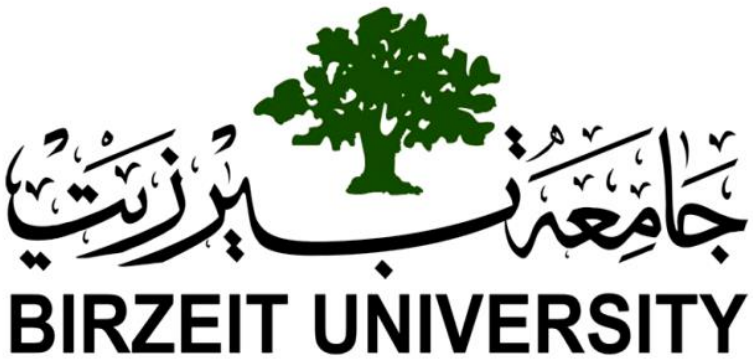


Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Computer Networks – ENEE2360

Project Report

Prepared by:

Name	ID	Section
Sara Al-lahaleh	1211083	1
Malak Ammar	1211470	1
Asmaa Fares	1210084	3

Instructor: Ibrahim Nemer.

Date: 06/05/2024.

Abstract

Our project is a multifaceted exploration bridging networking concepts and web development. Initially, we dive into network analysis using tools like Command Prompt and Wireshark, focusing on packet tracking and DNS message analysis. Transitioning to practical implementation, we venture into socket programming, emphasizing UDP for a peer-to-peer model, facilitating efficient communication. Building upon this groundwork, we construct a robust web server capable of managing HTTP requests and serving diverse file types. Additionally, we craft visually engaging web pages using HTML and CSS to spotlight project team information. Through this endeavor, our goal is to deepen our understanding of networking fundamentals while honing our web development proficiency.

Table of Contents

Table of figures

❖ Part 1

- **Project Questions**

1- In your own words, what are ping, tracert, nslookup, and telnet (write one sentence for each one).

2- Make sure that your computer is connected to the internet and then run the following commands:

   a) Ping a device in the same network, e.g. from a laptop to a smartphone

   b) ping www.stanford.edu

   c) From the ping results, do you think the response you got is from USA? Explain your answer briefly.

   d) tracert www.stanford.edu

   e) nslookup www.stanford.edu

3- use Wireshark to capture some DNS messages.

➢ Question 1

- **Ping:** Internet Package Grouper. It is a widely utilized utility tool for troubleshooting network connectivity. It assists in testing the reachability of hosts/servers, verifying internet connections, assessing network interface card functionality, and diagnosing DNS issues.

- **Traceroute**: A tool utilized to visualize the precise path data takes to reach its destination. This is achieved by sending Internet Control Message Protocol (ICMP) echo packets to the target destination and observing the route each packet takes through the network.

- **NSLOOKUP**: A utility used to retrieve the IP address or DNS record associated with a given host name. Additionally, it can identify the domain name associated with a specific IP address.

- **Telnet**: A network protocol facilitates two-way, text-based communication between two machines. It enables users to establish connections with remote computers over a TCP/IP network, providing a means for interactive communication and remote access to systems.

➢ Question 2

- Ping a device in the same network, e.g. from a laptop to a smartphone

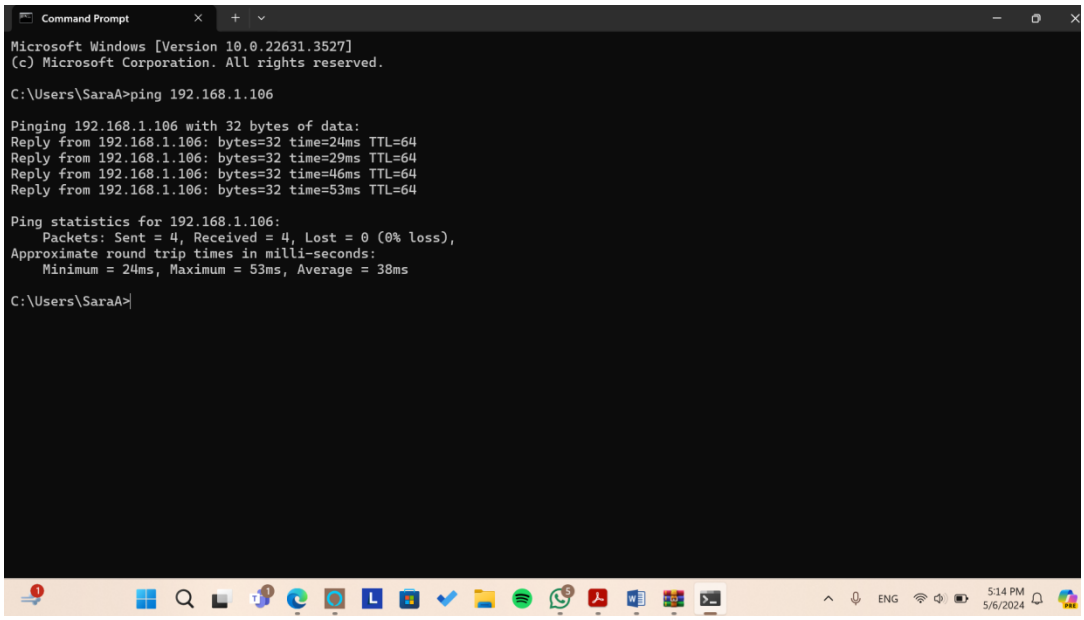


Figure 1: Ping a device

➢ Explanation

Ping Command Test Using Command Prompt

In our testing, we utilized the Command Prompt (CMD) to execute the ping command targeting the IP address 192.168.1.106, which corresponds to a mobile phone's IP. We sent four data packets using this command, each containing 32 bytes of data.
Each packet sent had the following parameters:

1. **Packet Size**: Each packet was 32 bytes in size.
2. **Round-Trip Time**: The time taken for each packet to be sent and received was measured in milliseconds.
3. **Time To Live (TTL)**: This parameter indicates the number of networks hops the packet is allowed to traverse before being discarded.

Upon execution, **all four packets were successfully sent and received**, as evidenced in the output. The average round-trip time for these packets was 38 milliseconds. This test demonstrates the functionality and performance of the network connection to the specified IP address.

- **Ping www.stanford.edu**

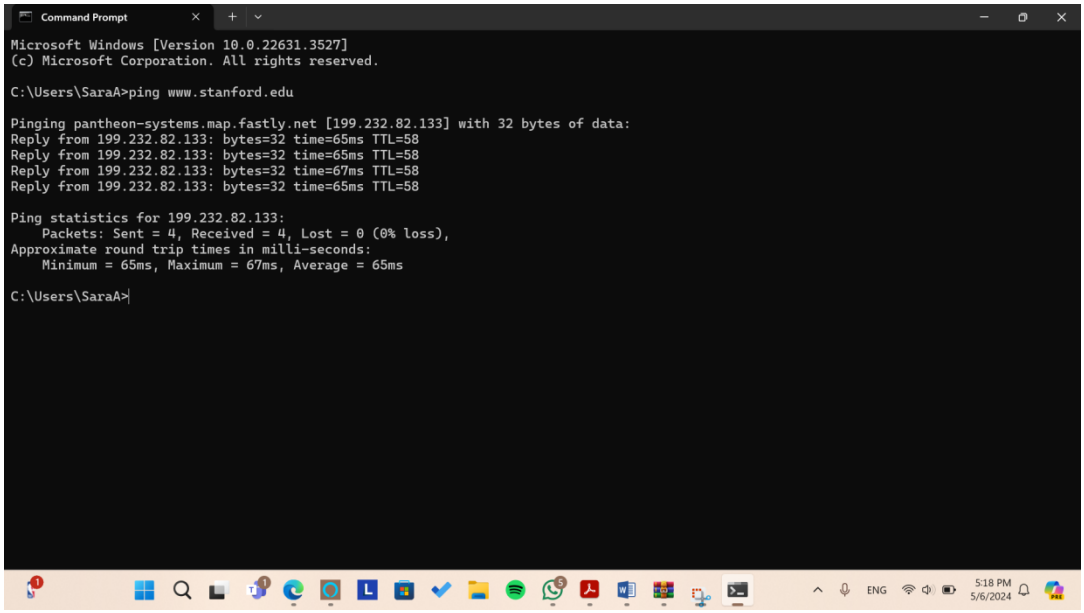


Figure 2: Ping www.stanford.edu

➢ **Explanation**

Ping Test on www.stanford.edu

In our investigation, we utilized Command Prompt (CMD) to perform a ping on the domain www.stanford.edu, expanding beyond simple device testing to include website analysis. This process involved several key steps:

1- DNS Resolution

The system first needed to resolve the domain name www.stanford.edu to its corresponding IP address by querying the DNS (Domain Name System) server.

2- Packet Transmission

Once the IP address (199.232.82.122) of the domain (pantheon-systems.map.fastly.net) was determined, the system proceeded to send ping requests to this address, similar to our previous example with a specific device.

3- Ping Response Details

The output of the ping command provided insightful details, including:

- **Round-Trip Time (RTT)**: The time taken (in milliseconds) for each packet to travel to the destination and back.
- **Packet Size**: Each packet's size in bytes.
- **Time from Sending to Receiving**: The duration it took for each packet to be transmitted and received.
- **TTL (Time to Live)**: The maximum number of networks hops each packet can traverse before being discarded.

By analyzing this information, we gained valuable insights into the responsiveness and network performance associated with www.stanford.edu. This comprehensive approach showcases the versatility of ping testing in evaluating website accessibility and connectivity.

- **From the ping results, do you think the response you got is from USA? Explain your answer briefly.**

  Based on the ping results for www.stanford.edu, the IP address 199.232.82.133 is associated with the hostname pantheon-systems.map.fastly.net. Determining the precise geographic location of this IP address is not straightforward solely from the address itself. Fastly.net is a global content delivery network (CDN) utilized by various websites, including Stanford University's site. The IP address could correspond to a server in the USA or elsewhere depending on Fastly's network configuration and server locations. To pinpoint the location accurately, a geolocation service like IPGeolocation.io or ipinfo.io would be necessary. These services suggest potential locations such as California, USA, or France based on the IP address, but it's important to note that these are speculative findings and the actual location cannot be conclusively determined from the ping results alone.

- **Tracert www.stanford.edu**

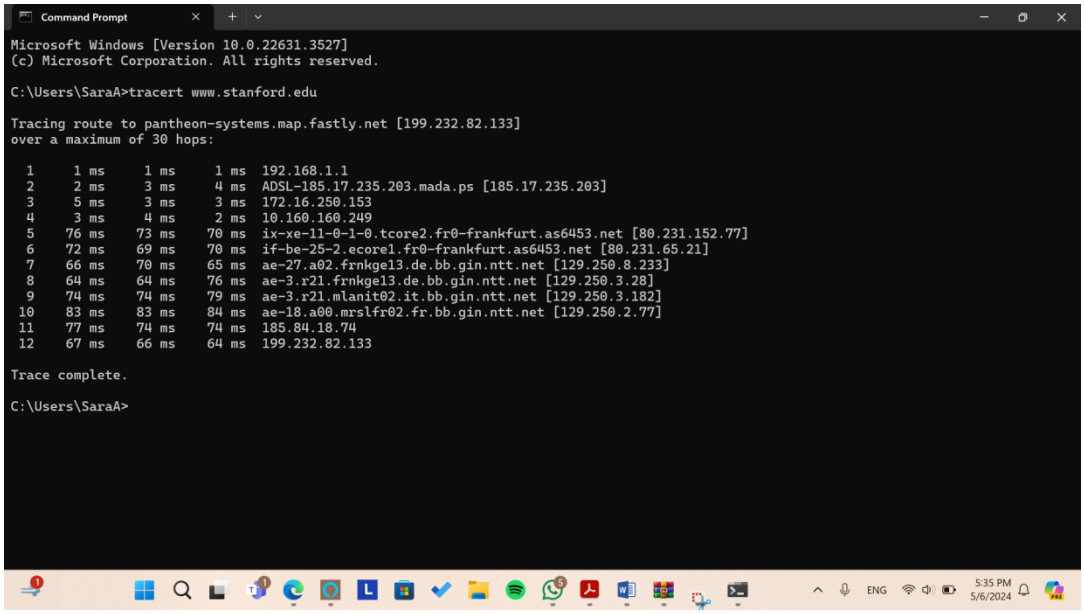


Figure 3: tracert www.stanford.edu

➢ **Explanation**

Traceroute Analysis for www.stanford.edu

We conducted a traceroute using Command Prompt (CMD) to analyze the route to www.stanford.edu. The initial output displays the target domain along with its corresponding IP address. The traceroute output is organized into columns with the following headers:

- **Hop:** Indicates the sequence number of the hop in the route.
- **Hostname (or IP address):** Shows the IP address or hostname of the router encountered at each hop.
- **Time (ms):** Represents the round-trip time in milliseconds for a packet to travel to the specified hop and back.

Each row in the traceroute output corresponds to a specific hop along the route. The first hop typically corresponds to our local router or gateway, with subsequent hops representing intermediate routers or switches that the packet traverses.

The time values displayed in milliseconds reflect the round-trip time for packets sent to each hop. These times are based on sending multiple packets (usually three) to each hop to assess network latency and route efficiency. Traceroute provides valuable insights into the path packets take through the network and helps identify potential bottlenecks or latency issues along the route to the destination.

- **Nslookup www.stanford.edu**

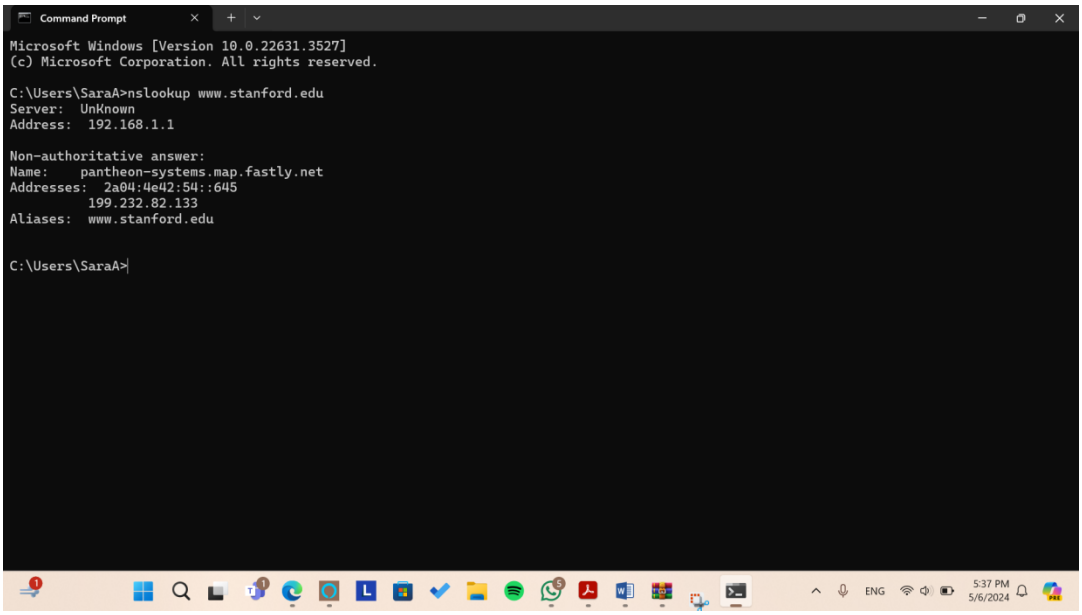


Figure 4: nslookup www.stanford.edu

➢ **Explanation**

NSLOOKUP Analysis for www.stanford.edu

We utilized Command Prompt (CMD) to execute the nslookup command, yielding the following output:

- **Server:** The output specifies the DNS server responsible for providing the resolution. In our example, the server is identified as "Unknown" with the corresponding IP address of "192.168.1.1".

- **Non-authoritative answer:** This indicates that the information obtained was retrieved from a DNS server that is not the authoritative source for the domain.

- **Name:** The resolved domain name is displayed as "www.stanford.edu", which resolves to "pantheon-system.map.fastly.net".
- **Addresses:** The output includes the IP addresses associated with the resolved domain, providing essential network routing information.
- **Aliases:** Additionally, any aliases or alternate domain names associated with the resolved IP addresses are listed, helping to identify related domains within the network context.

This nslookup analysis assists in understanding domain resolution processes and the underlying DNS infrastructure responsible for mapping domain names to corresponding IP addresses. It provides valuable insights into DNS server configurations and domain aliasing within network environments.

➢ Question 3

Use Wireshark to capture some DNS messages.

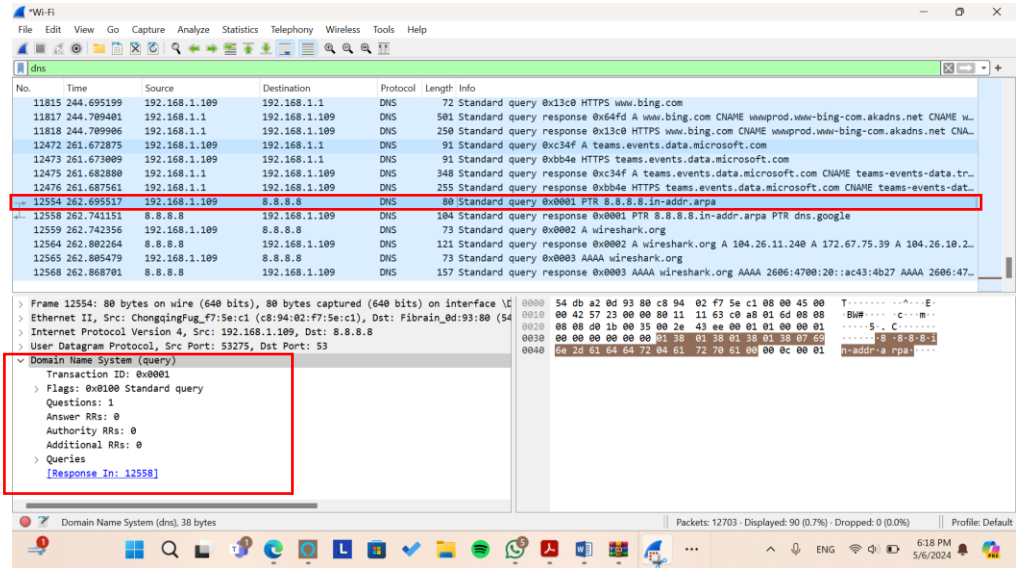


Figure 5: Wireshark DNS request

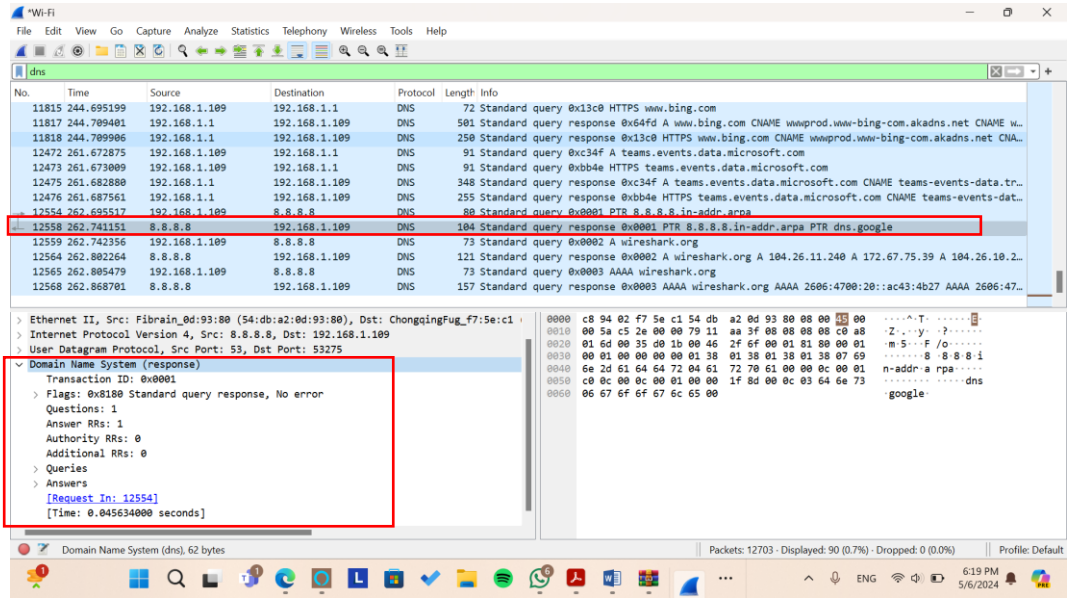


Figure 6: Wireshark DNS response

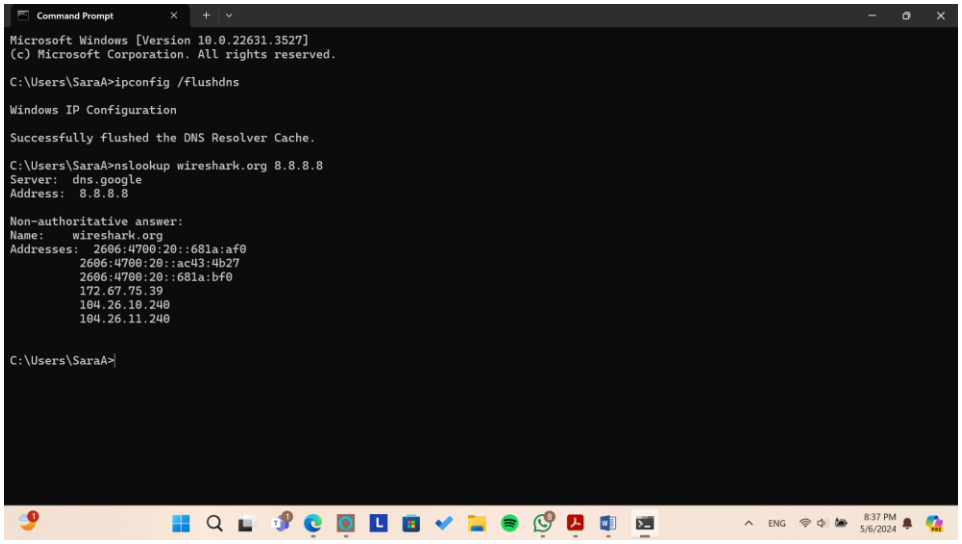


Figure 7: nslookup wireshark

When examining DNS (Domain Name System) messages in Wireshark, we observe DNS request and response packets, various DNS record types, queries for local domains, DNS flags, and more. The DNS protocol serves to translate human-readable domain names into corresponding IP addresses. This translation enables computers to communicate effectively over networks by identifying specific destinations based on domain names.

Below is an example showcasing DNS request/response captured in Wireshark:

In Figure 6, we executed the command `nslookup wireshark.org 8.8.8.8` on CMD to query the DNS server IP address (in this case, Google's Public DNS server) for domain name resolution (`wireshark.org`) and obtained the following results:

- **Server:** The output specifies that the DNS server used for this query is `dns.google` (Google's Public DNS) with the IP address `8.8.8.8`.
- **Non-authoritative answer:** Indicates that the response obtained is from a DNS server that is not the authoritative source for the domain `wireshark.org`.
- **Name:** The resolved domain name is `wireshark.org`.
- **Addresses:** The output provides a list of IP addresses associated with `wireshark.org`, including both IPv4 and IPv6 addresses.

Referring to Figure 4, the initial request is depicted where we queried `8.8.8.8`, the domain name server, with a **transaction ID** of `0x0001`. In Figure 5, the response to our query is shown, with the response having the same **transaction ID** as the initial request. Additionally, the timestamp of when the response arrived is visible within the query section.

This example demonstrates the DNS query process, including server selection, response handling, and the association of domain names with corresponding IP addresses, as captured and analyzed using Wireshark.

❖ Part 2

```
import socket
import threading
import datetime

# Peer details
PEER_FIRST_NAME = input("Enter your first name: ")
PEER_LAST_NAME = input("Enter your last name: ")

# IP and MASK details
IP = '192.168.88.13'
MASK = '255.255.255.0'
```

Figure 8: IP address and subnet mask

Here, the IP address and subnet mask are hardcoded.

```
1 usage
def get_ID(ip, mask):
    ip = ip.split('.')
    mask = mask.split('.')
    ip = [int(octet) for octet in ip]
    mask = [int(octet) for octet in mask]
    subnet = [str(ip_octet & mask_octet) for ip_octet, mask_octet in zip(ip, mask)]
    host = [str(ip_octet & ~mask_octet) for ip_octet, mask_octet in zip(ip, mask)]
    broadcast = [str(ip_octet | ~mask_octet & 0xFF) for ip_octet, mask_octet in
                 zip(ip, mask)]  # Fixing broadcast calculation
    subnet_mask = '.'.join(subnet)
    print('Subnet: {0}'.format(subnet_mask))
    print('Host: {0}'.format('.'.join(host)))
    print('Broadcast address: {0}'.format('.'.join(broadcast)))
    return subnet_mask
```

Figure 9: get_ID() function

This function get_ID() takes an IP address and subnet mask as input, calculates the subnet, host, and broadcast addresses, prints them, and returns the subnet mask.

```
1 usage
def receive_message():
    socket_instance = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    socket_instance.bind(('', SERVER_PORT))

    while not exit_flag:
        try:
            data, _ = socket_instance.recvfrom(BUFFER_SIZE)
            message = data.decode()
            sender_ip, sender_first_name, sender_last_name, message_content = message.split('|')
            print(f"Received a message from {sender_first_name} {sender_last_name} at {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
            print(f"Message Content: {message_content}")
            # Update last received message from sender
            last_received_messages[sender_ip] = (sender_first_name, sender_last_name, datetime.datetime.now())

            # Check if you are the receiver
            if sender_ip == IP:
                print("You are the receiver.")


        except socket.timeout:
            continue
        except KeyboardInterrupt:
            break
```

Figure 10: receive_message() function

This function receive_message() continuously listens for incoming messages using a UDP socket. It decodes the received message, extracts sender details and message content, prints them, updates the last received message from the sender, and checks if the current host is the receiver.

```
1 usage
def send_message():

    socket_instance = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    socket_instance.bind(('', 0))  # Bind to any available port on this machine

    while not exit_flag:
        message_content = input("\nEnter your message: \n")
        message = f"{socket.gethostbyname(socket.gethostname())}|{PEER_FIRST_NAME}|{PEER_LAST_NAME}|{message_content}"
        broadcast_address = "192.168.88.255"
        try:
            socket_instance.sendto(message.encode(), (broadcast_address, SERVER_PORT))
        except PermissionError:
            print("Permission denied. Make sure to run the script with appropriate permissions.")
            break
```

Figure 11: send_message() function

This function send_message() sends messages using a UDP socket. It takes user input for the message content, constructs the message with sender details, and sends it to a broadcast address.

```
# Display IP and MASK details
subnet_mask = get_ID(IP, MASK)
print("Subnet Mask:", subnet_mask)
```

Figure 12: get_ID() function call

This part calls the get_ID() function to calculate and print subnet, host, and broadcast addresses based on the provided IP address and subnet mask.

```
# Server details
SERVER_PORT = 5051
BUFFER_SIZE = 1024
```

Figure 13: Server and Buffer ports

Defines the server port and buffer size for receiving data.

```
# Dictionary to store the last received message from each peer
last_received_messages = {}
```

Figure 14: Initialize Dictionary

Initializes an empty dictionary to store the last received message from each peer.

```
# Flag to indicate when to exit the threads
exit_flag = False
```

Figure 15: Flag

Defines a flag to indicate when to exit the threads.

```
# Start receiving messages in a separate thread if the role is receiver

receive_thread = threading.Thread(target=receive_message, args=())
receive_thread.start()

# Start sending messages in a separate thread if the role is sender

send_thread = threading.Thread(target=send_message, args=())
send_thread.start()
```

Figure 16: Receive and send thread

Creates and starts a thread to execute the receive_message() function for receiving messages. Then

Creates and starts a thread to execute the send_message() function for sending messages.

```
send_thread.join()
receive_thread.join()
# Display the last received message from each peer
print("\nLast received messages from other peers:")
for ip, (first_name, last_name, timestamp) in last_received_messages.items():
    print(f"Received a message from {first_name} {last_name} at {timestamp.strftime('%Y-%m-%d %H:%M:%S')}")
```

Figure 17: Send and Receive thread join

Finally, it Waits for the send and receive threads to finish execution. Then Prints the last received message from each peer stored in the last_received_messages dictionary.

Running this code on three machines demonstrates a simple peer-to-peer messaging system using UDP communication, allowing real-time communication between the machines on the same network.

Here are some screenshots for the run on three machines on the same broadcast address:

- **The first machine of IP=192.168.88.13**

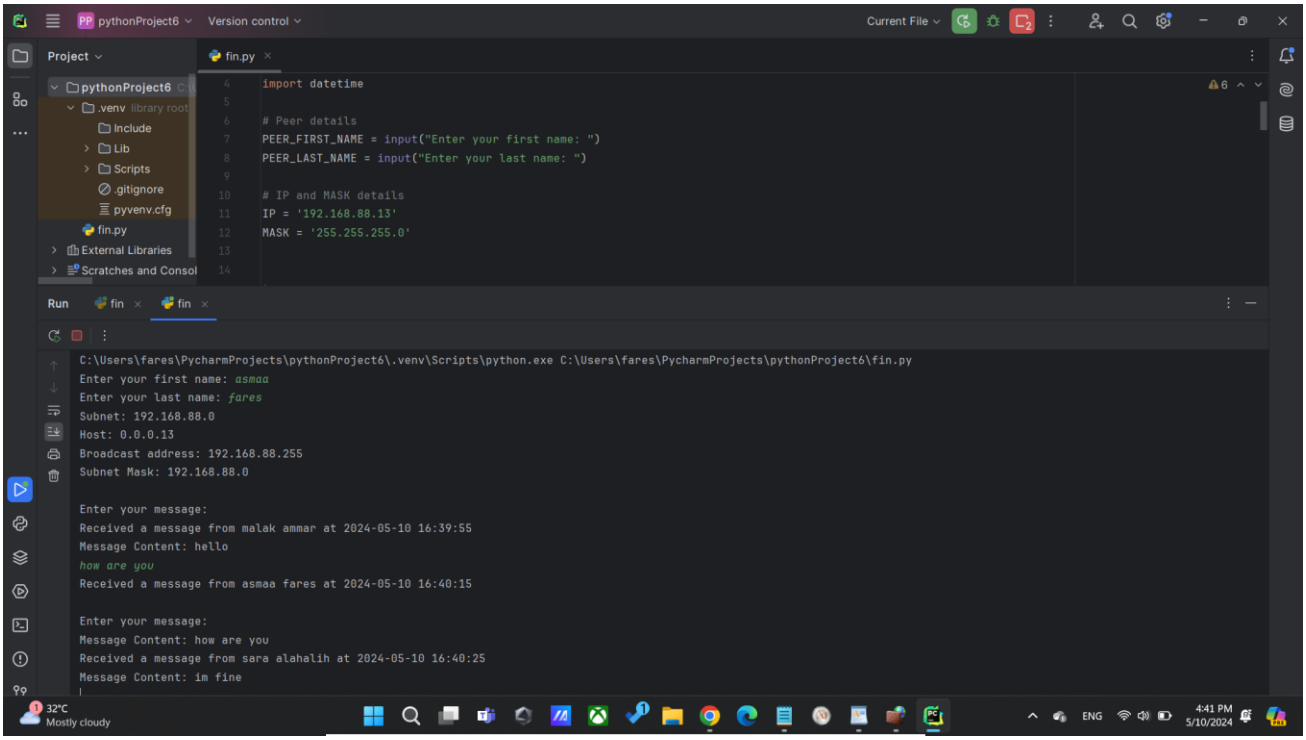


Figure 18: First machine

- **The second machine of IP=192.168.88.3**

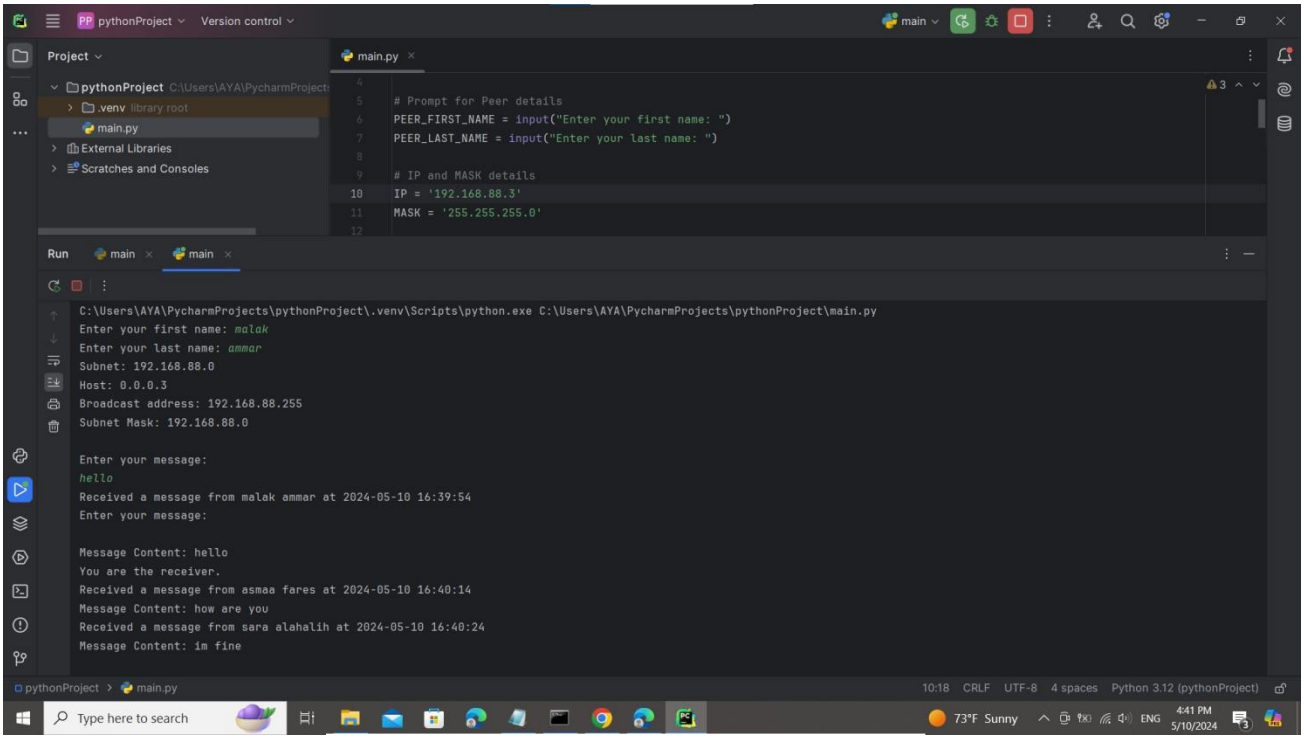


Figure 19: Second machine

- **The third machine of IP=192.168.88.1**

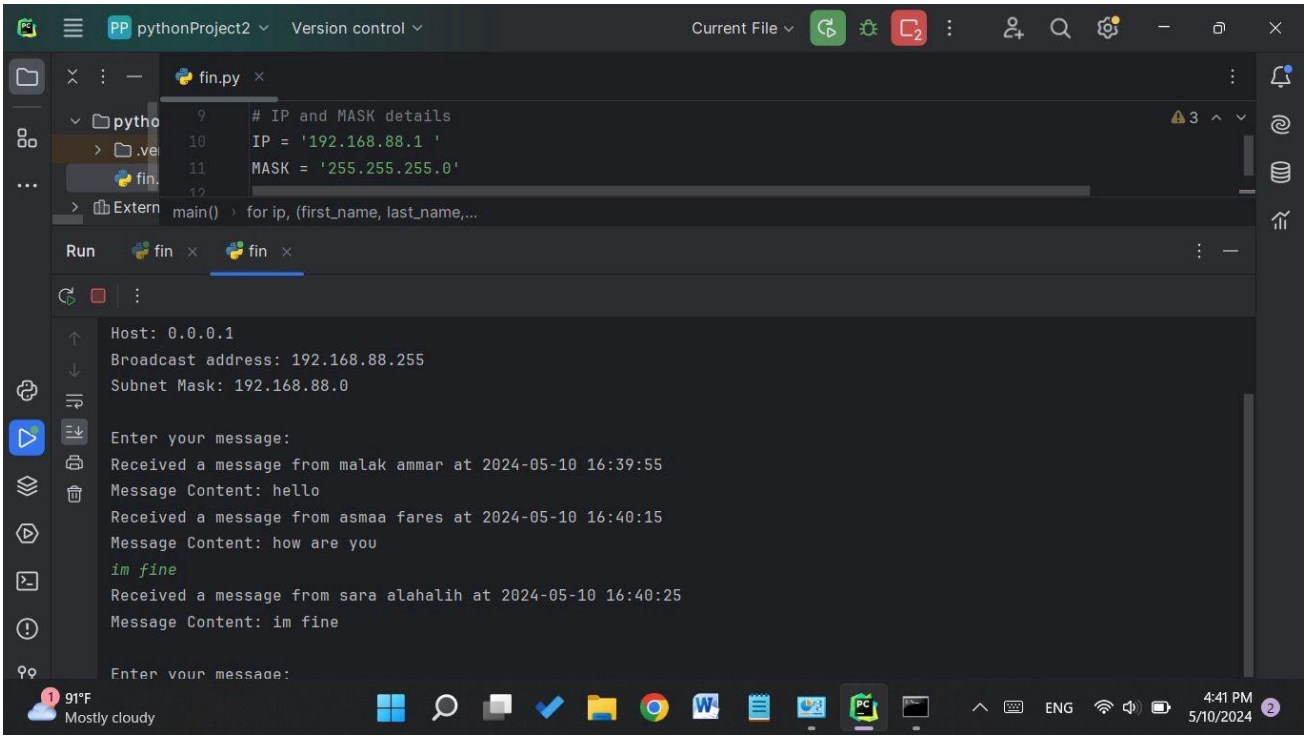


Figure 20: Third machine

❖ Part 3

➢ Objective: Implementing a Versatile Web Server Using Socket Programming

In alignment with the aim outlined for Part Three, our focus was on the formal implementation of a versatile web server utilizing Python, CSS, and HTML. Leveraging socket programming principles, we developed a comprehensive solution capable of effectively managing HTTP requests while maintaining flexibility and scalability on port 6060. The primary objective was to ensure that the web server could handle diverse file types and execute redirects as specified by project requirements. Additionally, emphasis was placed on creating a generalized codebase to maximize reusability across different scenarios. By adhering to these objectives, we aimed to deliver a robust system capable of accommodating various user inputs, such as different language preferences, and responding accordingly.

➢ RFC2616: Entity Tag Cache Validators in the HTTP Protocol

Entity Tag Cache Validators (ETags), delineated in RFC 2616 within the HTTP protocol framework, serve as unique identifiers for resources hosted on web servers. They facilitate streamlined interactions between web clients and servers. Each resource is assigned a distinct ETag, akin to a digital signature, enabling servers to ascertain whether modifications have occurred since a prior request. Consequently, upon revisiting a webpage, clients can query the server regarding any alterations since the previous visitation. If none are detected, the server promptly notifies the client, thereby conserving both time and bandwidth. ETags thus function as catalysts for enhancing browsing efficiency and optimizing server performance.

➢ main.py: Python Web Server Implementation

The Python script `**main.py**` embodies the core functionality of our web server implementation, showcasing the fusion of socket programming and HTTP protocol handling. Through meticulous design and implementation, this script serves as the backbone of our server, orchestrating the reception and processing of HTTP requests and the subsequent dispatching of appropriate responses. At its essence, `main.py` acts as the gatekeeper, efficiently managing client-server interactions while adhering to the intricacies of the HTTP standard.

One of the script's key strengths lies in its a**daptability to various request types and resource types**, empowering our server to handle diverse web content seamlessly. Leveraging conditional logic and file-based routing, it intelligently serves **HTML, CSS, image files, and more**, catering to the dynamic needs of modern web applications. Additionally, the **integration of error handling** mechanisms ensures robustness and reliability, mitigating potential issues such as file not found errors and providing informative responses to clients.

- **Functionality Overview: Core Features and Operations of the Python Web Server**

  - **Part 1: Server Initialization and Socket Binding**

    In this part, the Python script initializes a socket server using the socket module. It specifies the host and port to which the server should bind (**serverPort = 6060**). After binding, the server enters a listening state to accept incoming connections from clients.

  - **Part 2: Parsing HTTP Requests**

    This part focuses on extracting relevant information from incoming HTTP requests. The **parse_request()** function dissects each request to identify elements like the requested file path, HTTP method (GET or POST), and any additional parameters or headers.

  - **Part 3: Handling GET Requests**

    GET requests, which are typically used to retrieve resources **like HTML pages, CSS files, and images**, are managed here. The server examines the requested file path, determines its content type, and sends an appropriate HTTP response containing the requested resource.

  - **Part 4: Handling POST Requests**

    POST requests, commonly used to submit data to a server, are addressed in this part. When the server receives a POST request, it extracts the relevant data (e.g., image name) from the request body and processes it accordingly. In this case, it **retrieves the corresponding image file and sends it back to the client**.

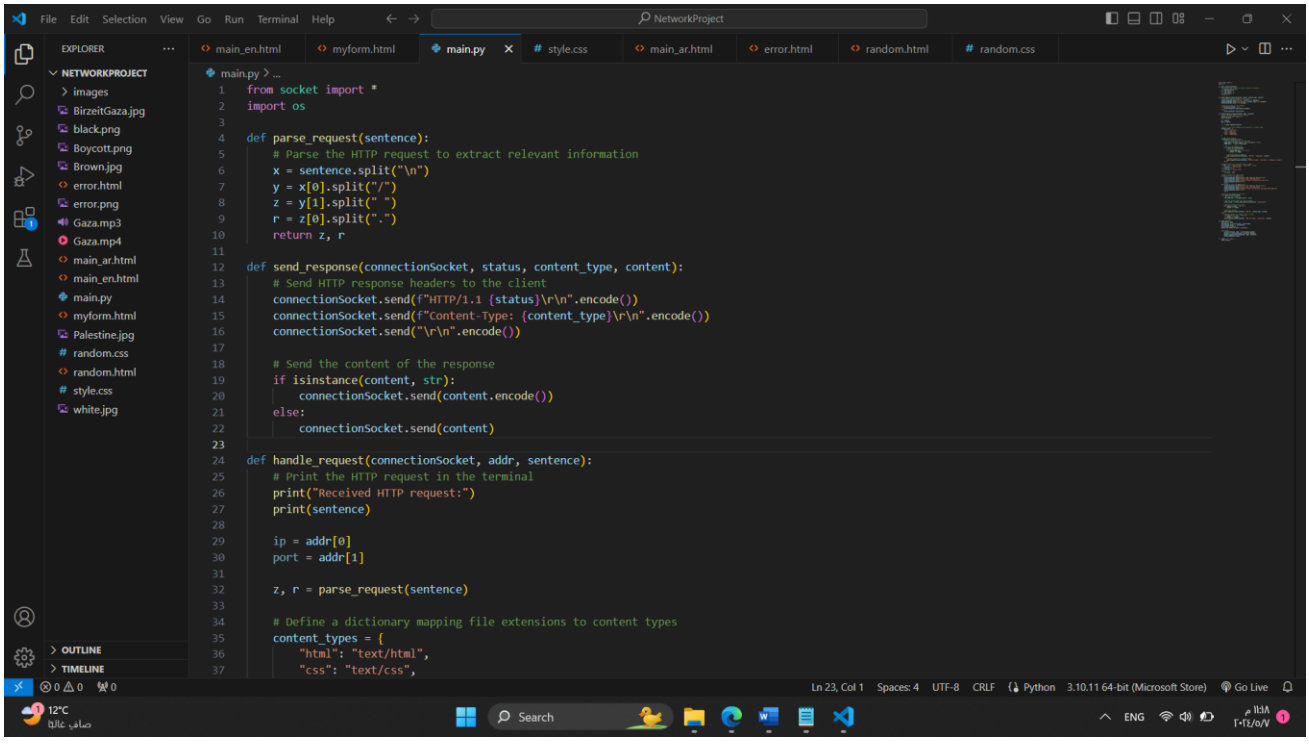
- **Part 5: Redirection Handling**

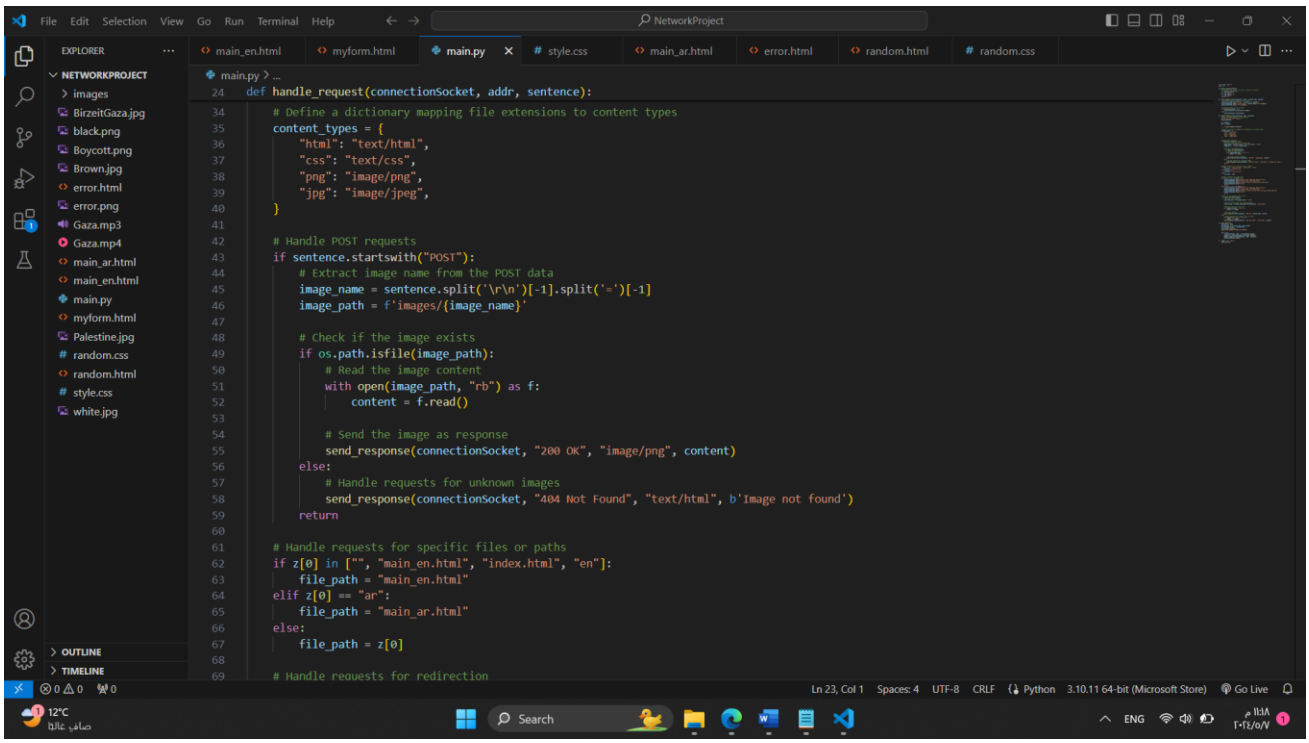
  This section deals with **redirecting clients to different URLs**. If the requested path matches predefined redirection paths (e.g., **/so for Stack Overflow**), the server issues a temporary redirect response (status code 307) to direct the client to the specified URL.

- **Part 6: Error Handling**

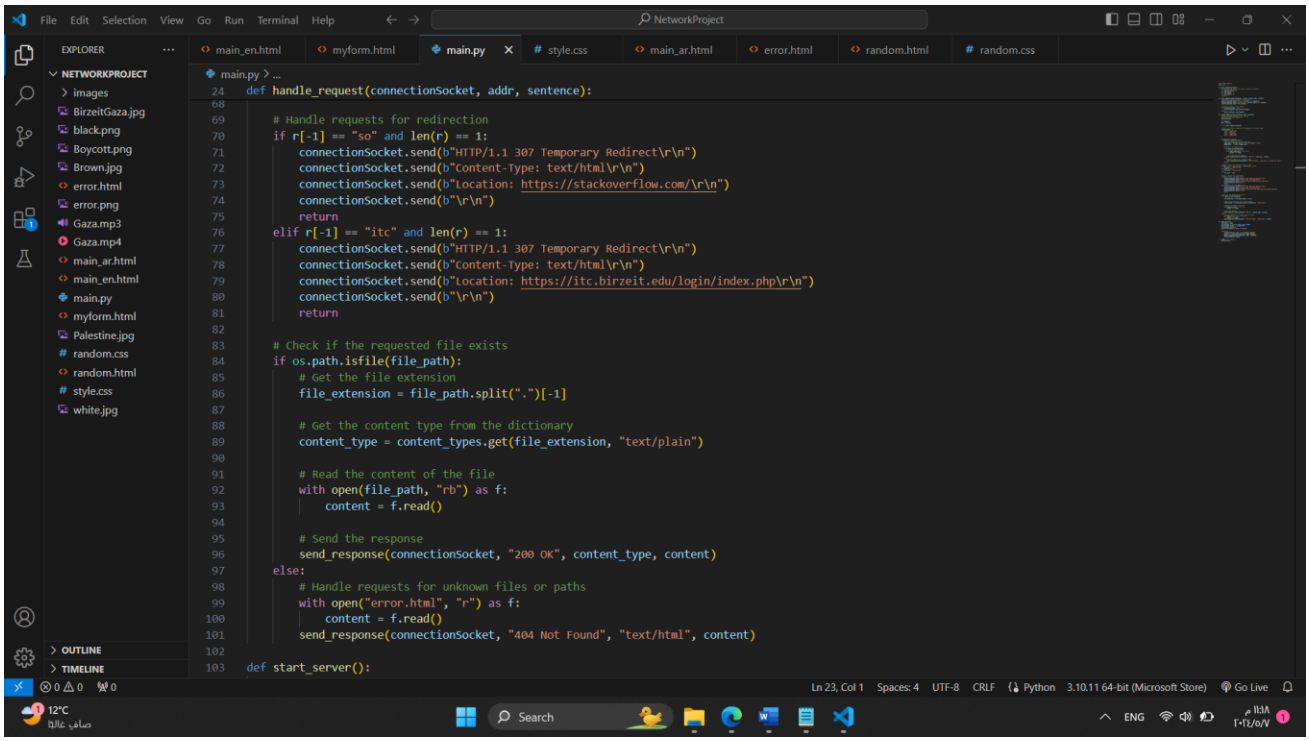
  Error handling is crucial for **informing clients when requested resources are unavailable**. If a requested file or path does not exist, the server generates a 404 Not Found error response. It then serves a custom error page to notify the client that the requested resource could not be located.

- **Python Code**





```python
def handle_request(connectionSocket, addr, sentence):
    # Define a dictionary mapping file extensions to content types
    content_types = {
        "html": "text/html",
        "css": "text/css",
        "png": "image/png",
        "jpg": "image/jpeg",
    }

    # Handle POST requests
    if sentence.startswith("POST"):
        # Extract image name from the POST data
        image_name = sentence.split('\r\n')[-1].split('=')[-1]
        image_path = f'images/{image_name}'

        # Check if the image exists
        if os.path.isfile(image_path):
            # Read the image content
            with open(image_path, "rb") as f:
                content = f.read()

            # Send the image as response
            send_response(connectionSocket, "200 OK", "image/png", content)
        else:
            # Handle requests for unknown images
            send_response(connectionSocket, "404 Not Found", "text/html", b'Image not found')
        return

    # Handle requests for specific files or paths
    if z[0] in ["", "main_en.html", "index.html", "en"]:
        file_path = "main_en.html"
    elif z[0] == "ar":
        file_path = "main_ar.html"
    else:
        file_path = z[0]

    # Handle requests for redirection
```



```python
def handle_request(connectionSocket, addr, sentence):

    # Handle requests for redirection
    if r[-1] == "so" and len(r) == 1:
        connectionSocket.send(b"HTTP/1.1 307 Temporary Redirect\r\n")
        connectionSocket.send(b"Content-Type: text/html\r\n")
        connectionSocket.send(b"Location: https://stackoverflow.com/\r\n")
        connectionSocket.send(b"\r\n")
        return
    elif r[-1] == "itc" and len(r) == 1:
        connectionSocket.send(b"HTTP/1.1 307 Temporary Redirect\r\n")
        connectionSocket.send(b"Content-Type: text/html\r\n")
        connectionSocket.send(b"Location: https://itc.birzeit.edu/login/index.php\r\n")
        connectionSocket.send(b"\r\n")
        return

    # Check if the requested file exists
    if os.path.isfile(file_path):
        # Get the file extension
        file_extension = file_path.split(".")[-1]

        # Get the content type from the dictionary
        content_type = content_types.get(file_extension, "text/plain")

        # Read the content of the file
        with open(file_path, "rb") as f:
            content = f.read()

        # Send the response
        send_response(connectionSocket, "200 OK", content_type, content)
    else:
        # Handle requests for unknown files or paths
        with open("error.html", "r") as f:
            content = f.read()
        send_response(connectionSocket, "404 Not Found", "text/html", content)

def start_server():
```

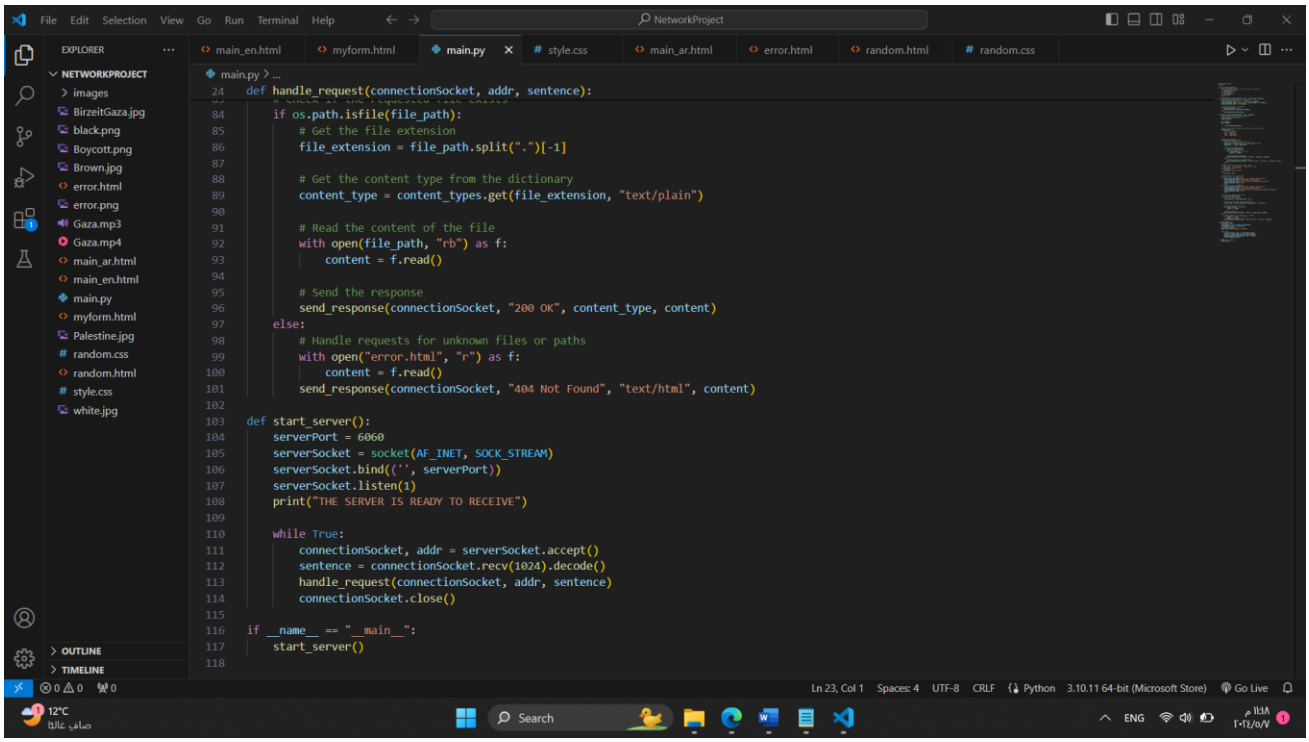
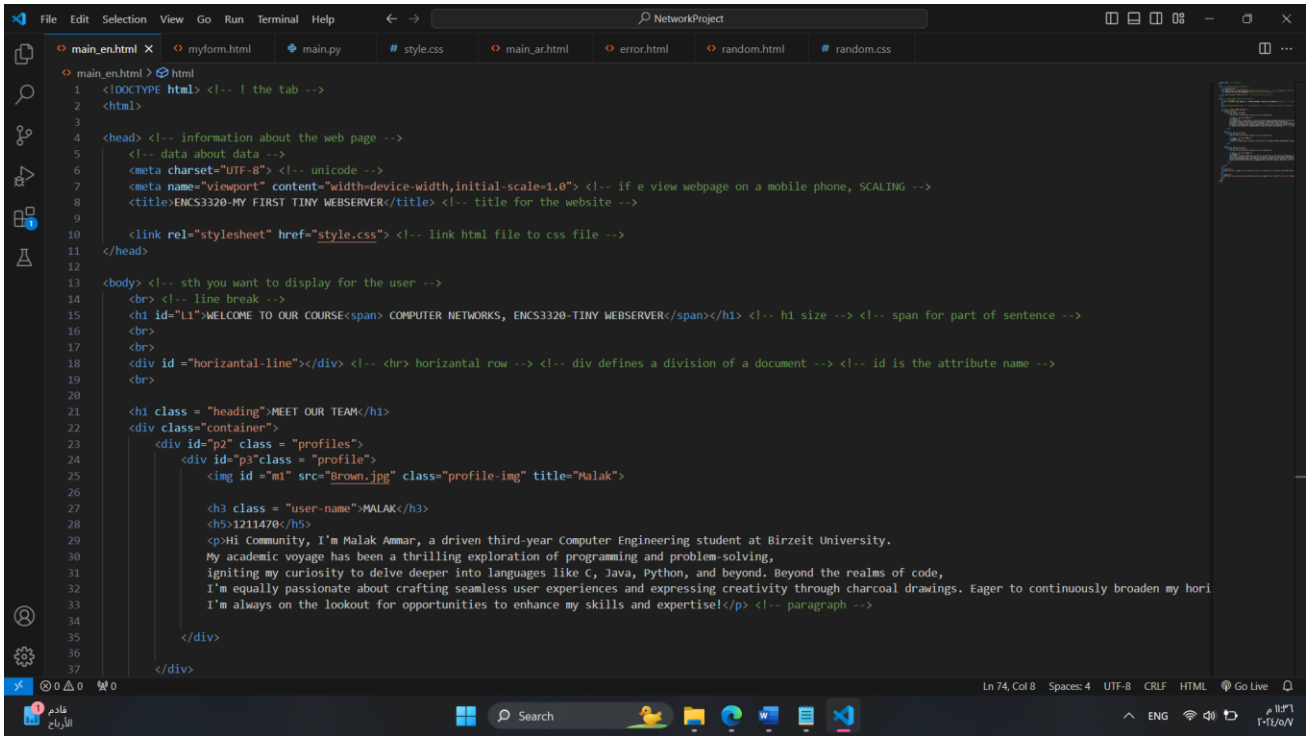


Figure 21: Python Code

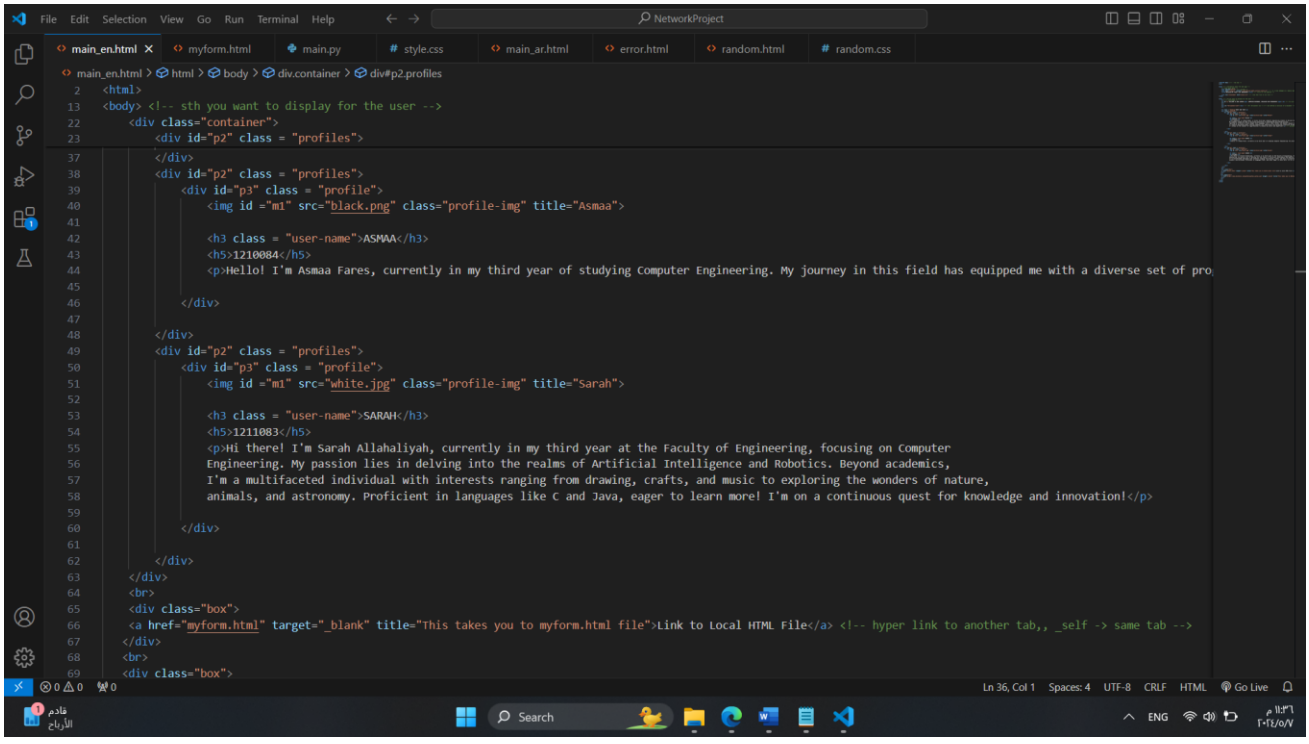
➢ main_en.html: Design and Functionality of main_en.html

The design and content of the **main_en.html** file play a crucial role in **providing a welcoming and informative experience for users accessing the web server**. When users access the root URL or specific paths such as **`/`, `/index.html`, `/main_en.html`, or `/en`** (e.g., **`localhost:6060/` or `localhost:6060/en`),** the **server responds by serving the `main_en.html` file**, ensuring a consistent entry point to the web server's main webpage. This HTML document, titled "ENCS3320-My First Tiny Webserver," introduces users to the course "Computer Networks, ENCS3320-tiny webserver" with a prominent heading. The page features details about the team members involved in the project, including their names, student IDs, and personal statements showcasing their skills and interests. Divided into distinct sections, the webpage **employs CSS styling** for enhanced visual appeal and organization. It incorporates **images with extensions `.jpg` and `.png`**, **links to a local HTML file (`myform.html`)**, and an external resource, enriching the user experience with additional content and interactivity. Furthermore, the server specifies the `Content-Type` as **`text/html`** for proper rendering in client browsers, ensuring seamless presentation and navigation.

- **main_en.html Code**



```
<!DOCTYPE html> <!-- ! the tab -->
<html>

<head> <!-- information about the web page -->
    <!-- data about data -->
    <meta charset="UTF-8"> <!-- unicode -->
    <meta name="viewport" content="width=device-width,initial-scale=1.0"> <!-- if e view webpage on a mobile phone, SCALING -->
    <title>ENCS3320-MY FIRST TINY WEBSERVER</title> <!-- title for the website -->

    <link rel="stylesheet" href="style.css"> <!-- link html file to css file -->
</head>

<body> <!-- sth you want to display for the user -->
    <br> <!-- line break -->
    <h1 id="L1">WELCOME TO OUR COURSE<span> COMPUTER NETWORKS, ENCS3320-TINY WEBSERVER</span></h1> <!-- h1 size --> <!-- span for part of sentence -->
    <br>
    <br>
    <div id ="horizantal-line"></div> <!-- <hr> horizantal row --> <!-- div defines a division of a document --> <!-- id is the attribute name -->
    <br>

    <h1 class = "heading">MEET OUR TEAM</h1>
    <div class="container">
        <div id="p2" class = "profiles">
            <div id="p3"class = "profile">
                <img id ="m1" src="Brown.jpg" class="profile-img" title="Malak">

                <h3 class = "user-name">MALAK</h3>
                <h5>1211470</h5>
                <p>Hi Community, I'm Malak Ammar, a driven third-year Computer Engineering student at Birzeit University.
                My academic voyage has been a thrilling exploration of programming and problem-solving,
                igniting my curiosity to delve deeper into languages like C, Java, Python, and beyond. Beyond the realms of code,
                I'm equally passionate about crafting seamless user experiences and expressing creativity through charcoal drawings. Eager to continuously broaden my hori
                I'm always on the lookout for opportunities to enhance my skills and expertise!</p> <!-- paragraph -->

            </div>

        </div>
```



```
        </div>
        <div id="p2" class = "profiles">
            <div id="p3" class = "profile">
                <img id ="m1" src="black.png" class="profile-img" title="Asmaa">

                <h3 class = "user-name">ASMAA</h3>
                <h5>1210084</h5>
                <p>Hello! I'm Asmaa Fares, currently in my third year of studying Computer Engineering. My journey in this field has equipped me with a diverse set of pro

            </div>

        </div>
        <div id="p2" class = "profiles">
            <div id="p3" class = "profile">
                <img id ="m1" src="white.jpg" class="profile-img" title="Sarah">

                <h3 class = "user-name">SARAH</h3>
                <h5>1211083</h5>
                <p>Hi there! I'm Sarah Allahaliyah, currently in my third year at the Faculty of Engineering, focusing on Computer
                Engineering. My passion lies in delving into the realms of Artificial Intelligence and Robotics. Beyond academics,
                I'm a multifaceted individual with interests ranging from drawing, crafts, and music to exploring the wonders of nature,
                animals, and astronomy. Proficient in languages like C and Java, eager to learn more! I'm on a continuous quest for knowledge and innovation!</p>

            </div>

        </div>
    </div>
    <br>
    <div class="box">
    <a href="myform.html" target="_blank" title="This takes you to myform.html file">Link to Local HTML File</a> <!-- hyper link to another tab,, _self -> same tab -->
    </div>
    <br>
    <div class="box">
```

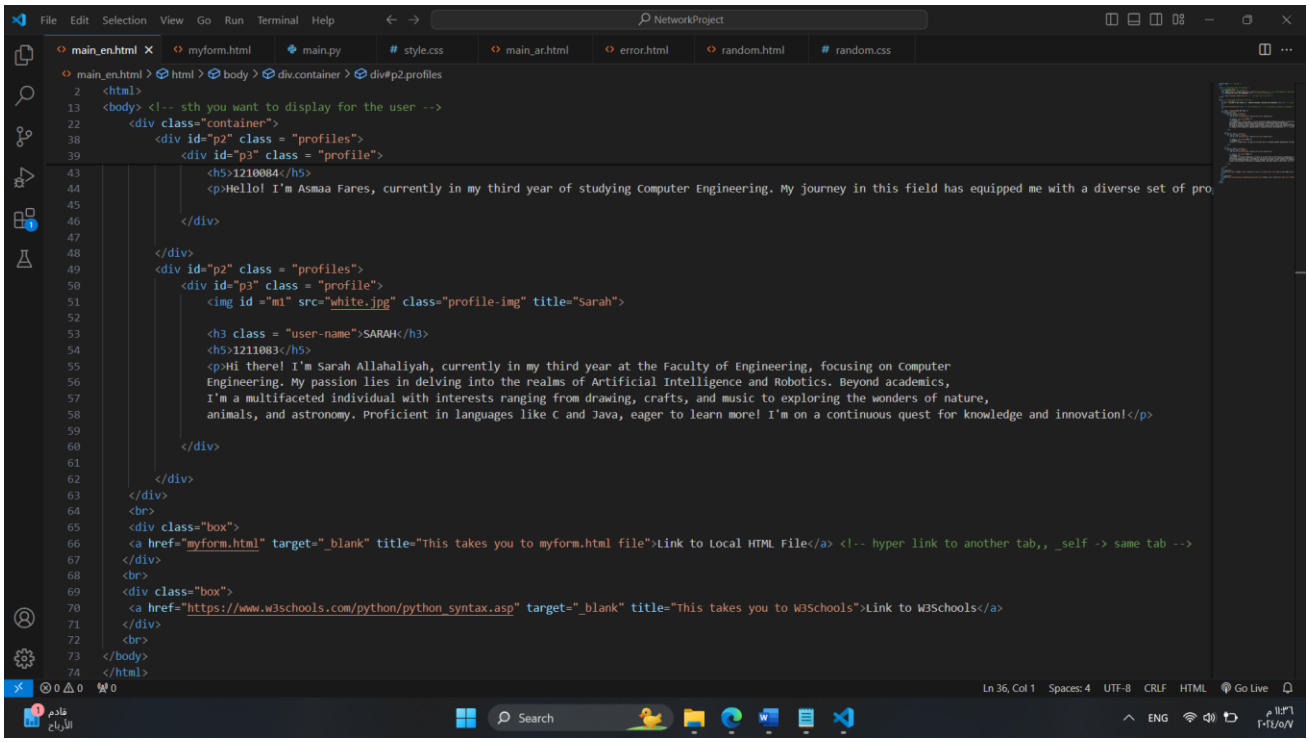
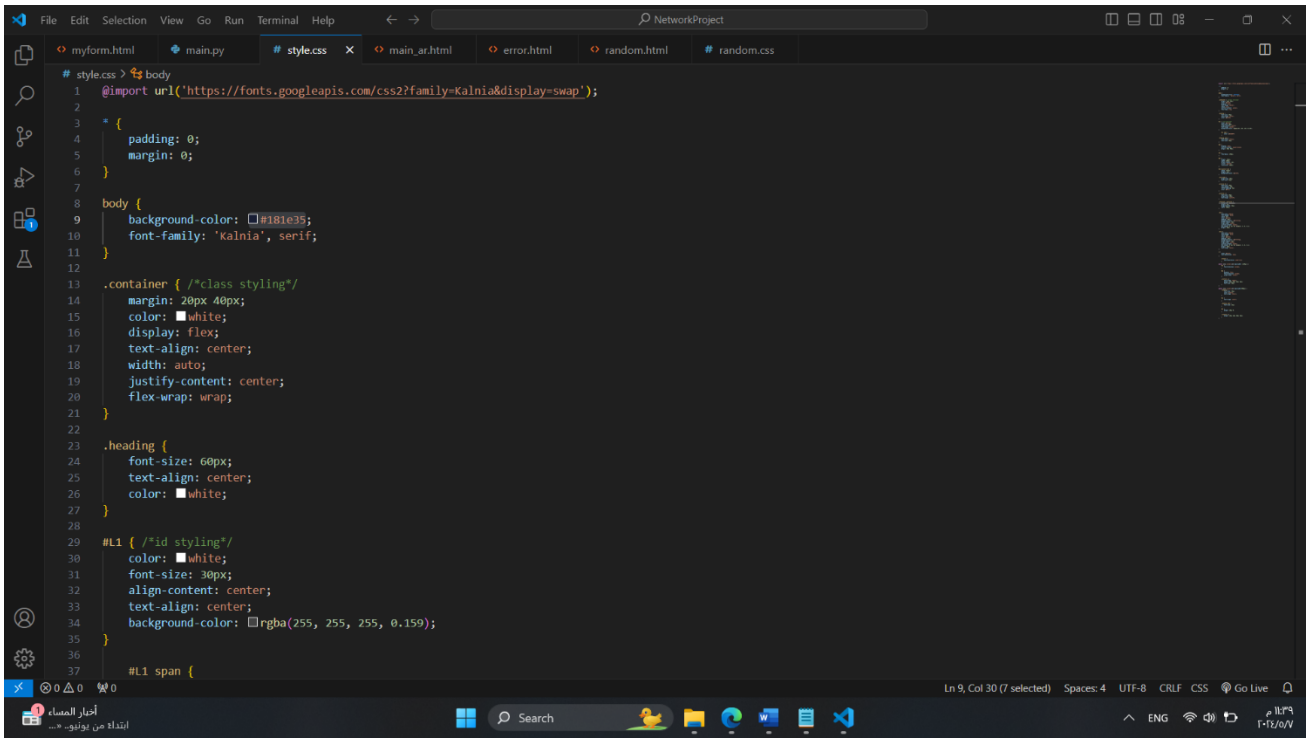
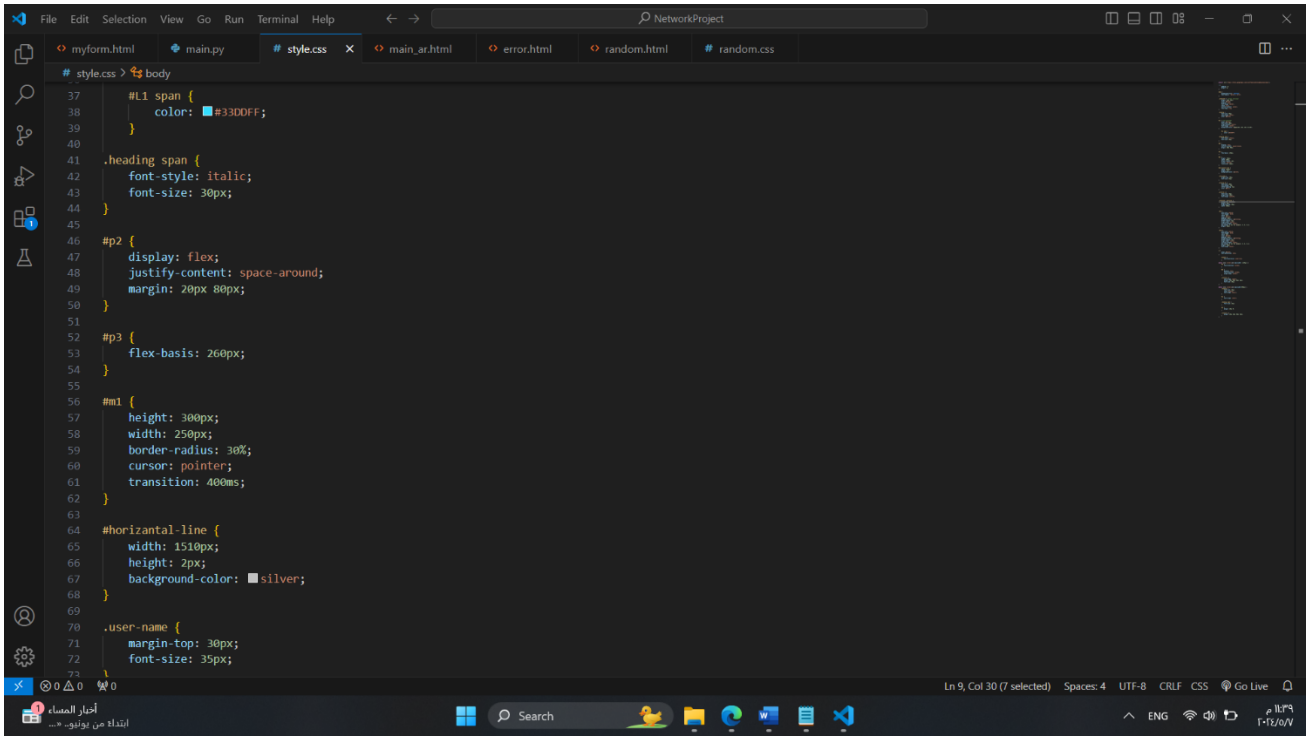


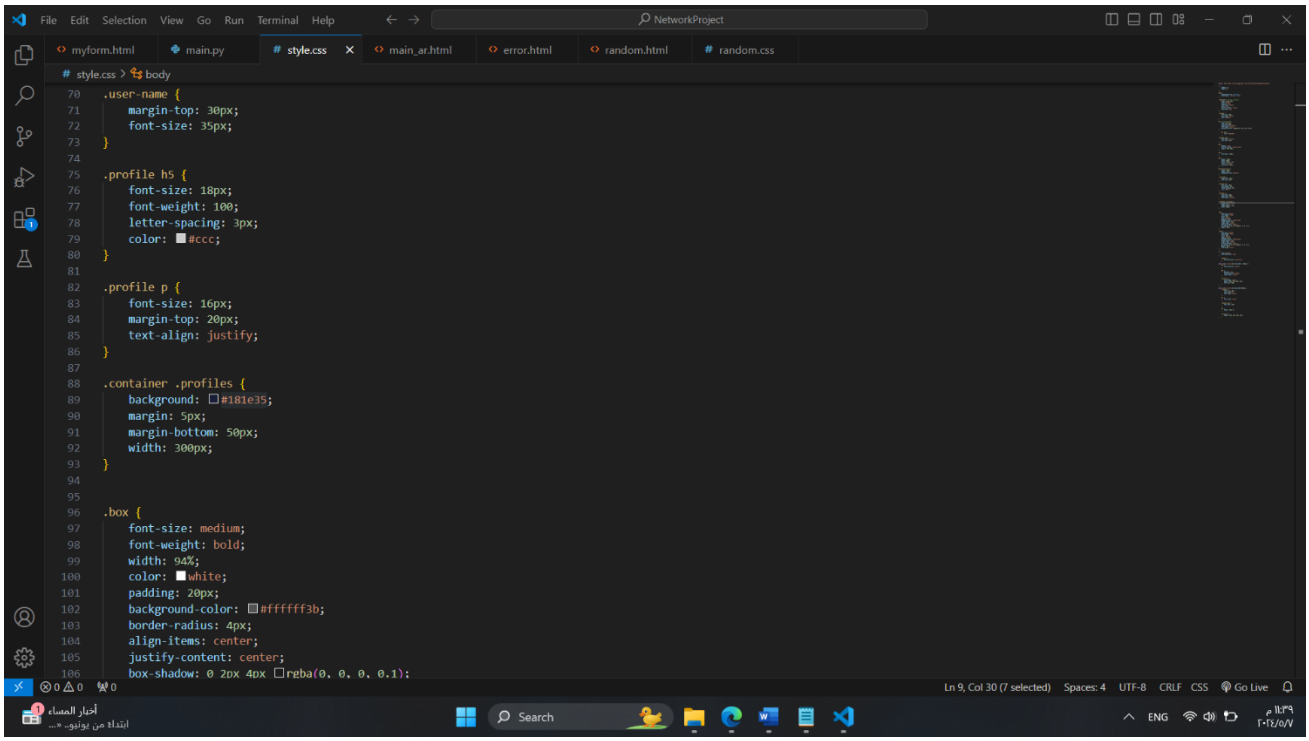
Figure 22: main_en.html Code

- **style.css Code**

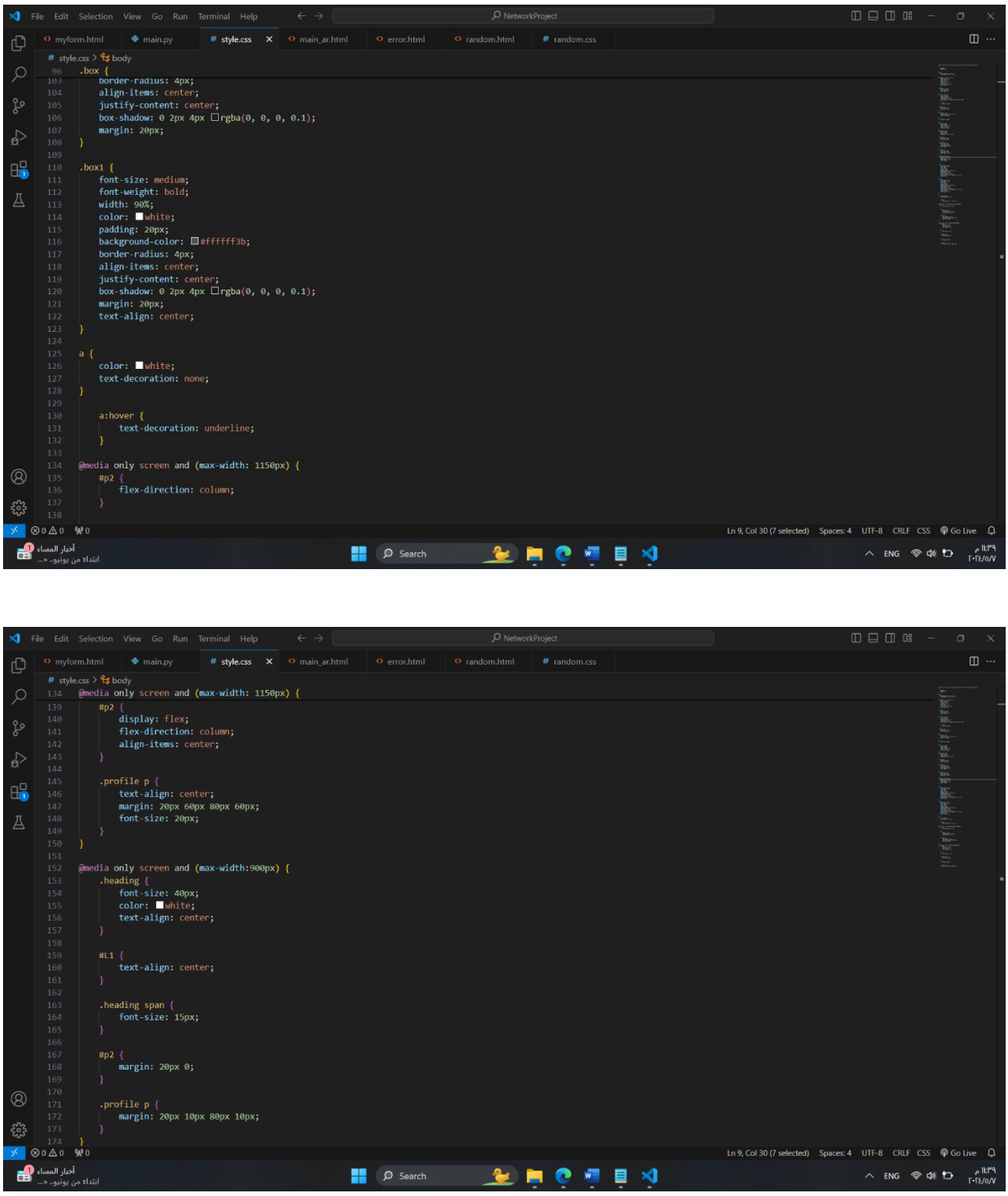




```
#L1 span {
    color: #33DDFF;
}

.heading span {
    font-style: italic;
    font-size: 30px;
}

#p2 {
    display: flex;
    justify-content: space-around;
    margin: 20px 80px;
}

#p3 {
    flex-basis: 260px;
}

#m1 {
    height: 300px;
    width: 250px;
    border-radius: 30%;
    cursor: pointer;
    transition: 400ms;
}

#horizantal-line {
    width: 1510px;
    height: 2px;
    background-color: silver;
}

.user-name {
    margin-top: 30px;
    font-size: 35px;
}
```



```
.user-name {
    margin-top: 30px;
    font-size: 35px;
}

.profile h5 {
    font-size: 18px;
    font-weight: 100;
    letter-spacing: 3px;
    color: #ccc;
}

.profile p {
    font-size: 16px;
    margin-top: 20px;
    text-align: justify;
}

.container .profiles {
    background: #181e35;
    margin: 5px;
    margin-bottom: 50px;
    width: 300px;
}


.box {
    font-size: medium;
    font-weight: bold;
    width: 94%;
    color: white;
    padding: 20px;
    background-color: #ffffff3b;
    border-radius: 4px;
    align-items: center;
    justify-content: center;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
```



```
.box {
    border-radius: 4px;
    align-items: center;
    justify-content: center;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    margin: 20px;
}

.box1 {
    font-size: medium;
    font-weight: bold;
    width: 90%;
    color: white;
    padding: 20px;
    background-color: #ffffff3b;
    border-radius: 4px;
    align-items: center;
    justify-content: center;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    margin: 20px;
    text-align: center;
}

a {
    color: white;
    text-decoration: none;
}

    a:hover {
        text-decoration: underline;
    }

@media only screen and (max-width: 1150px) {
    #p2 {
        flex-direction: column;
    }
```



```
@media only screen and (max-width: 1150px) {
    #p2 {
        display: flex;
        flex-direction: column;
        align-items: center;
    }

    .profile p {
        text-align: center;
        margin: 20px 60px 80px 60px;
        font-size: 20px;
    }
}

@media only screen and (max-width:900px) {
    .heading {
        font-size: 40px;
        color: white;
        text-align: center;
    }

    #L1 {
        text-align: center;
    }

    .heading span {
        font-size: 15px;
    }

    #p2 {
        margin: 20px 0;
    }

    .profile p {
        margin: 20px 10px 80px 10px;
    }
}
```

Figure 23: style.css Code

- **The client is redirected to our webpage when the HTTP request '/en' is received**

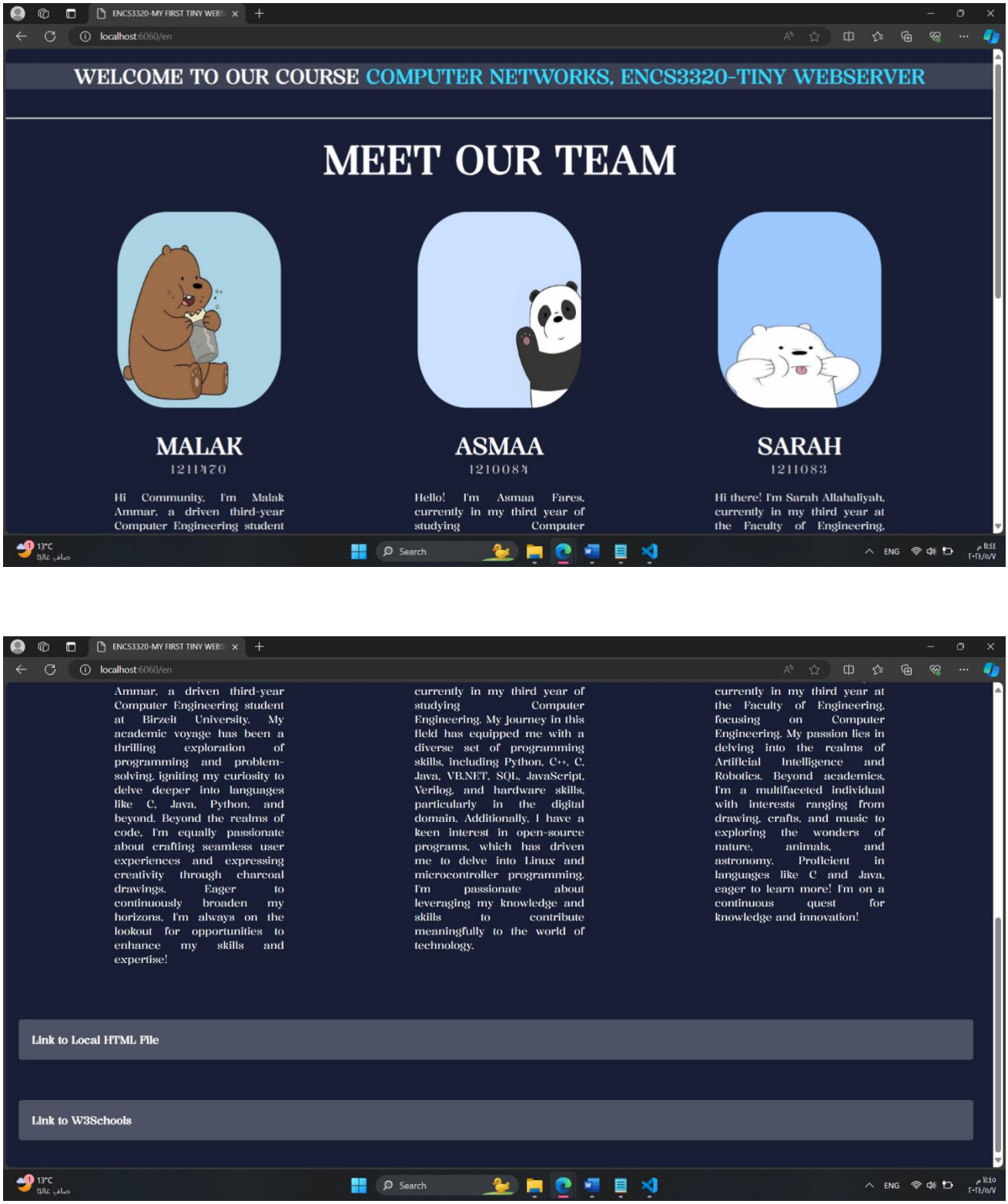


Figure 24: English Webserver with HTTP request '/en'

- **The client is redirected to our webpage when the HTTP request '/main_en.html' is received**

Figure 25: English Webserver with HTTP request '/main_en.html'

- **The client is redirected to our webpage when the HTTP request '/' is received**

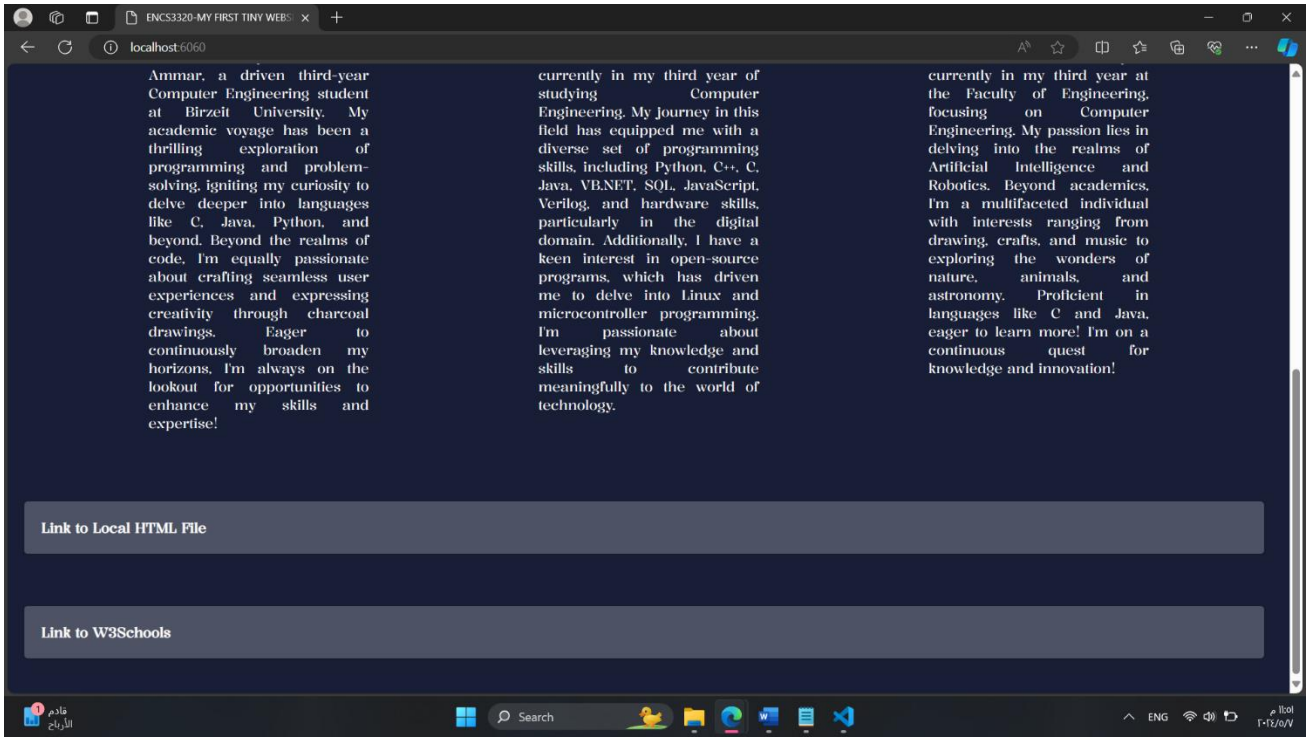
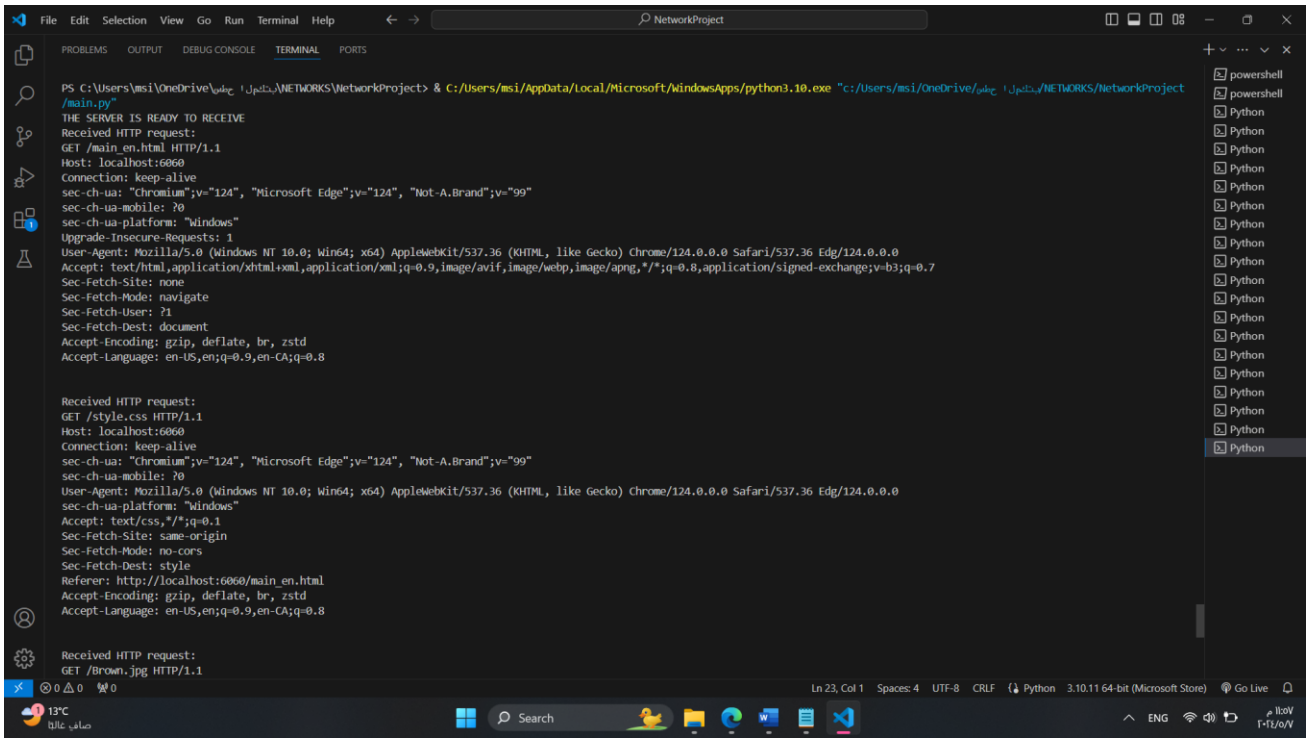


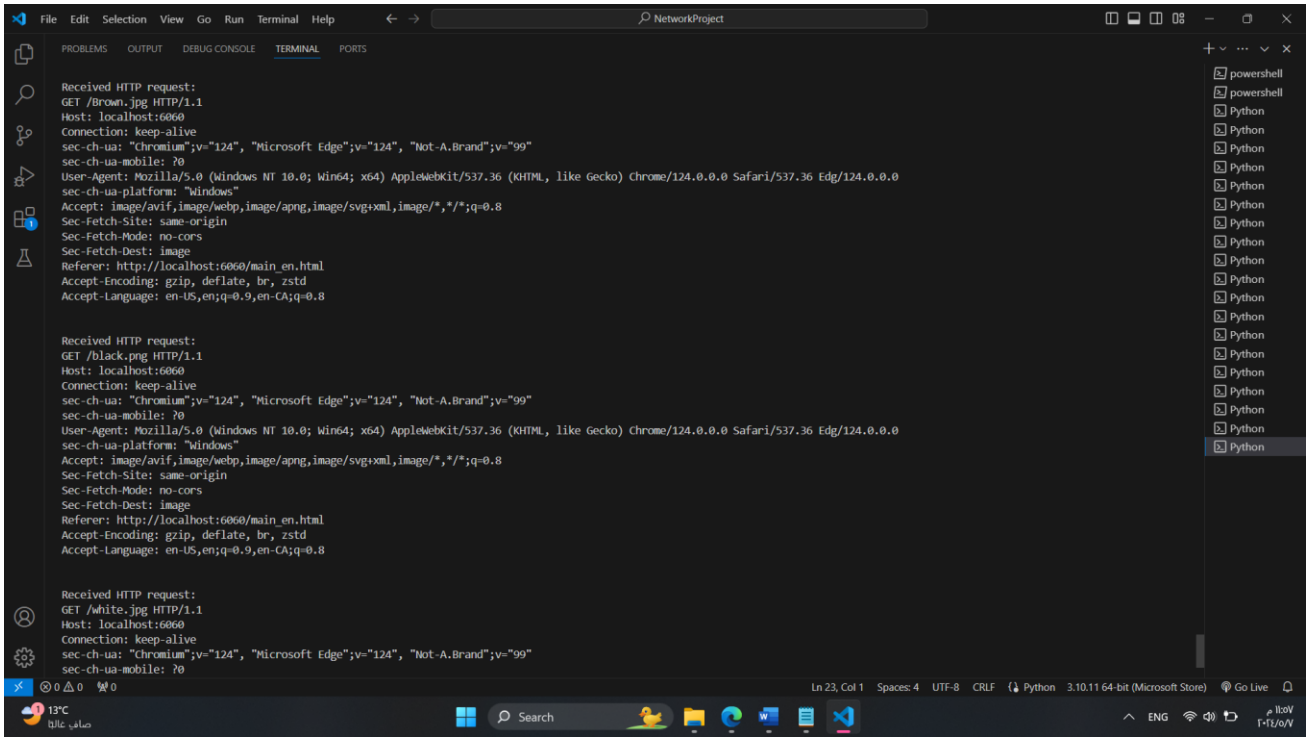
Figure 26: English Webserver with HTTP request '/'

- **The client is redirected to our webpage when the HTTP request '/index.html' is received**

Figure 27: English Webserver with HTTP request '/index.html'

- **HTTP Request for English Webpage**





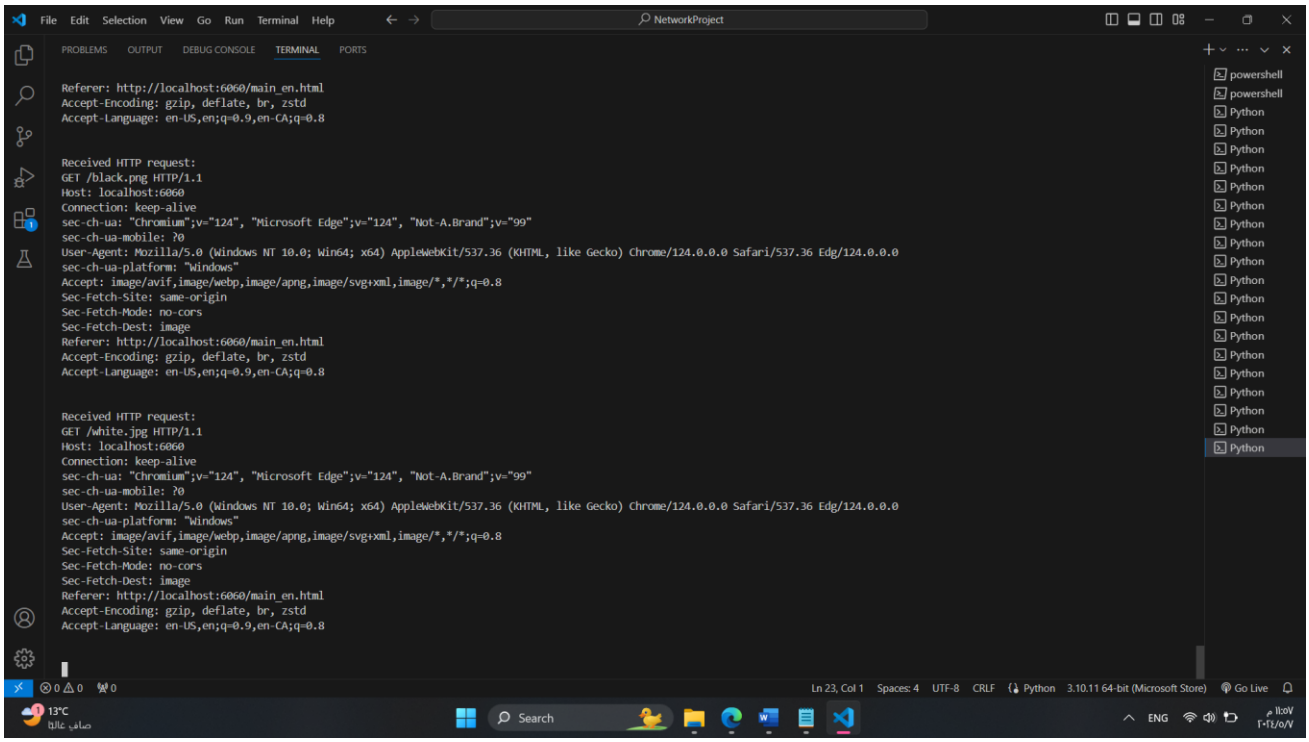


Figure 28: HTTP Request for English Webpage

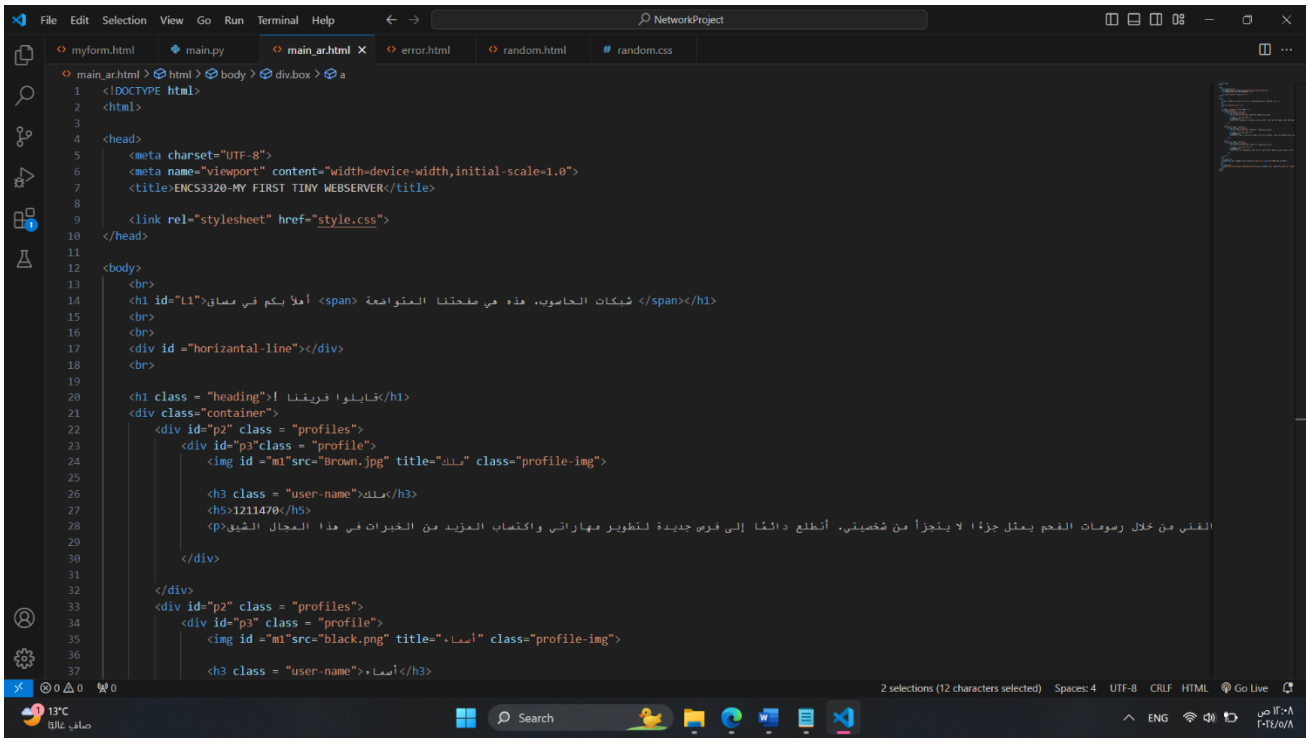
- **Explanation**

The **HTTP request** log demonstrates the **interactions between the client and the server**, specifically focusing on requests related to **main_en.html** and its associated resources. Each request begins with a "**GET**" method followed by the URL of the requested resource and the **HTTP/1.1 protocol** version. The subsequent lines detail various headers providing additional context, such as the user agent, accept headers, and referrer information. Notably, the requests include references to **style.css** and several image files (**Brown.jpg, black.png, and white.jpg**) referenced within the main_en.html file, along with their respective headers indicating the desired content types. These requests are essential for the server to fulfill the client's demands by providing the requested HTML and associated resources, contributing to the seamless rendering of the webpage on the client's browser.

➢ main_ar.html: Design and Functionality of main_ar.html

main_ar.html is structured to provide a user-friendly **Arabic version of the webpage**, utilizing the **style.css** for consistent styling **as in the English HTML**. It incorporates Arabic language elements, adhering to cultural standards, and ensures responsiveness across different devices for an optimal viewing experience.

- **main_ar.html Code**



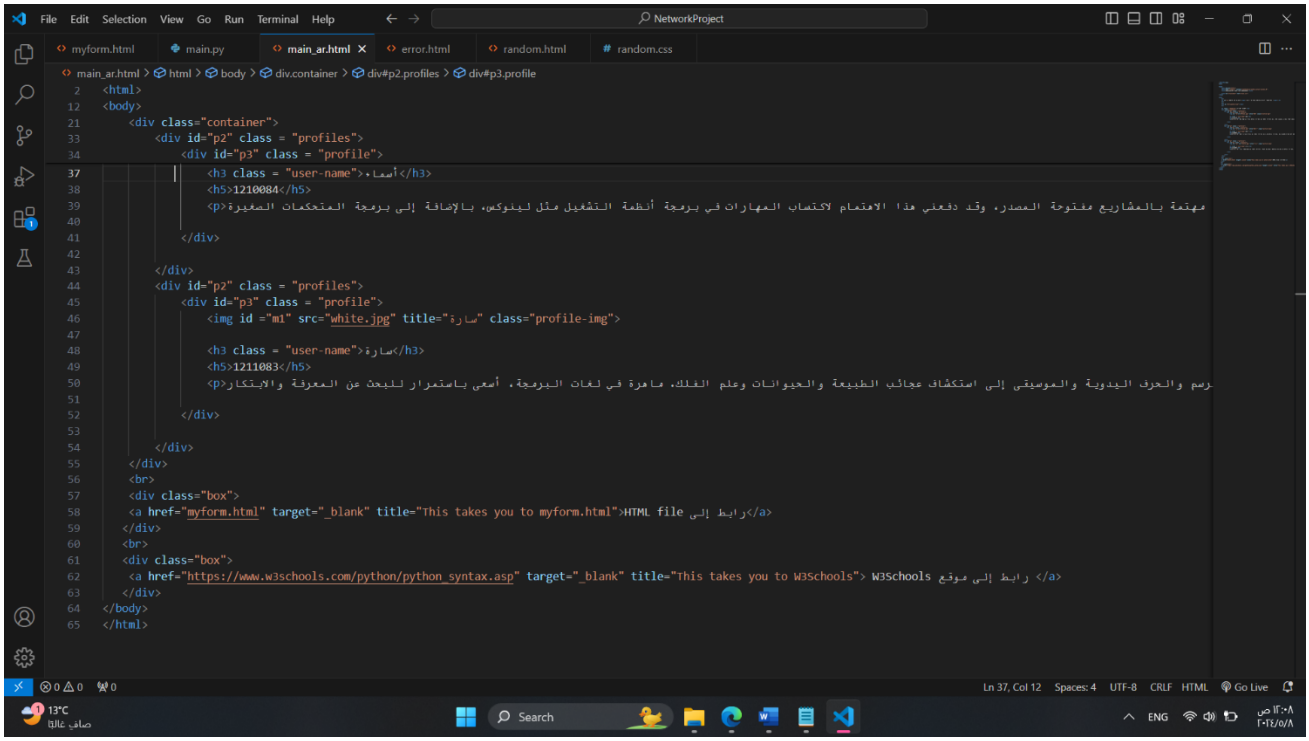


Figure 29: main_ar.html Code

- **The client is redirected to our Arabic webpage when the HTTP request '/main_ar.html' is received**

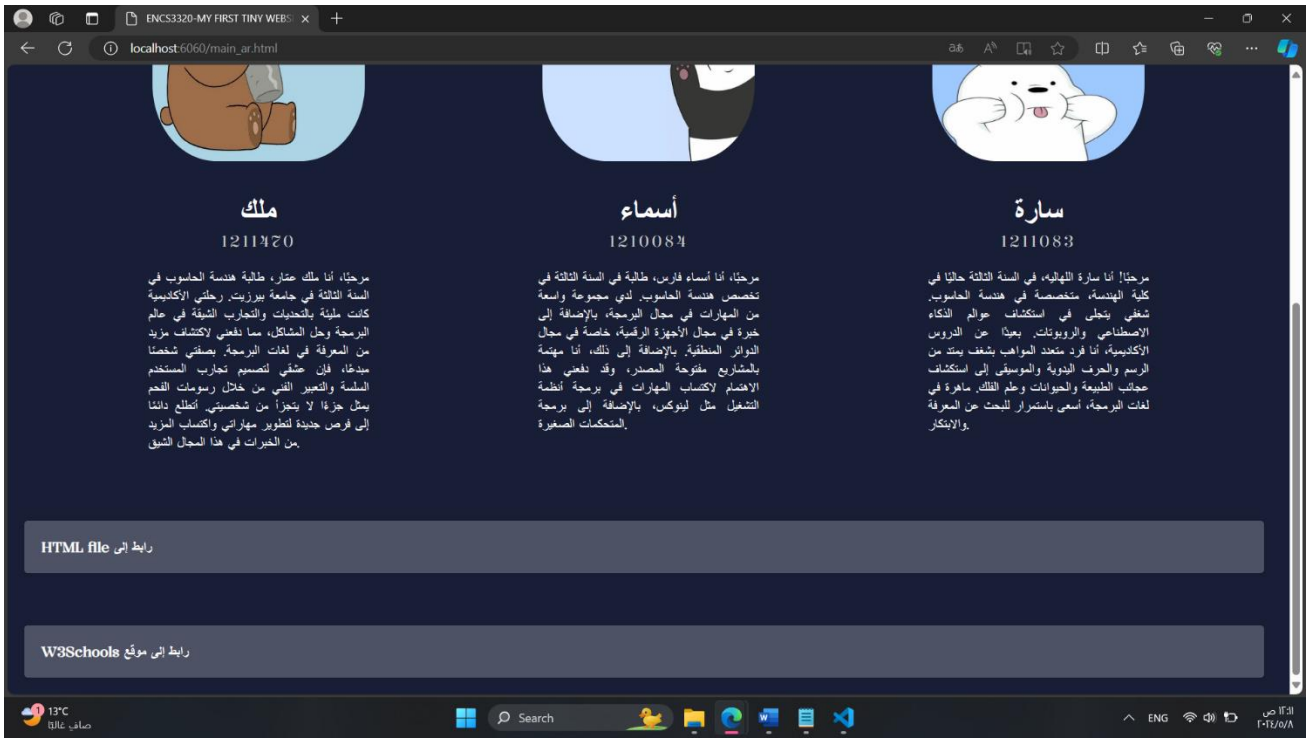


Figure 30: Arabic Webserver with HTTP request '/main_ar.html'

- **HTTP Request for Arabic Webpage**

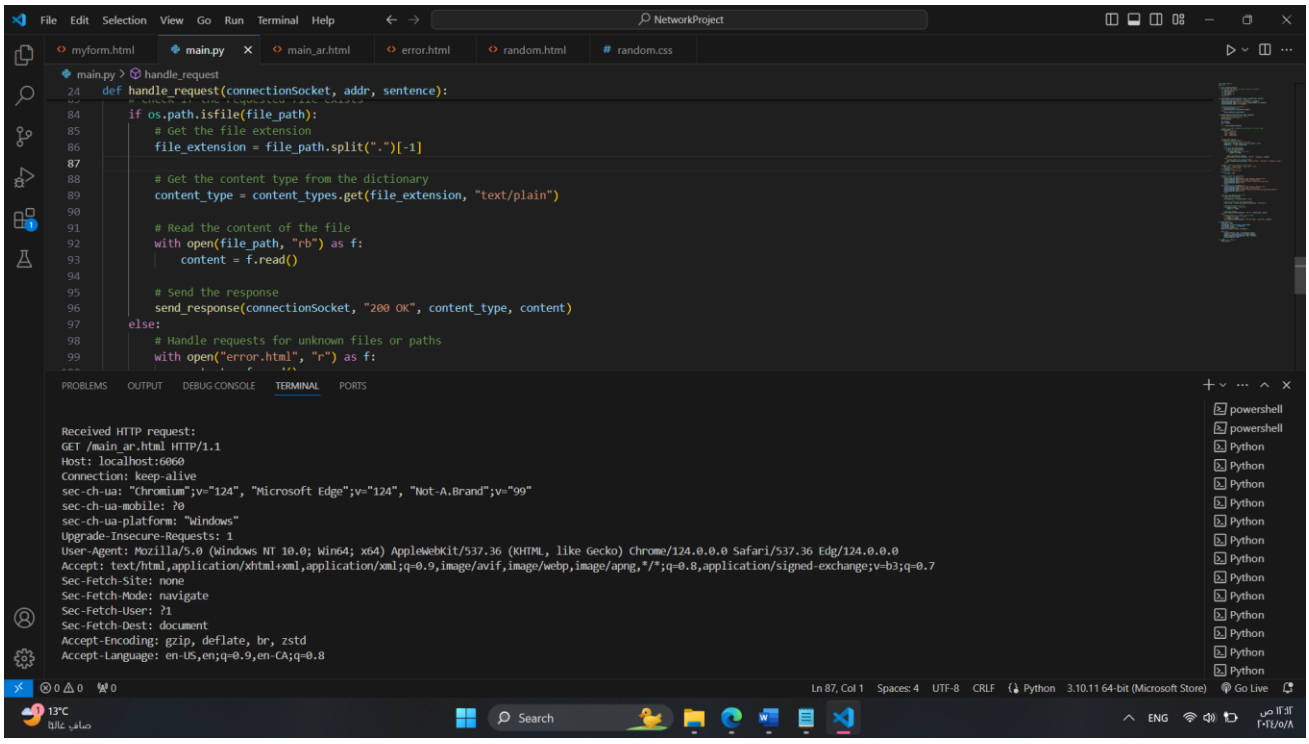


Figure 31: HTTP Request for Arabic Webpage

➢ random.html: Design and Functionality of random.html

random.html file presents a poignant call-to-action in support of Palestine and Gaza. It features **the flag of Palestine** prominently displayed urging visitors to stand in solidarity. A **video titled "HELP GAZA" is auto played**, likely showcasing the ongoing conflict in Gaza to raise awareness. Additionally, a donation appeal is made, encouraging viewers to contribute to the cause through an **embedded GoFundMe link**. The layout and design are carefully crafted to captivate attention and convey the urgency of the situation. FREE PALESTINE!

- **random.html Code**

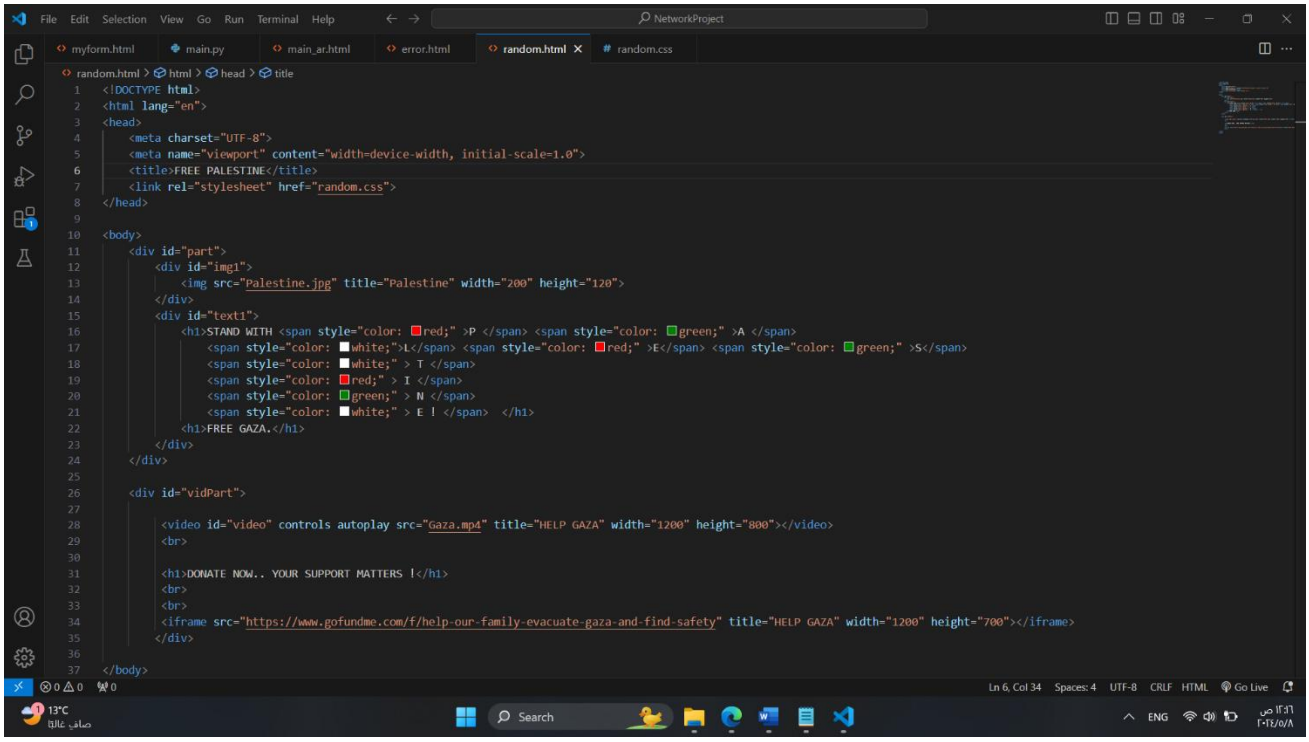


Figure 32: random.html Code

- **random.css Code**

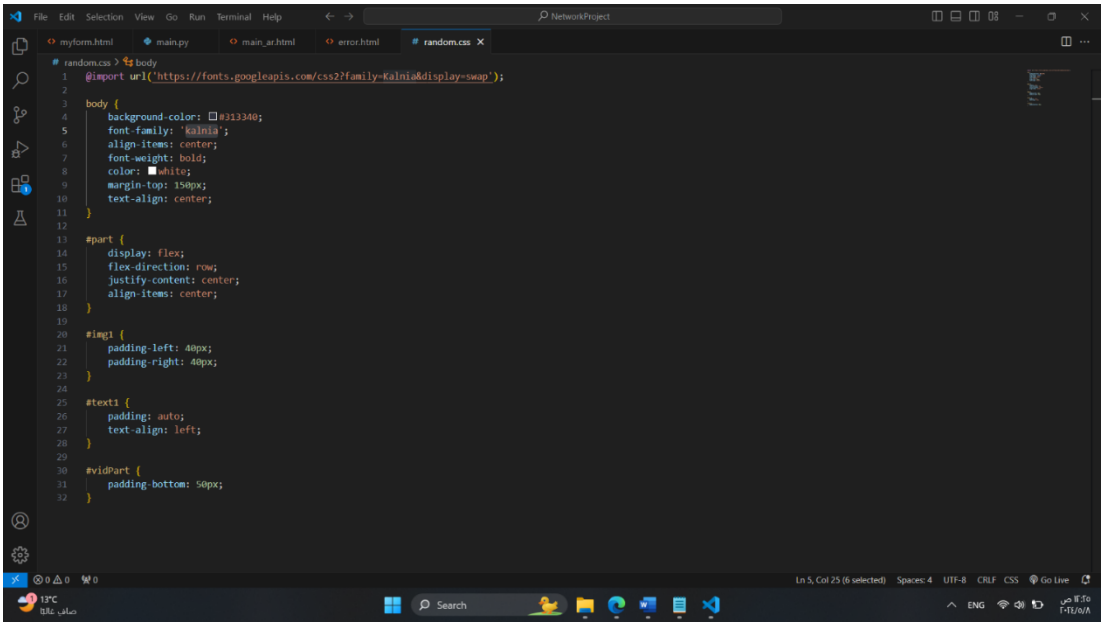
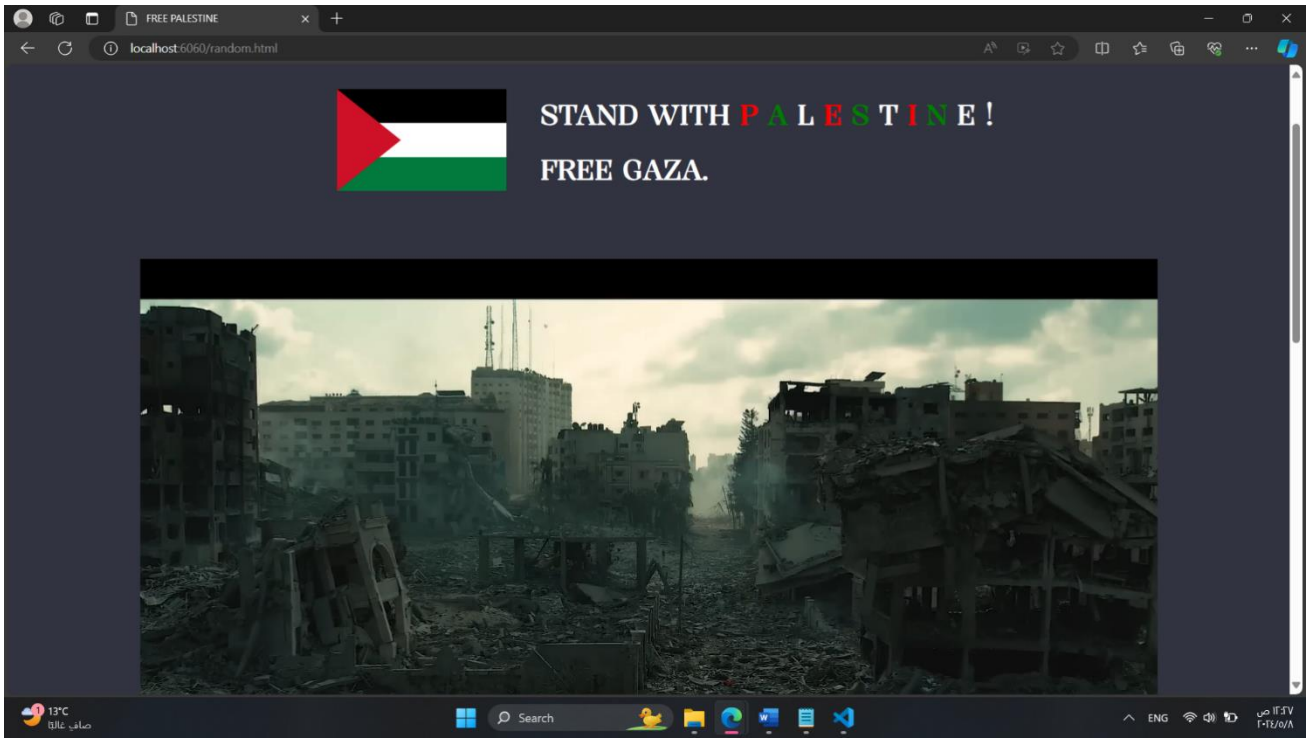


Figure 33: random.css Code

- **The client is redirected to our random webpage when the HTTP request '/random.html' is received**



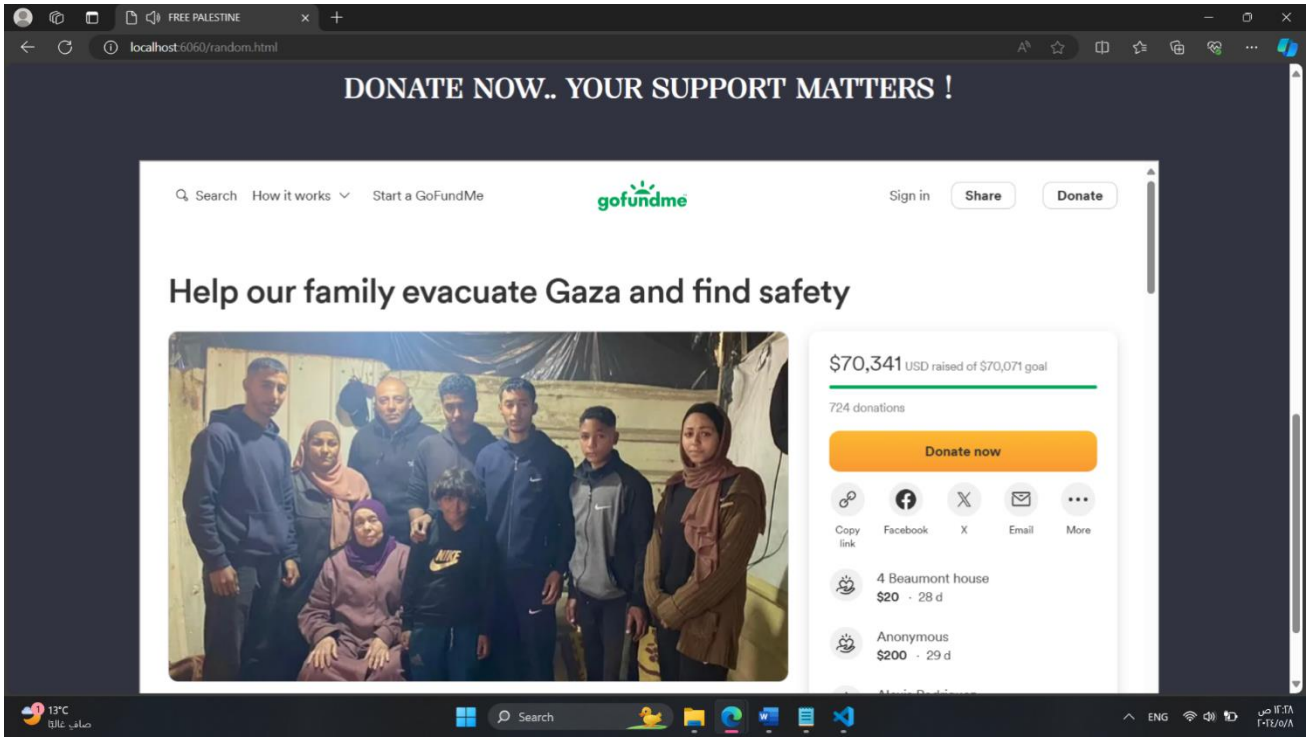
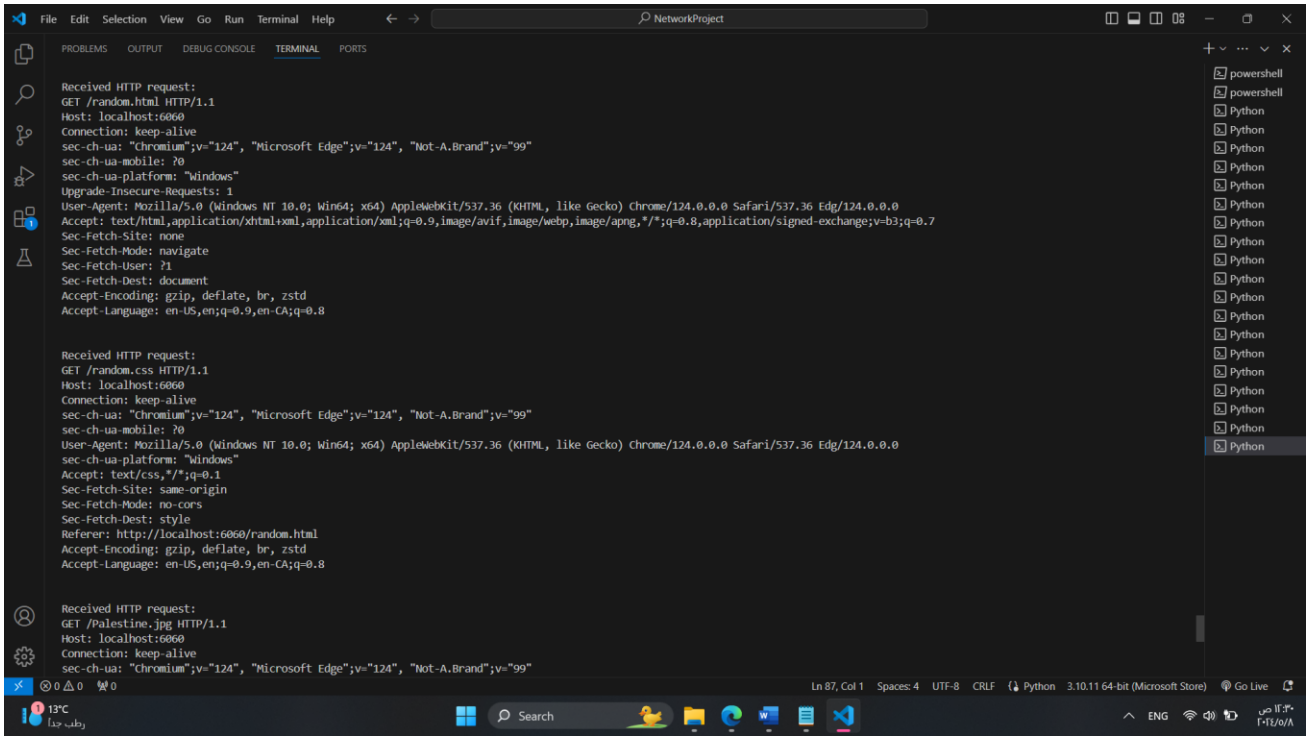


Figure 34: random webpage when the HTTP request '/random.html'

- **HTTP Request for random Webpage**



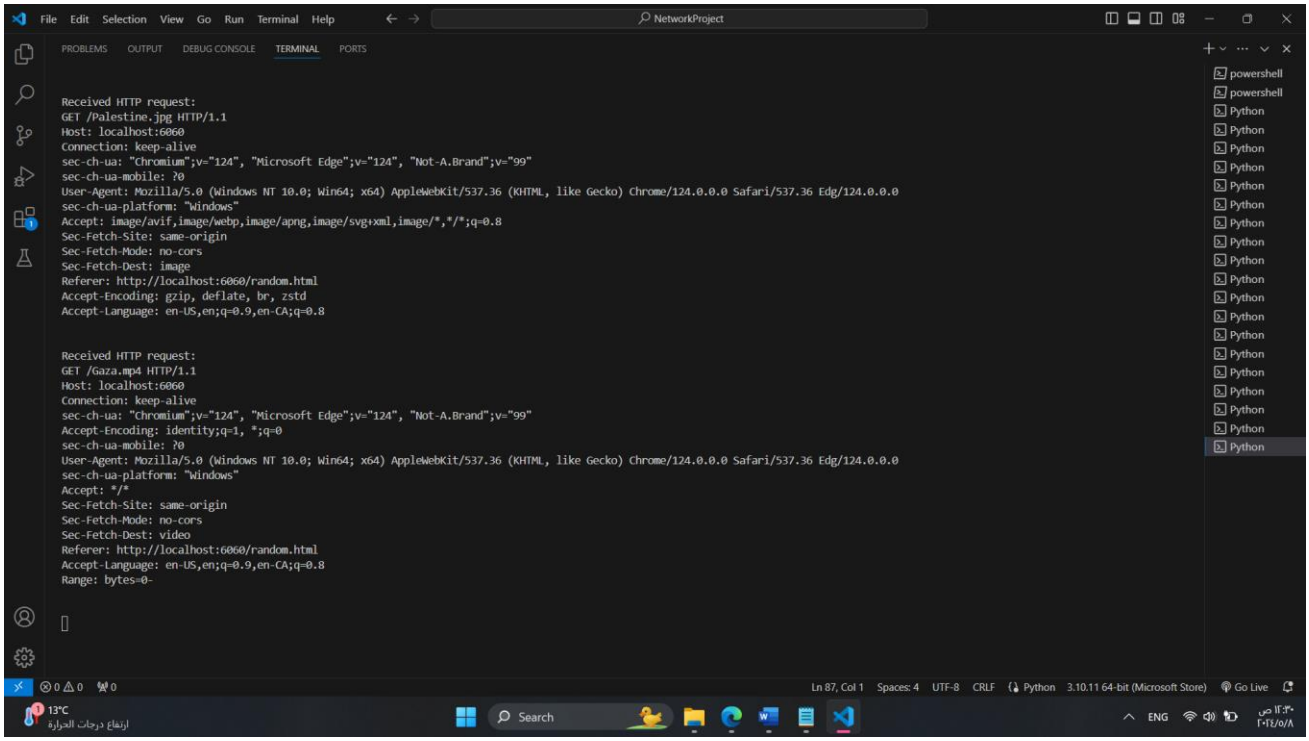


Figure 35: HTTP Request for random Webpage

➢ random.css: Design and Functionality of random.css

- **The client is redirected to our random CSS code when the HTTP request '/random.css' is received**

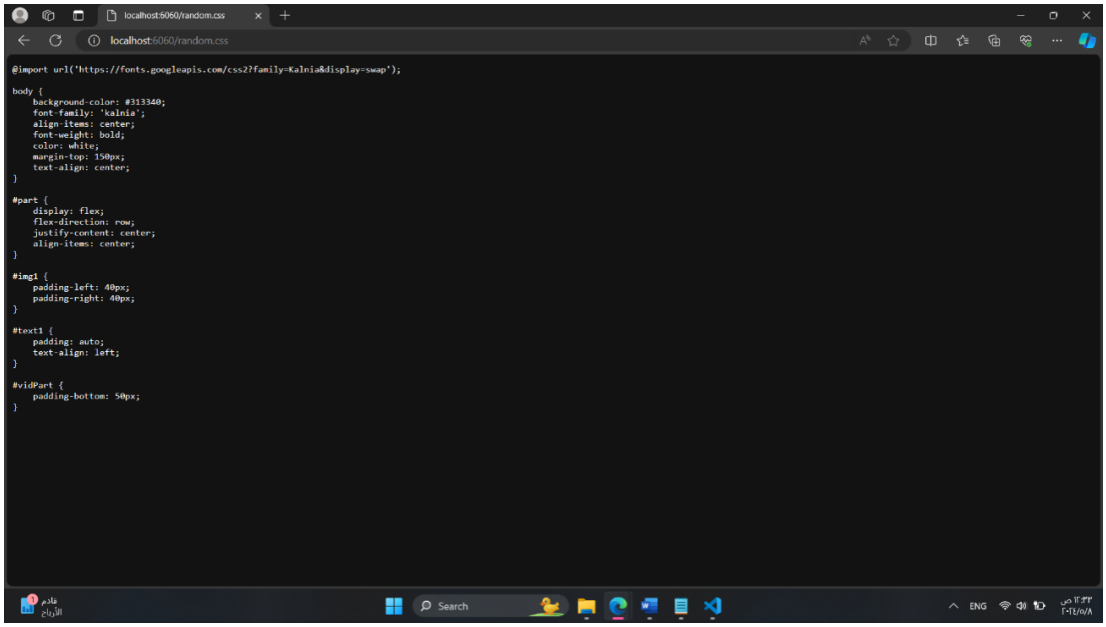


Figure 36: random CSS code

- **HTTP Request for random CSS**

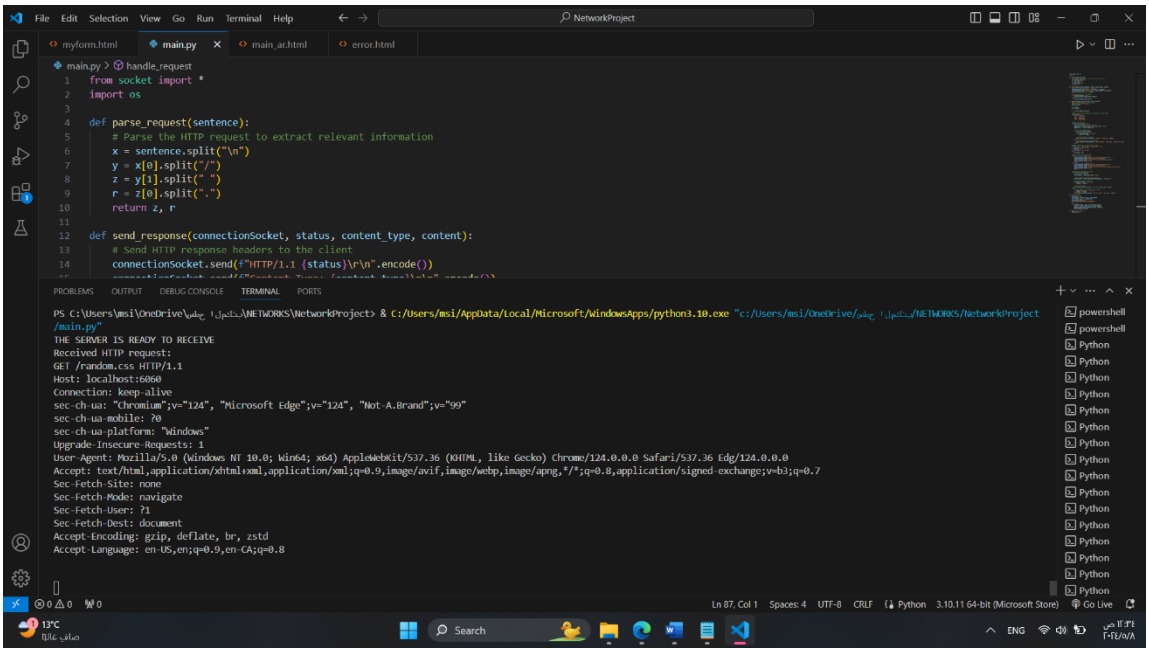


Figure 37: HTTP Request for random CSS

➢ BirzeitGaza.jpg: Design and Functionality of BirzeitGaza.jpg

- **The client is redirected to our JPG image when the HTTP request '/BirzeitGaza.jpg' is received**

Figure 38: JPG image with HTTP request '/BirzeitGaza.jpg'

- **HTTP Request for BirzeitGaza.jpg**

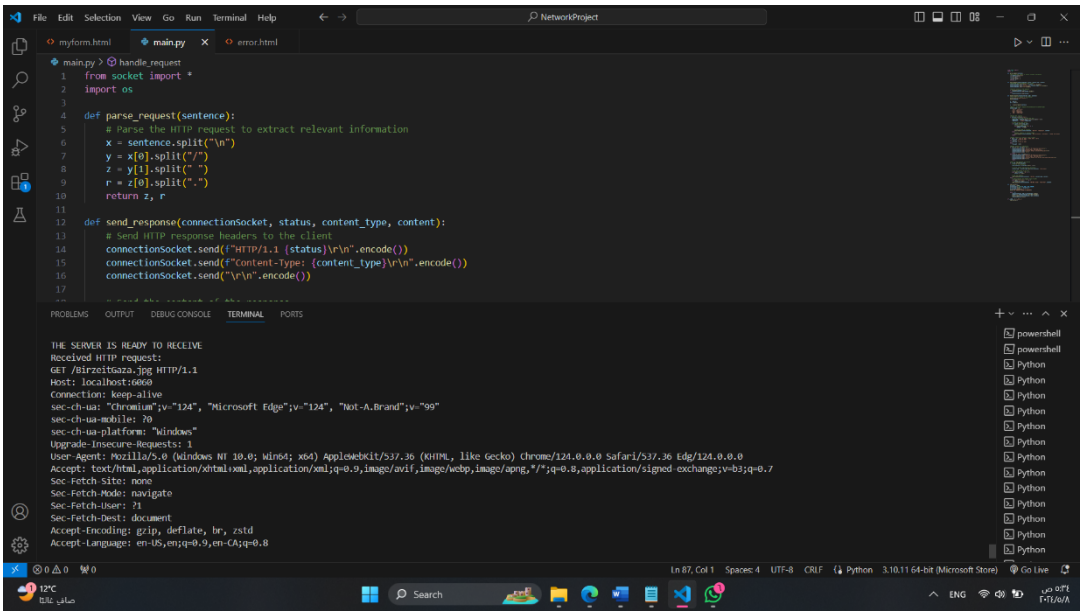


Figure 39: HTTP request '/BirzeitGaza.jpg'

➢ Boycott.png: Design and Functionality of BirzeitGaza.jpg

- **The client is redirected to our PNG image when the HTTP request '/Boycott.png' is received**

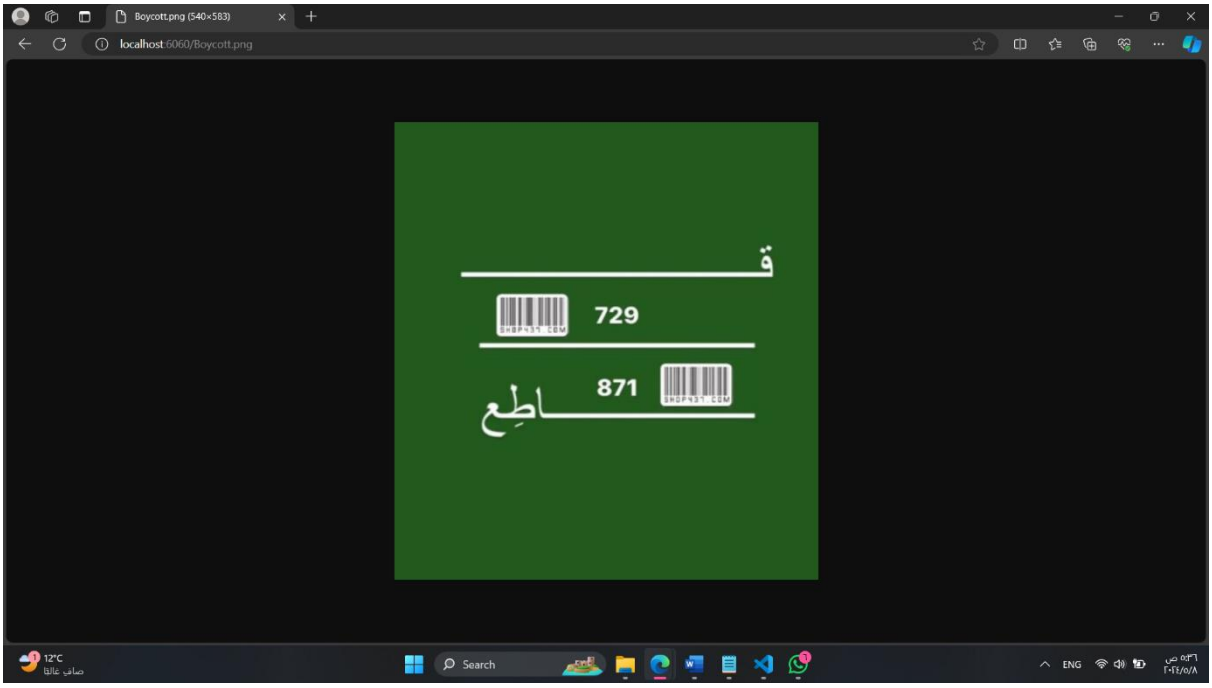


Figure 40: PNG image when the HTTP request '/Boycott.png'

- **HTTP Request for Boycott.png**

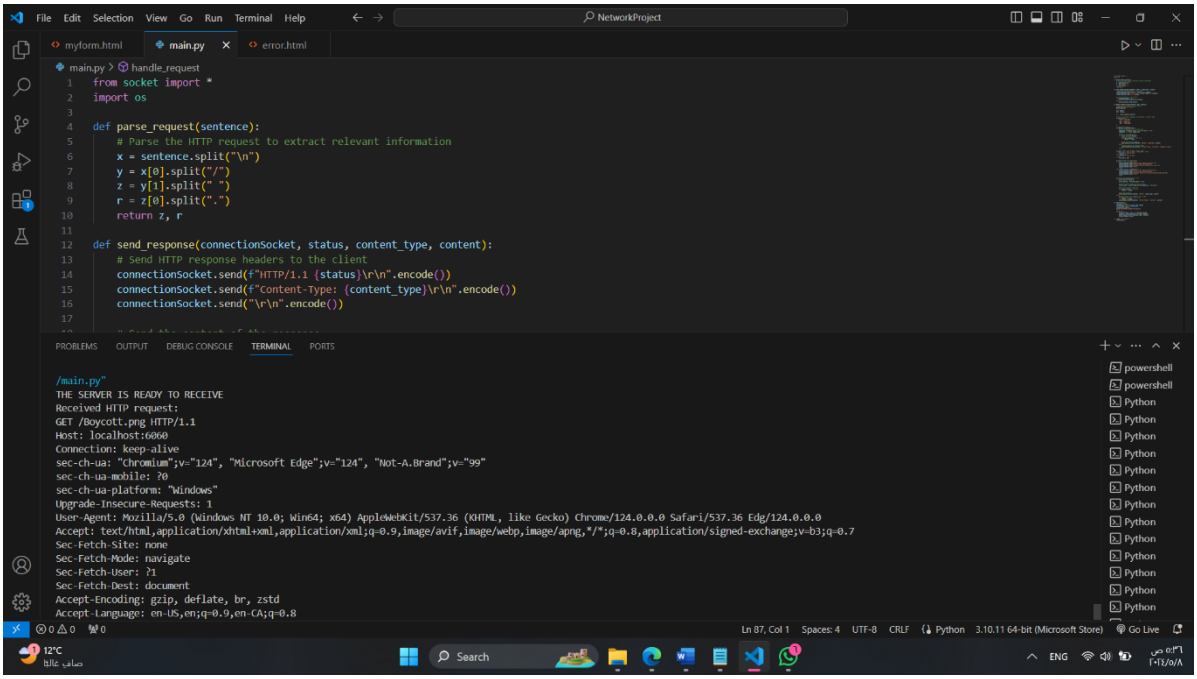


Figure 41: HTTP request '/Boycott.png'

➢ myform.html: Design and Functionality of myform.html

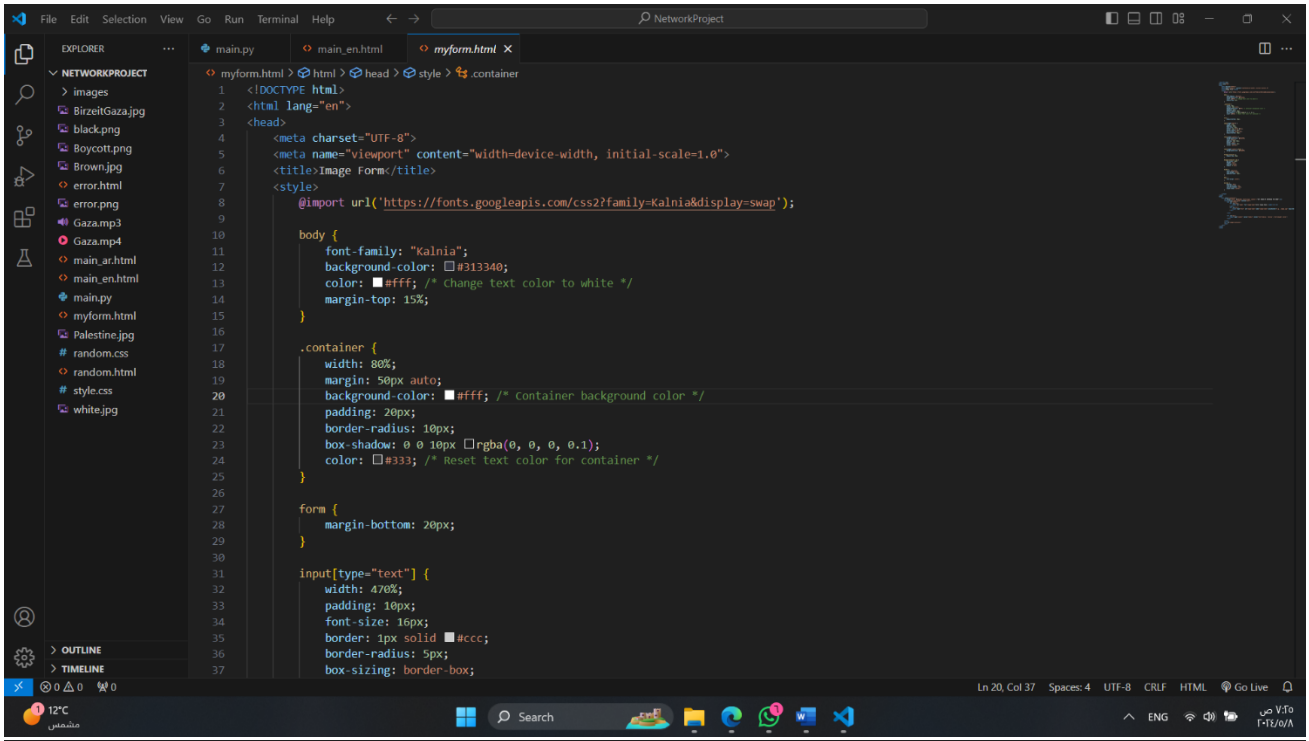
The "**myform.html**" page is designed to facilitate user **interaction with the server to request images**. Its layout emphasizes simplicity and functionality. The form includes a single **input field** where users can **enter the name of the image they want to retrieve**. The design ensures clarity with clear **labeling and placeholder text**, and it highlights that the **input field is required**. **CSS styling** enhances the visual appeal, with a modern font and a contrasting color scheme that improves readability.

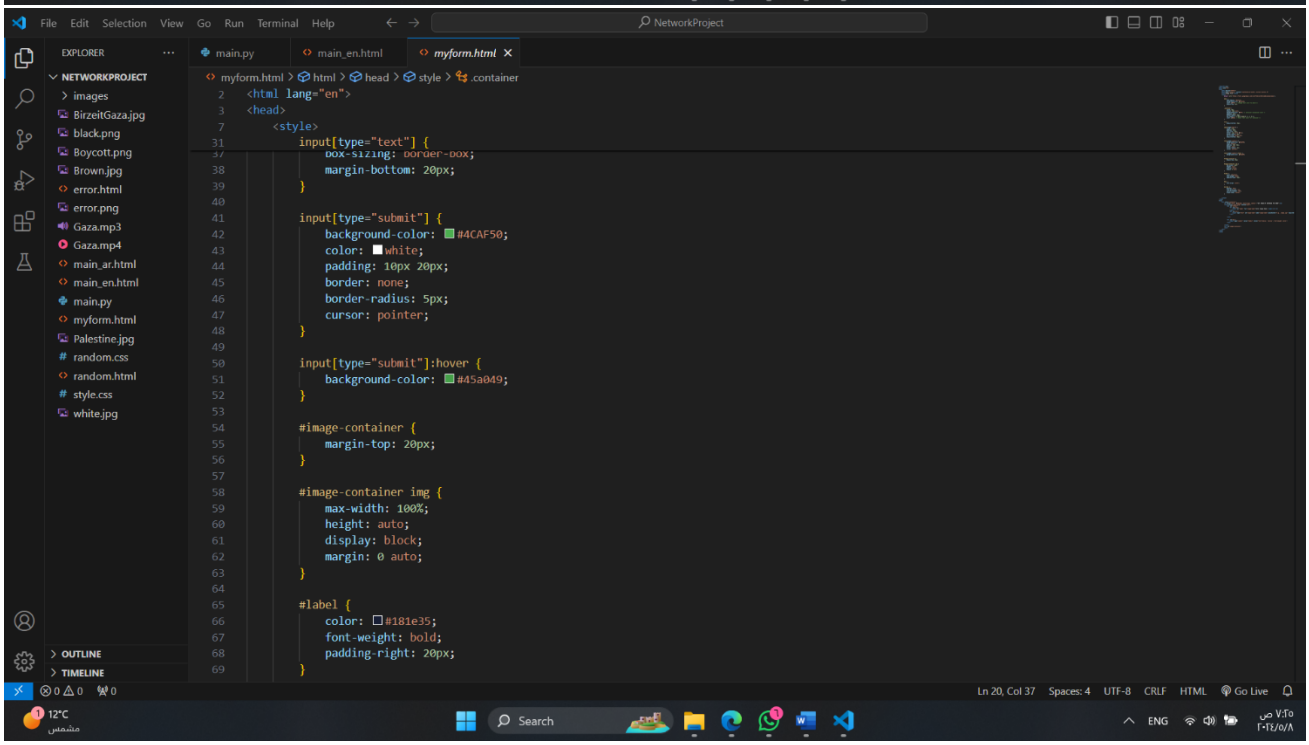
- **Submission Operation**

The provided Python script includes functionality for h**andling HTTP POST requests in the handle_request function**. When a POST request is received, the script extracts the image name from the request data and attempts to retrieve the corresponding **image file** from the server. **If the image exists, it sends the image content as a response**. However, **if the image is not found, it sends a "404 Not Found" error response**. This functionality is specifically designed to handle requests made to the "/get_image" endpoint, as specified in the **form action** of the HTML file named "myform.html".

Additionally, "**myform.html**" defines a form **where users can enter the name of an image file**, and upon submission, a **POST** request is sent to the server to retrieve the requested image.

- **myform.html Code**





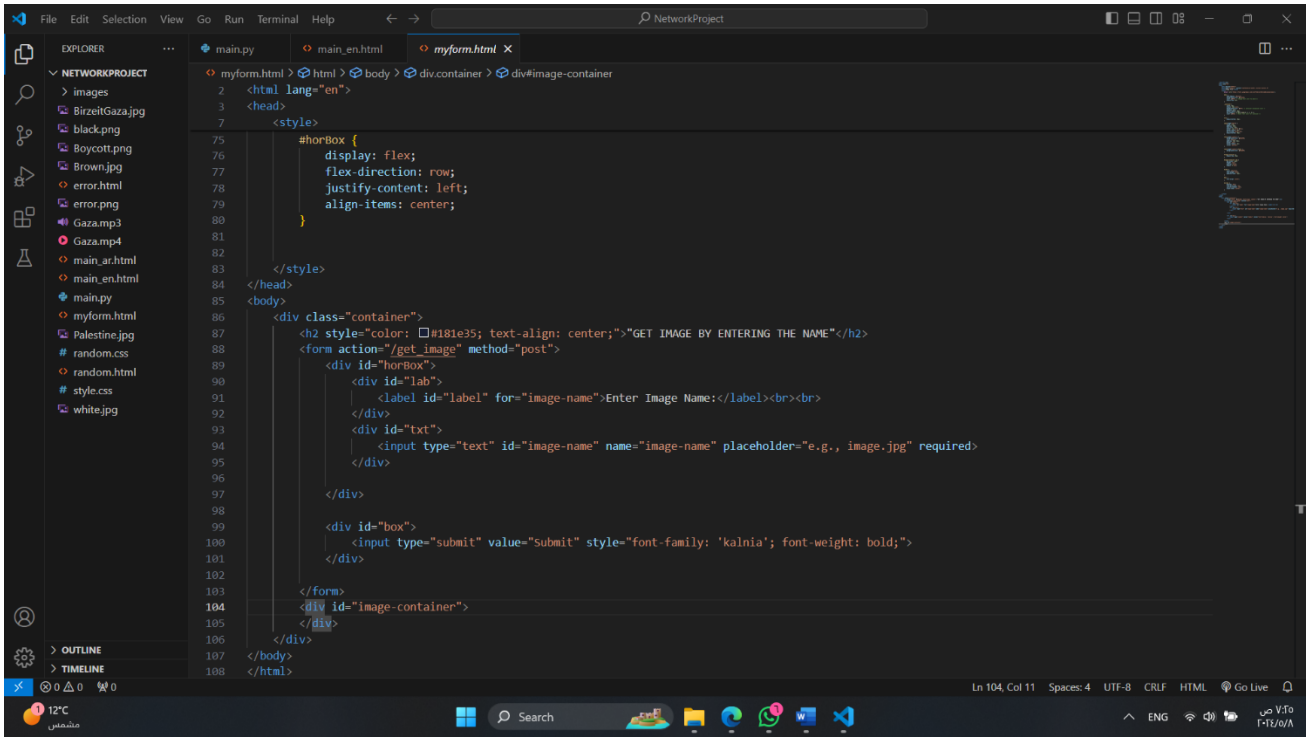
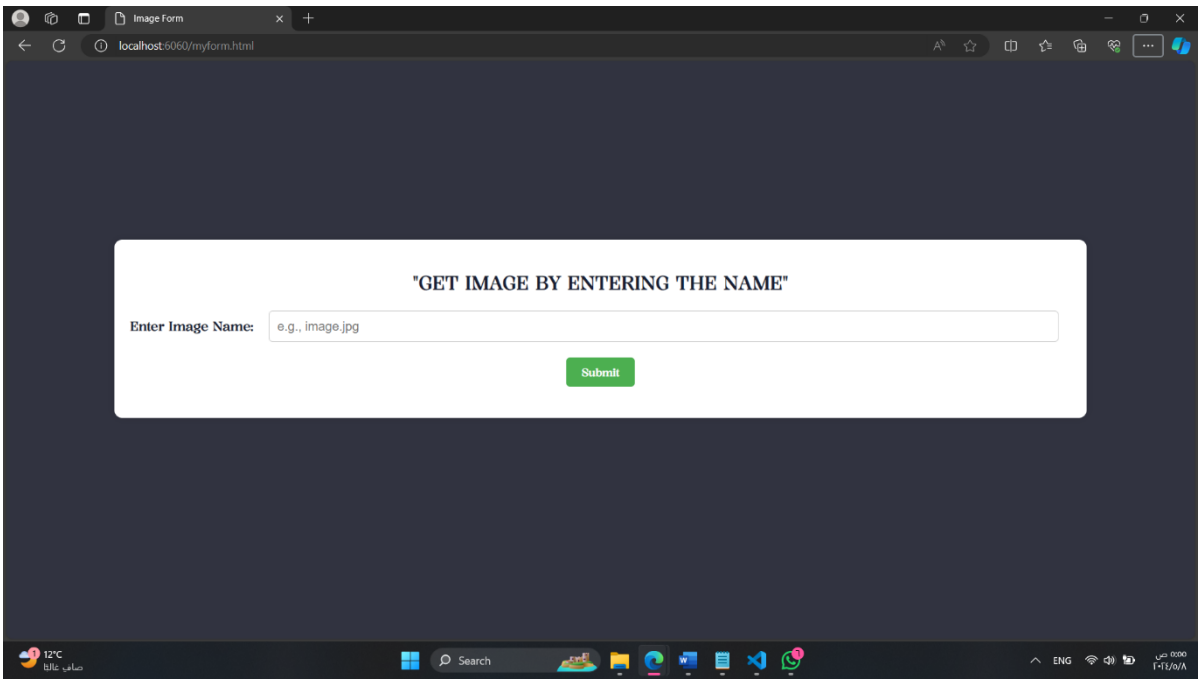


Figure 42: myform.html Code

- **The client is redirected to our image mapping form when the HTTP request '/myform.html' is received**



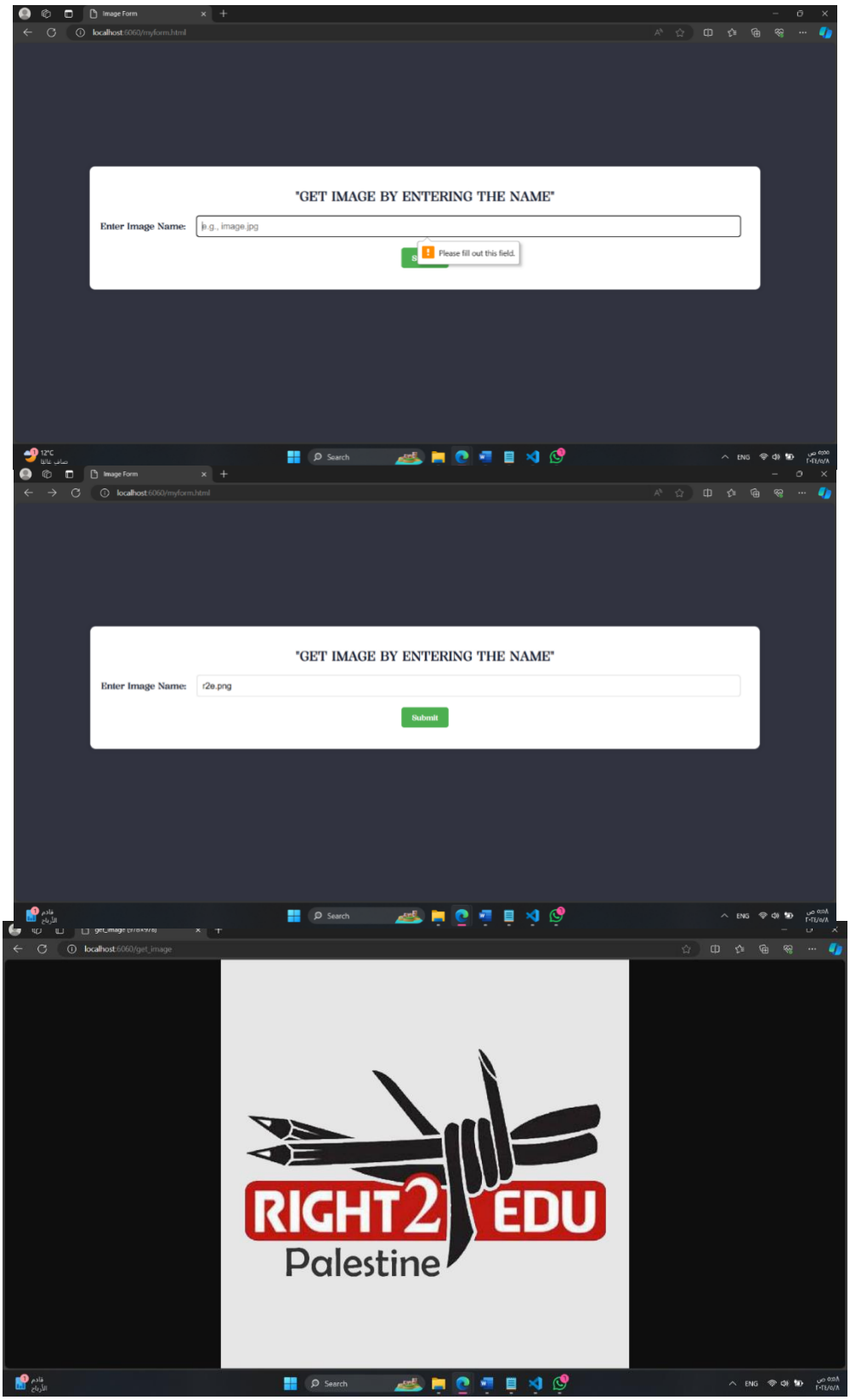


Figure 43: Image mapping form with HTTP request '/myform.html'

- **HTTP Request for myform.html**

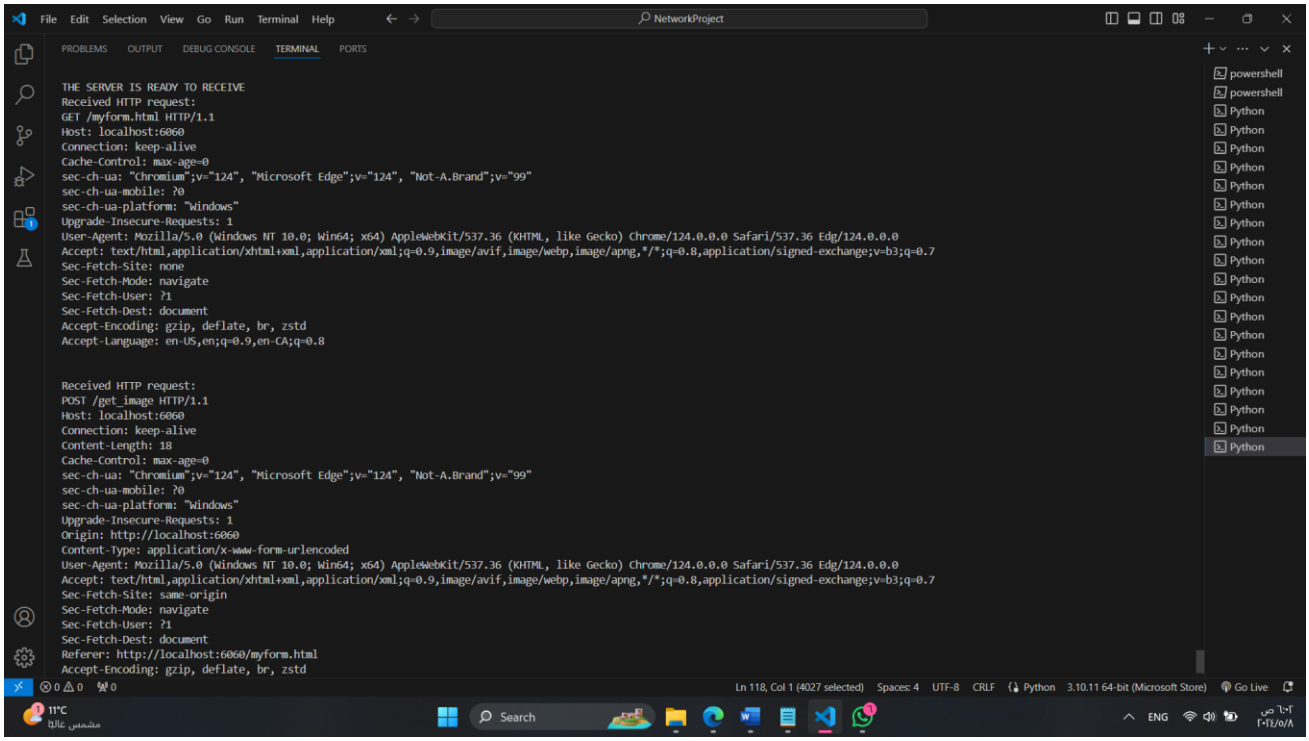


Figure 44: HTTP request '/myform.html'

- **Non-Existing Image**

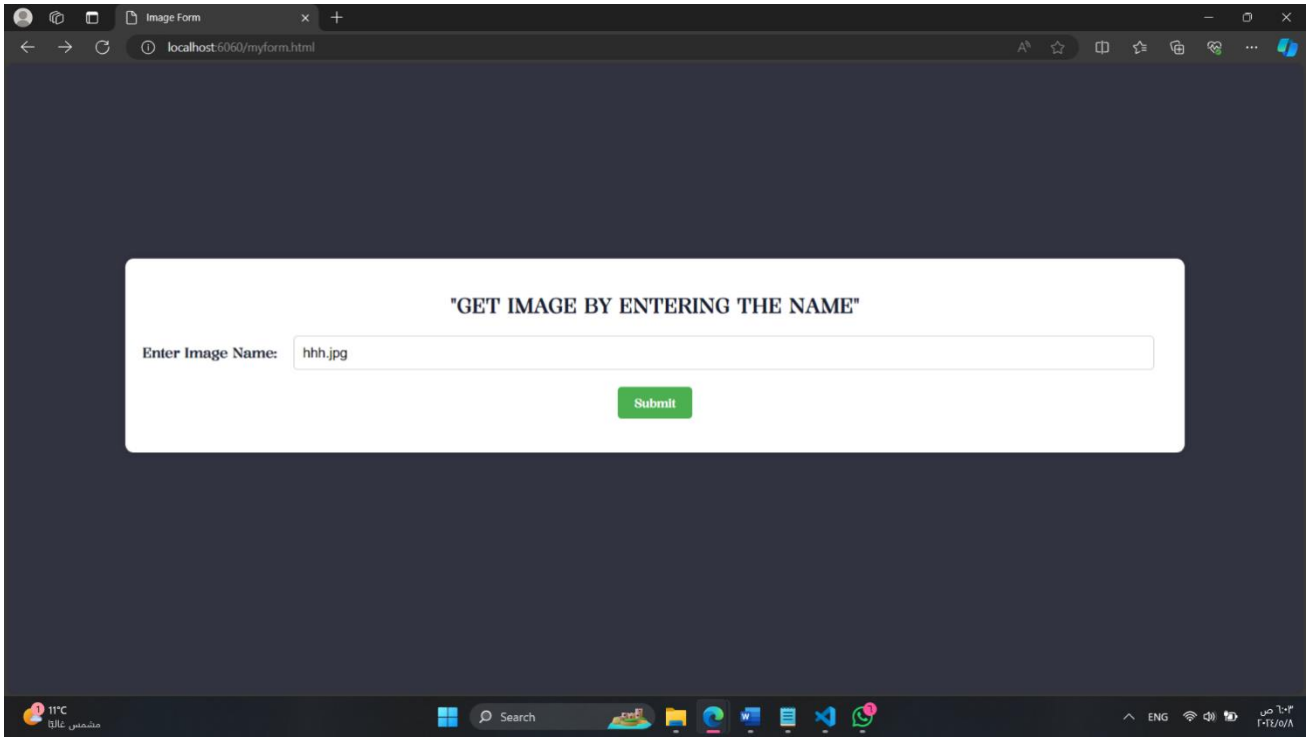


Figure 45: Non-Existing Image

- **The client is redirected to error file when the image is not found**

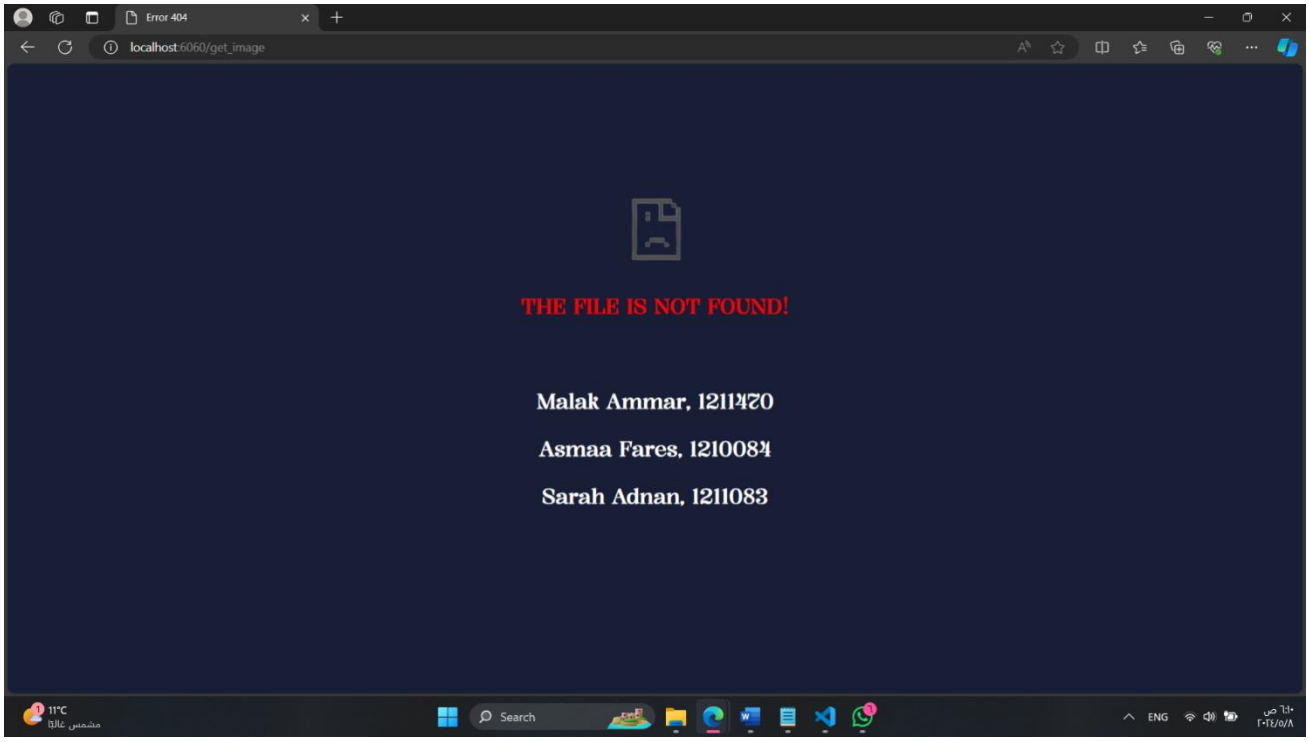


Figure 46: File not found error

- **HTTP Request for non-existing image**

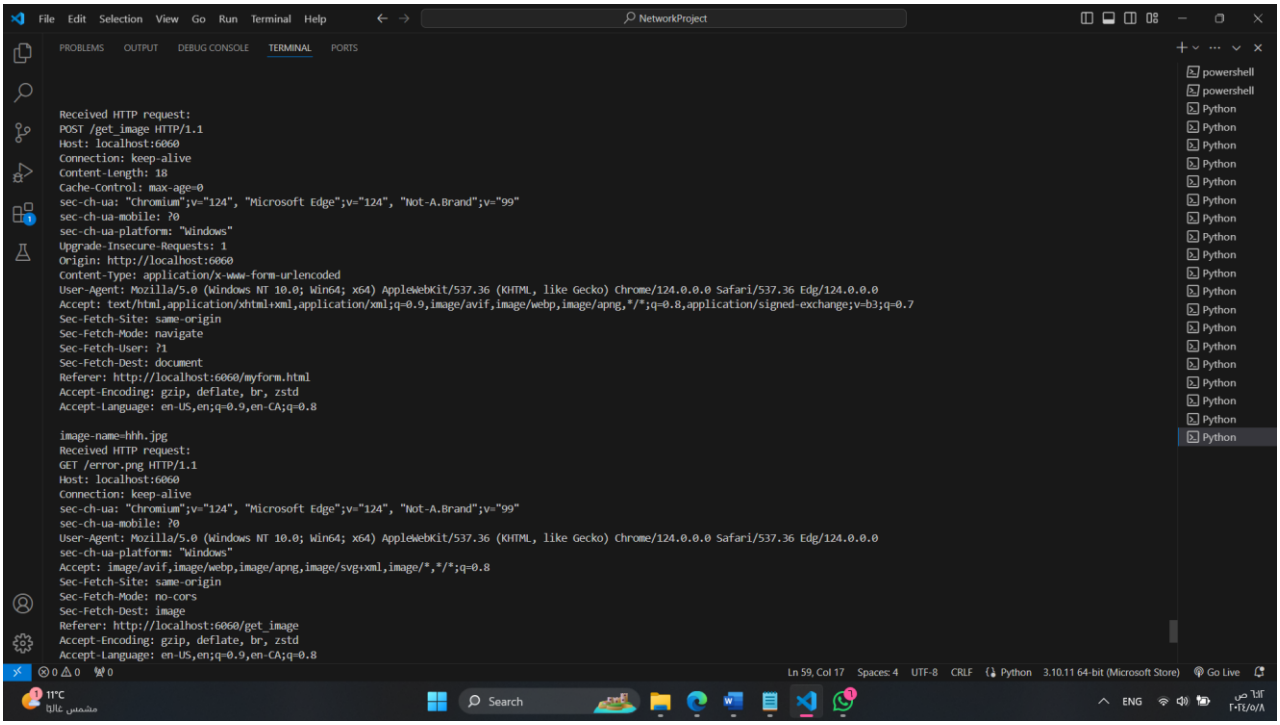


Figure 47: HTTP Request for non-existing image

➢ ‘SO’ HTTP Request

- **The client is redirected to stackoverflow.com when the HTTP request '/so' is received**

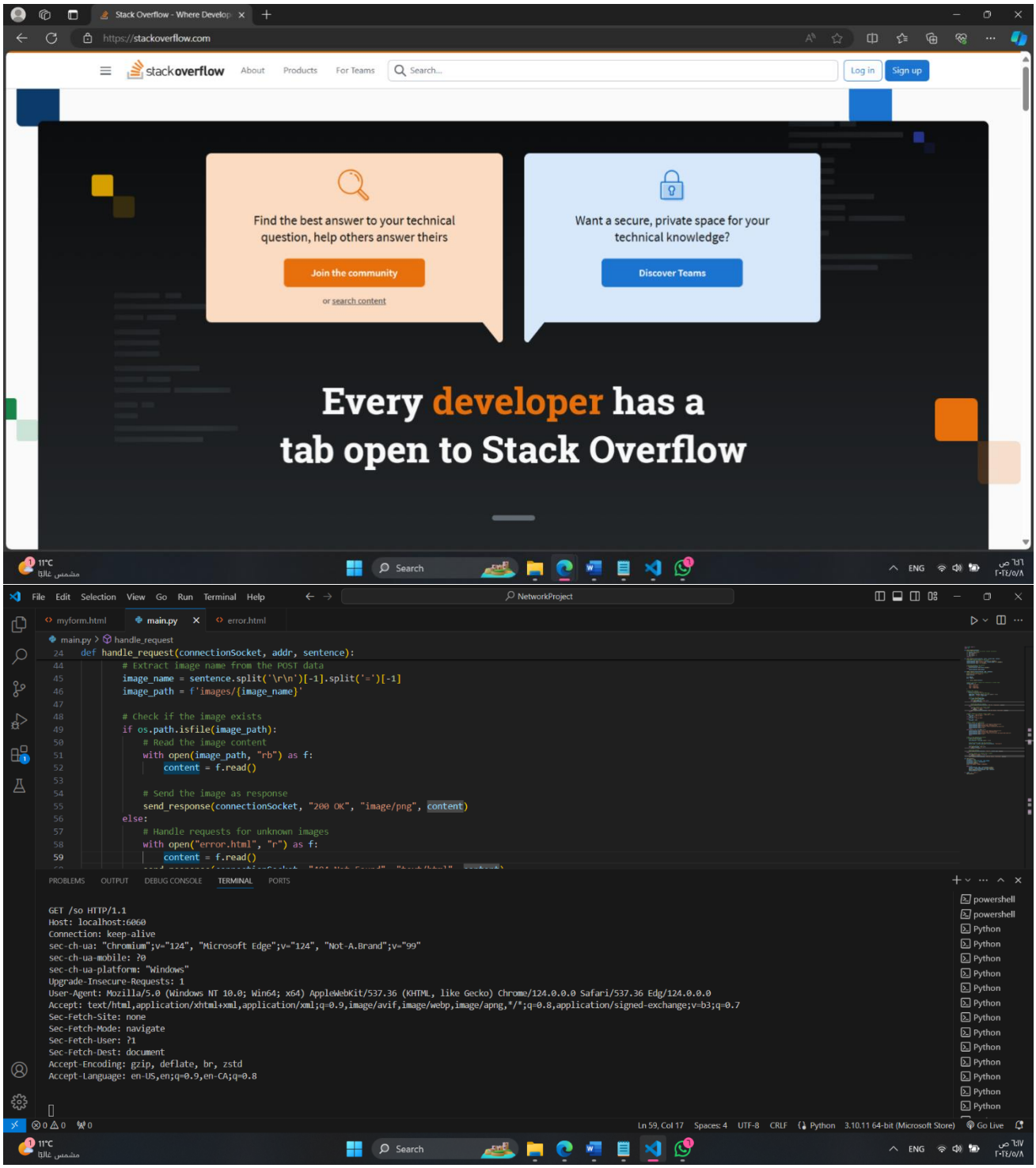


Figure 48: stackoverflow.com with HTTP request '/so'

➢ ‘ITC’ HTTP Request

- **The client is redirected to stackoverflow.com when the HTTP request '/itc' is received**

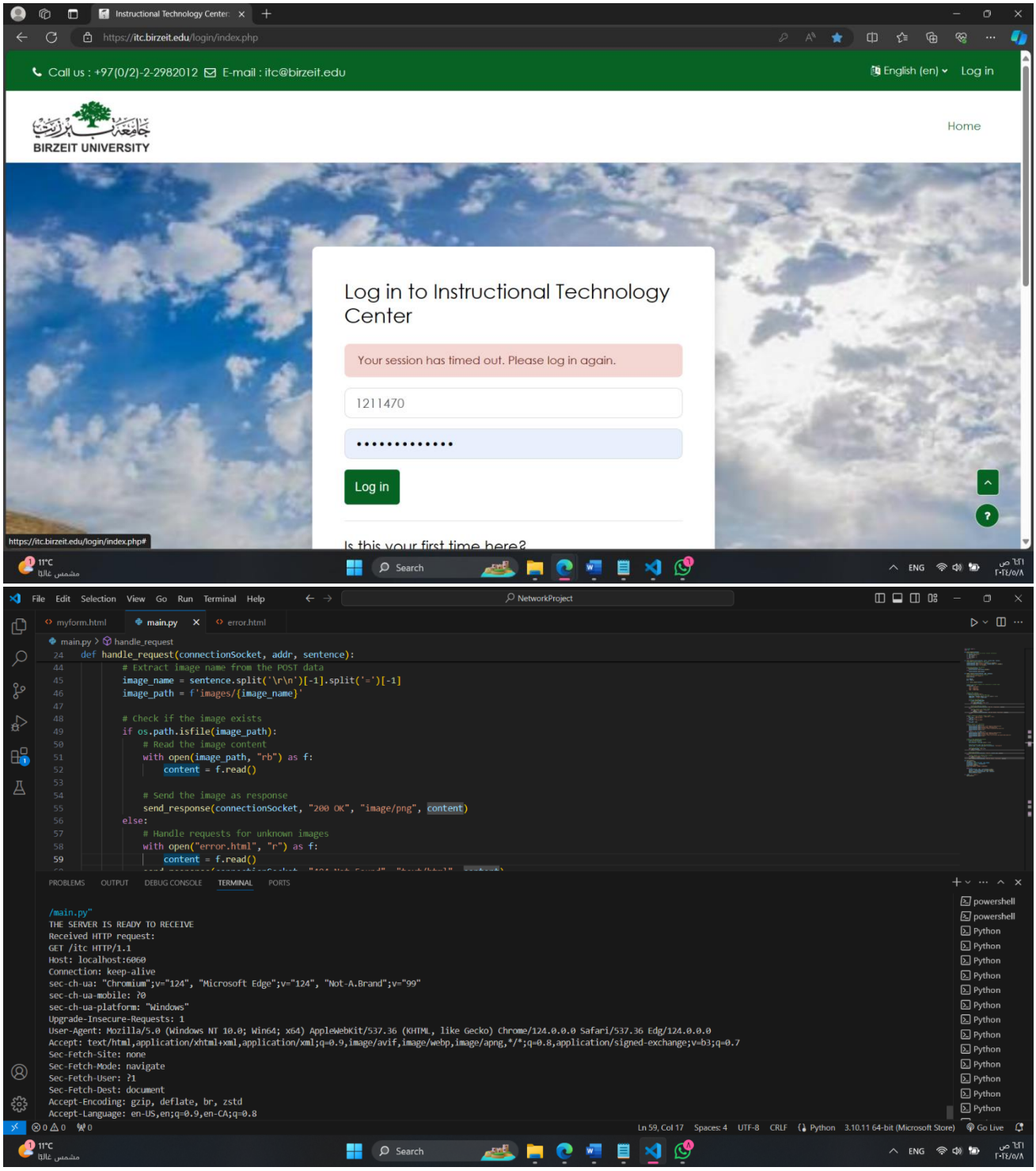


Figure 49: stackoverflow.com with HTTP request '/itc'

➢ error.html: Design and Functionality of error.html

The "**error.html**" page serves as a user-friendly interface to **inform users about errors encountered while accessing specific web content**. **CSS styling** is utilized to enhance visual appeal and maintain consistency with the overall website design.

- **error.html Code**

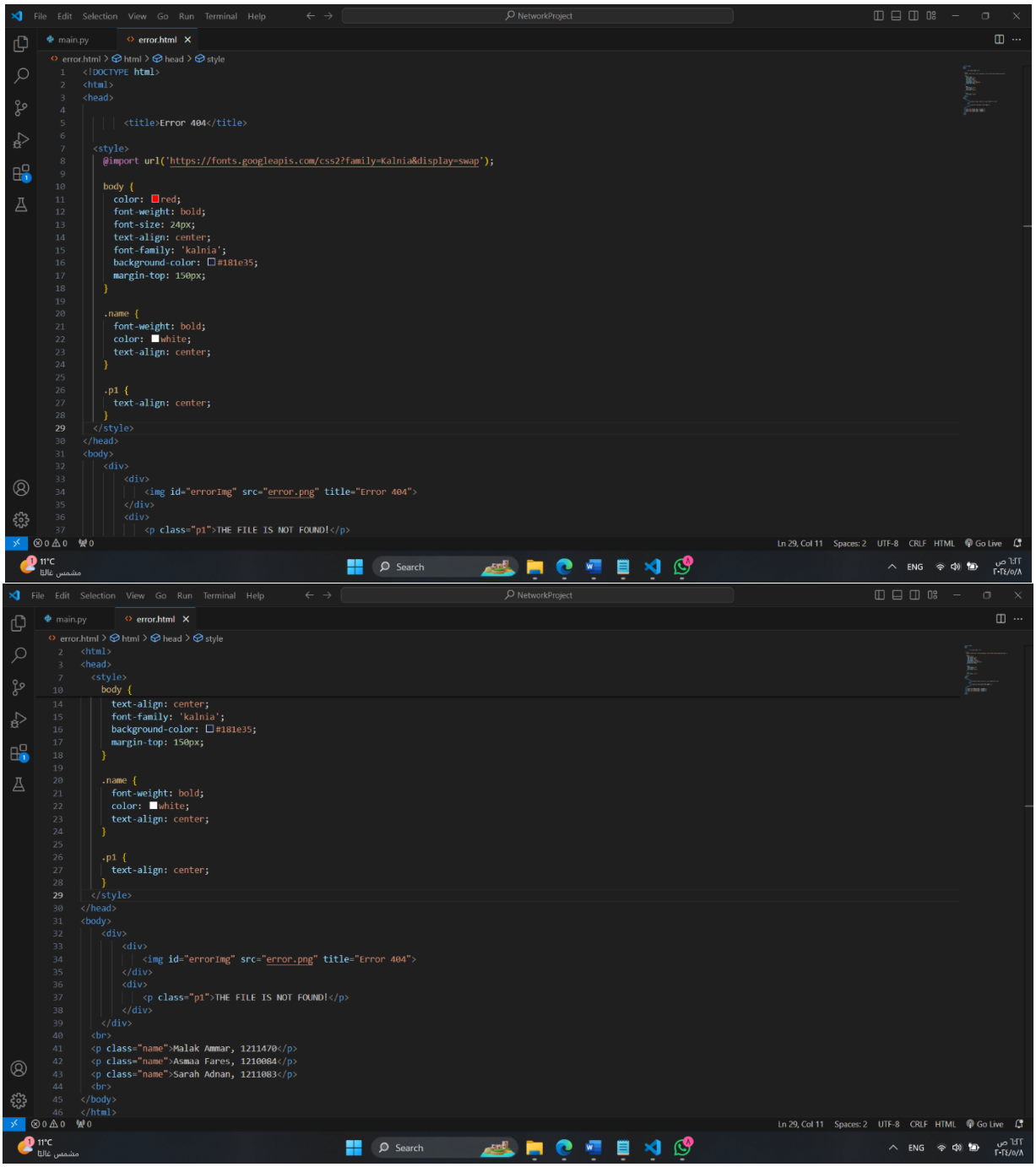


Figure 50: error.html Code

- **The client is redirected to error message file when the HTTP request '/error.html' is received**

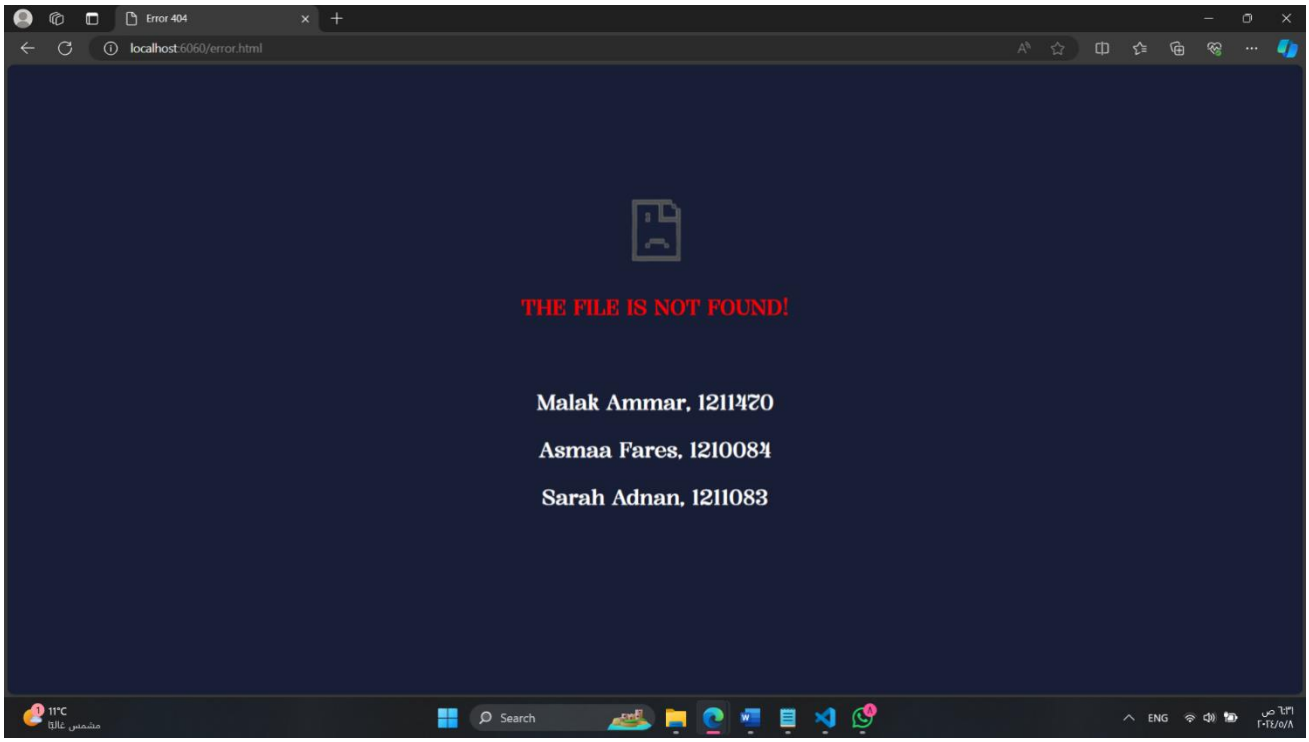


Figure 51: Error message file when the HTTP request '/error.html'

➢ Running Webpage on the Phone

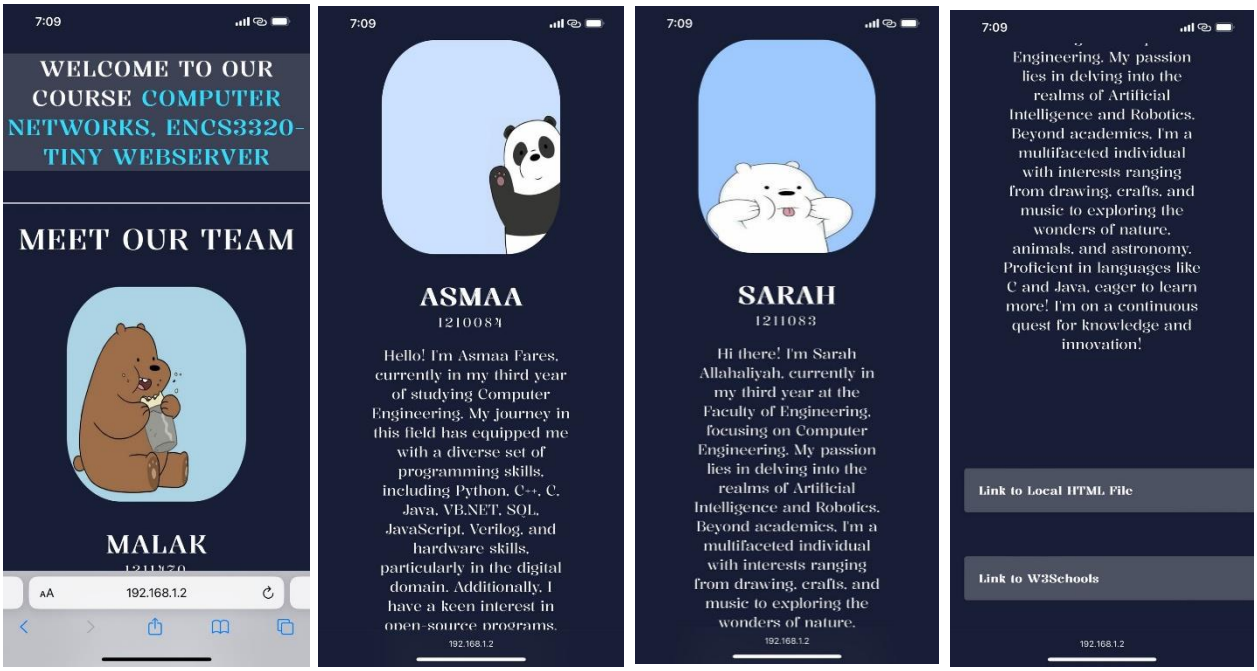


Figure 52: Running Webpage on the Phone

➢ Opening W3School from Main Webpage

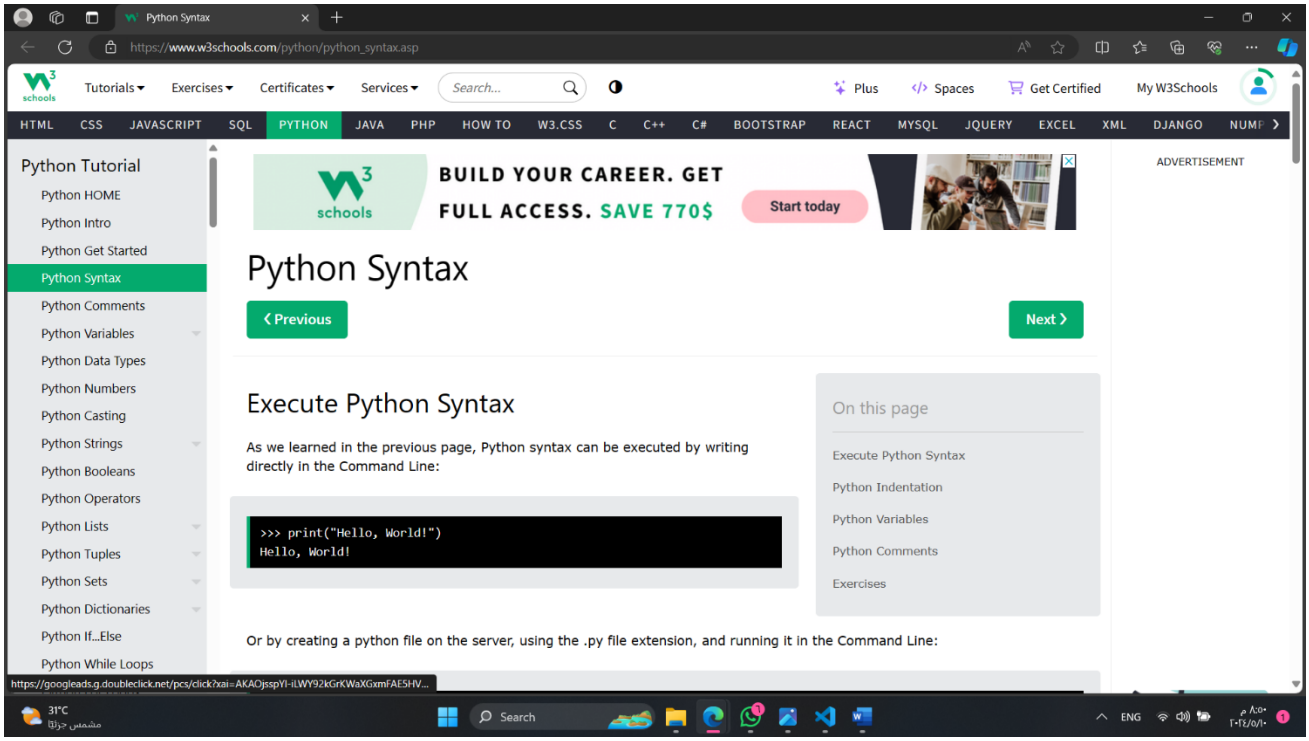


Figure 53: Running W3School website

Conclusion

In conclusion, our project has been an enriching journey blending theoretical knowledge with hands-on application. We began by dissecting networking concepts through analysis tools, delving into packet tracking and DNS message interpretation. Transitioning to practical implementation, we delved into socket programming, emphasizing UDP for peer-to-peer communication. The culmination of our efforts resulted in the creation of a functional web server capable of handling HTTP requests and serving diverse content. Additionally, we curated visually appealing web pages to showcase project team information, demonstrating our proficiency in web development. Overall, this project not only enhanced our understanding of networking fundamentals but also sharpened our skills in web development, paving the way for future endeavors in both domains.