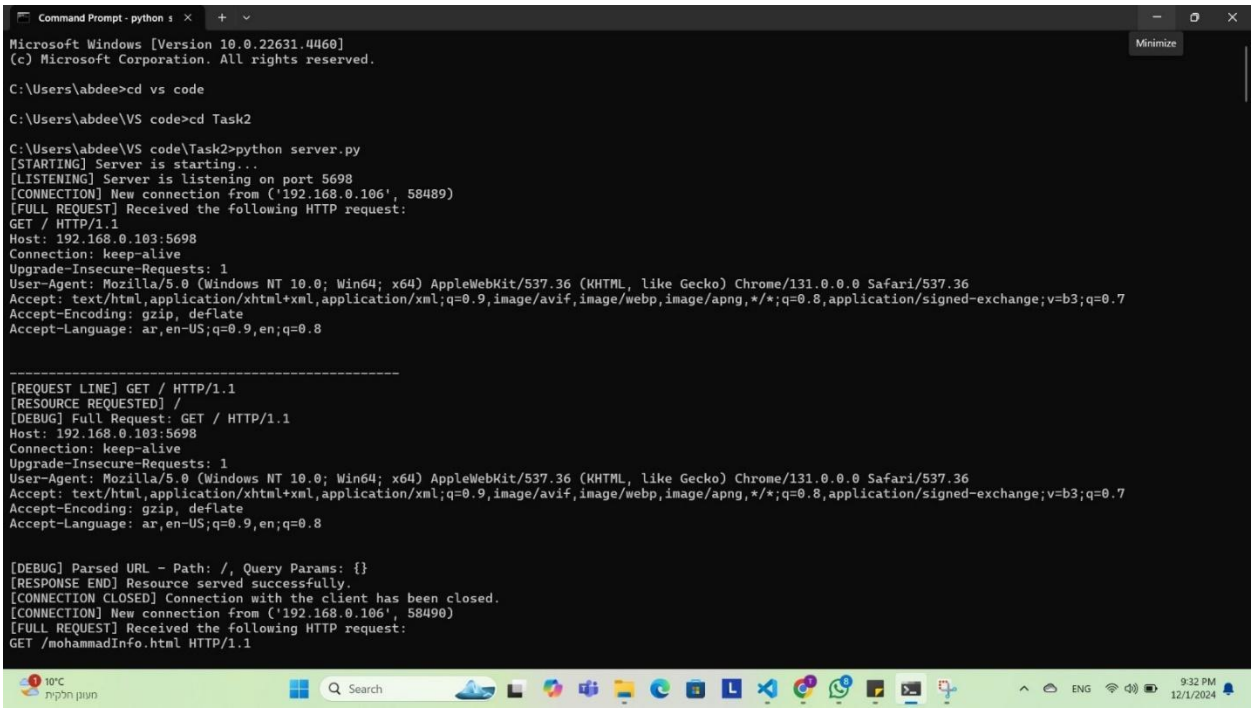


Figure 18: Form of CMD when requesting main_en from another laptop -1

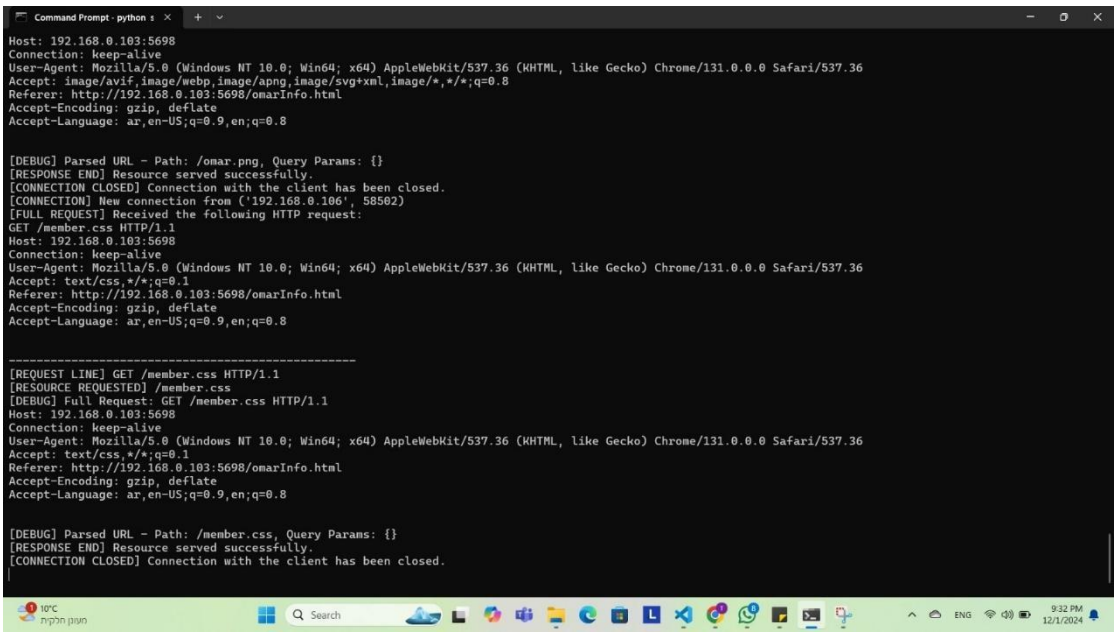


Figure 18: Form of CMD when requesting main_en from another laptop -1

Browser page images for supporting_material_en:

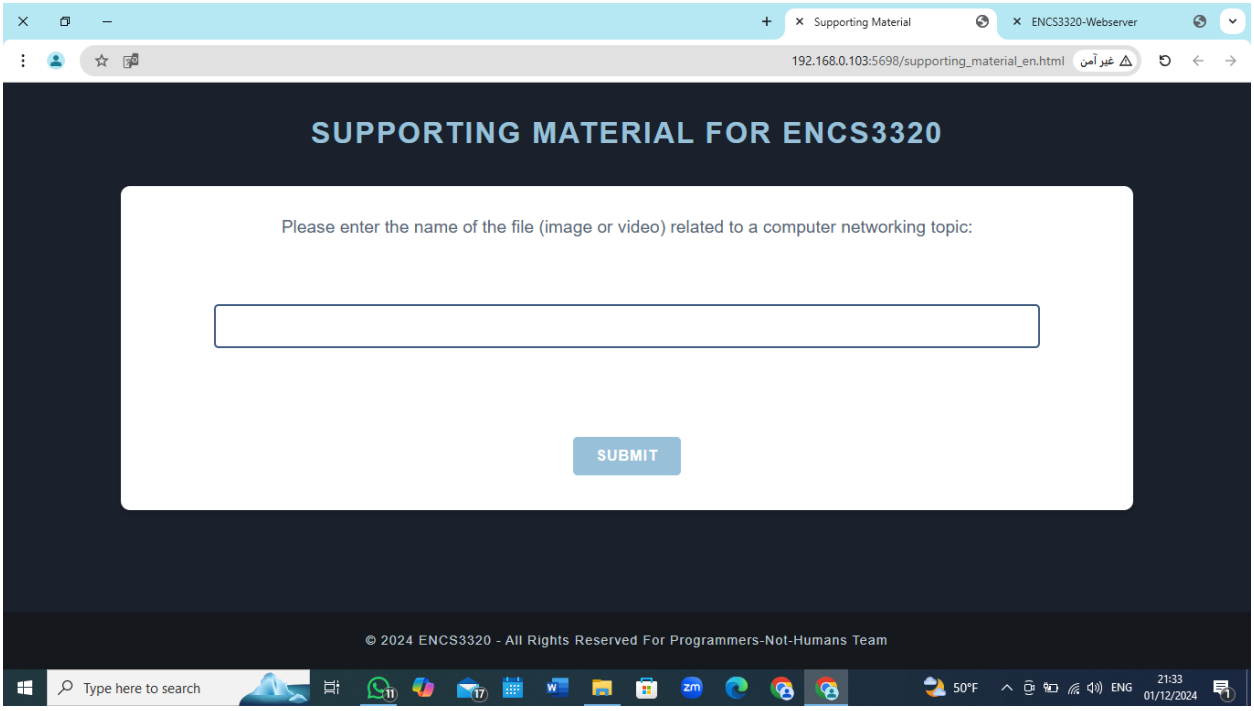


Figure 19: Open supporting_material_en web page from another laptop

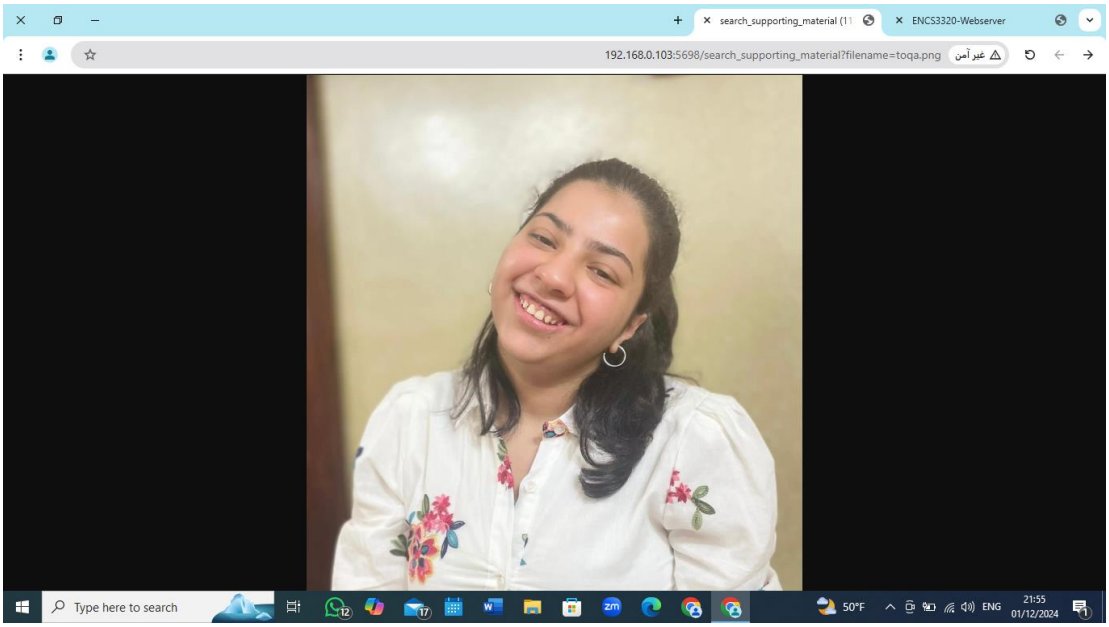


Figure 20: Request toqa.png in supporting_material_en web page from another laptop

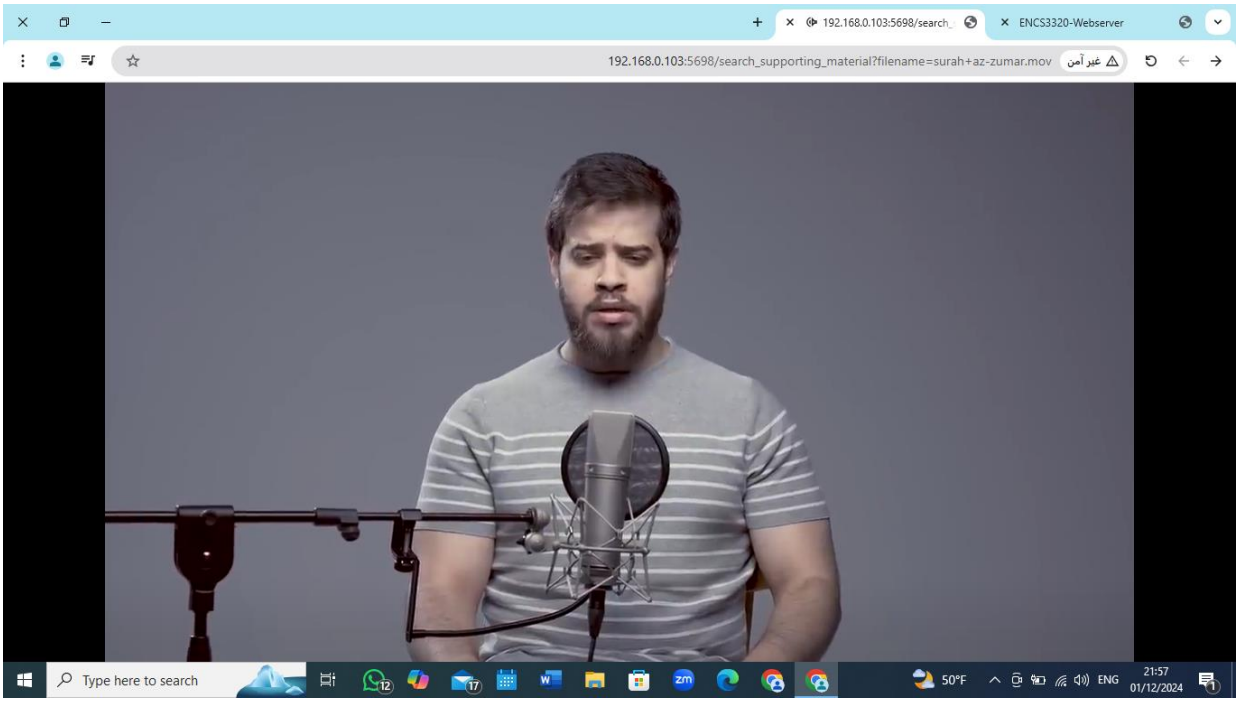


Figure 21: Request Surah Az-Zumar.mov in supporting_material_en web page from another laptop

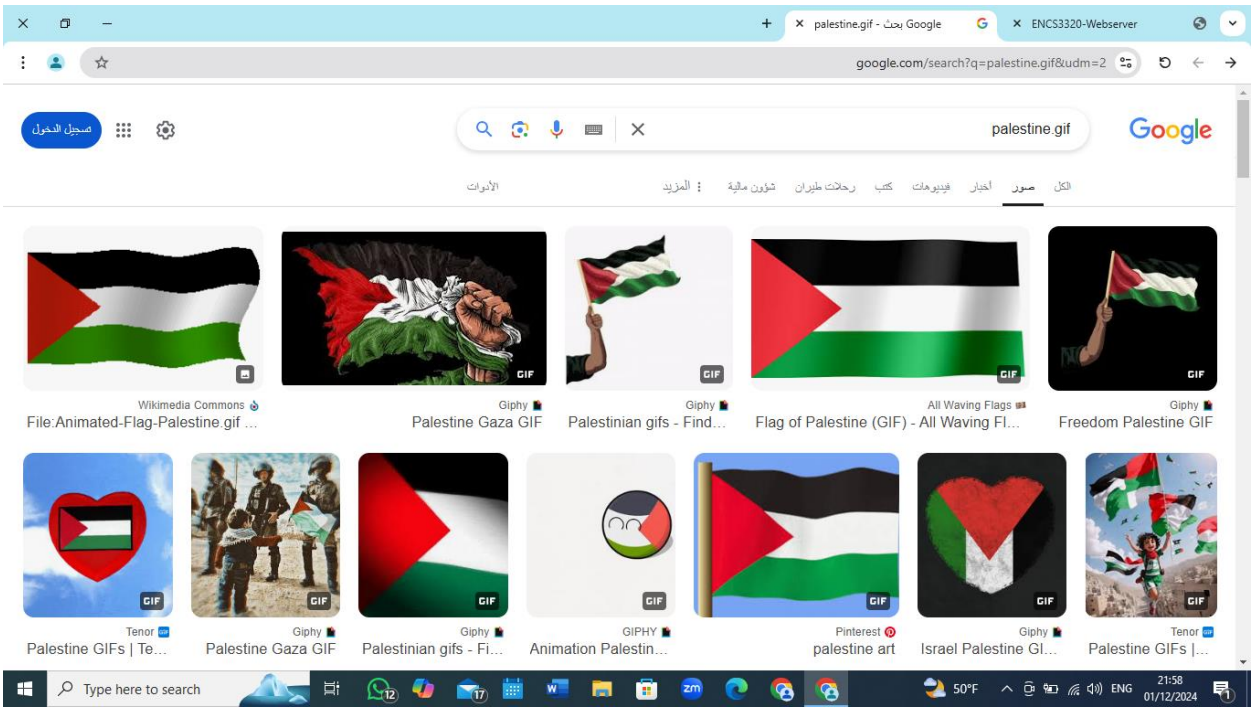


Figure 22: Request palestine.gif in supporting_material_en web page from another laptop

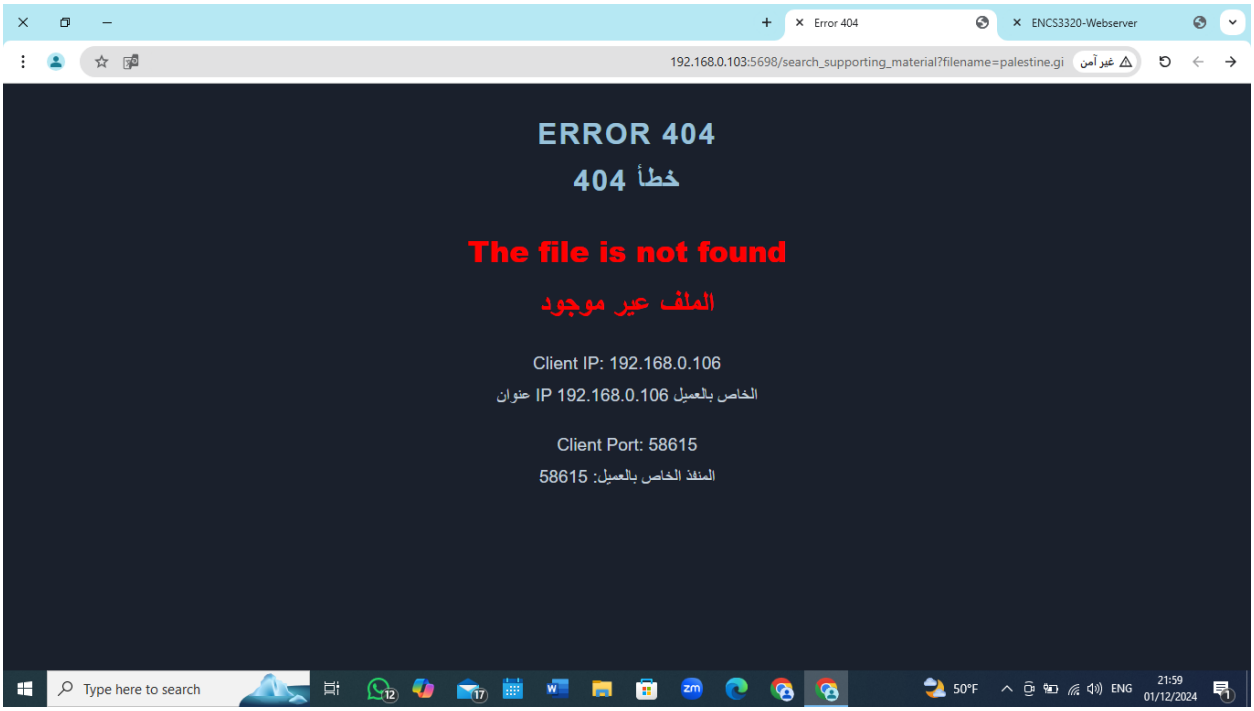


Figure 23: Request palestine.gi in supporting_material_en web page from another laptop

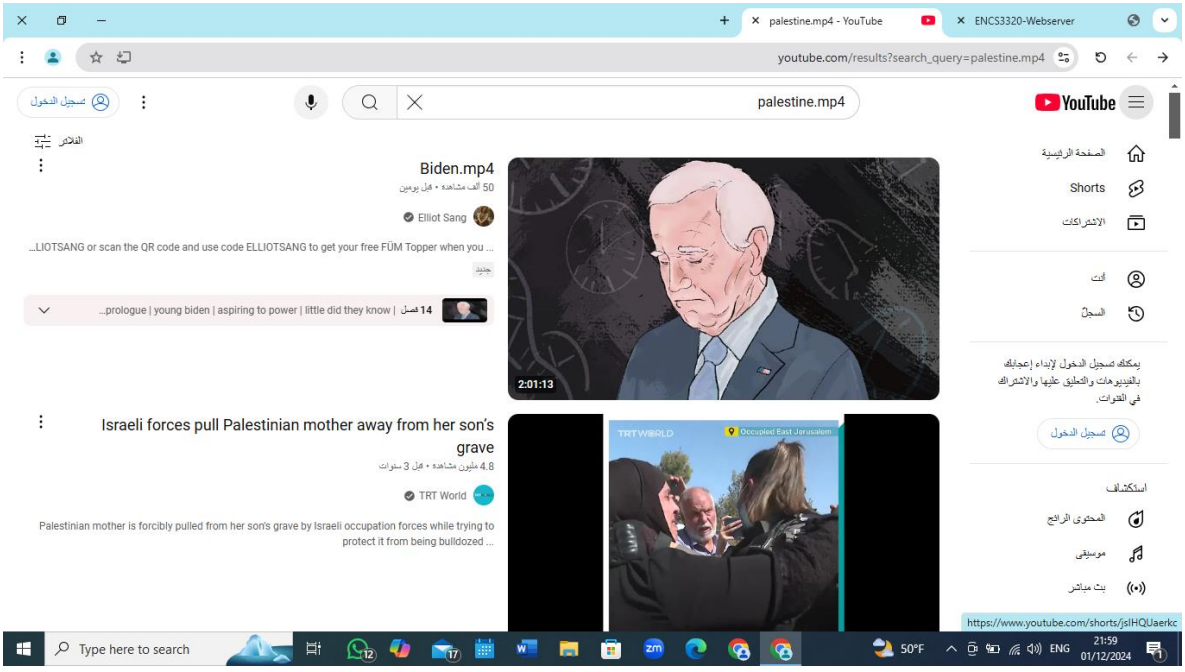


Figure 24: Request palestine.mp4 in supporting_material_en web page from another laptop

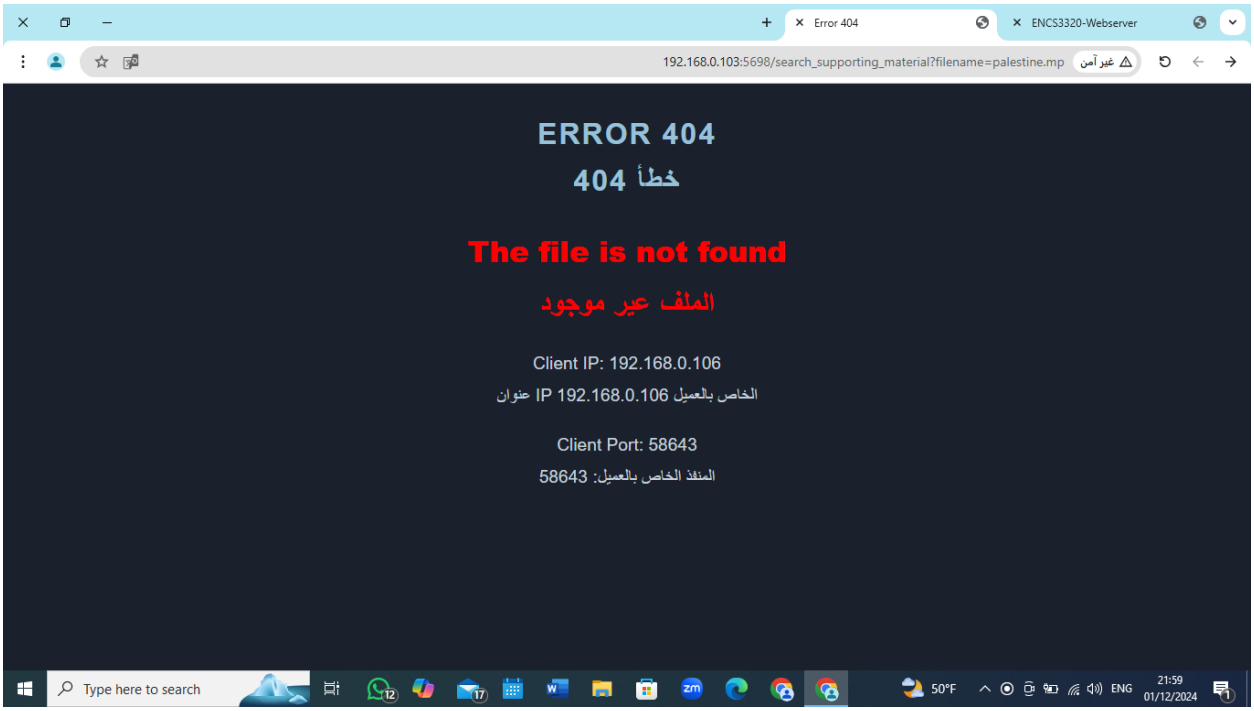


Figure 25: Request palestine.mp in supporting_material_en web page from another laptop

CMD images for supporting_material_en request:

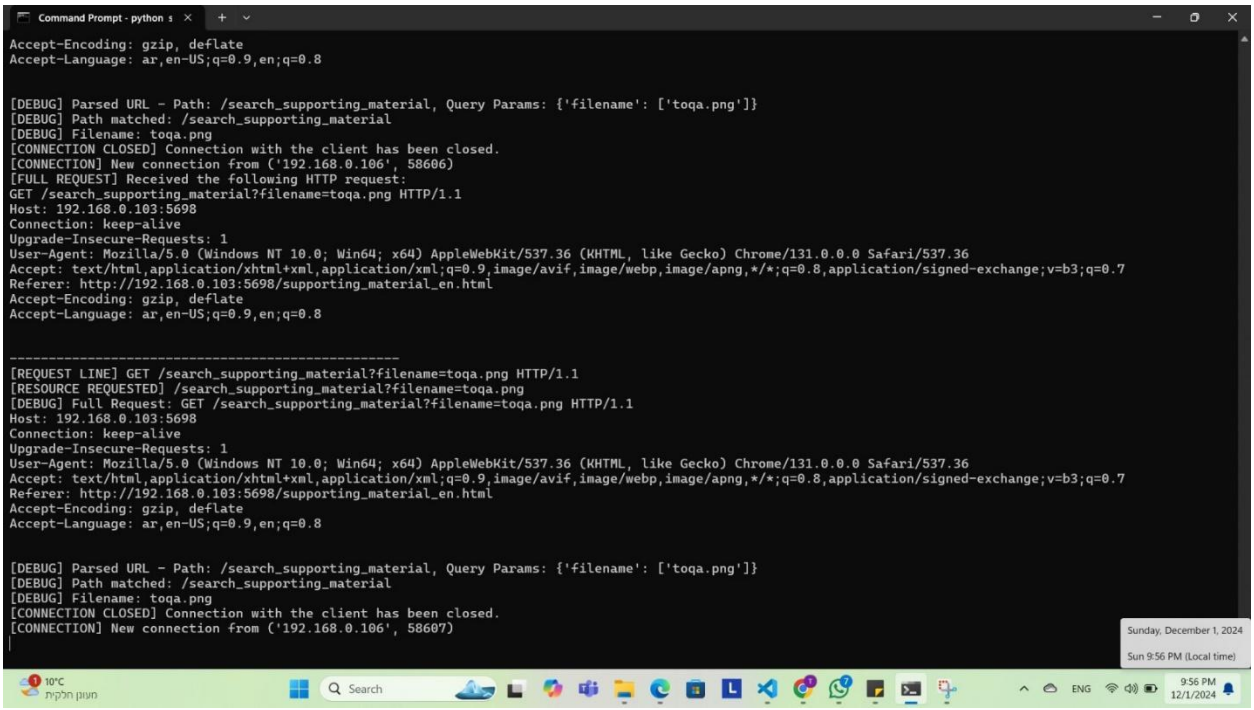


Figure 26: Form of CMD when requesting toqa.png from another laptop

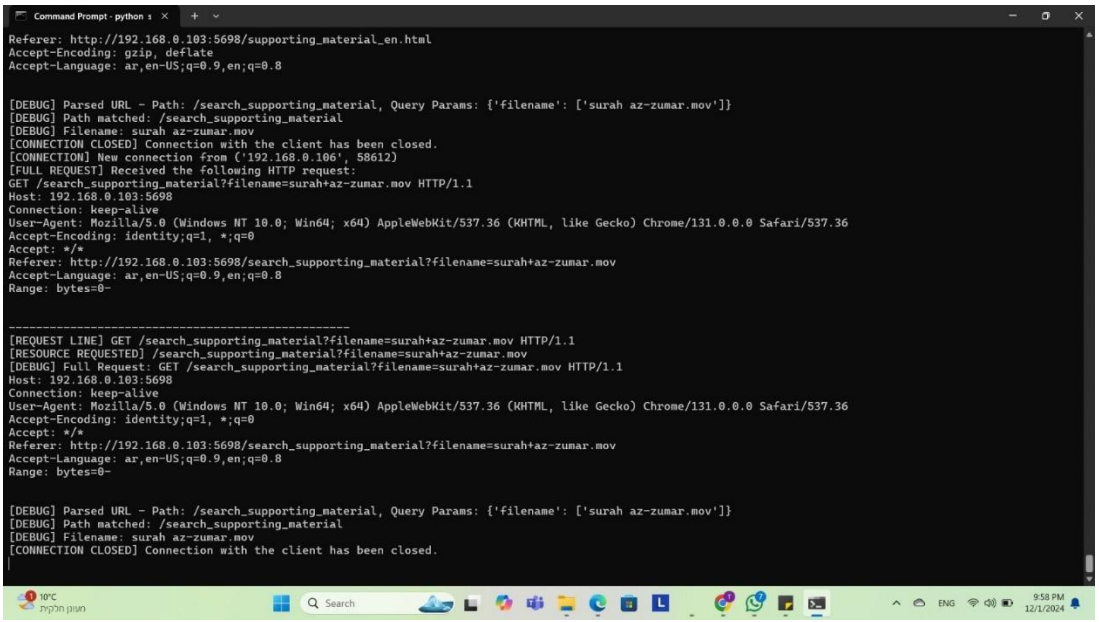


Figure 27: Form of CMD when requesting Surah Az-Zumar.mov from another laptop

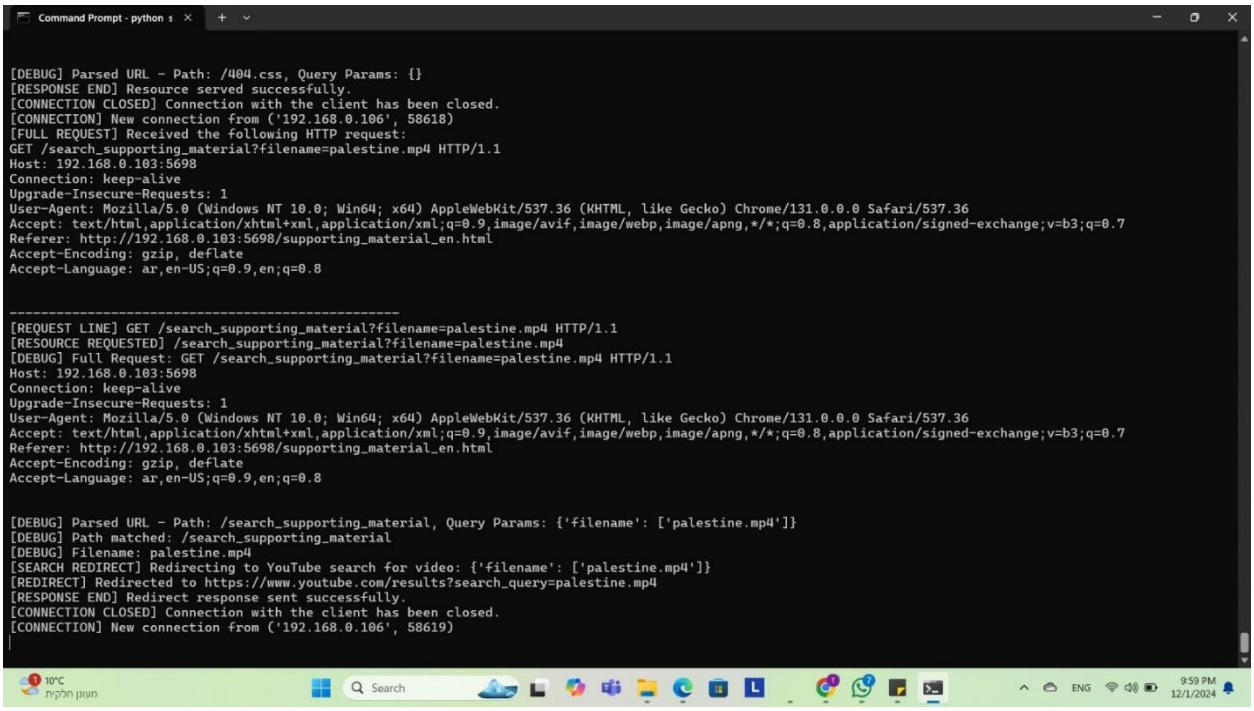


Figure 28: Form of CMD when requesting palestine.gi from another laptop

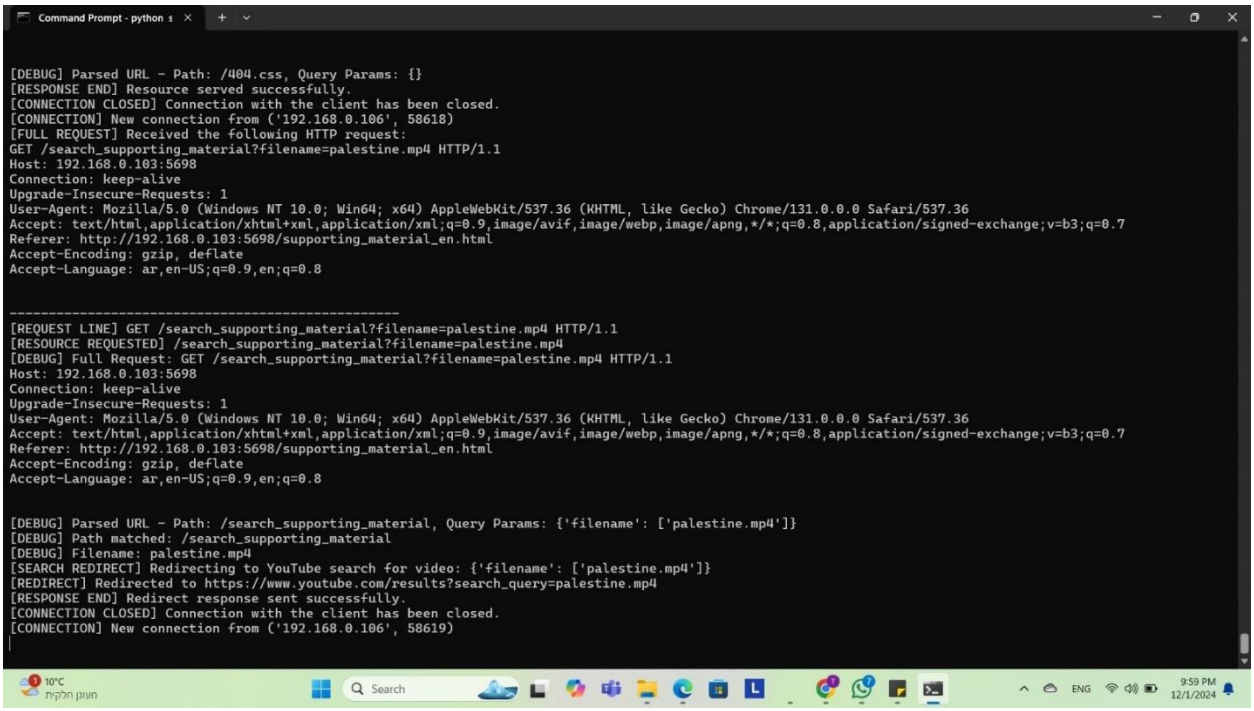


Figure 29: Form of CMD when requesting palestine.mp4 from another laptop

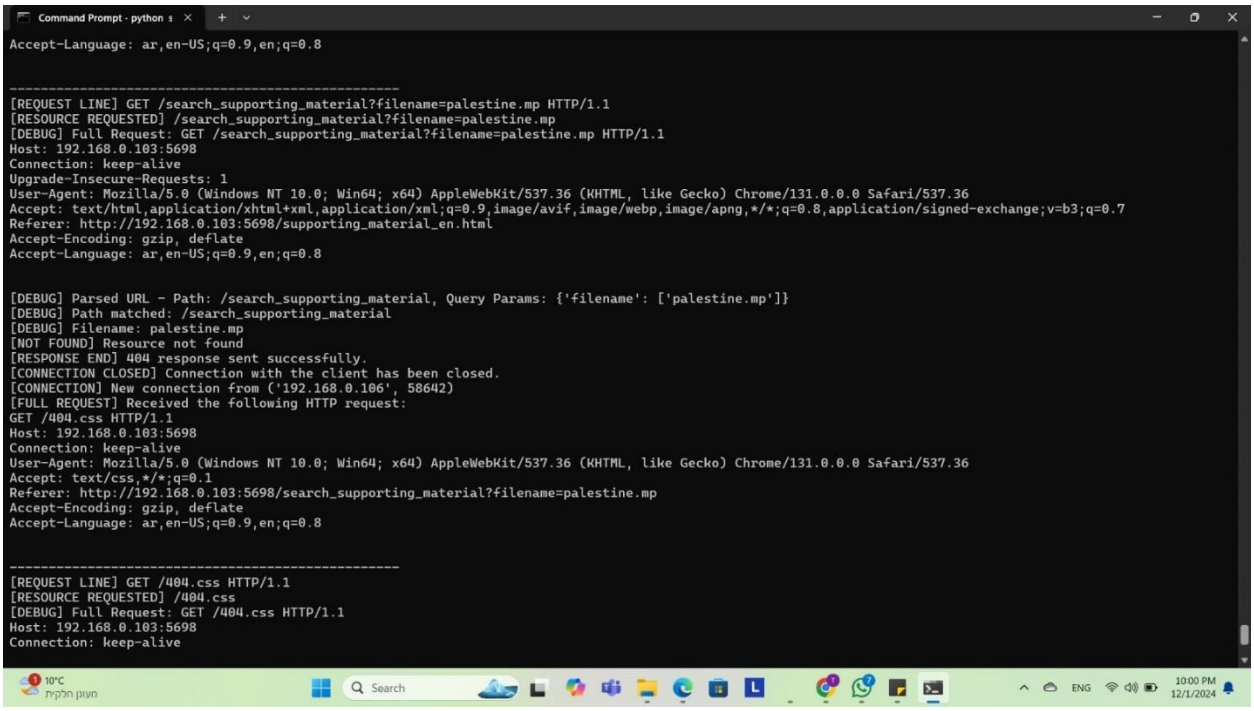


Figure 30: Form of CMD when requesting palestine.mp from another laptop

When the second laptop sends a request to the main laptop (such as a ping), the data travels over the local network, using the main laptop's IP address as the destination. If Network Discovery and File Sharing are enabled, the request can also be used to discover shared resources. However, communication can fail if firewalls block incoming traffic or if client isolation is enabled on the router, preventing devices from communicating even if they're on the same network.

➢ Task 3:

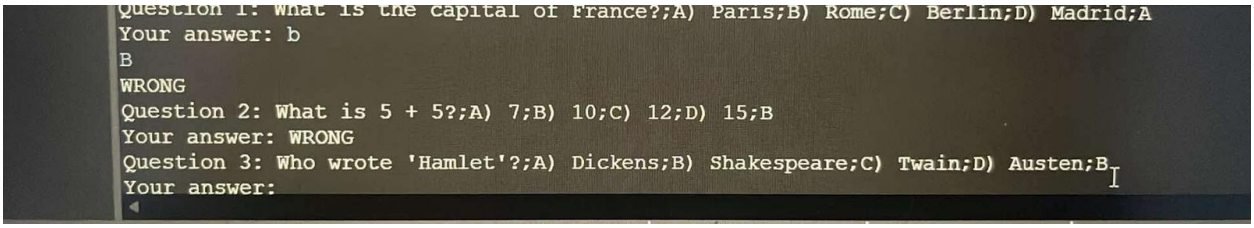


Figure 31: Trivia server and client - 1

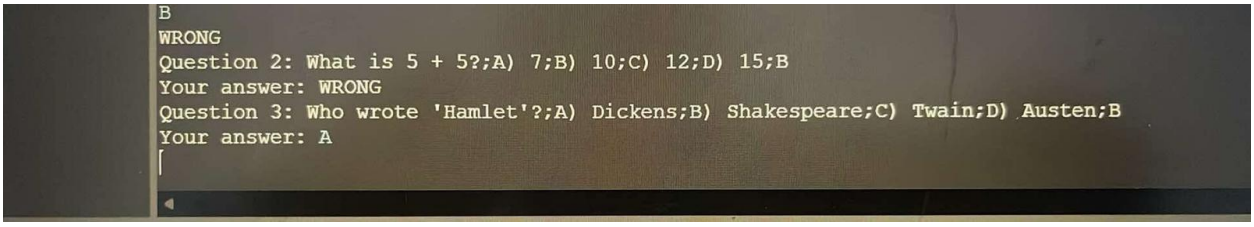


Figure 31: Trivia server and client - 2

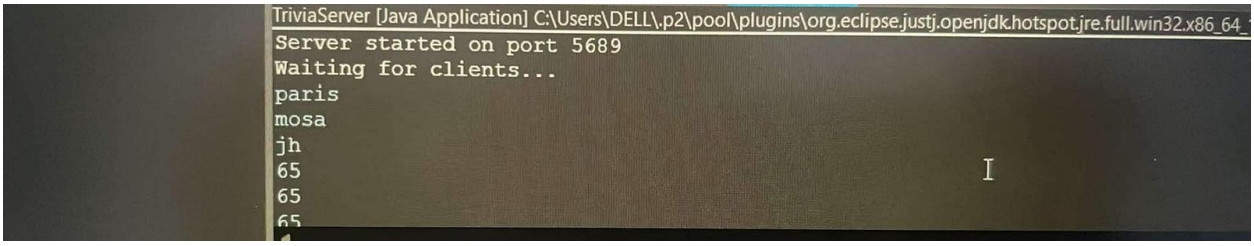


Figure 31: Trivia server and client - 3

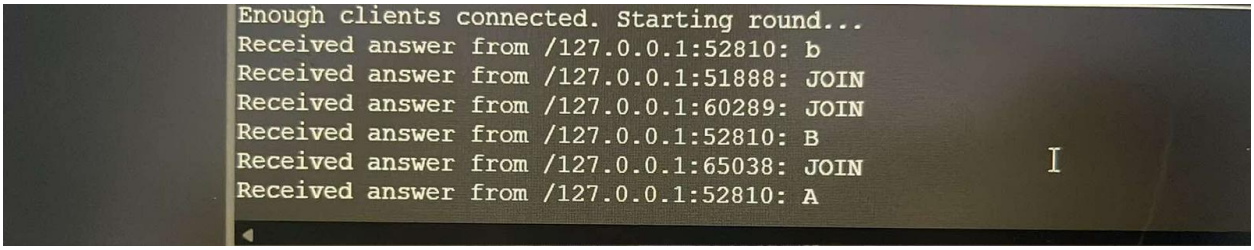


Figure 31: Trivia server and client – 4

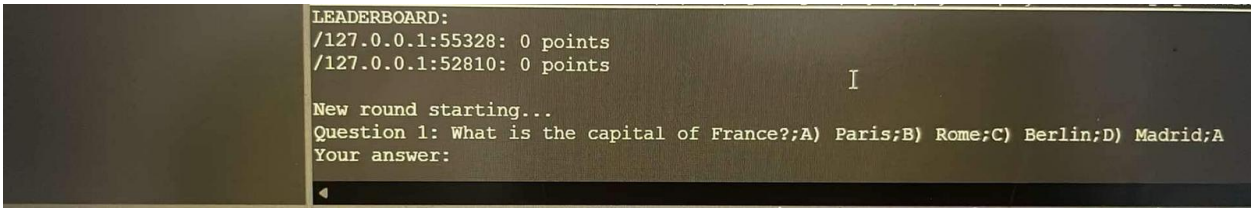


Figure 31: Trivia server and client – 5

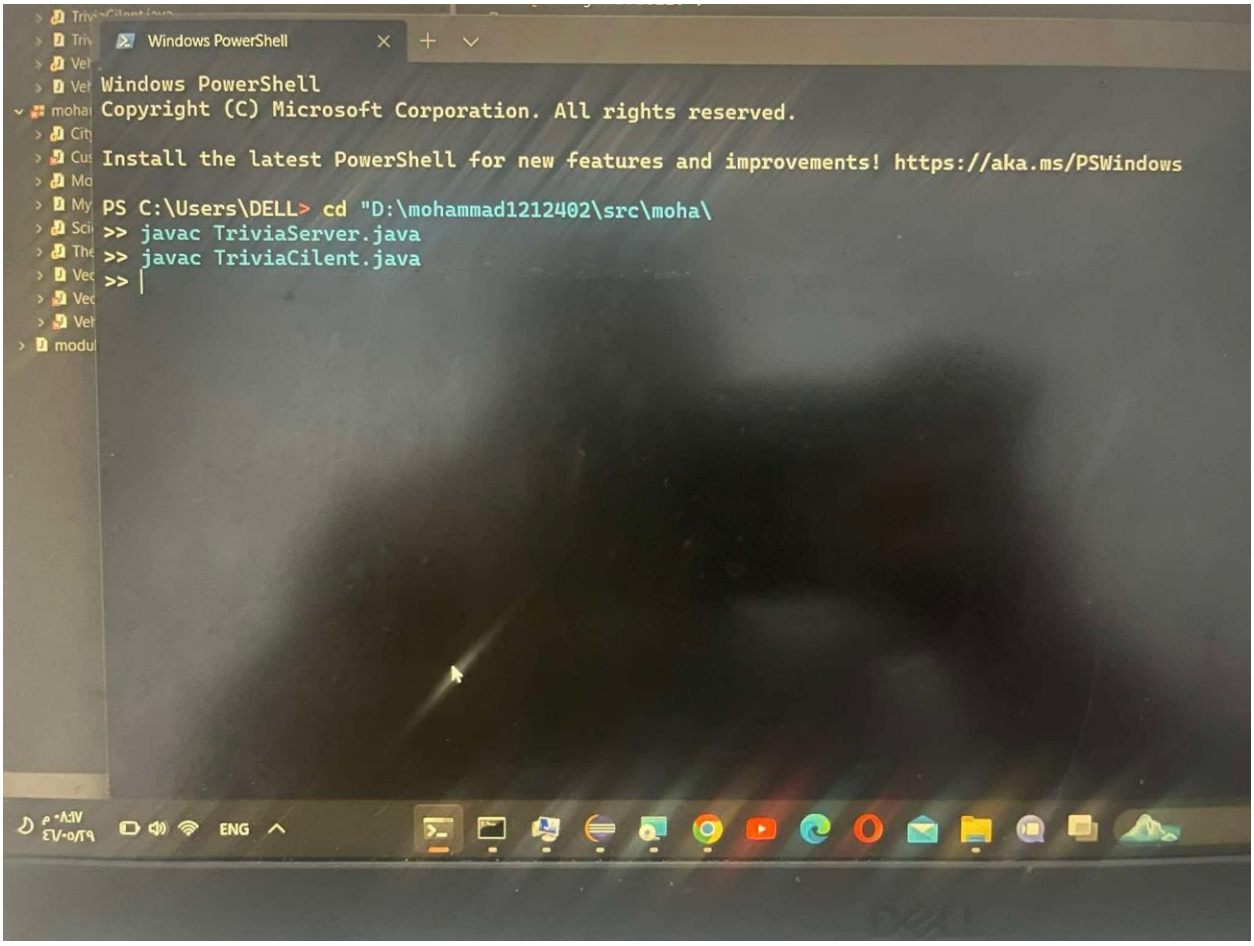


Figure 31: Trivia server and client – 6

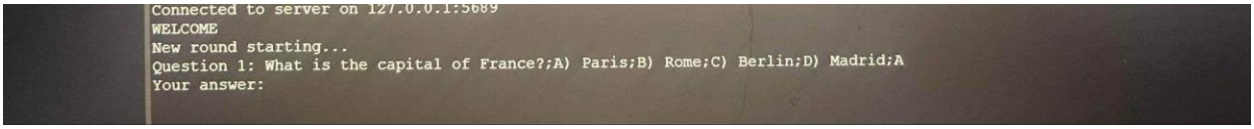


Figure 31: Trivia server and client – 7

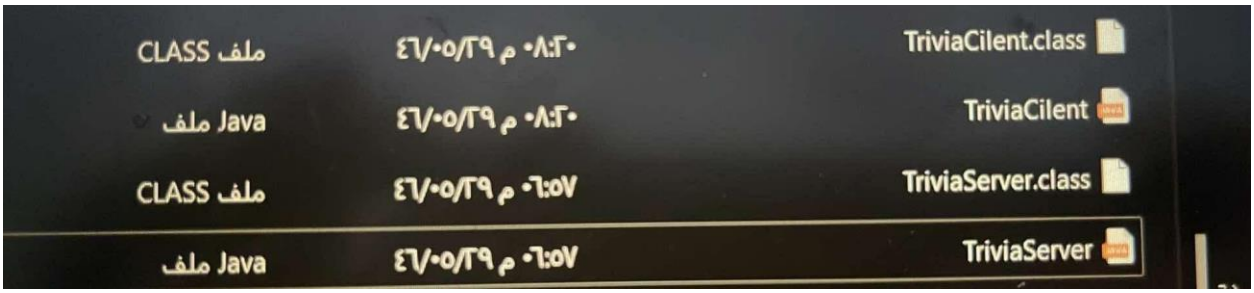


Figure 31: Trivia server and client – 8

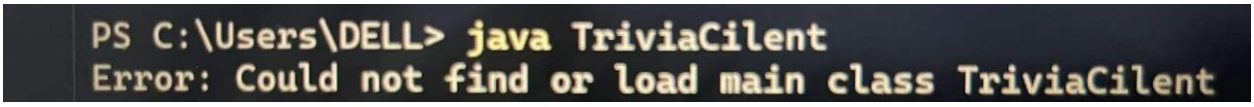


Figure 31: Trivia server and client – 9

Trivia Server and Client Overview

- **TriviaServer:**

Listens for client connections on port 5689. Sends trivia questions to connected clients. Collects answers, checks correctness, updates scores, and broadcasts results. Pauses between rounds and repeats the process.

- **TriviaClient:**

Connects to the server, listens for questions, and sends answers. Displays the leaderboard after each round.

- **Communication:**

Uses UDP (via DatagramSocket) for sending and receiving messages (questions, answers, and scores) between the server and clients.

- **Game Flow:**

Server waits for clients. Server starts the game, sending questions to clients. Client's answer, and server checks answers, updates scores. Server broadcasts the leaderboard after each round. Game repeats until it is finished.

Alternative Solutions, Issues, and Limitations:

In task 2, while creating the main_en.html and main_ar.html web page as required in the project file, but because we faced the following problem, which is because of placing the image of each member with his private information in the box, we faced two problems: 1. The image didn't appear in its correct position in the box. 2. The box was tilted to the left of the page, and therefore the largest part of the member's image wasn't shown. According to our analysis of the cause of the problem, we found that the reason for it was the overlap of the special style in the main HTML page with the special style of the box. We tried to solve this problem by creating a separate CSS file for it, but this solution didn't work because of the continued overlap (such as the overlap of the body of each file due to their difference and other overlaps). Therefore, after searching the Internet (link in references [11]), we found that the ideal solution is to create an HTML file for each member of ours and link it to the same CSS file. We used these working files in the main HTML file, and we also hid the scroll bar for these 3 files to appear to the client as if they were one web page. Because of these interferences, we also created a special CSS file for the chapter topic and used it in the HTML file in this part.

Doing it the same way as before, we worked on the Arabic version of the file as required in HTML.

The new files we had to add:

- fotTopic.css
- member.css
- mohammadInfo.html
- mohammadInfo_ar.html
- toqaInfo.html
- toqaInfo_ar.html
- omarInfo.html
- omarInfo_ar.html

Teamwork

Task Chart

Task 1

- Assigned to Omar:
    - Subtasks: a, b, c

Task 2

- Assigned to Toqa:
    - Subtasks: a, c, server
- Assigned to Mohammad:
    - Subtasks: b, c

Task 3

- Assigned to Mohammad:
    - Subtasks: client, server

Report Sections

Toqa's Responsibilities:

- Sections to write:
    - Cover page
    - Theory and Procedure
    - Results and Discussions (for Task 2)
    - Alternative Solutions
    - Issues and Limitations
    - Teamwork
    - References
    - Table of contents
    - List of figures
    - List of tables
    - Flow chart

Omar's Responsibilities:

- Sections to write:
    - Results and Discussions (for Task 1)

Mohammad's Responsibilities:

- Sections to write:
    - Results and Discussions (for Task 3)

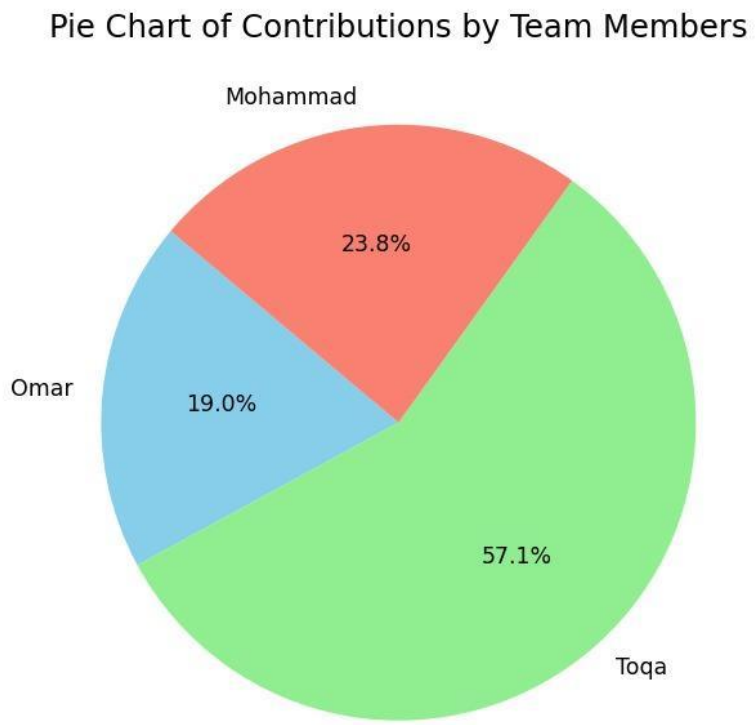


Figure 32: Contribution Distribution Among Team Members

References:

[1]:https://www.computernetworkingnotes.com/networking-tutorials/basic-networking-commands-explained-with-examples.html#google_vignette

[2]:https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/nslookup

[3]:https://www.namecheap.com/support/knowledgebase/article.aspx/9667/2194/what-are-traceroute-ping-telnet-and-nslookup-commands/#telnet

[4]:https://developer.mozilla.org/en-US/docs/Web/HTTP/Evolution_of_HTTP#http0.9_%E2%80%93_the_one-line_protocol

[5]:https://en.wikipedia.org/wiki/HTTP

[6]:https://datatracker.ietf.org/doc/html/rfc2616

[7]: https://ritaj.birzeit.edu/bzu-msgs/attach/2681408/Chapter_2_v8.0V1.pdf

[8]: https://www.wireshark.org/about.html

[9]: https://vahid.blog/post/2020-12-15-how-the-internet-works-part-i-infrastructure/

[10]:https://www.researchgate.net/figure/Approaches-to-web-access_fig1_228644561

[11]:https://www.youtube.com/watch?v=dIAG3cWAOlU