



BIRZEIT UNIVERSITY

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

**ENCS 3320**

**Computer Networks**

**Project #1 – Report**

---

**Prepared by :**

Heba Fialah	1211315	Sec2
Layan Jarar	1210112	Sec3
Lama Khateeb	1213515	Sec3

**Instructor :**

Dr. Abdalkarim Awad & Dr. Ibrahim Nimer

**Date : 10-05-2024**

## Abstract

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

## Table Of Content

### Contents

Abstract .....	II
Table Of Content .....	III
Table Of Figure .....	V
Pare One : Ping and trace and nslookup the www.stanford.edu .....	1
1.1: Ping a device in the same network, e.g. from a laptop to a smartphone.....	1
1.2: By using command line to perform ping www.stanford.edu.....	1
1.3 : By using command line to perform tracert www.stanford.edu. ....	2
1.4 : By using command line to perform nslookup www.stanford.edu .....	3
1.5 : Using wireshark to capture some DNS messages .....	4
Part Two : : Socket programming .....	6
Key Features.....	6
1. Interactive Message Display .....	6
2. Timestamps .....	6
3. Network Addressing .....	6
Technical Details.....	6
1. Server Enhancements .....	6
2. Client Upgrades .....	7
Usage Scenario .....	7
Student Collaboration .....	7
<b>Project Outcome</b> .....	7
<b>Screenshots and Documentation</b> .....	7
Code and result .....	8
Part Three : Web Server .....	11
Web server code .....	11
From rfce2616, what is Entity Tag Cache Validators in the HTTP protocol and why do we need it?.....	13
3.1 : If the request is / or /index.html or /main_en.html or /en (for example localhost:6060/ or localhost:6060/en) then the server should send main_en.html file with Content-Type: text/html. ....	14
3.2 : If the request is /ar then the server response with main_ar.html which is an Arabic version of main_en.html.....	17

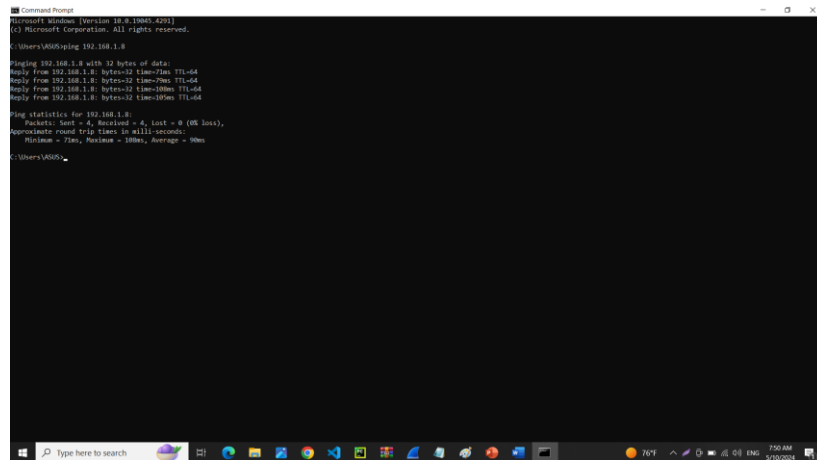
3.3 : If the request is an .html file then the server should send the requested html file with Content-Type: text/html. use any html file. ....	19
3.4 : If the request is a .css file then the server should send the requested css file with Content Type: text/css. use any CSS file. ....	19
3.5 : If the request is a .png then the server should send the png image with Content-Type: image/png. You can use any image. ....	20
3.6 : If the request is a .jpg then the server should send the jpg image with Content-Type: image/jpeg. use any image. ....	21
3.7 : Use myform.html to get image by typing the name of the image in a box For instance.....	22
3.8 : Use the status code 307 Temporary Redirect to redirect the following .....	22
3.8.1 : If the request is /so then redirect to stackoverflow.com website.....	22
3.8.2 : If the request is /itc then redirect to itc.birzeit.edu website .....	23
3.9 : If the request is wrong or the file doesn't exist the server should return a simple HTML webpage that contains (Content-Type: text/html) .....	24

## Table Of Figure

Figure 1 : results of pinging a device in the same network.....	1
Figure 2 : resulting of pinging www.stanford.edu.....	1
Figure 3 :Results of using the command tracert www.stanford.edu .....	2
Figure 4 : results of using the command nslookup www.stanford.edu .....	3
Figure 5 : DNS query message captured by Wireshark .....	5
Figure 6 : DNS query response message captured by Wireshark .....	5
Figure 7 : Code for server .....	8
Figure 8 :code for first client .....	9
Figure 9 : screenshot for testing the server output .....	9
Figure 10 : screenshot for testing the first client output .....	9
Figure 11 :screenshot for testing the second client output .....	10
Figure 12 : web serve (1).....	11
Figure 13 : web server (2) .....	11
Figure 14 : web server(3) .....	12
Figure 15 : web server(4) .....	12
Figure 16 : server code(5) .....	12
Figure 17 : The HTTP request '/en' redirects the client to our webpage. ....	14
Figure 18 : main_en.....	14
Figure 19 : The HTTP request '/index.html' redirects the client to our webpage.....	15
Figure 20 : HTTP request for english .....	16
Figure 21 : The HTTP request '/ar' redirects the client to our webpage. ....	17
Figure 22 : HTTP request for arabic.....	17
Figure 23 : main_ar .....	17
Figure 24 : HTTP request for Arabic .....	18
Figure 25 : Style code .....	19
Figure 26 : HTTP request for css.....	20
Figure 27 : png image.....	20
Figure 28 : HTTP request for png .....	21
Figure 29 : jpg image.....	21
Figure 30 : HTTP request for jpg.....	22
Figure 31 : stachoverflow.com .....	22
Figure 32 : HTTP request for stackoverflow .....	23
Figure 33 : itc.edu.....	23
Figure 34 : HTTP request for itc.....	24
Figure 35 : request is wrong or the file doesn't exist.....	24
Figure 36 : wrong HTTP request.....	25
Figure 37 : Arabic webserver (1) .....	25
Figure 38 : Arabic webserver(2) .....	26
Figure 39 : Arabic webserver(3) .....	26
Figure 40 : English webserver(1).....	27
Figure 41 : English webserver (2) .....	27
Figure 42 :English webserver(3).....	28

## Pare One : Ping and trace and nslookup the [www.stanford.edu](http://www.stanford.edu)

### 1.1: Ping a device in the same network, e.g. from a laptop to a smartphone.



```
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\A002>ping 192.168.1.8

Pinging 192.168.1.8 with 32 bytes of data:
Reply from 192.168.1.8: bytes=32 time=79ms TTL=64
Reply from 192.168.1.8: bytes=32 time=79ms TTL=64
Reply from 192.168.1.8: bytes=32 time=105ms TTL=64
Reply from 192.168.1.8: bytes=32 time=108ms TTL=64

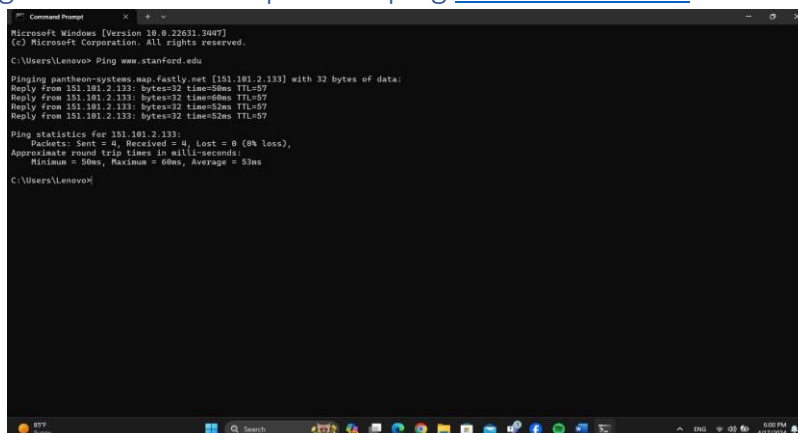
Ping statistics for 192.168.1.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 79ms, Maximum = 108ms, Average = 90ms

C:\Users\A002>
```

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

As shown above after I pinged my smartphone that the IP address is 192.168.1.8 . The target IP address receives four ping packets by default each of them 32 bytes in size .The time needed for each packet to send and received measured by milliseconds It is varied between 71ms, 79ms,105ms, 108ms .TTL is the time to live for packet, which is the number of network devices along the path that the packet can pass through , Thus, a packet here is allowed to pass through 64 devices. These replies are called "echo reply to requests" and the outcome of each ping is displayed on a line.

### 1.2: By using command line to perform ping [www.stanford.edu](http://www.stanford.edu).



```
Microsoft Windows [Version 10.0.22621.3407]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>ping www.stanford.edu

Pinging pantheon-systems.map.fastly.net [151.101.2.133] with 32 bytes of data:
Reply from 151.101.2.133: bytes=32 time=58ms TTL=57
Reply from 151.101.2.133: bytes=32 time=60ms TTL=57
Reply from 151.101.2.133: bytes=32 time=52ms TTL=57
Reply from 151.101.2.133: bytes=32 time=52ms TTL=57

Ping statistics for 151.101.2.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 58ms, Maximum = 60ms, Average = 53ms

C:\Users\Lenovo>
```

Figure 2 : resultng of pinging [www.stanford.edu](http://www.stanford.edu)

**Ping:** Ping is a network administration utility used to test the reachability of a host on an Internet Protocol (IP) network. It operates by sending Internet Control Message Protocol (ICMP) echo request packets to the target host and waits for an echo reply, measuring the time it takes for the round-trip.

**Interpreting Ping Results:** The ping results to “www.stanford.edu” show response times ranging from **50ms to 60ms** with a **Time To Live (TTL)** of **57**. These response times are relatively low, which typically indicates that the server is geographically closer to the user or has a highly efficient network route. Given that Stanford University is located in the USA, the low response times could suggest that the server is indeed in the USA. However, it’s also possible that the server is part of a content delivery network (CDN), which could provide low-latency responses globally.

**Brief Explanation of the Output:** The output indicates a successful connection to the server with no packet loss, which means all the data packets sent were received back, confirming the server’s reachability. The consistent response times and the absence of packet loss denote a stable network connection.

1.3 : By using command line to perform `tracert www.stanford.edu`.

```
C:\Users\Lenovo> Tracert www.stanford.edu

Tracing route to pantheon-systems.map.fastly.net [151.101.2.133]
over a maximum of 30 hops:
  0  5 ms  1 ms  1 ms  192.168.1.1 [192.168.1.1]
  1  18 ms  8 ms  18 ms  10.74.32.246 [10.74.32.246]
  2  75 ms  75 ms  70 ms  10.74.19.113 [10.74.19.113]
  3  109 ms  117 ms  92 ms  10.74.19.166 [10.74.19.166]
  4  78 ms  88 ms  93 ms  10.74.59.198 [10.74.59.198]
  5  *      *      *      Request timed out.
  6  *      *      *      Request timed out.
  7  50 ms  49 ms  49 ms  151.101.2.133

Trace complete.

C:\Users\Lenovo>
```

Figure 3 :Results of using the command `tracert www.stanford.edu`

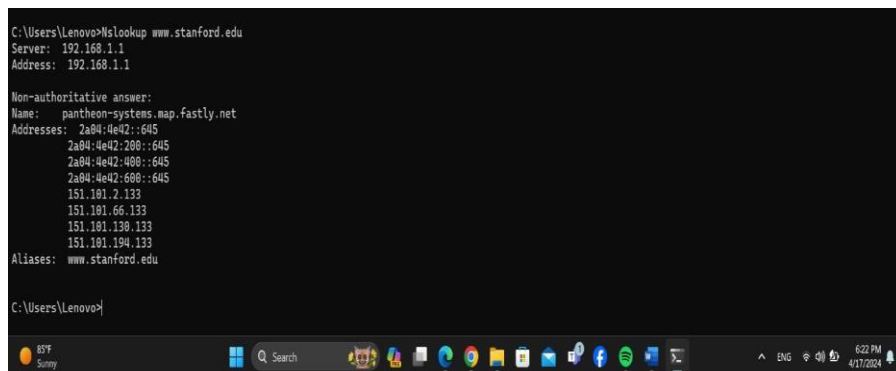
**Tracert:** The tracert command is a network diagnostic tool used to determine the path packets take to reach a specific destination across an IP network. It identifies each hop along this path and measures the time each hop takes to respond, which is useful for pinpointing where potential delays or failures occur in the network.

### Explanation of the Tracert Output:

- **Hop 1 [192.168.1.1]:** The first hop is typically the local router or gateway, with a very low response time (1ms to 5ms), indicating a quick and healthy connection within the local network.
- **Hop 2 [10.74.32.246]** and subsequent hops until **Hop 5 [10.74.59.198]:** These are intermediate hops, likely within the ISP's network or along the route to the destination. The increasing response times (from 8ms to 93ms) are normal as the distance to the destination increases.
- **Hop 6:** The asterisks (\*) indicate that the request timed out. This hop did not respond, which could be due to a variety of reasons, such as firewalls or routers configured not to reply to ICMP requests.
- **Hop 7 [151.101.2.133]:** The final hop reaches the destination server associated with "pantheon-systems.map.fastly.net," with a consistent and relatively low response time (around 50ms). This indicates that the packets have successfully reached the target server, and the path is clear and functioning well.

The trace was completed in 7 hops, and despite the timeout at hop 6, the overall path is efficient, and the connection to the destination is stable.

#### 1.4 : By using command line to perform nslookup [www.stanford.edu](http://www.stanford.edu).



```
C:\Users\Lenovo>nslookup www.stanford.edu
Server: 192.168.1.1
Address: 192.168.1.1

Non-authoritative answer:
Name:   pantheon-systems.map.fastly.net
Address: 2a04:4e42::645
        2a04:4e42:208::645
        2a04:4e42:408::645
        2a04:4e42:608::645
        151.101.2.133
        151.101.66.133
        151.101.130.133
        151.101.194.133
Aliases: www.stanford.edu

C:\Users\Lenovo>
```

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

**Nslookup:** The nslookup command is a network administration tool used for querying the Domain Name System (DNS) to obtain domain name or IP address mapping for any specific



DNS record. It is useful for diagnosing DNS infrastructure and for verifying the correct DNS server is responding.

#### **Explanation of the Nslookup Output:**

- **Server and Address:** The first two lines indicate the DNS resolver that the user's system is using, which in this case is the local gateway with an IP address of **192.168.1.1**.
- **Non-authoritative answer:** This label means that the response comes from a server that does not have authoritative knowledge of the domain, typically a cached response from a local or intermediate DNS server.
- **Name:** The domain name that was looked up, which resolved to "pantheonsystems.map.fastly.net," indicating that this is the server responding for the queried domain.
- **Addresses:** A list of IPv6 and IPv4 addresses associated with the name. The IPv6 addresses are shown in hexadecimal colon-separated notation, and the IPv4 addresses are in decimal dot notation.
- **Aliases:** This shows any alternative domain names associated with the queried domain. In this case, "www.stanford.edu" is an alias for "pantheon-systems.map.fastly.net."

The output indicates that "www.stanford.edu" is served by a server under the domain "pantheonsystems.map.fastly.net," which has multiple IP addresses, suggesting a robust and redundant hosting setup, likely part of a content delivery network (CDN).

#### **1.5 : Using wireshark to capture some DNS messages**

In this part will be use the Wireshark program to capture DNS messages as a result of pinging [www.stanford.edu](http://www.stanford.edu) and get a detailed notice that it is an Ethernet II , Internet Protocol Version 4 , User Datagram Protocol and Domain Name System (response) frame. As shown in the figure below .DNS query requesting ping assigns the IP address to the URL [www.stanford.edu](http://www.stanford.edu) , as a result of being pinged. for An A record which stores the IP address of the host name.

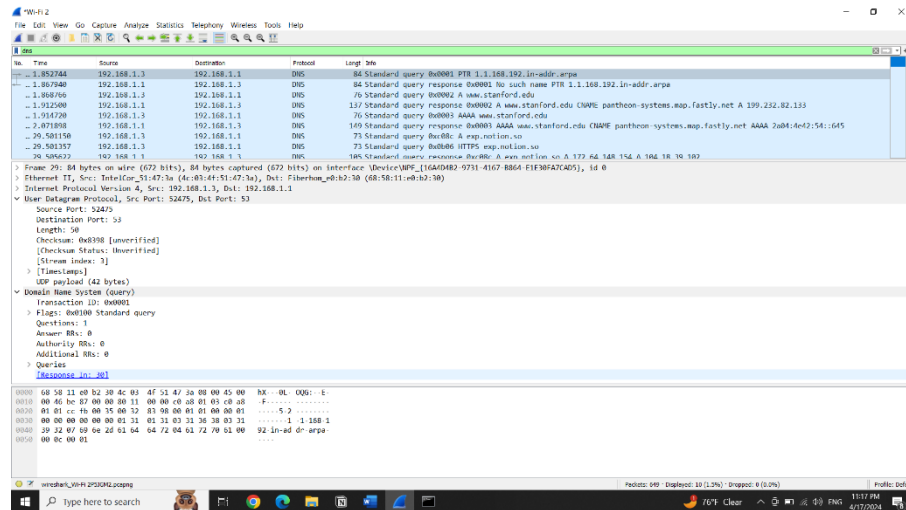


Figure 5 : DNS query message captured by Wireshark

The below DNS query response message contains two types of records, namely A and CNAME. These hold an IP address and an alias, respectively, The default port for DNS in Wireshark is 53, and the protocol is UDP (User Datagram Protocol).

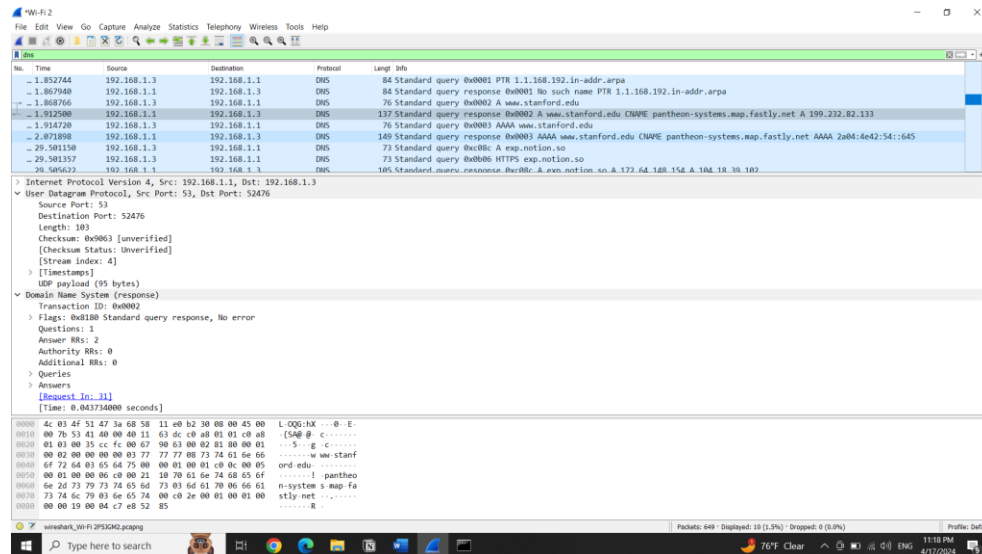


Figure 6 : DNS query response message captured by Wireshark

## Part Two : : Socket programming

Overview In this phase of our project, we enhance the basic UDP chat application to create a more robust and interactive chat system. The advanced UDP chat system allows multiple clients to communicate within the same local network. Let's dive into the key features and technical details

### Key Features

#### 1. Interactive Message Display

- Clients can now view specific messages interactively.
- By entering the line number followed by 'D' (e.g., 2D), clients retrieve the content of a particular message.
- This feature enhances usability and allows users to review past conversations easily.

#### 2. Timestamps

- Each received message is timestamped.
- The server records the exact time when a message arrives from a specific sender.
- Timestamps provide a chronological order of messages and help track communication history.

#### 3. Network Addressing

- The application uses the sender's IP address to uniquely identify clients.
- This ensures that messages are correctly attributed to the respective senders.
- Network addressing plays a crucial role in distinguishing between different peers.

### Technical Details

#### 1. Server Enhancements

- The server continues to listen on port 5051.
- It now maintains a list of the most recent messages received from each client.
- When a new message arrives, the server updates this list and displays the last message from each sender.

## 2. Client Upgrades

- Clients can send messages to the server as before.
- Additionally, they can request specific message content using the interactive line number and 'D' command.
- The client-side experience is more intuitive and user-friendly.

### Usage Scenario

#### Student Collaboration

- Ideally, three students participate: at least two as clients and one as the server.
- The system facilitates peer-to-peer communication within the same subnet.
- Students can exchange messages, view historical conversations, and explore the system's capabilities.

### Project Outcome

The Advanced UDP Chat System demonstrates practical network programming concepts. It serves as an educational tool for understanding socket communication, broadcasting, and real-world applications of UDP.

### Screenshots and Documentation

- Include screenshots of the server and client terminals.
- Accompany each screenshot with a brief description, highlighting the context and actions taking place.

## Code and result

```
import socket
import threading
import datetime
import queue

# Constants for the user's first and last name
FIRST_NAME = "Lama"
LAST_NAME = "Khattib"

# Server details
BROADCAST_IP = '26.255.255.255'
SERVER_PORT = 5051
server_address = (BROADCAST_IP, SERVER_PORT)

# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Enable broadcasting mode
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
sock.bind(('', SERVER_PORT))

# Dictionary to store client info
clients = {}
client_messages = []

# Queue for messages to be sent
message_queue = queue.Queue()

# Function to handle receiving messages
# usage
def receive_message():
    data, address = sock.recvfrom(4096)
    if address != sock.getsockname(): # Check if the sender is not the current user
        message = data.decode()
        first_name, last_name, msg = message.split(';')
        timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        client_info = {'name': f'{first_name} {last_name}', 'timestamp': timestamp, 'message': msg}
        clients[address] = client_info
        client_messages.append(client_info)
        print(f"Received a message from {first_name} {last_name} at {timestamp}")

# Function to handle sending messages
# usage
def send_message():
    while True:
        message = message_queue.get()
        if message == "QUIT":
            break
        full_message = f'{FIRST_NAME};{LAST_NAME};{message}'
        sock.sendto(full_message.encode(), server_address)

# Main function to run the send and receive functions
# usage
def main():
    recv_thread = threading.Thread(target=receive_message)
    send_thread = threading.Thread(target=send_message)

    recv_thread.start()
    send_thread.start()

    while True:
        command = input("Enter a line number and 'D' to display a message (e.g., 2D), 'Q' to quit or 'S' to enter message: ")
        if command.upper() == 'Q':
            message_queue.put("QUIT")
            break
        elif command[-1].upper() == 'D' and command[:-1].isdigit():
            line_number = int(command[:-1]) - 1
            if 0 <= line_number < len(client_messages):
                selected_message = client_messages[line_number]
                print(f"Message from {selected_message['name']} at {selected_message['timestamp']}: {selected_message['message']}")
            else:
                print("Invalid line number.")
        elif command.upper() == 'S':
            message = input("Enter your message: ")
            message_queue.put(message)
        else:
            print("Invalid command.")

    recv_thread.join()
    send_thread.join()

if __name__ == "__main__":
    main()
```

```
while True:
    command = input("Enter a line number and 'D' to display a message (e.g., 2D), 'Q' to quit or 'S' to enter message: ")
    if command.upper() == 'Q':
        message_queue.put("QUIT")
        break
    elif command[-1].upper() == 'D' and command[:-1].isdigit():
        line_number = int(command[:-1]) - 1
        if 0 <= line_number < len(client_messages):
            selected_message = client_messages[line_number]
            print(f"Message from {selected_message['name']} at {selected_message['timestamp']}: {selected_message['message']}")
        else:
            print("Invalid line number.")
    elif command.upper() == 'S':
        message = input("Enter your message: ")
        message_queue.put(message)
    else:
        print("Invalid command.")

    recv_thread.join()
    send_thread.join()

if __name__ == "__main__":
    main()
```

Figure 7 : Code for server

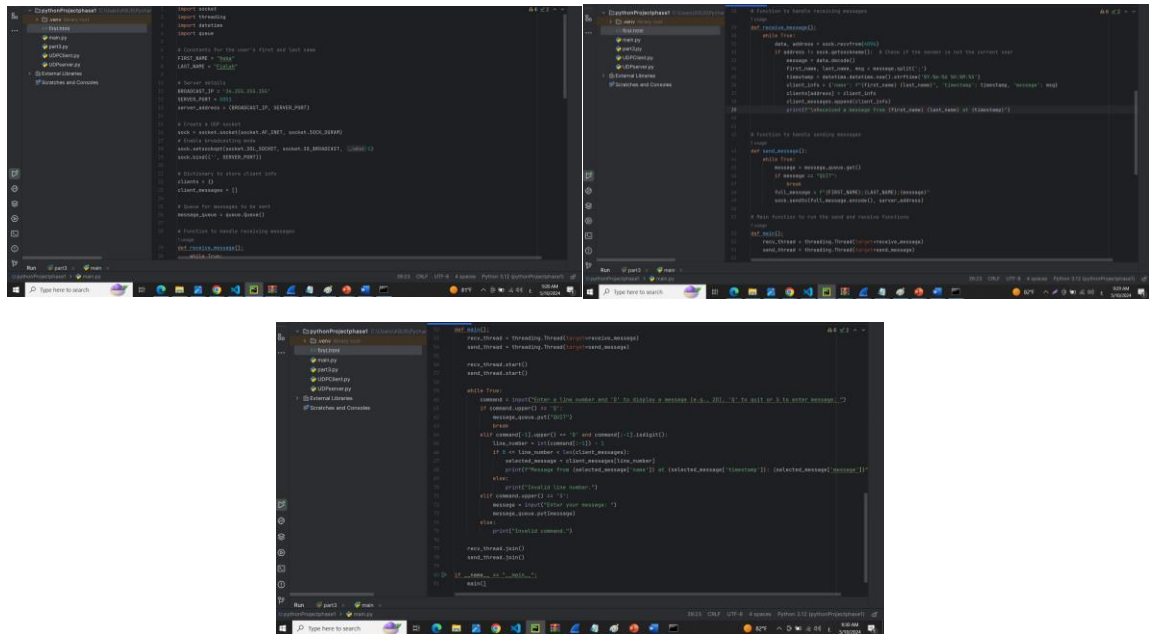


Figure 8 :code for first client

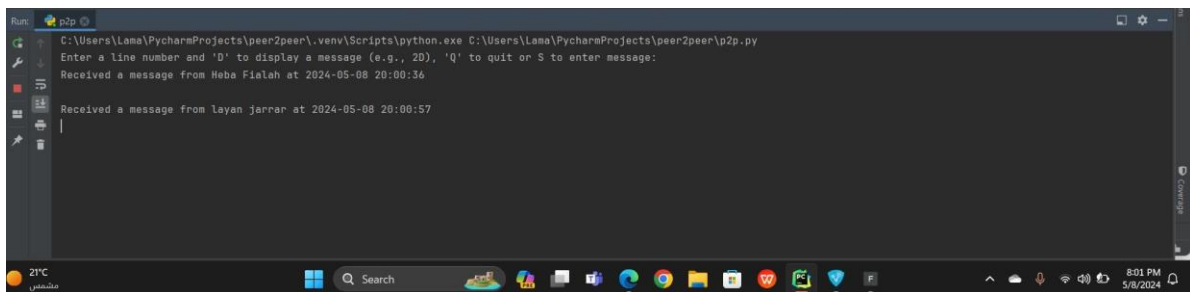


Figure 9 : screenshot for testing the server output

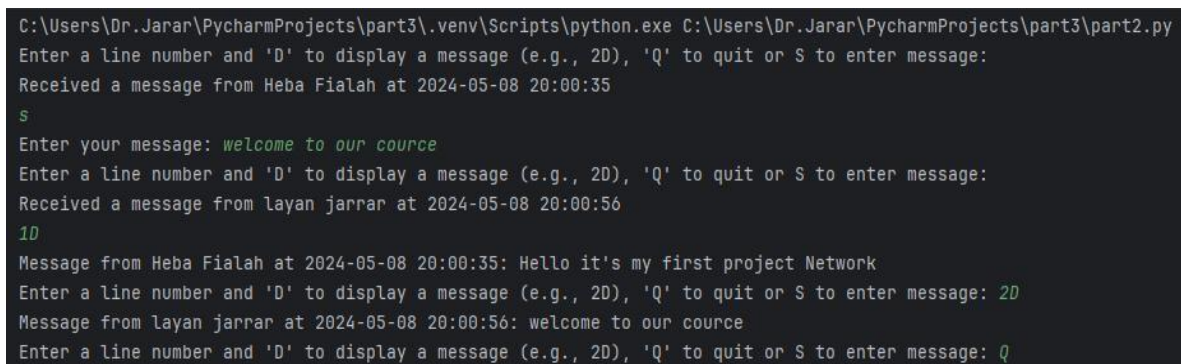


Figure 10 : screenshot for testing the first client output

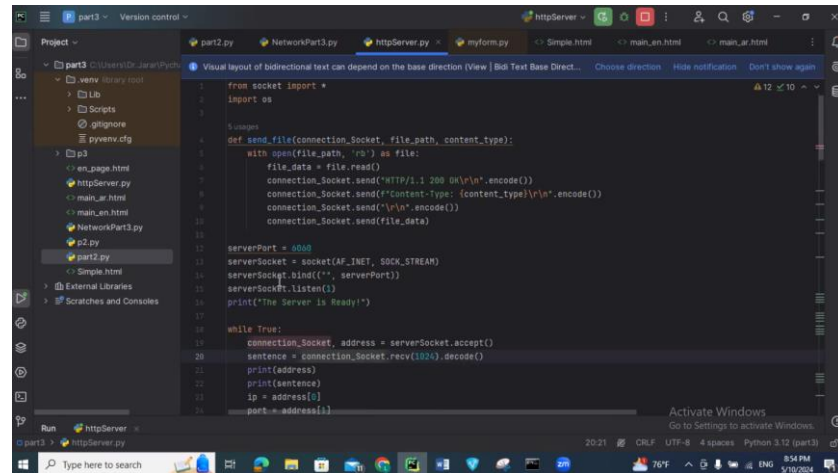
```
Run  part3 x  main x
C:\Users\ASUS\PycharmProjects\pythonProjectphase1\.venv\Scripts\python.exe C:\Users\ASUS\PycharmProjects\pythonProjectphase1\main.py
Enter a line number and 'D' to display a message (e.g., 20), 'Q' to quit or S to enter message: S
Enter your message: Hello it's my first project Network
Enter a line number and 'D' to display a message (e.g., 20), 'Q' to quit or S to enter message:
Received a message from Heba Fialah at 2024-05-08 20:00:35
10
Message from Heba Fialah at 2024-05-08 20:00:35: Hello it's my first project Network
Enter a line number and 'D' to display a message (e.g., 20), 'Q' to quit or S to enter message:
Received a message from layan jarrar at 2024-05-08 20:00:56
|
```

Figure 11 :screenshot for testing the second client output

## Part Three : Web Server

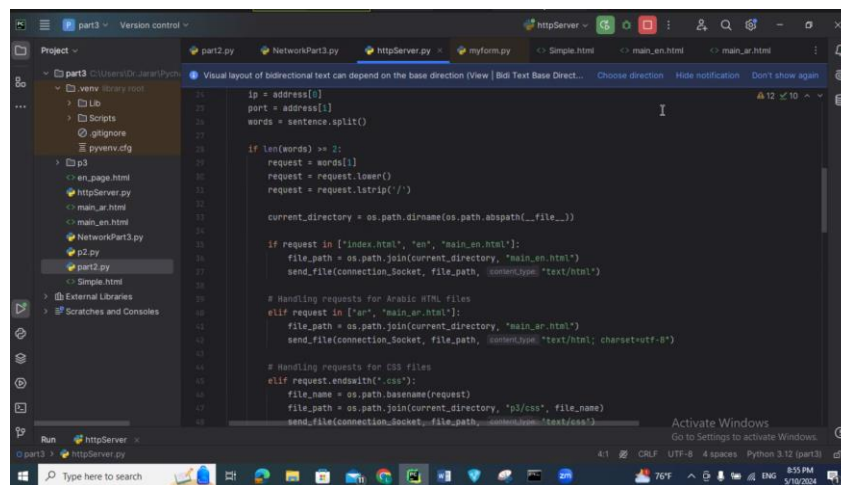
Using socket programming, implement a simple but a complete web server in go, python, java or C that is listening on port 6060.

### Web server code



```
1 from socket import *
2 import os
3
4 def send_file(connection_socket, file_path, content_type):
5     with open(file_path, 'rb') as file:
6         file_data = file.read()
7         connection_socket.send("HTTP/1.1 200 OK\r\n".encode())
8         connection_socket.send("Content-Type: %s\r\n".encode() % content_type)
9         connection_socket.send("\r\n".encode())
10        connection_socket.send(file_data)
11
12 serverPort = 6060
13 serverSocket = socket(AF_INET, SOCK_STREAM)
14 serverSocket.bind(('', serverPort))
15 serverSocket.listen(1)
16 print("The server is ready!")
17
18 while True:
19     connection_socket, address = serverSocket.accept()
20     sentence = connection_socket.recv(1024).decode()
21     print(address)
22     print(sentence)
23     ip = address[0]
24     part = address[1]
```

Figure 12 : web serve (1)



```
25 ip = address[0]
26 part = address[1]
27 words = sentence.split()
28
29 if len(words) >= 2:
30     request = words[1]
31     request = request.lower()
32     request = request.lstrip('/')
33
34     current_directory = os.path.dirname(os.path.abspath(__file__))
35
36     if request in ['index.html', 'en', 'main_en.html']:
37         file_path = os.path.join(current_directory, 'main_en.html')
38         send_file(connection_socket, file_path, content_type="text/html")
39
40     # Handling requests for Arabic HTML files
41     elif request in ['ar', 'main_ar.html']:
42         file_path = os.path.join(current_directory, 'main_ar.html')
43         send_file(connection_socket, file_path, content_type="text/html; charset=utf-8")
44
45     # Handling requests for CSS files
46     elif request.endswith('.css'):
47         file_name = os.path.basename(request)
48         file_path = os.path.join(current_directory, 'p3/css', file_name)
49         send_file(connection_socket, file_path, content_type="text/css")
```

Figure 13 : web server (2)



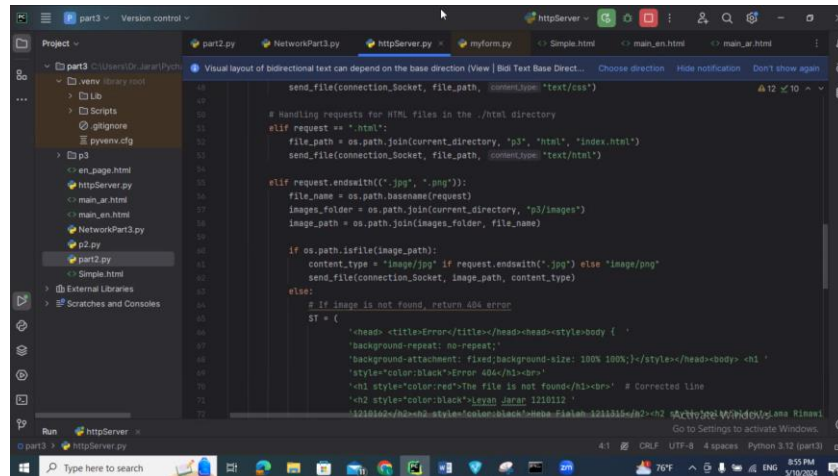


Figure 14 : web server(3)

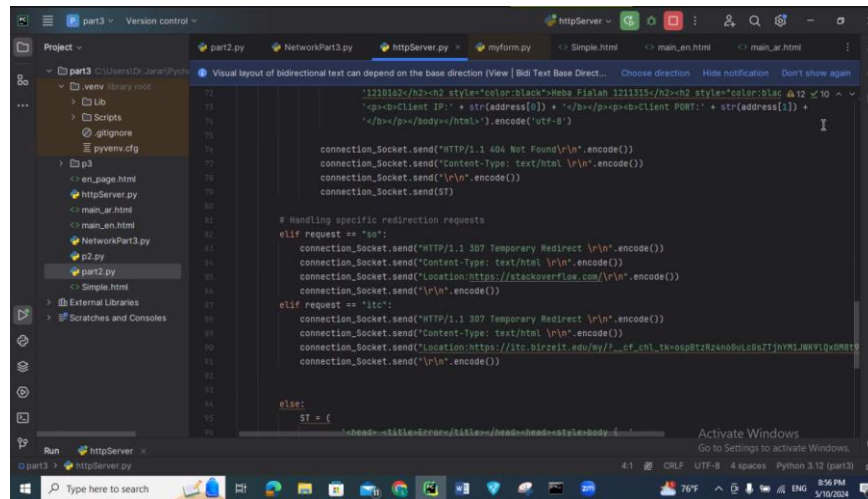


Figure 15 : web server(4)

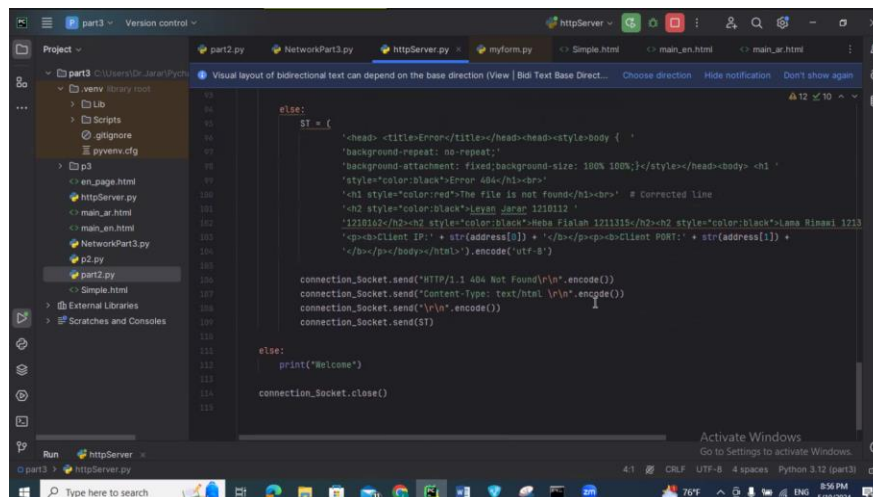


Figure 16 : server code(5)

Explanation: In this part, we use socket programming to create a s web server in python that accepts connections, and listen on port 6060, then it takes requests from client, and gives responses according to these requests. The serve() function reads the content of the file, constructs the HTTP request, then returns the HTTP response. The Handle Requests() function receives client requests, returns the response(depending on the client address), and closes the connection. The Process Requests() function checks the method (HTTP request message) if it is a GET(for sending data to server), extracts the path from the request, and responses with the requested file. The error MSG() function handles the error state, so if the request is wrong, or the file does not exist, it shows the Error page we have designed. The start() function starts listens for incoming connections from the client, and creates a thread for each client.

From rfce2616, what is Entity Tag Cache Validators in the HTTP protocol and why do we need it?

The ETag response-header field value, an entity tag, provides for an "opaque" cache validator. This might allow more reliable validation in situations where it is inconvenient to store modification dates, where the one-second resolution of HTTP date values is not sufficient, or where the origin server wishes to avoid certain paradoxes that might arise from the use of modification dates.

3.1 : If the request is / or /index.html or /main\_en.html or /en (for example localhost:6060/ or localhost:6060/en) then the server should send main\_en.html file with Content-Type: text/html.

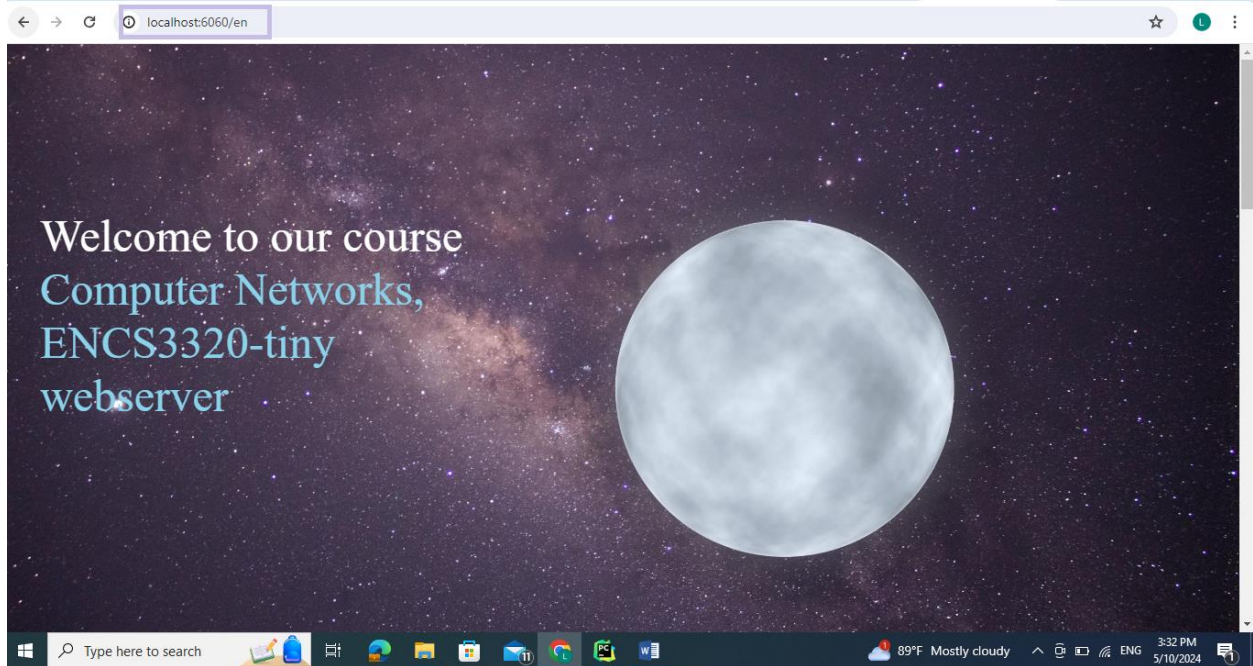


Figure 17 : The HTTP request '/en' redirects the client to our webpage.

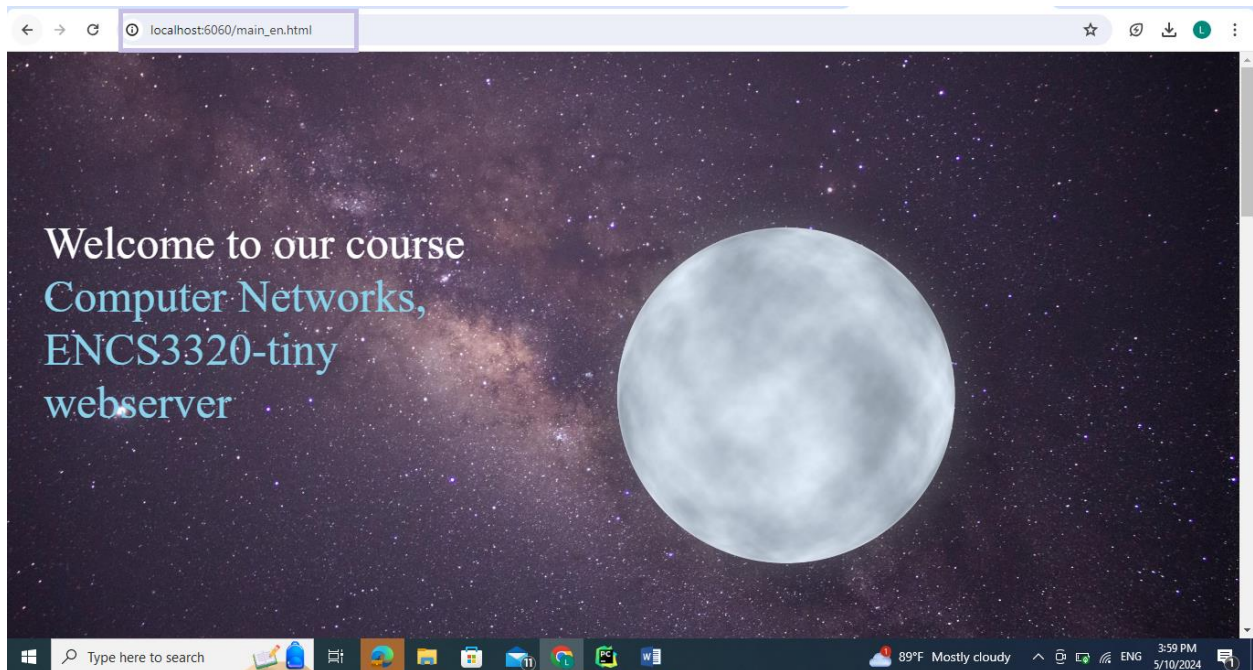


Figure 18 : main\_en



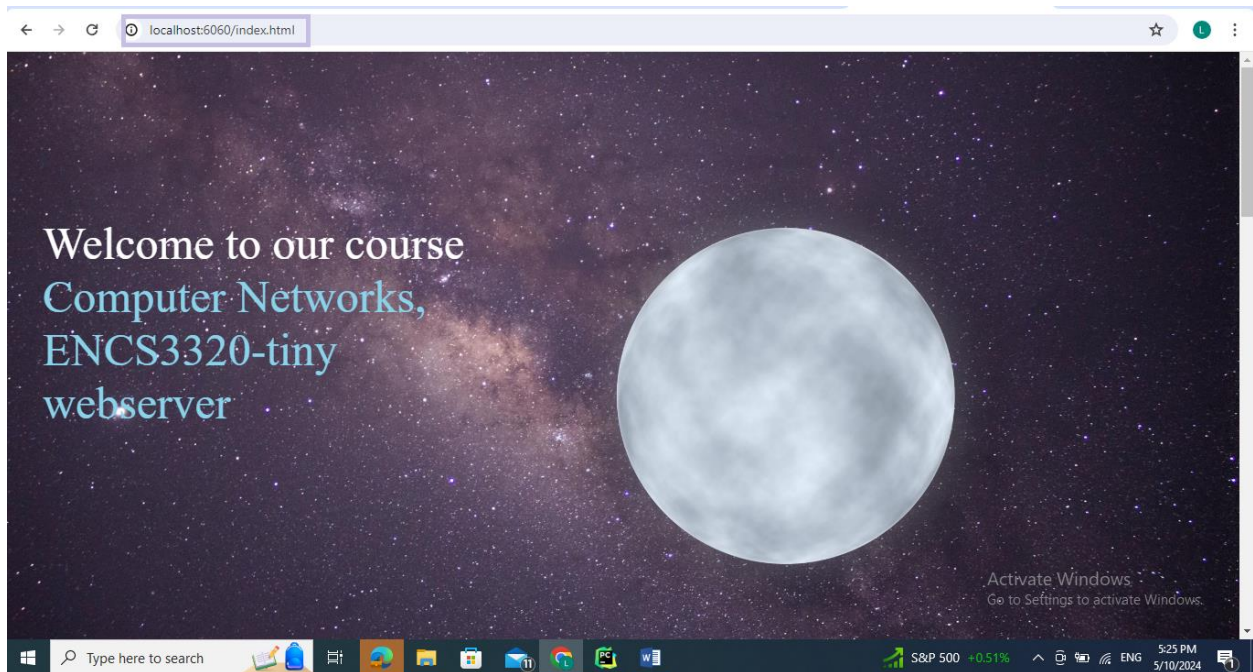


Figure 19 : The HTTP request `'/index.html'` redirects the client to our webpage

```
The Server is Ready!
('127.0.0.1', 60175)
GET /en HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Sec-Purpose: prefetch;prerender
Purpose: prefetch
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408
```

```
( '127.0.0.1', 60229)
GET /main_en.html HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc98d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408
```

*Figure 20 : HTTP request for english*

The server should send the main\_en.html file if the request contains / or /index.html or /main\_en.html or / en . The main html file will be sent to the client if the request is /or main\_ar.html. The server will transmit the header containing the response status that has been encoded. Ultimately, the encoded file that it previously read will be sent by the server.

Explanation: The previous HTTP request messages: (taking main\_en.html as an example)

- The request line: “GET /main\_en.html HTTP/1.1”. GET: The method for sending data to server. /main\_en.html: The URL of the HTML file requested by the client. HTTP/1.1: The version of the protocol used by the client.
- The next line in the terminal shows the IP address and the port number
- The header lines: the rest of the request message are header lines each with a header field name and a value.
- The body: the body is empty.

3.2 : If the request is /ar then the server response with main\_ar.html which is an Arabic version of main\_en.html

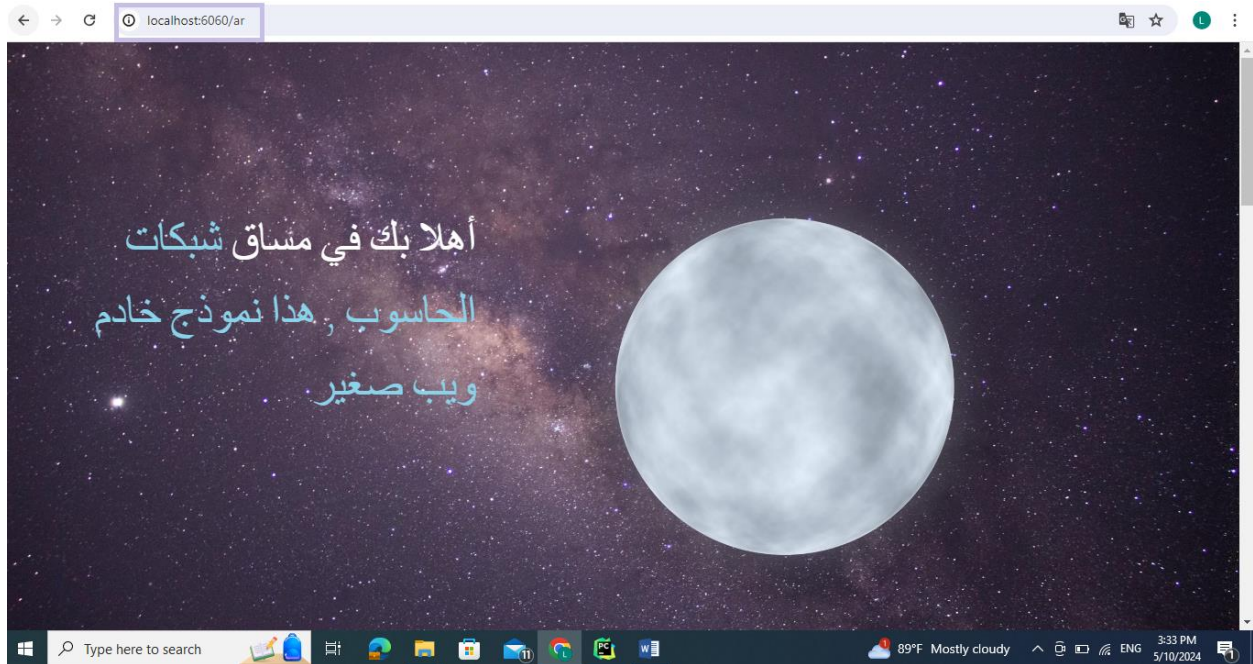


Figure 21 : The HTTP request '/ar' redirects the client to our webpage.

Figure 22 : HTTP request for arabic

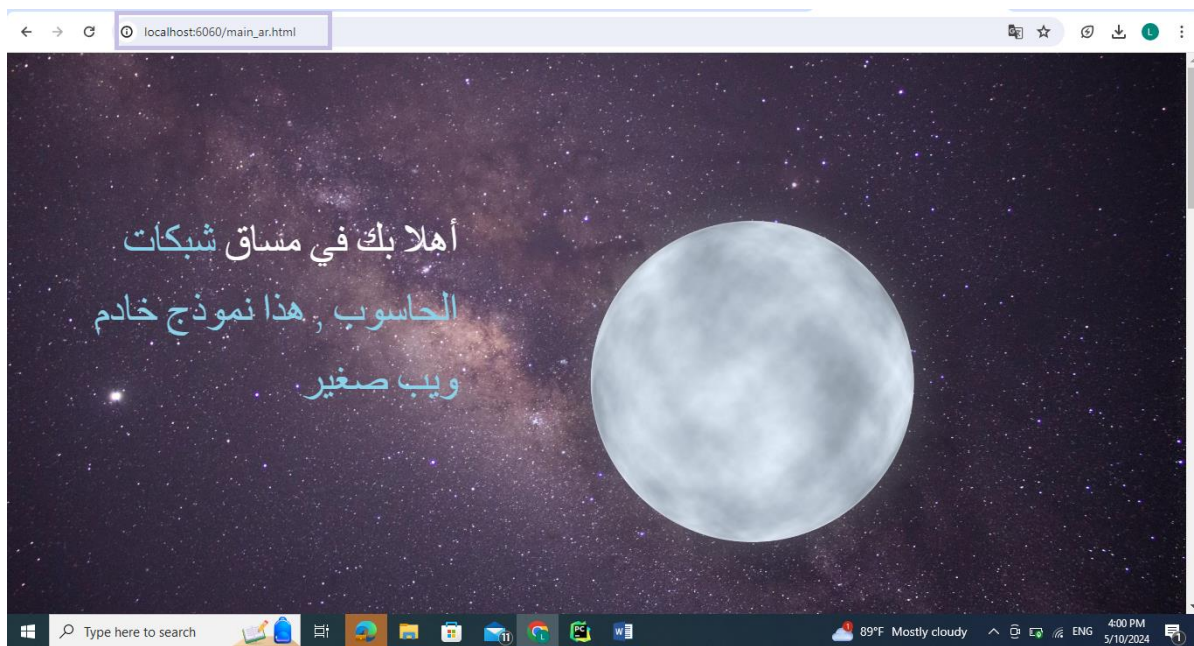


Figure 23 : main\_ar

The HTTP request ‘/main\_ar.html’ redirects the client to our webpage.

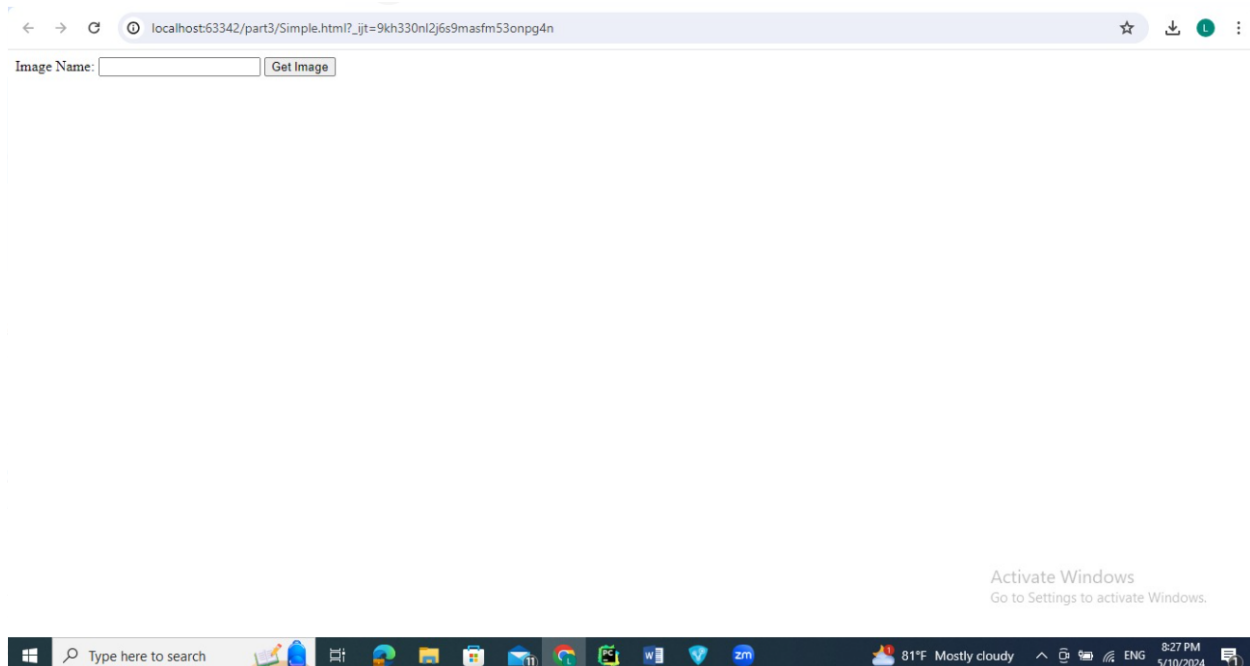
```
('127.0.0.1', 60227)
GET /main_ar.html HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408
```

```
('127.0.0.1', 60178)
GET /ar HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Sec-Purpose: prefetch;prerender
Purpose: prefetch
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408
```

Figure 24 : HTTP request for Arabic



3.3 : If the request is an .html file then the server should send the requested html file with Content-Type: text/html. use any html file.



3.4 : If the request is a .css file then the server should send the requested css file with Content Type: text/css. use any CSS file.

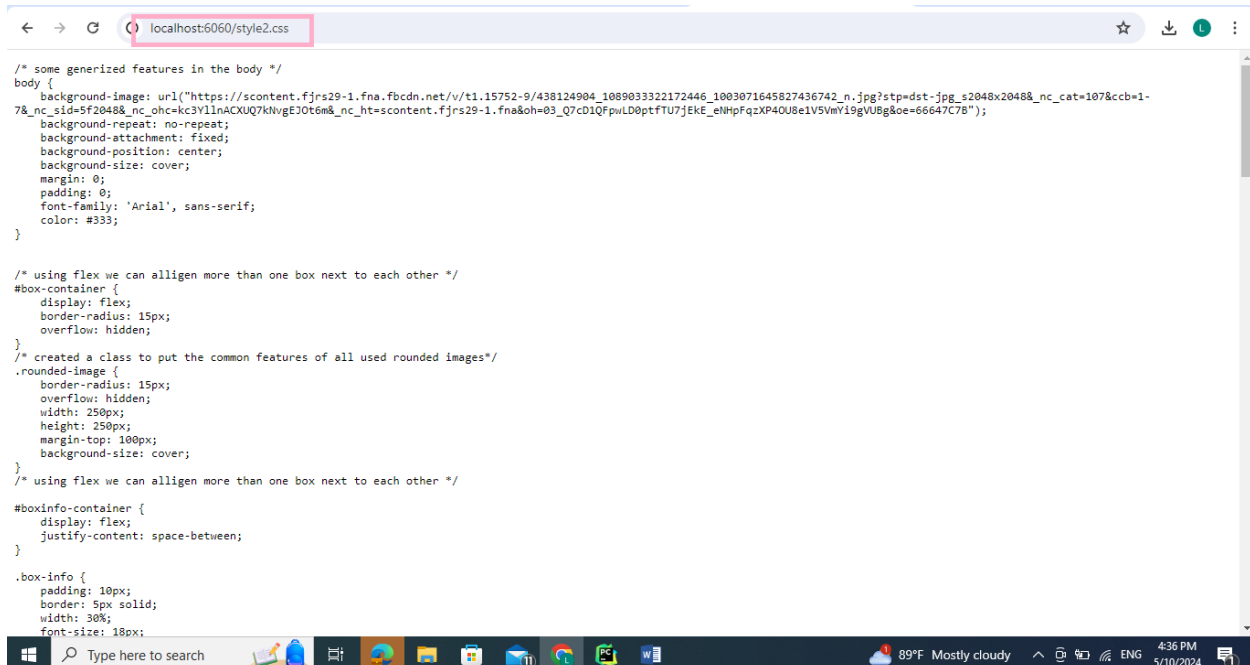


Figure 25 : Style code



```
(('127.0.0.1', 60177)
GET /style2.css HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
Sec-Purpose: prefetch;prerender
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: text/css,*/*;q=0.1
Purpose: prefetch
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://localhost:6060/en
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408
```

Figure 26 : HTTP request for css

3.5 : If the request is a .png then the server should send the png image with Content-Type: image/png. You can use any image.

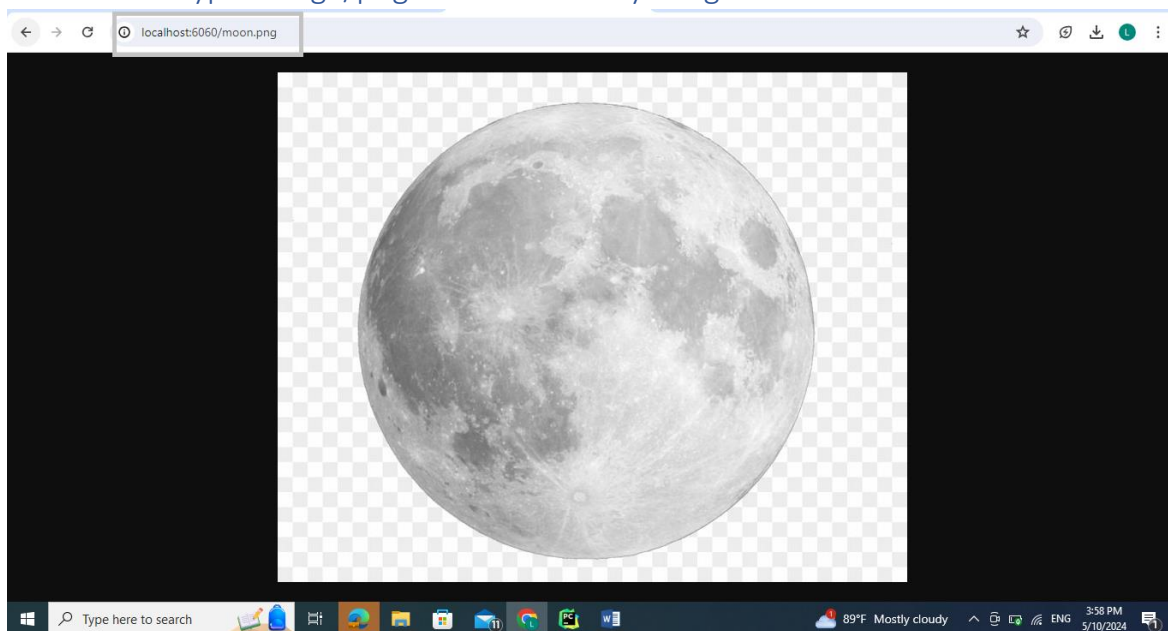


Figure 27 : png image

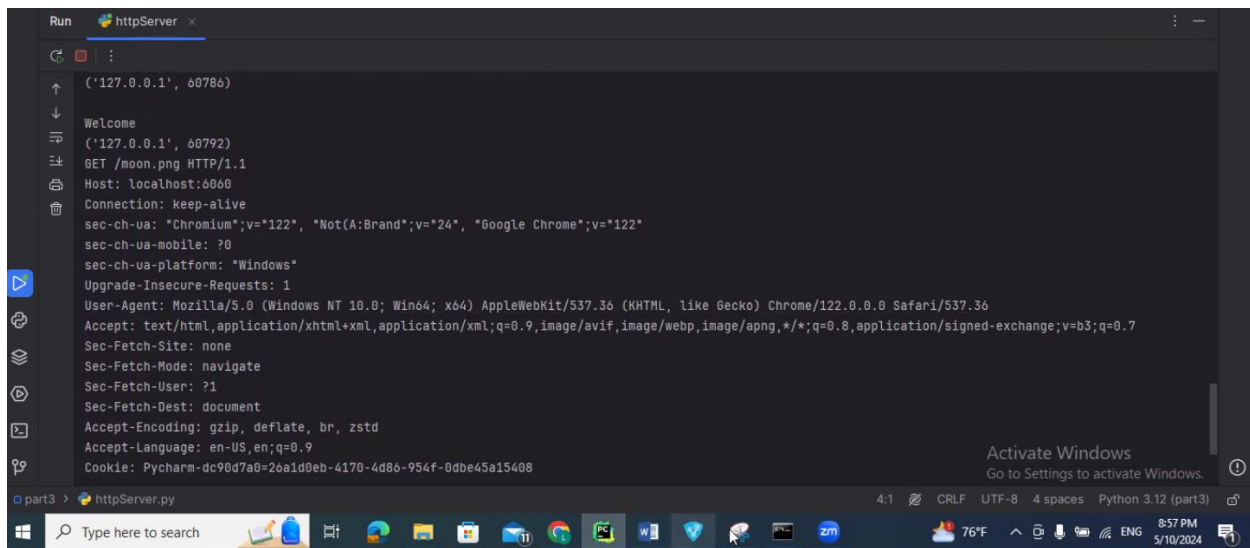


Figure 28 : HTTP request for png

The HTTP request ‘/image.png’ redirects the client to a png image stored on the computer (The same path as the web server).

3.6 : If the request is a .jpg then the server should send the jpg image with Content-Type: image/jpeg. use any image.

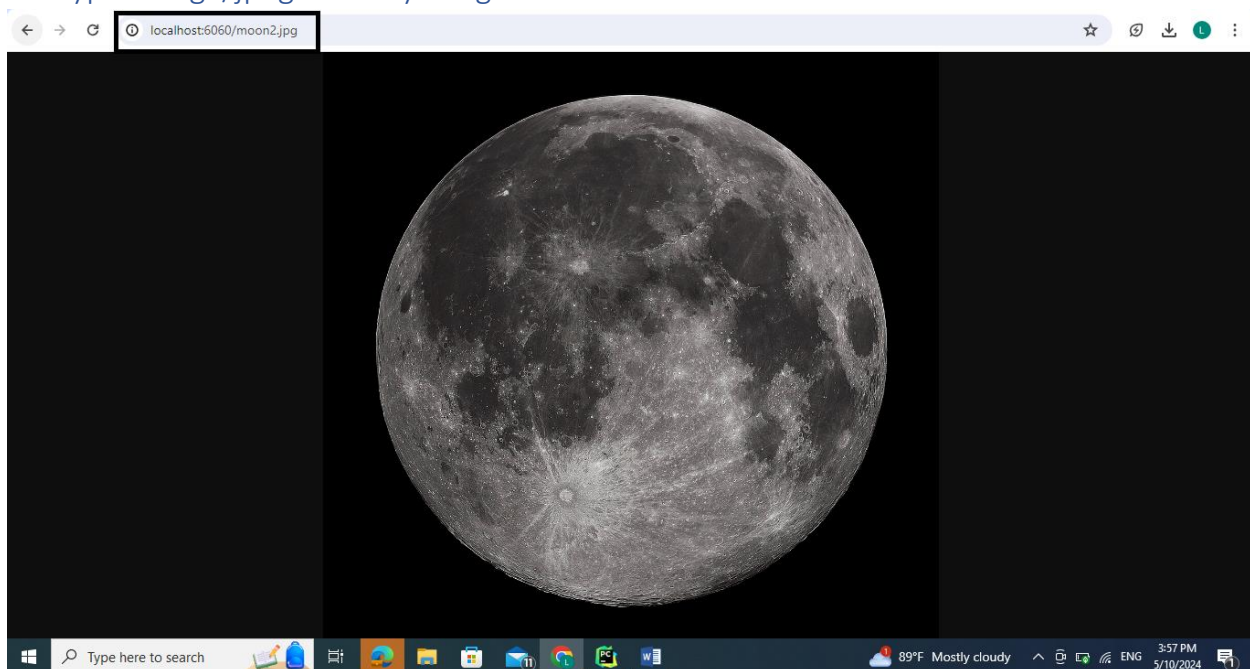
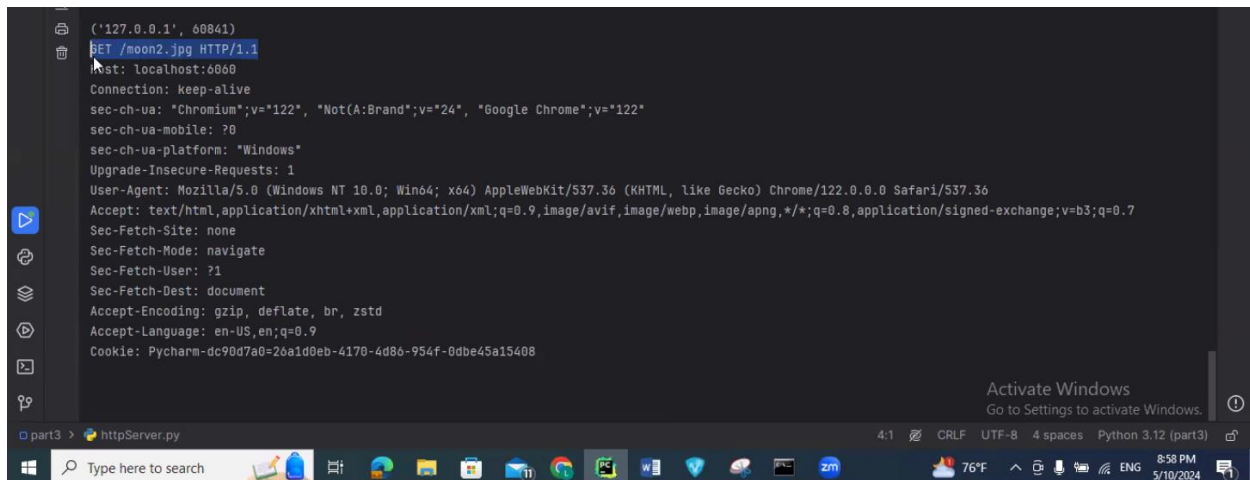


Figure 29 : jpg image



```
(127.0.0.1', 6060)
GET /moon2.jpg HTTP/1.1
Host: localhost:6060
Connection: Keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0-26a1d0eb-4170-4d86-954f-0dbe45a15408
```

Figure 30 : HTTP request for jpg

The HTTP request ‘/image.jpg’ redirects the client to a jpg image stored on the computer (The same path as the web server).

3.7 : Use myform.html to get image by typing the name of the image in a box For instance

3.8 : Use the status code 307 Temporary Redirect to redirect the following

3.8.1 : If the request is /so then redirect to stackoverflow.com website

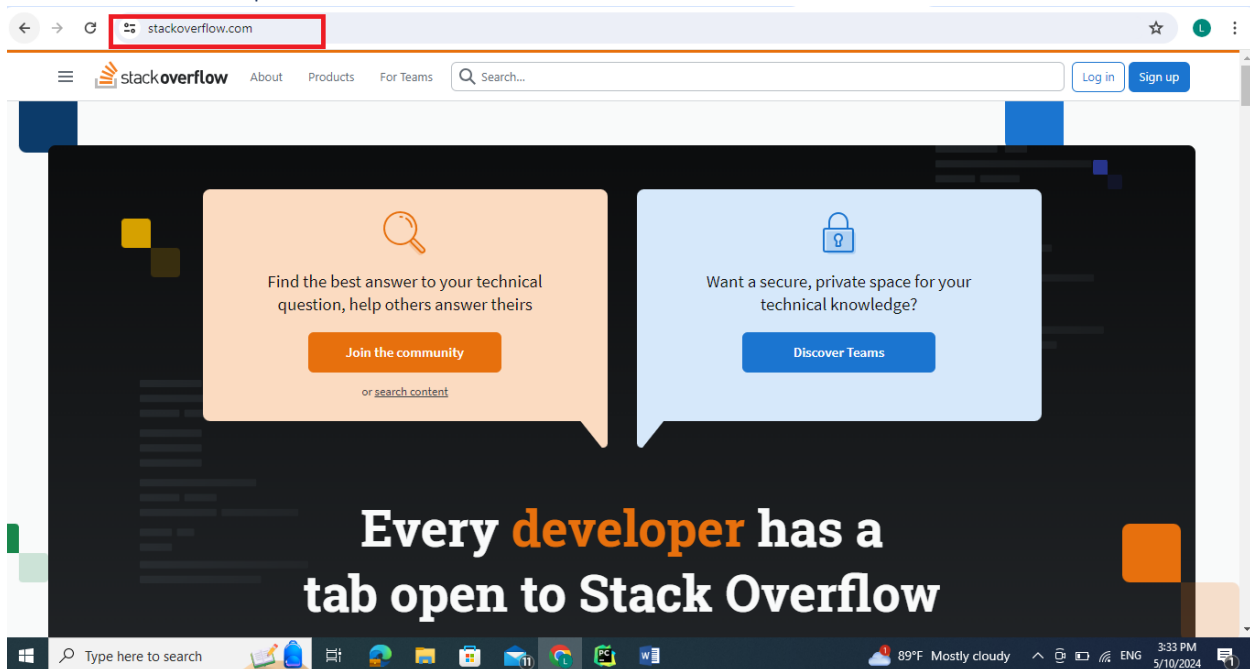


Figure 31 : stackoverflow.com

```

('127.0.0.1', 60182)
GET /so HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Sec-Purpose: prefetch;prerender
Purpose: prefetch
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408

```

Figure 32 : HTTP request for stackoverflow

### 3.8.2 : If the request is /itc then redirect to itc.birzeit.edu website

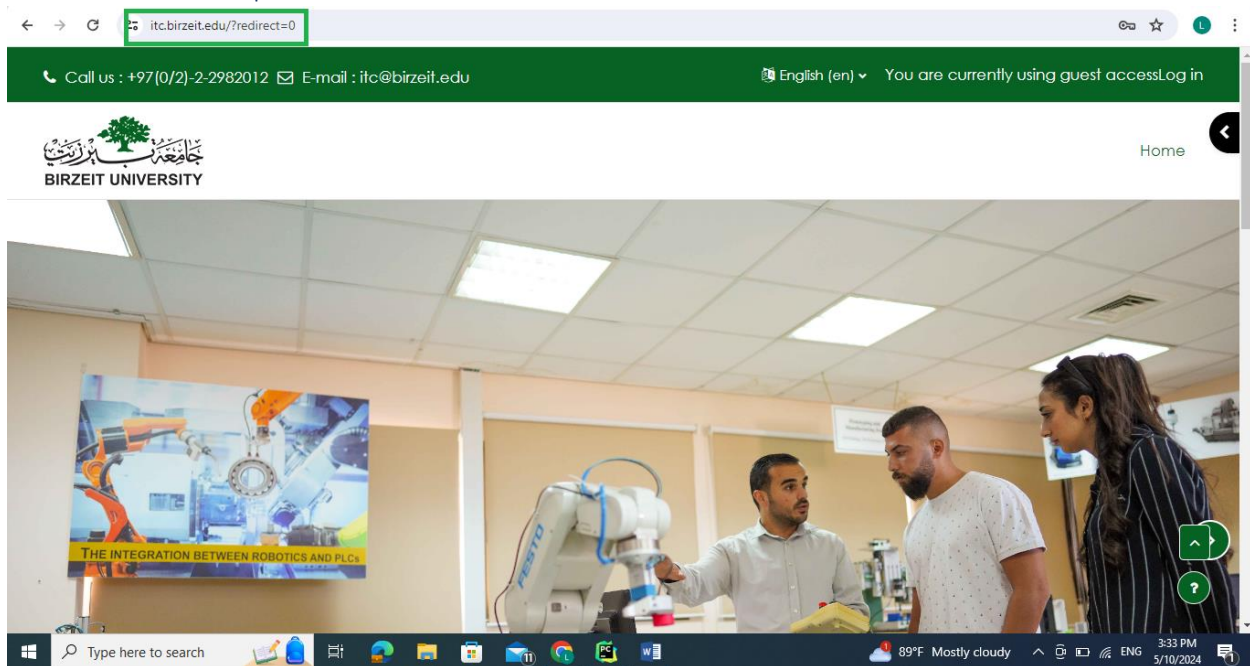


Figure 33 : itc.edu

```

('127.0.0.1', 60206)
GET /itc HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Sec-Purpose: prefetch;prerender
Purpose: prefetch
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408

```

Figure 34 : HTTP request for itc

3.9 : If the request is wrong or the file doesn't exist the server should return a simple HTML webpage that contains (Content-Type: text/html)



**Error 404**

**The file is not found**

**Leyan Jarar 1210112**

**Heba Fialah 1211315**

**Lama Rimawi 1213515**

Client IP:127.0.0.1

Client PORT:64699



Figure 35 : request is wrong or the file doesn't exist



```
('127.0.0.1', 60861)
GET /error HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="122", "Not(A:Brand";v="24", "Google Chrome";v="122"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: Pycharm-dc90d7a0=26a1d0eb-4170-4d86-954f-0dbe45a15408
```

part3 > httpServer.py

Figure 36 : wrong HTTP request

## Web Server in Arabic

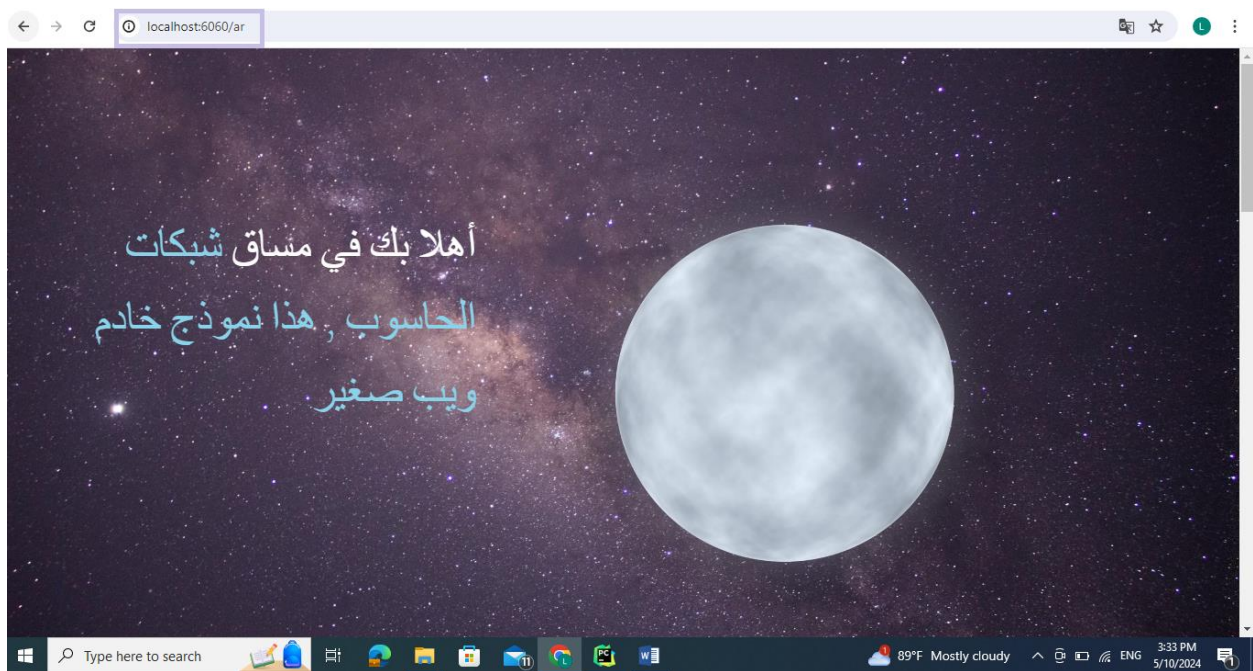


Figure 37 : Arabic webserver (1)

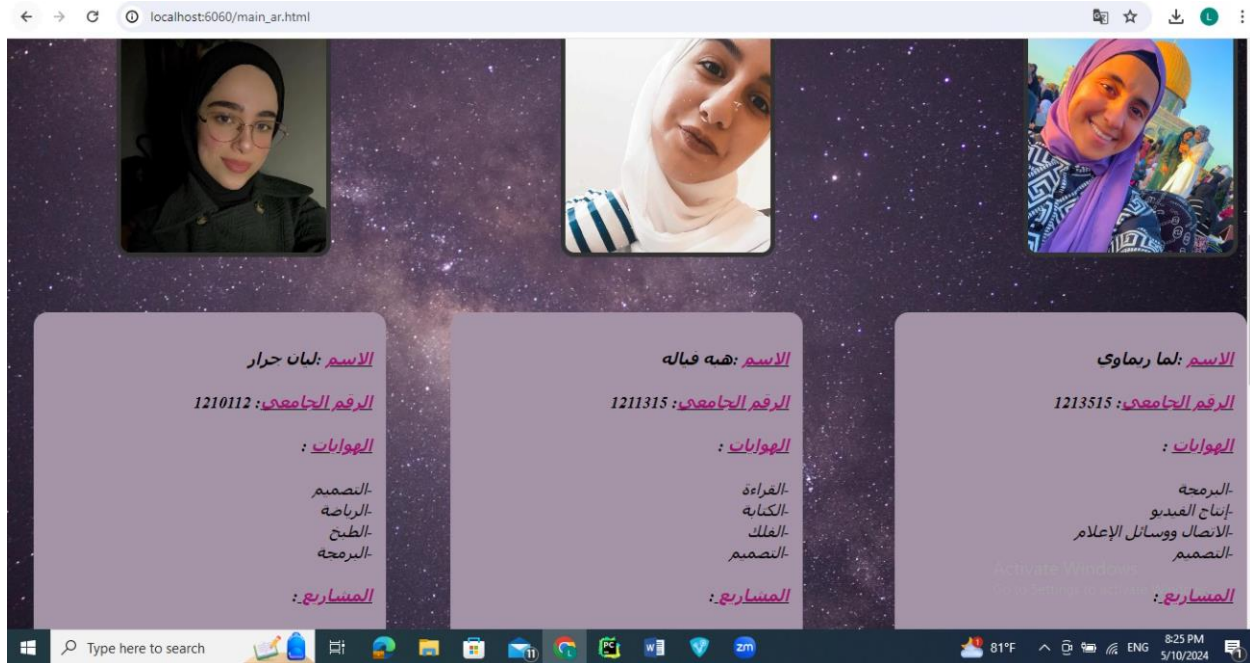


Figure 38 : Arabic webserver(2)

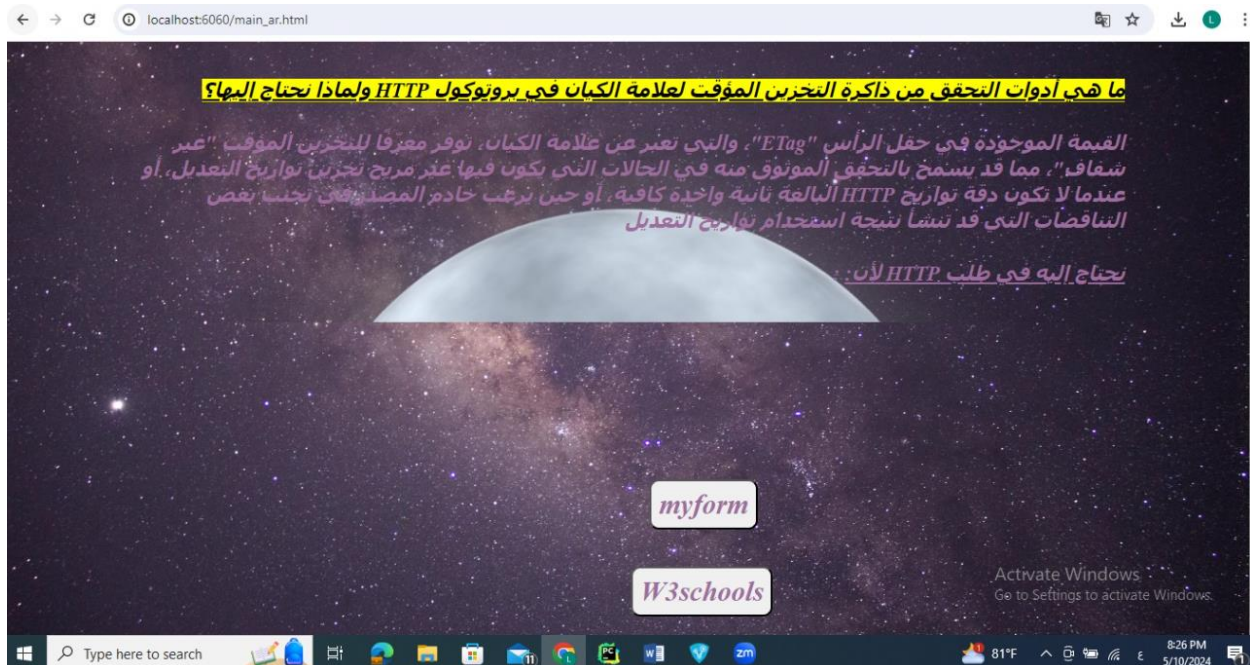


Figure 39 : Arabic webserver(3)



## Web Server in English

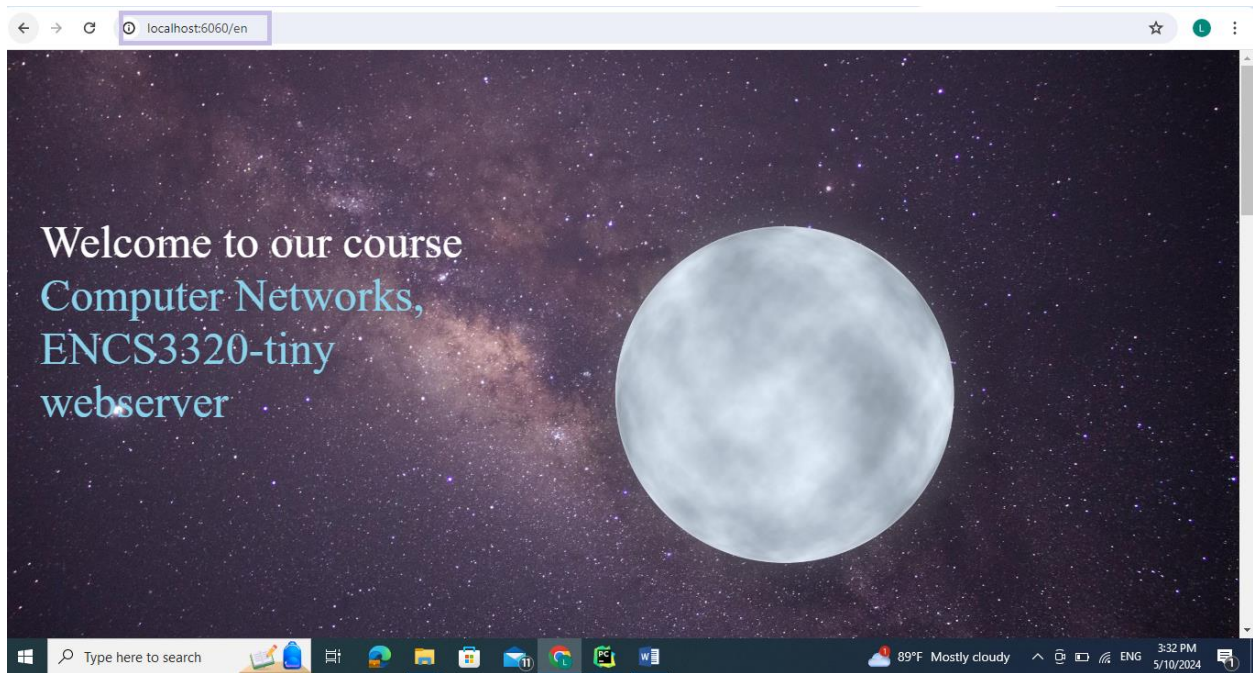


Figure 40 : English webserver(1)

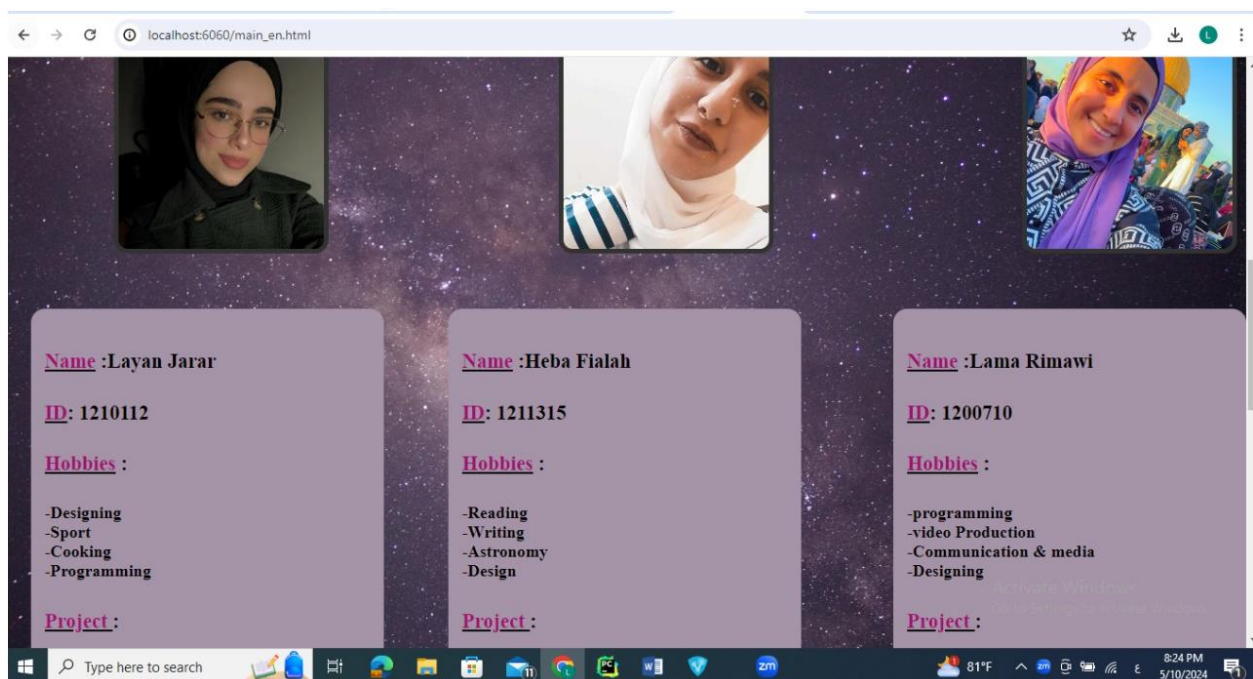


Figure 41 : English webserver (2)



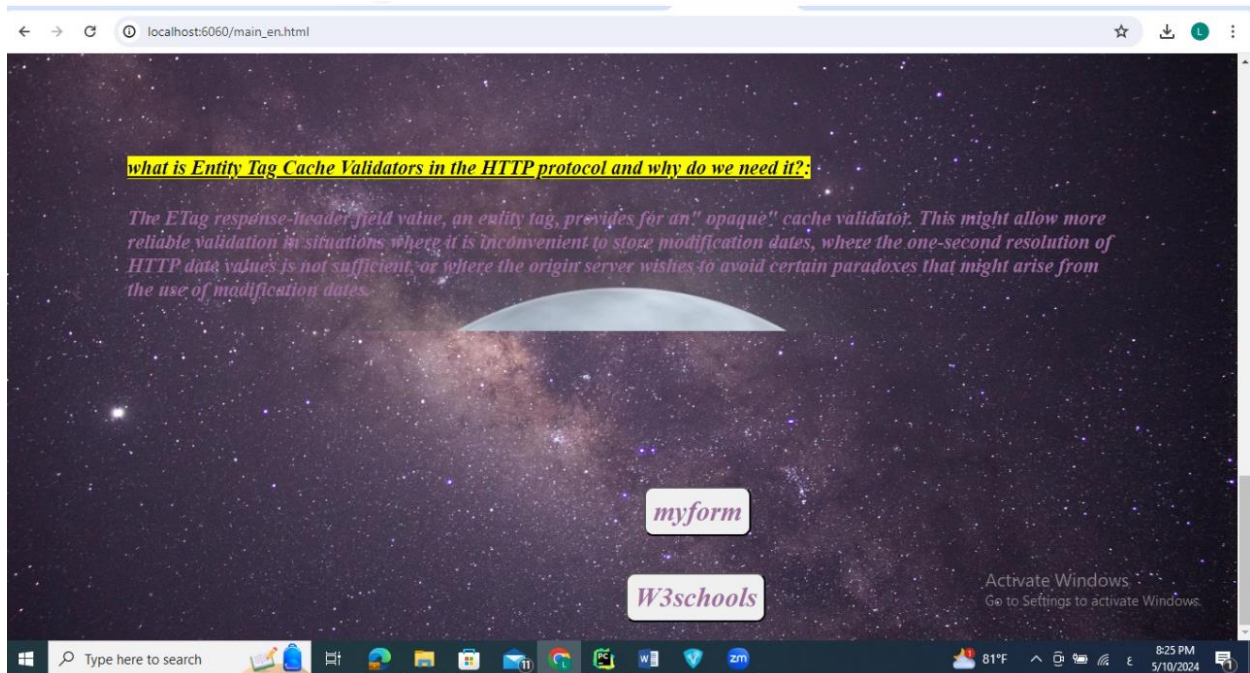


Figure 42 :English webserver(3)