



**Department of Electrical and Computer Engineering**

**Computer Networks**

**ENCS3320**

**Project#1**

***Student name and ID: Mohammed Salem - 1203022***

***Student name and ID: Yousef Eyad – 1201742***

***Section: 4***

***Instructor's name: Dr. Ibrahim Nimer***

- **Part One:**

1. In your own words, what are **ping**, **tracert**, **nslookup**, and **telnet**:

**a) Ping:** Ping is a tool for network troubleshooting that checks if a specific host can be reached across an IP network and measures how long it takes for messages to travel back and forth.

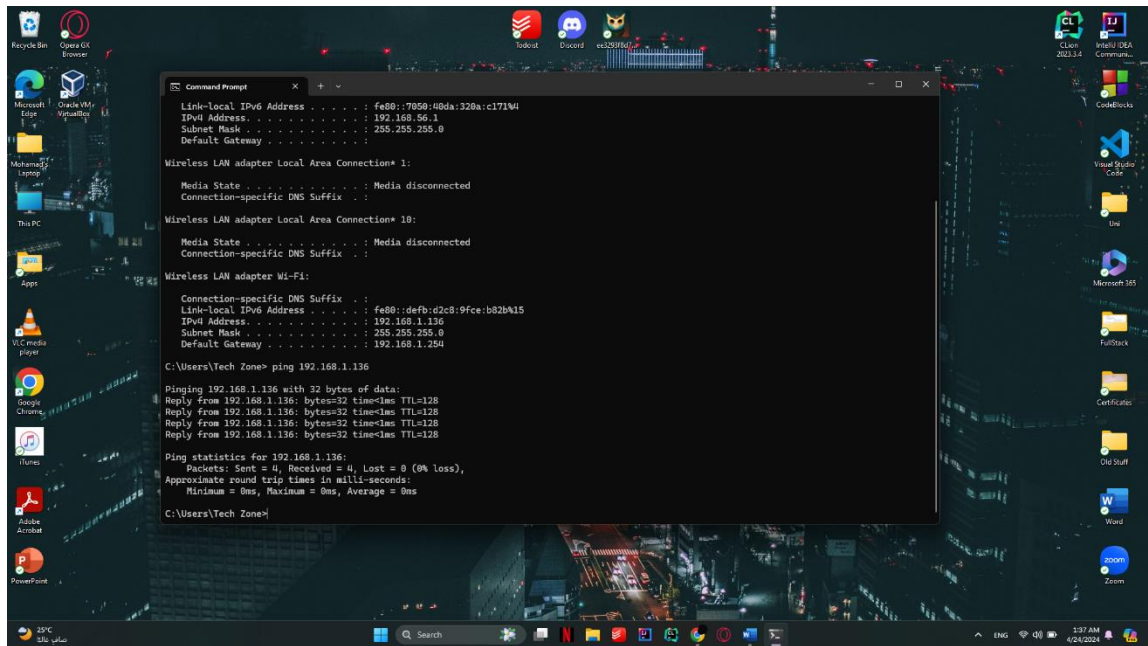
**b) Tracert (Trace Route):** Tracert is a command-line tool that traces the path packets take to a network host, revealing the routers they pass through along the way.

**c) Nslookup (Name Server Lookup):** Nslookup is a utility for querying DNS servers to retrieve domain name or IP address associations, effectively translating hostnames into IP addresses and vice versa.

**d) Telnet:** Telnet is a protocol for remotely accessing another computer, primarily through command-line tasks, although it's considered outdated and less secure compared to SSH for such connections.

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:



The screenshot shows a Windows 10 desktop with a Command Prompt window open. The window displays the following information:

```
Link-local IPv6 Address . . . . . : fe80::7059:40da:320a:c171%4
IPv6 Address. . . . . : 192.168.1.56
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :

Wireless LAN adapter Local Area Connection* 1:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:
Connection-specific DNS Suffix  . :
Link-local IPv6 Address . . . . . : fe80::defb:d2c8:9fce:b02b%15
IPv6 Address. . . . . : 192.168.1.136
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.254

C:\Users\Tech Zone> ping 192.168.1.136

Pinging 192.168.1.136 with 32 bytes of data:
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128

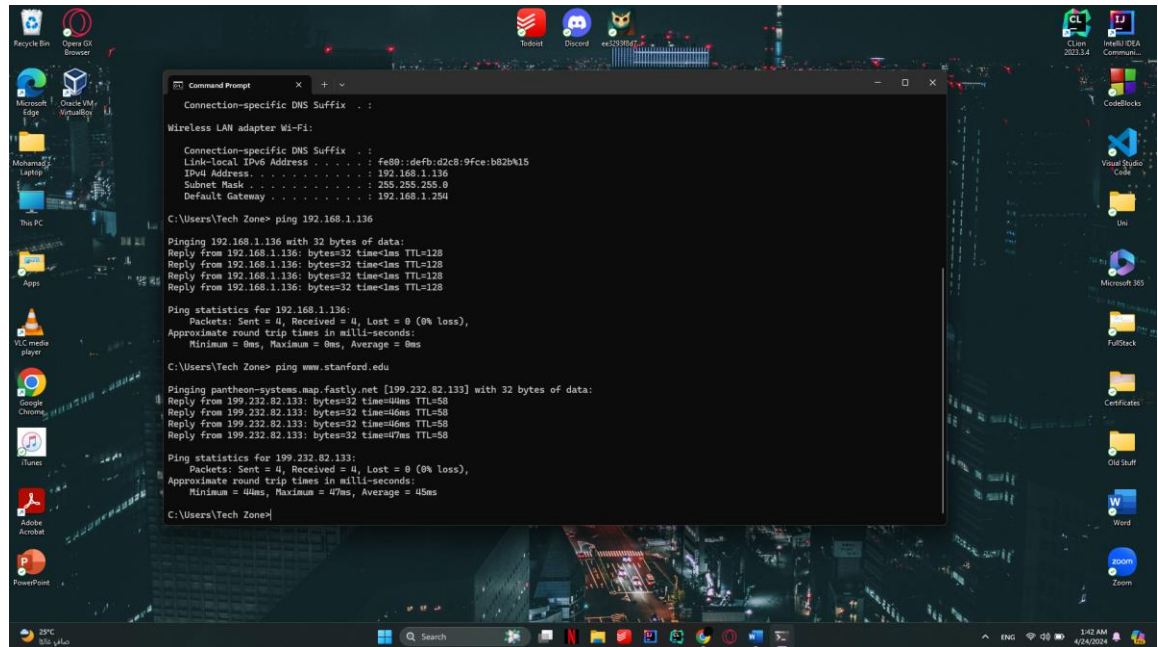
Ping statistics for 192.168.1.136:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Tech Zone>
```

The desktop background is a cityscape at night. The taskbar at the bottom shows the Start button, a search bar, and several pinned applications. The system tray in the bottom right corner shows the date and time as 1:07 AM on 4/24/2020.

This screenshot shows the result of someone using the "ping" command on a Windows PC to check the network connection to the device with the IP address 192.168.1.136. It looks like all the attempts to reach the device were successful because there are replies for each ping sent. Each round trip took about 32 milliseconds, which is pretty good and means the network is responding quickly. The "TTL=128" part is just a default setting that tells you how many more network jumps the ping could make if needed.

b) ping [www.stanford.edu](http://www.stanford.edu)



```
Command Prompt
Connection-specific DNS Suffix . : 
Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix . : 
    Link-local IPv6 Address . . . . . : fe80::defb:d2c8:9fce:b82b%15
    IPv4 Address. . . . . : 192.168.1.136
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.254

C:\Users\Tech Zone> ping 192.168.1.136

Pinging 192.168.1.136 with 32 bytes of data:
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128
Reply from 192.168.1.136: bytes=32 time=1ms TTL=128

Ping statistics for 192.168.1.136:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Tech Zone> ping www.stanford.edu

Pinging pantheon-systems.map.fastly.net [199.232.82.133] with 32 bytes of data:
Reply from 199.232.82.133: bytes=32 time=58ms TTL=58
Reply from 199.232.82.133: bytes=32 time=56ms TTL=58
Reply from 199.232.82.133: bytes=32 time=46ms TTL=58
Reply from 199.232.82.133: bytes=32 time=47ms TTL=58

Ping statistics for 199.232.82.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 44ms, Maximum = 47ms, Average = 45ms

C:\Users\Tech Zone>
```

Here's a screenshot of a Windows Command Prompt after running a "ping" command to check the network response for [www.stanford.edu](http://www.stanford.edu). The command actually resolved to the domain [pantheon-systems.map.fastly.net](http://pantheon-systems.map.fastly.net), which likely means that Stanford's website is using Fastly a content delivery network (CDN). The ping was successful with all four packets sent receiving a reply and it took about 58 milliseconds for each round trip. No packets were lost so the network connection to the server seems to be stable. The average time for the pings is pretty quick which is a good sign if you're trying to visit the website and want it to load fast.

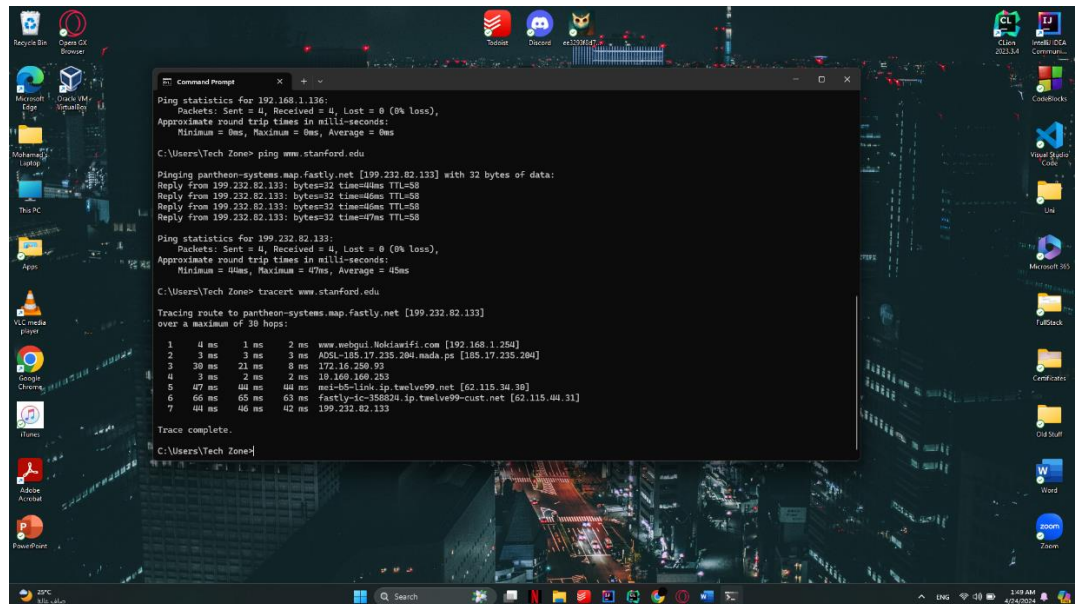
- c) From the ping results, do you think the response you got is from USA?  
Explain your answer briefly.

The ping command results show that `www.stanford.edu` resolved to the IP address `199.232.82.133` and the responses were consistently around 44-47 milliseconds. The "TTL" value is 58 which stands for Time to Live and indicates the number of hops a packet can take before it is discarded.

However from this ping result alone we cannot definitively determine the geographic location of the server IP addresses do not inherently contain location information and while there are databases that map IP ranges to general locations the ping command does not access this information.

Moreover with the presence of content delivery networks (CDNs) like Fastly as indicated by the hostname `pantheon-systems.map.fastly.net` the server responding to your ping could be located at any of Fastly's global network nodes which are designed to serve content from a location nearest to the requester to reduce latency. Considering Stanford University is in the USA it's likely but not guaranteed that the ping response is from a server located in the USA.

- d) tracert [www.stanford.edu](http://www.stanford.edu)



```
Command Prompt
Ping statistics for 199.168.1.254:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Tech Zone> ping www.stanford.edu

Pinging pantheon-systems.map.fastly.net [199.232.82.133] with 32 bytes of data:
Reply from 199.232.82.133: bytes=32 time=44ms TTL=58
Reply from 199.232.82.133: bytes=32 time=47ms TTL=58
Reply from 199.232.82.133: bytes=32 time=45ms TTL=58
Reply from 199.232.82.133: bytes=32 time=46ms TTL=58

Ping statistics for 199.232.82.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 44ms, Maximum = 47ms, Average = 45ms

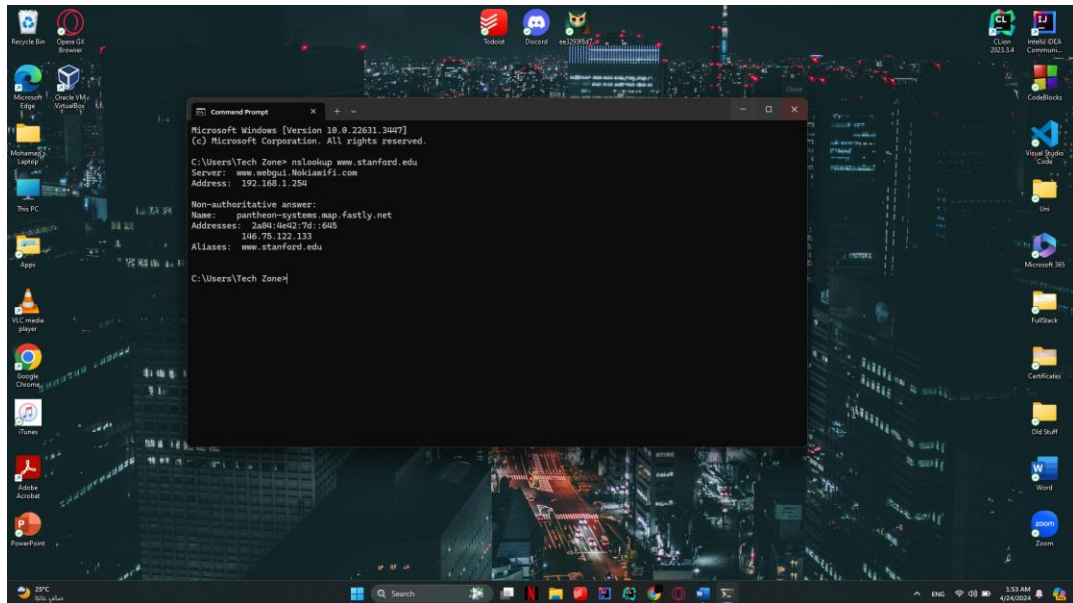
C:\Users\Tech Zone> tracert www.stanford.edu

Tracing route to pantheon-systems.map.fastly.net [199.232.82.133]
over a maximum of 30 hops:
  0  0 ms  0 ms  0 ms  www.mugui.Mediacraft.com [192.168.1.254]
  1  3 ms  3 ms  3 ms  ADSL-165-17.235.204.mada.ps [165.17.235.204]
  2  30 ms  21 ms  8 ms  172.16.250.93
  3  3 ms  2 ms  2 ms  19.169.160.253
  4  47 ms  44 ms  44 ms  msi-46-16-line.ip.twelve99.net [62.115.34.30]
  5  66 ms  65 ms  63 ms  fastly-ic-358824.ip.twelve99-cust.net [62.115.44.31]
  6  44 ms  46 ms  42 ms  199.232.82.133
Trace complete.

C:\Users\Tech Zone>
```

This is a traceroute result, which shows the path that packets from your computer take to get to Stanford's website. The site is again using Fastly and you can see each "hop" along the way with the time it takes to get from one point to the next. The times are all pretty low which is good. It means there's not much delay. It's like getting a peek at the roadmap of the internet as data zips from your place to the server hosting the Stanford site.

e) nslookup www.stanford.edu



```
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Tech Zone> nslookup www.stanford.edu
Server: www.webui.hoklawifi.com
Address: 192.168.1.254

Non-authoritative answer:
Name: pantheon-systems.map.fastly.net
Addresses: 2a84:4e42:7d::645
          146.78.122.133
Aliases: www.stanford.edu

C:\Users\Tech Zone>
```

This is the output of an nslookup command. It's showing the details for Stanford University's website. My computer asked a local DNS server probably My WiFi router given the private IP address to find where "www.stanford.edu" is on the internet. It came back with an address that's managed by Fastly the content delivery network. This kind of information helps computers figure out where to send their information when you're trying to visit a website. Also, it's saying that the answer isn't the final word on where Stanford.edu is located hence the 'non-authoritative answer'. It's like asking for directions and getting a point to the nearest signpost instead of the full path to your destination.

## • Part Two:

Using socket programming, implement UDP client and server applications in go, python, java or C. The server should listen on port 5051.

```
1 import socket
2 import threading
3 import time
4
5 # Configuration settings for the UDP server
6 localIP = "0.0.0.0" # Listens on all network interfaces
7 localPort = 5051 # Port number to listen on
8 bufferSize = 1024 # Buffer size for receiving data
9 broadcastIP = "192.168.88.255" # Broadcast IP address for sending messages
10
11 # Create a UDP socket
12 UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
13
14 # Enable broadcasting mode on the socket
15 UDPServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
16
17 # Bind the socket to the specified IP and port
18 UDPServerSocket.bind((localIP, localPort))
19 print("UDP server up and listening on port 5051")
20
21 # List to store received messages
22 messages = []
23
24 # Function to handle receiving messages
25 def receiveMessages():
26     while True:
27         bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
28         message = bytesAddressPair[0]
29         address = bytesAddressPair[1]
30         sender = f'{address[0]}'
31         received_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())
32
33         # Decode the received message
34         message = message.decode('utf-8')
35
36         # Append message to the messages list with sender and timestamp
37         messages.append((sender, received_time, message))
38
39         # Print received message details
40         print(f"Received a message from {sender} at {received_time}")
41         for idx, msg in enumerate(messages):
42
43
44 # Function to handle sending messages and processing commands
45 def sendMessages():
46     while True:
47         input_str = input("Enter your first name, last name, and message or command: ")
48         if 'D' in input_str:
49             # Display the detail of a specific message if 'D' is included in the input
50             line_number = int(input_str.replace('D', ' ').split(' ')) - 1
51             if 0 <= line_number < len(messages):
52                 sender, time_received, message = messages[line_number]
53                 print(f"Detail of message {line_number + 1}: From {sender} at {time_received}: '{message}'")
54             else:
55                 print("Invalid message number.")
56         else:
57             # Send a message if no command is included
58             try:
59                 firstName, lastName, msg = input_str.split(maxsplit=2)
60                 fullMessage = f'{firstName} {lastName}: {msg}'
61                 UDPServerSocket.sendto(fullMessage.encode(), (broadcastIP, localPort))
62             except ValueError:
63                 print("Error: Please enter both your first name, last name and a message.")
64
65 # Create threads for receiving and sending messages
66 receiveThread = threading.Thread(target=receiveMessages)
67 sendThread = threading.Thread(target=sendMessages)
68
69 # Start both threads
70 receiveThread.start()
71 sendThread.start()
72
73 # Prevent the main thread from exiting by joining the child threads
74 receiveThread.join()
75 sendThread.join()
76
```

The provided Python script is designed as a simple UDP-based chat application that enables each instance to function both as a client and a server. This dual functionality allows users to send and receive messages across a local network, making it ideal for environments like classrooms or small offices where quick and easy communication is necessary.

### **Key Features and Functionality:**

The script uses Python's ``socket`` library to manage network communications. It sets up a UDP socket that broadcasts messages, meaning that any message sent by one user is received by all users within the same subnet. This is achieved by binding the socket to ``0.0.0.0``, which allows the program to accept messages on all network interfaces of the host machine. The socket is also configured to listen on port 5051 and to allow broadcasting, which is crucial for sending messages to multiple recipients simultaneously.

Messages are handled through two main functions: one for sending and one for receiving. Users can send messages by inputting their first and last names followed by their message. This input is then broadcast to all peers. Concurrently, the script listens for incoming messages, which, upon receipt, are decoded, timestamped, and stored. Each received message is displayed on the console along with the sender's information and the time it was received.

### **Interactive User Experience:**

To enhance interactivity, the script includes threading capabilities using Python's ``threading`` library. This allows the script to handle sending and receiving messages simultaneously, providing a seamless user experience. Users can interact with the script through a command-line interface where they can also execute commands to view detailed information about specific messages. For instance, typing a line number followed by 'D' (e.g., "2D") retrieves detailed information about a particular message, including the sender's IP address and the exact time it was received.

### **Practical Application and Use:**

This chat application is straightforward and lightweight, making it a practical solution for internal communications where setting up a more complex system might be unnecessary. It leverages the simplicity of UDP broadcasting for message distribution, ensuring that setup and operation are both uncomplicated and accessible even to users with minimal technical background.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Tech Zone\Networks\Project\Network\Project-PartTwo> python UDPPeer.py
UDP server up and listening on port 5051
Enter your first name, last name, and message or command: mhmd salem hi
Received a message from 192.168.88.17 at 2024-05-09 15:13:23
1- Received a message from 192.168.88.17 at 2024-05-09 15:13:23
Enter your first name, last name, and message or command: Received a message from 192.168.88.16 at 2024-05-09 15:13:25
1- Received a message from 192.168.88.17 at 2024-05-09 15:13:23
2- Received a message from 192.168.88.16 at 2024-05-09 15:13:25
mhmd salem twoo
Received a message from 192.168.88.17 at 2024-05-09 15:13:50
1- Received a message from 192.168.88.17 at 2024-05-09 15:13:23
2- Received a message from 192.168.88.16 at 2024-05-09 15:13:25
2- Received a message from 192.168.88.17 at 2024-05-09 15:13:50
Enter your first name, last name, and message or command: Received a message from 192.168.88.16 at 2024-05-09 15:14:01
1- Received a message from 192.168.88.17 at 2024-05-09 15:13:23
2- Received a message from 192.168.88.16 at 2024-05-09 15:13:25
3- Received a message from 192.168.88.17 at 2024-05-09 15:13:50
4- Received a message from 192.168.88.16 at 2024-05-09 15:14:01
```

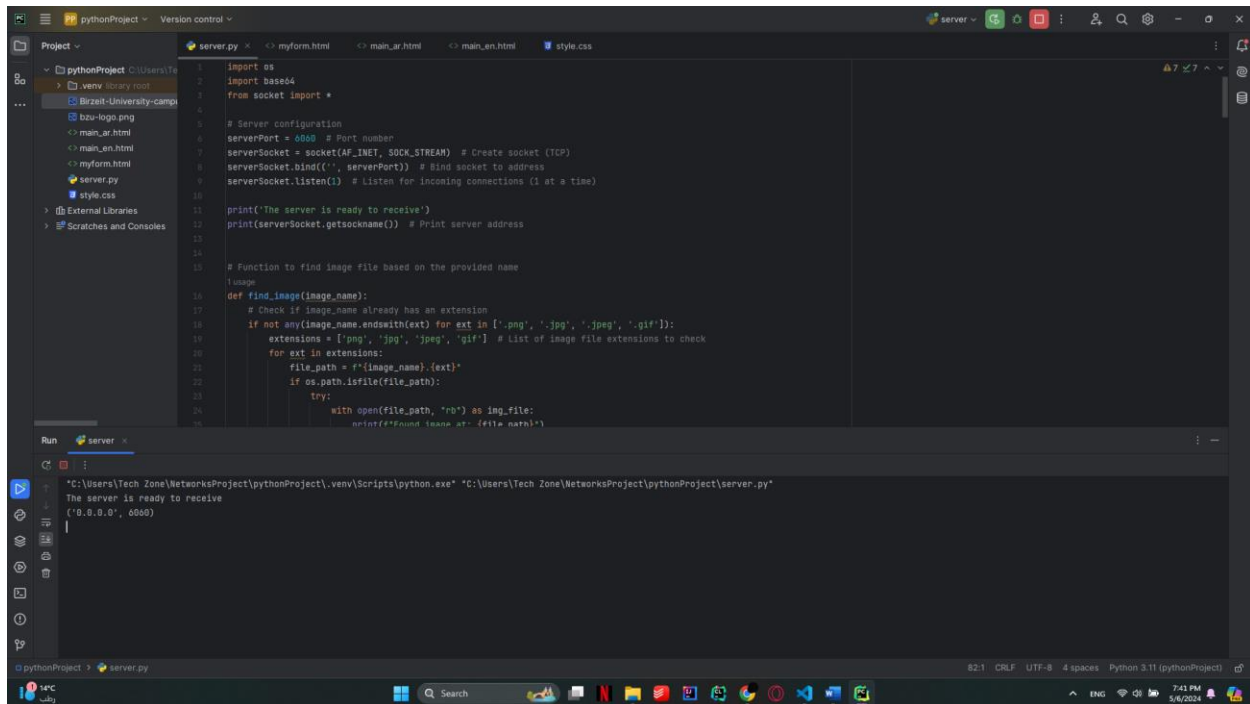
The image shows a Windows PowerShell window from the first user "Mohammed Salem" where I was running a Python script called "UDPPeer.py". The script sets up a UDP server on port 5051 and displays interactions between users on the network. In the session, we can see that the user "mhmd salem" sending a message, "hi". The console logs confirm that messages are being sent and received between computers with IP addresses "192.168.88.17" and "192.168.88.16", showing the chat application is working as expected in facilitating real-time communication across the network.

```
UDP server up and listening on port 5051
Enter your first name, last name, and message or command: Received a message from 192.168.88.17 at 2024-05-09 15:14:02
1- Received a message from 192.168.88.17 at 2024-05-09 15:14:02
yousef eyad halo
Received a message from 192.168.88.16 at 2024-05-09 15:14:14
1- Received a message from 192.168.88.17 at 2024-05-09 15:14:02
2- Received a message from 192.168.88.16 at 2024-05-09 15:14:14
Enter your first name, last name, and message or command: Received a message from 192.168.88.17 at 2024-05-09 15:14:29
1- Received a message from 192.168.88.17 at 2024-05-09 15:14:02
2- Received a message from 192.168.88.16 at 2024-05-09 15:14:14
3- Received a message from 192.168.88.17 at 2024-05-09 15:14:29
yousef eyad three
Received a message from 192.168.88.16 at 2024-05-09 15:14:39
1- Received a message from 192.168.88.17 at 2024-05-09 15:14:02
2- Received a message from 192.168.88.16 at 2024-05-09 15:14:14
3- Received a message from 192.168.88.17 at 2024-05-09 15:14:29
4- Received a message from 192.168.88.16 at 2024-05-09 15:14:39
Enter your first name, last name, and message or command: |
```

This image shows a Windows PowerShell window from the second user "Yousef Eyad" displaying a chat application in action. "Yousef Eyad," sends a greeting message, "halo." The program records and shows this message along with others from different network addresses. Yousef also send another message by "three." The window clearly lists all the messages sent and received, demonstrating the program's ability to handle chat data in real-time on a local network.

- **Part Three:**

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



```
1 import os
2 import base64
3 from socket import *
4
5 # Server configuration
6 serverPort = 6060 # Port number
7 serverSocket = socket(AF_INET, SOCK_STREAM) # Create socket (TCP)
8 serverSocket.bind(('', serverPort)) # Bind socket to address
9 serverSocket.listen(1) # Listen for incoming connections (1 at a time)
10
11 print('The server is ready to receive')
12 print(serverSocket.getsockname()) # Print server address
13
14 # Function to find image file based on the provided name
15 image
16 def find_image(image_name):
17     # Check if image_name already has an extension
18     if not any(image_name.endswith(ext) for ext in ['.png', '.jpg', '.jpeg', '.gif']):
19         extensions = ['.png', '.jpg', '.jpeg', '.gif'] # List of image file extensions to check
20         for ext in extensions:
21             file_path = f'{image_name}.{ext}'
22             if os.path.isfile(file_path):
23                 try:
24                     with open(file_path, 'rb') as img_file:
25                         print(f'Found image at: {file_name}')
```

Run server

```
"C:\Users\Tech Zone\NetworksProject\pythonProject\venv\Scripts\python.exe" "C:\Users\Tech Zone\NetworksProject\pythonProject\server.py"
The server is ready to receive
('0.0.0.0', 6060)
```

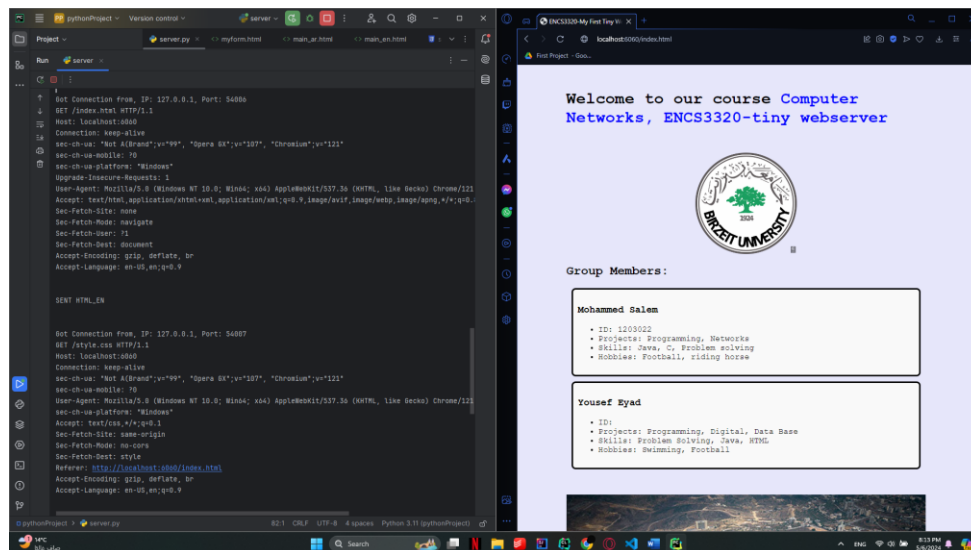
Let's get started by creating a basic TCP server that can connect to clients over the internet, specifically using IPv4. We'll use the “**socket**” library for handling connections, and we'll also bring in some tools from the “**os**” and “**base64**” libraries for managing system tasks and working with URLs. Our server will be set up to listen for connections on port 6060. Once we've connected the server to this port, we'll start listening for incoming connections. When a connection comes in, the server will print "The server is ready to receive" to let us know it's ready to handle client requests.

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; charset=UTF-8

```
The server is ready to receive
('0.0.0.0', 6060)
Got Connection from IP: 127.0.0.1, Port: 53936
GET / HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Not A(Brand";v="99", "Opera GX";v="107", "Chromium";v="121"
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/121.0.0.0 Safari/537.36 OPR/107.0.0.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

SENT HTML_EN

Got Connection from IP: 127.0.0.1, Port: 53941
```



When a client requests a path such as `/`, `/index.html`, `/main_en.html`, or `/en`, the server responds by retrieving the `"main_en.html"` file. It then sets the `"Content-Type"` HTTP header to `"text/html; charset=UTF-8"`, indicating that the content is HTML encoded in UTF-8. The server loads the `"main_en.html"` file from its storage and prepares an HTTP response that includes this header along with the HTML content. This response is sent back to the client, whose browser then displays the HTML content from `"main_en.html"`.

Here's the content from the index.html file located within main\_en.html:

The screenshot displays a VS Code editor with a Python project open. The file 'myform.html' is being edited, containing the following HTML code:

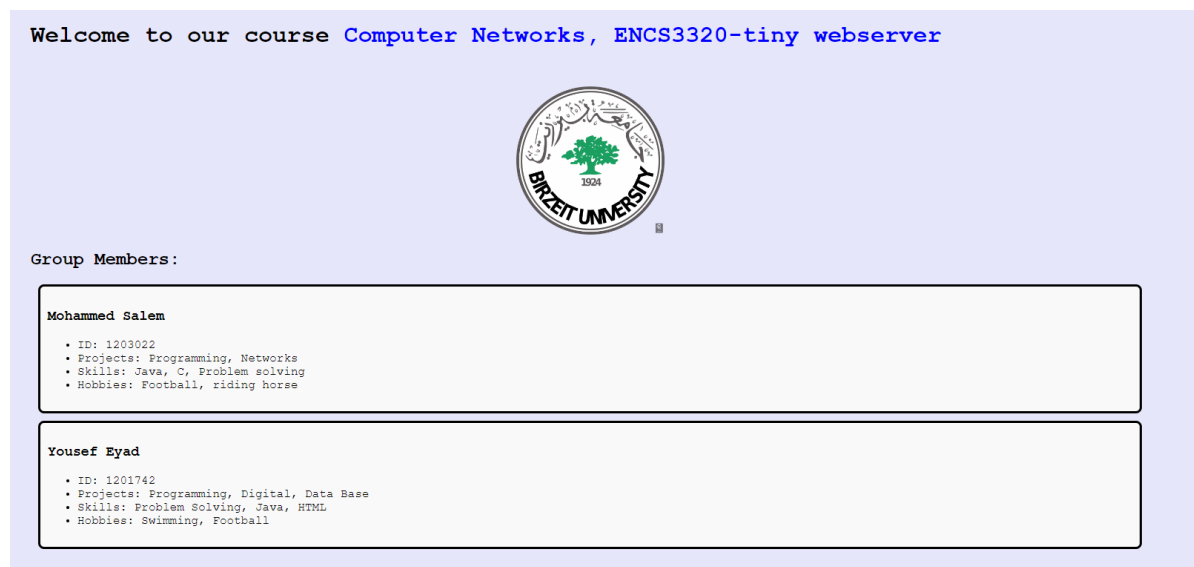
```

1 <!DOCTYPE html> <!-- Begins the HTML document and specifies the language as English -->
2 <html lang="en">
3 <head>
4 <!-- ENCS3320-My First Tiny Webserver/Hello! --> <!-- Sets the title of the web page -->
5 <title>ENCS3320-My First Tiny Webserver/Hello!</title>
6 <!-- Link rel="stylesheet" href="style.css"> <!-- Links to an external CSS file for styling -->
7 <link rel="stylesheet" href="style.css">
8 </head>
9
10 <body>
11 <div class="container"> <!-- Main container for content, styled by CSS -->
12 <h1>Welcome to our course <span class="highlight">Computer Networks, ENCS3320- tiny webserver</span>!</h1>
13 <!-- Header with highlighted course name -->
14  <!-- Image of Birzeit University logo -->
15 <h2>Group Members:</h2> <!-- Header for group members section -->
16
17 <div class="box"> <!-- Container for a group member's details -->
18 <h3>Mohammed Saleem</h3> <!-- Group member name -->
19 <ul>
20 <li>ID: 1203222/1/
21 <li>Projects: Programming, Networks/1/
22 <li>Skills: Java, C, Problem solving/1/
23 <li>Hobbies: Football, riding horse/1/
24 </li>
25 </ul>
26 </div>
27
28 <div class="box"> <!-- Container for another group member's details -->
29 <h3>Yusef Elyadi</h3> <!-- Group member name -->
30 <ul>
31 <li>ID: 1201742/1/
32 <li>Projects: Programming, Digital Data Base/1/
33 <li>Skills: Problem Solving, Java, HTML/1/
34 <li>Hobbies: Swimming, Football/1/
35 </li>
36 </ul>
37 </div>
38
39  <!-- Image of Birzeit University campus -->
40 <br> <!-- Line break for spacing -->
41 <div class="box">
42 <a href="https://www.w3schools.com/python/python_syntax.asp" target="_blank">Link to w3schools Website</a>
43 <!-- External Link to w3schools with the attribute to open in a new tab -->
44 </div>
45 <div>
46 <a href="myform.html" class="btn">Access My Form</a> <!-- Link to an internal HTML form -->
47 </div>
48
49 </body> </div> </html>

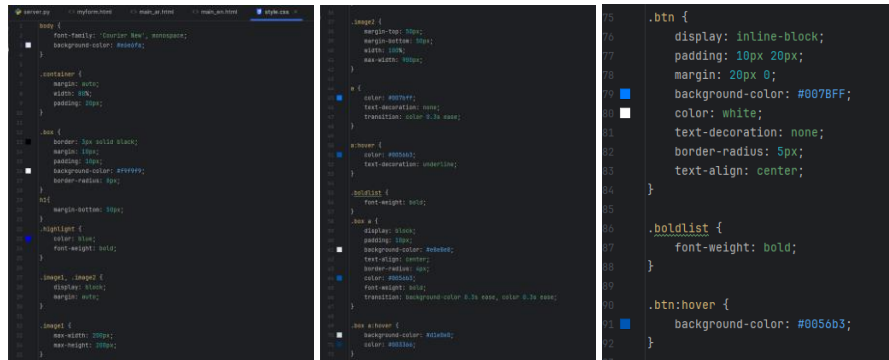
```

The interface includes a Project Explorer on the left showing the file structure, an Editor on the right displaying the code, and a bottom status bar indicating the current file is 'main\_en.html' and the editor is in 'Python 3.11 (pythonProject)' mode.

2. The **main\_en.html** file should contain HTML webpage, which contains the following:
  - a. “ENC53320-My Tiny Webserver” in the title
  - b. “Welcome to our course **Computer Networks, This is a tiny webserver**” (part of the phrase is in **Blue**)
  - c. Group members names and IDs
  - d. Some information about the group members. For instance, projects you have done during different course (programming, electrical, math, etc), skills, hobbies, etc.



- e. We used the CSS to styling the page and make it look nicer.
- f. We also divide the page into boxes as the photo shows and put student's information in the different boxes.
- g. We made the CSS in a separate file.



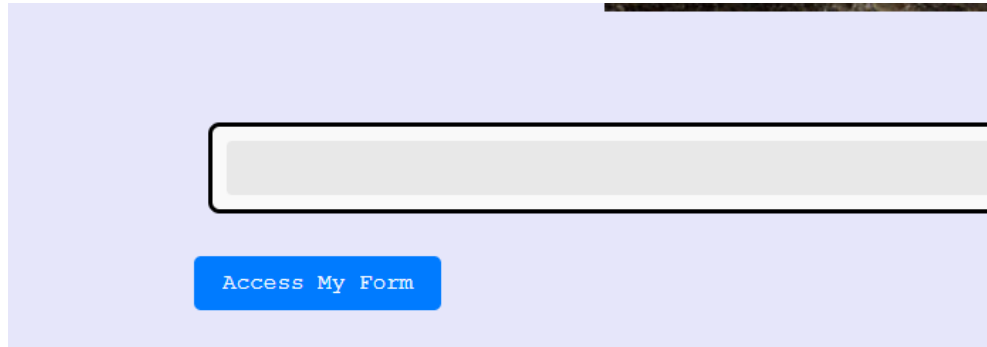
In our CSS code, We set up styles for different parts of a webpage. We chose a specific font for all the text and a light purple background for the page. We arranged content neatly using margins and padding, and boxes have a border, a light gray background, and rounded corners. Headings have a big space below them. Some texts are bold and blue to stand out. Pictures in our page fit within specific sizes, and links change color when you hover over them, making them easier to notice. Buttons are blue with white text, and they change to a darker blue when the user hover over them. This helps make our webpage look nice and organized.

- h. The page should contain at least an image with extension.jpg and an image with extension .png:



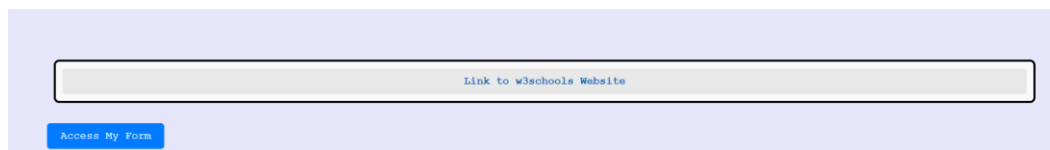
we include in our website two images the first with .PNG extension and the second with .jpg extension and these images appears in both arabic and english versions.

- i. A link to a local html file (myform.html):



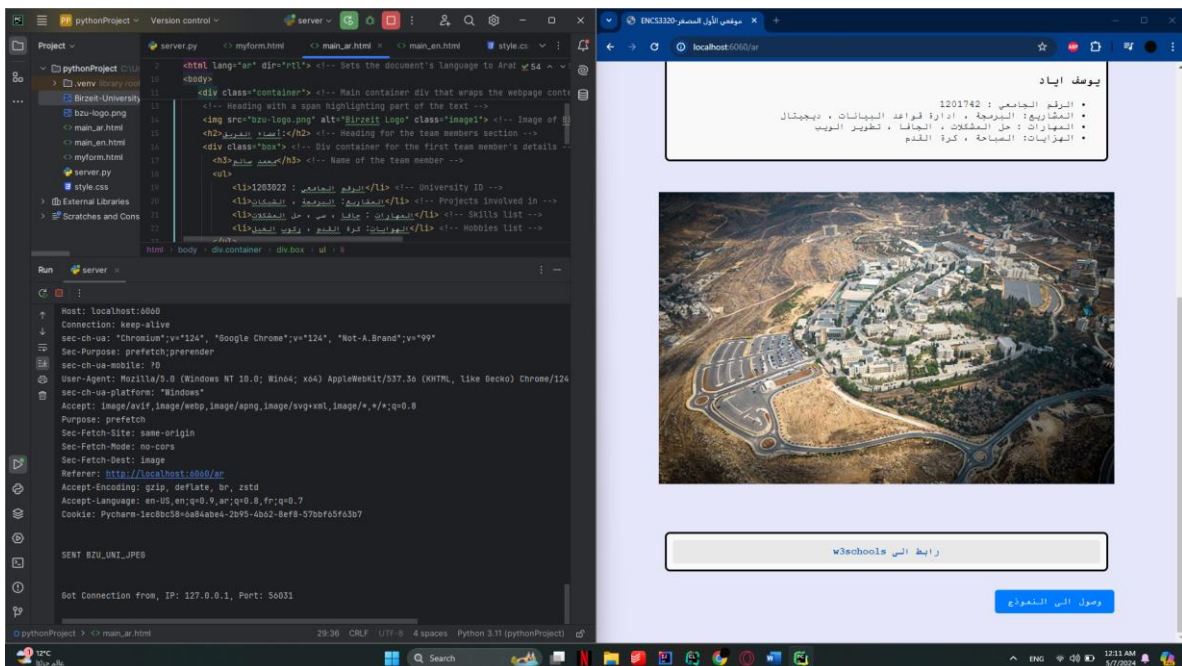
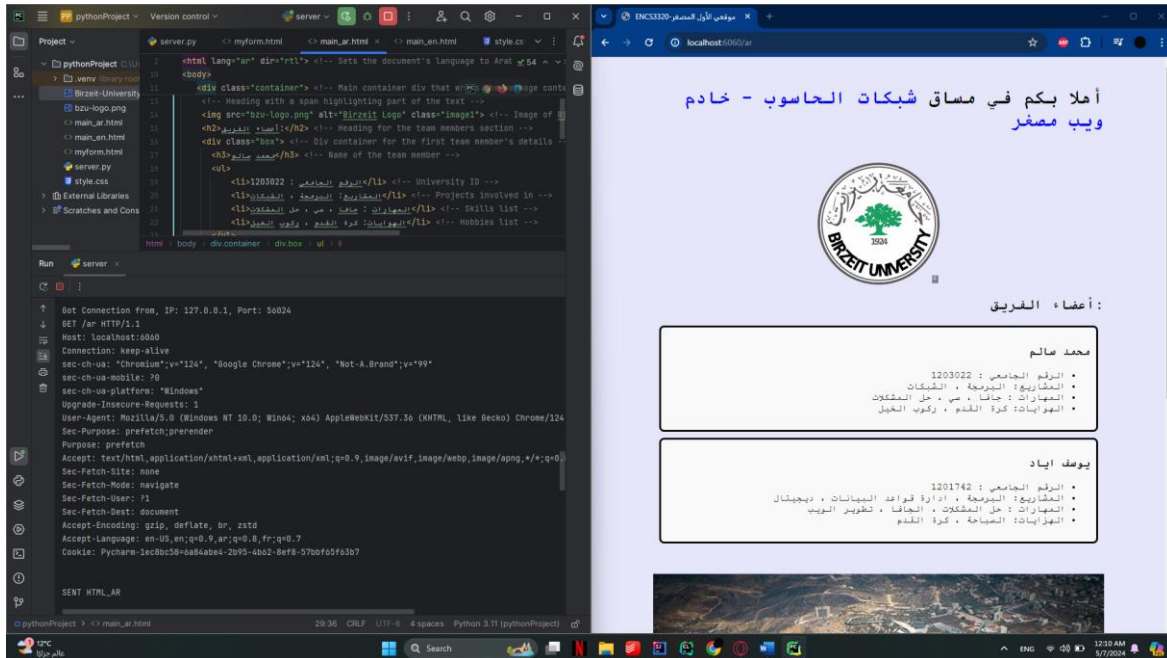
In our webpage, we've included a link that directs users to a local HTML file (myform.html) using the `<a>` HTML tag. We styled this link with CSS to resemble a button, enhancing its interactivity by changing its color when clicked. This creative touch improves user engagement.

- j. a link to [https://www.w3schools.com/python/python\\_syntax.asp](https://www.w3schools.com/python/python_syntax.asp)



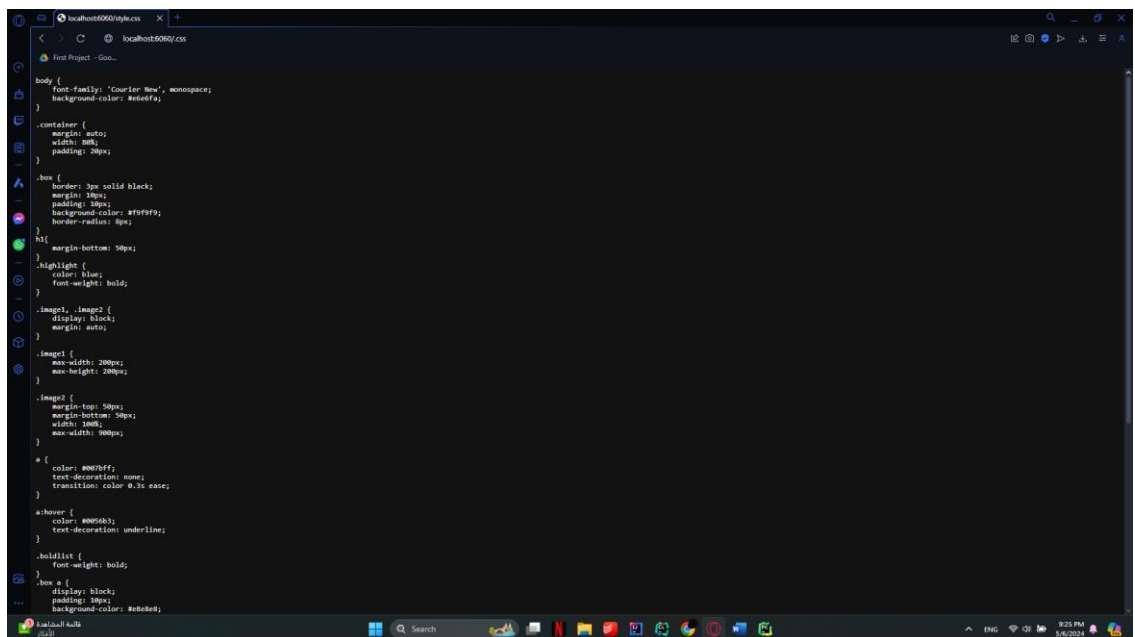
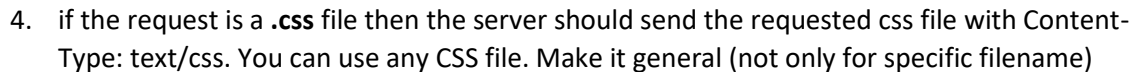
We've also included a link that directs you straight to the W3Schools website when you click on the 'Link to W3Schools Website'.

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





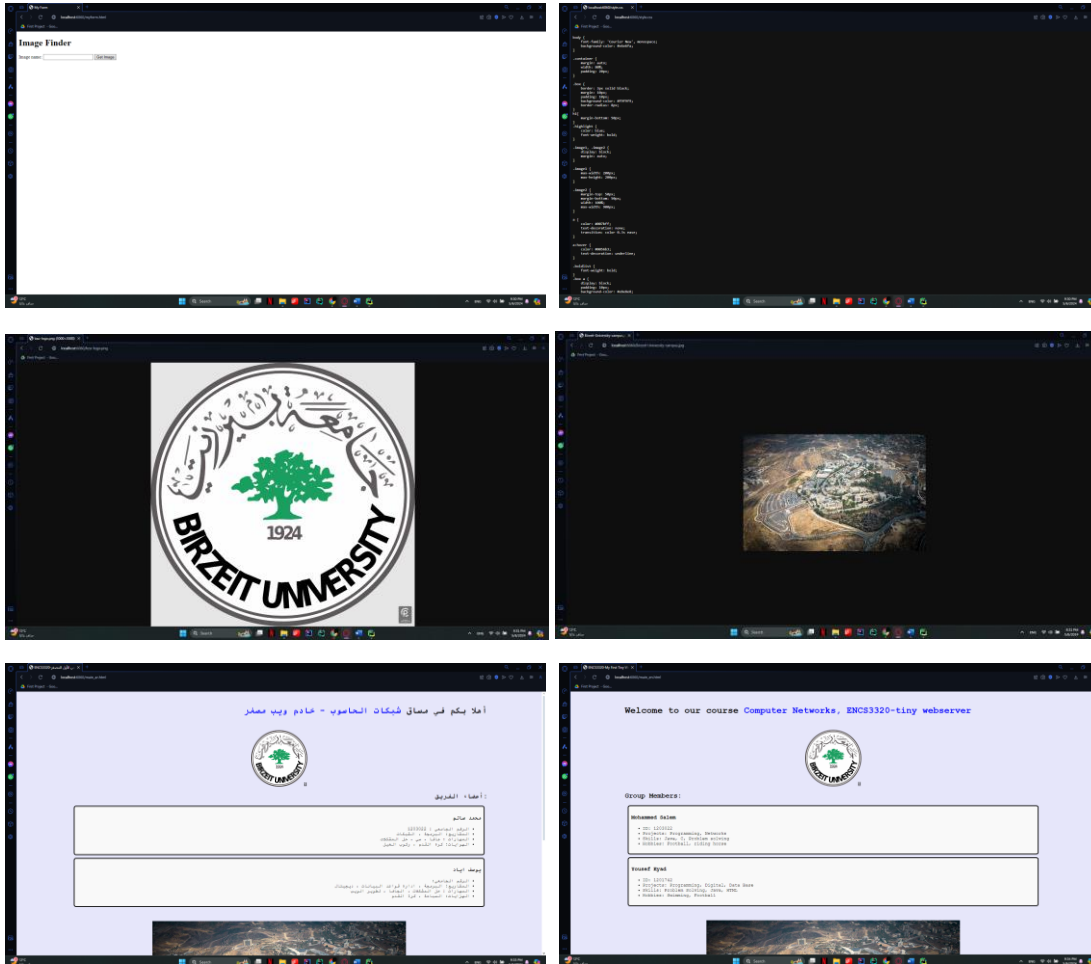
**This is the full code for main\_ar.html file :**



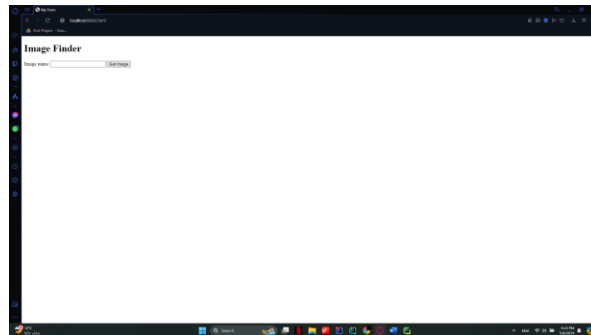


You can easily access any file in my project by entering its name in this format:  
**localhost:6060/filename.extension**. This includes HTML files, images, and the CSS file, making it straightforward to find whatever you're looking for.

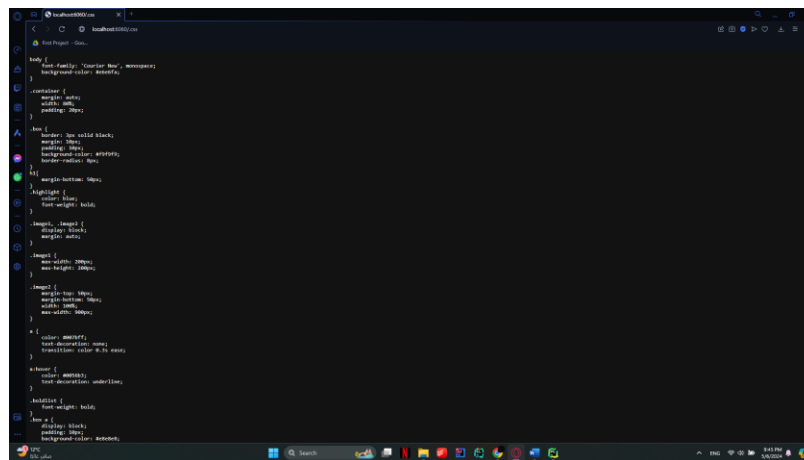
Here are some screenshots to illustrate what I'm talking about:



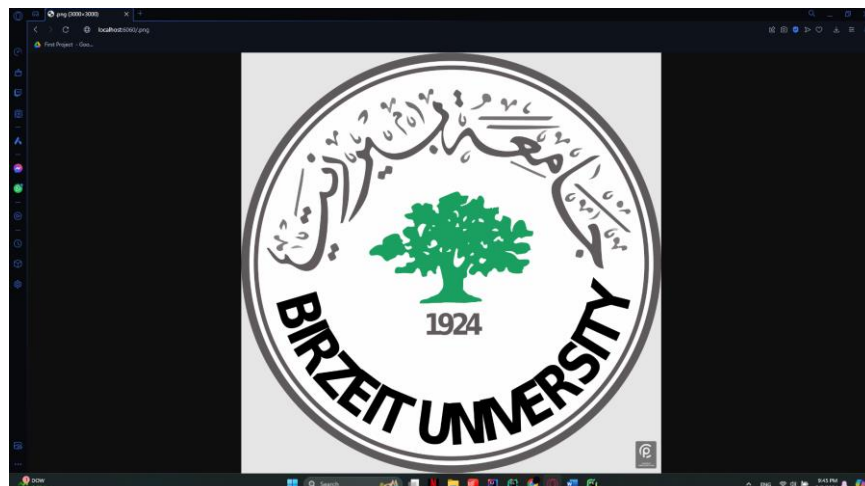
5. if the request is an **.html** file then the server should send the requested html file with Content-Type: text/html. You can use any html file. Make it general (not only for specific filename)



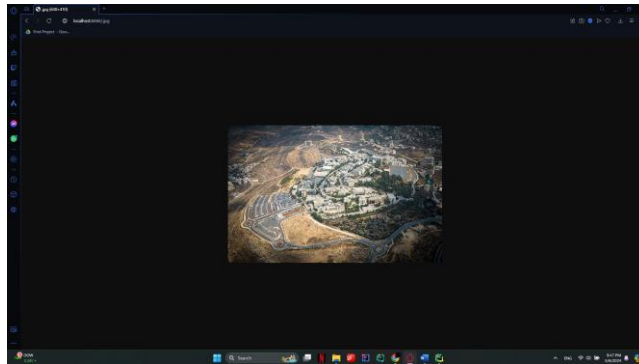
6. if the request is a **.css** file then the server should send the requested css file with Content-Type: text/css. You can use any CSS file. Make it general (not only for specific filename)



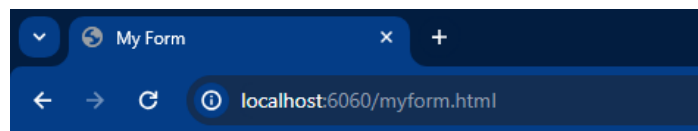
7. if the request is a **.png** then the server should send the png image with Content-Type: image/png. You can use any image. Make it general (not only for specific filename)



8. if the request is a **.jpg** then the server should send the jpg image with Content-Type: image/jpeg. You can use any image. Make it general (not only for specific filename)



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

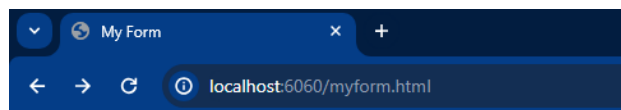


## Image Finder

Image name:

The "**myform.html**" on our web page allows users to enter the name of an image they want to find. When you submit the form by pressing on "**Get Image**" button, the name you entered is sent to the server. The server, set up by the **server.py** script, receives this name and looks for the image file in its storage. If the image exists, the server reads it and sends it back to your web browser. The browser then displays the image on the screen by embedding it in the webpage using a technique called base64 encoding. This process allows the image to be shown directly in the HTML content of the page. If the server cannot find the image, it will show an error message instead.

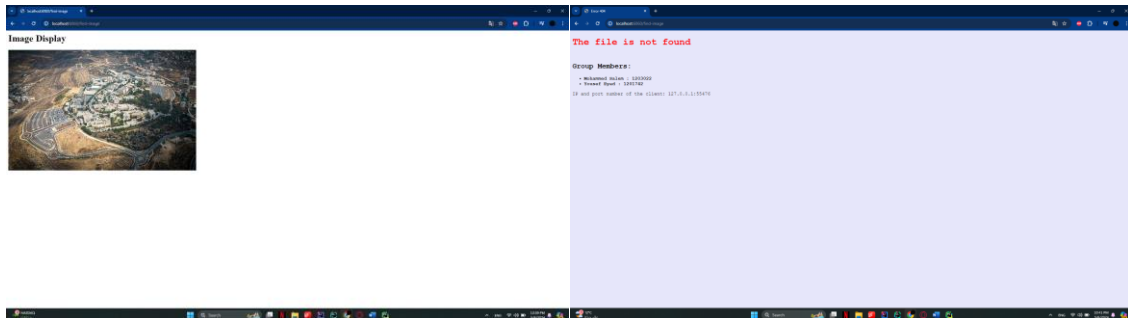
At first we entered the name of the image we want the press **Get Image**.



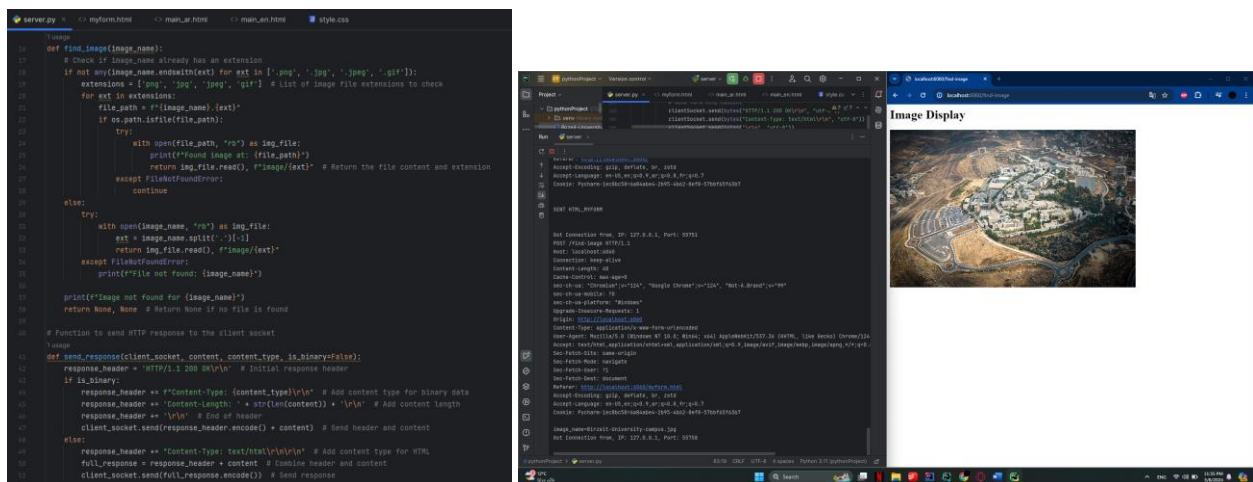
## Image Finder

Image name:

Then the server receives this name and looks for the image file in its storage. If the image exists, the server reads it and sends it back to your web browser. The browser then displays the image on the screen. And if the server can't find it, it'll show an error message listed.



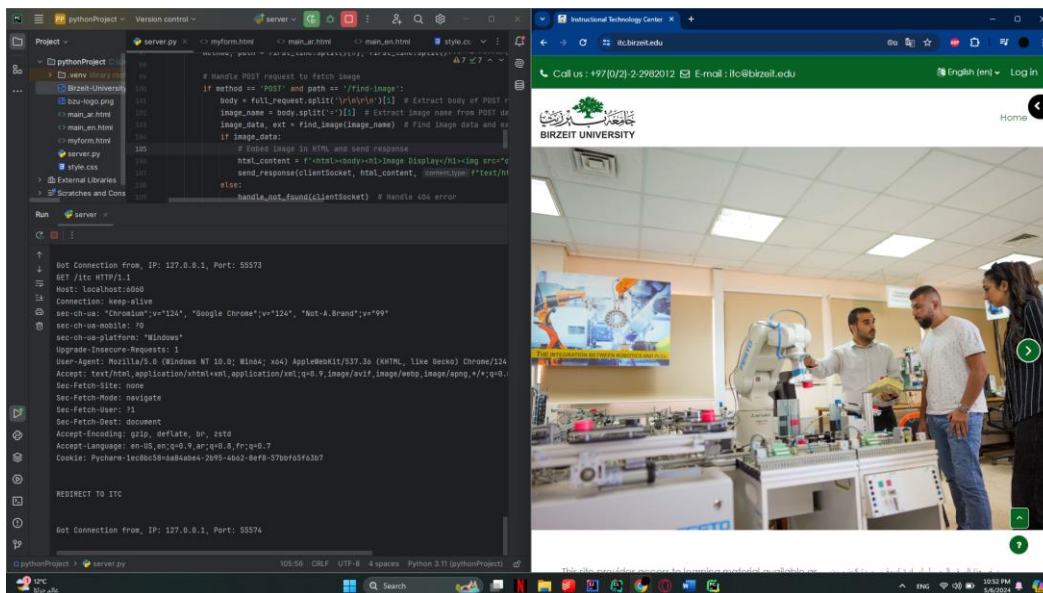
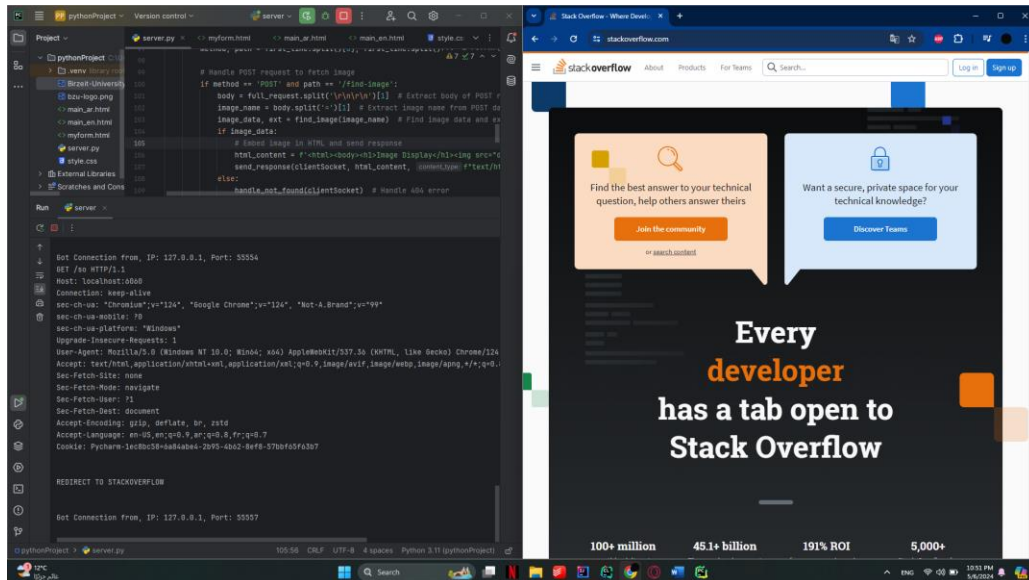
The functions in this script help a web server handle image requests from users. The `find_image` function searches for an image by its name, checking common file extensions if the name doesn't include one. If the image is found, it reads and prepares the image data to be sent back. The `send_response` function then sends this data to the user's browser, either as binary data for images or as HTML for text. When a user submits an image name through a form, the server extracts this name, finds the corresponding image using `find_image`, and displays it in the browser using `send_response`. If no image is found, the server sends an error message back to the user.



```
# Handle POST request to fetch image
if method == 'POST' and path == '/find-image':
    body = full_request.split('\n\n\r\n\r\n')[1] # Extract body of POST request
    image_name = body.split('=')[1] # Extract image name from POST data
    image_data, ext = find_image(image_name) # Find image data and extension
    if image_data:
        # Embed image in HTML and send response
        html_content = f'<html><body><div></div></body></html>'
        send_response(clientSocket, html_content, content_type='text/html')
    else:
        handle_not_found(clientSocket) # Handle 404 error
```

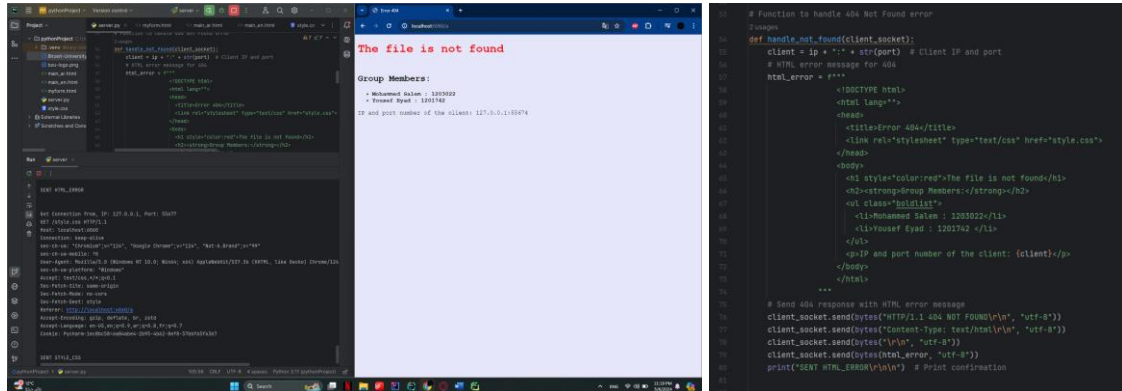
10. Use the status code **307 Temporary Redirect** to redirect the following

- If the request is `/so` then redirect to `stackoverflow.com` website
- If the request is `/itc` then redirect to `itc.birzeit.edu` website



As appear in the screenshot, the server is configured to handle specific routing rules: a request to "localhost:6060/so" automatically redirects to `https://stackoverflow.com/`, and similarly, a request to "localhost:6060/itc" redirects to `https://itc.birzeit.edu/`. This configuration ensures efficient navigation by directly forwarding users to the respective external websites.

11. 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)



The image shows a web browser window displaying a 404 error page. The title of the page is "The file is not found". The content of the page includes a heading "Group Members:" followed by a list of names and IDs: "Muhammad Salan : 1203022" and "Tareeq Syed : 1203762". Below this, it states "IP and port number of the client: 127.0.0.1:5074". To the right of the browser window, there is a Python code snippet that defines a function `handle\_not\_found(client\_socket)`. This function constructs an HTML response for a 404 error, including the title, a stylesheet link, a message that the file is not found, and the group members' names and IDs. It also includes the client's IP and port number. The function sends the response to the client using `client\_socket.send` and prints a confirmation message.

```
def handle_not_found(client_socket):
    client = ip + ":" + str(port) # Client IP and port
    # HTML error message for 404
    html_error = """
    <!DOCTYPE html>
    <html lang="">
    <head>
    <title>Error 404/Title</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
    <div style="color:red">The file is not found</div>
    <div><strong>Group Members:</strong></div>
    <ul class="list">
    <li>Muhammad Salan : 1203022</li>
    <li>Tareeq Syed : 1203762</li>
    </ul>
    <p>IP and port number of the client: {client}</p>
    </body>
    </html>
    """
    # Send the response with HTML error message
    client_socket.send(bytes("HTTP/1.1 404 NOT FOUND\r\n", "utf-8"))
    client_socket.send(bytes("Content-Type: text/html\r\n", "utf-8"))
    client_socket.send(bytes("\r\n", "utf-8"))
    client_socket.send(bytes(html_error, "utf-8"))
    print("SENT HTML_ERROR\r\n") # Print confirmation
```

The function `handle_not_found(client_socket)` manages HTTP 404 errors on a web server by taking a `client_socket` parameter, which represents the client connection. Initially, it combines the client's IP address and port into a string to identify the client. It then constructs a detailed HTML error message indicating a **404 error**, which includes the title, a stylesheet link, and a message that the file is not found, along with the group members' names and IDs, and the client's IP and port. This HTML content is sent to the client through the `client_socket` using a sequence of messages that include the HTTP status line, headers indicating the content type, a separator for the end of headers, and the actual HTML content. The process concludes with a print statement on the server console confirming that the error message has been sent, effectively notifying the client about the missing resource while providing additional server and client details.

And here is “server.py” full code:

```
server.py x  myform.html  < main_ar.html  < main_en.html  style.css
1  import os
2  import base64
3  from socket import *
4
5  # Server configuration
6  serverPort = 4040 # Port number
7  serverSocket = socket(AF_INET, SOCK_STREAM) # Create socket (TCP)
8  serverSocket.bind(('', serverPort)) # Bind socket to address
9  serverSocket.listen(1) # Listen for incoming connections (1 at a time)
10
11 print('The server is ready to receive')
12 print(serverSocket.getsockname()) # Print server address
13
14 # Function to find image file based on the provided name
15 def find_image(image_name):
16     # Check if image_name already has an extension
17     if not any(image_name.endswith(ext) for ext in ['.png', '.jpg', '.jpeg', '.gif']):
18         extensions = ['.png', '.jpg', '.jpeg', '.gif'] # List of image file extensions to check
19         for ext in extensions:
20             file_path = f'{image_name}{ext}'
21             if os.path.isfile(file_path):
22                 try:
23                     with open(file_path, 'rb') as img_file:
24                         print(f'Found image at: {file_path}')
25                         return img_file.read(), f'image/{ext}' # Return the file content and extension
26                 except FileNotFoundError:
27                     continue
28         else:
29             try:
30                 with open(image_name, 'rb') as img_file:
31                     ext = image_name.split('.')[-1]
32                     return img_file.read(), f'image/{ext}'
33             except FileNotFoundError:
34                 print(f'File not found: {image_name}')
35         return None, None # Return None if no file is found
36
37 # Function to send HTTP response to the client socket
38 def send_response(client_socket, content, content_type, is_binary=False):
39     response_header = 'HTTP/1.1 200 OK\r\n' # Initial response header
40     if is_binary:
41         response_header += f'Content-Type: {content_type}\r\n' # Add content type for binary data
42         response_header += f'Content-Length: {str(len(content))}\r\n' # Add content length
43         response_header += '\r\n' # End of header
44         client_socket.send(response_header.encode()) # Send header and content
45     else:
46         response_header += 'Content-Type: text/html\r\n\r\n' # Add content type for HTML
47         full_response = response_header + content # Combine header and content
48         client_socket.send(full_response.encode()) # Send response
49
50 # Function to handle 404 Not Found error
51 def handle_not_found(client_socket):
52     client = ip + ':' + str(port) # Client IP and port
53     # HTML error message for 404
54     html_error = f'''
55     <!DOCTYPE html>
56     <html lang="en">
57     <head>
58     <title>Error 404</title>
59     <link rel="stylesheet" type="text/css" href="style.css">
60     </head>
61     <body>
62     <h1 style="color:red">The file is not found</h1>
63     <h2>strong Group Members:</strong></h2>
64     <ul class="listlist">
65     <li>Mohamed Salem : 1201802</li>
66     <li>Youssef Eyad : 1201742</li>
67     </ul>
68     <p>IP and port number of the client: {client}</p>
69     </body>
70     </html>
71     '''
72     # Send 404 response with HTML error message
73     client_socket.send(bytes('HTTP/1.1 404 NOT FOUND\r\n', 'utf-8'))
74     client_socket.send(bytes('Content-Type: text/html\r\n', 'utf-8'))
75     client_socket.send(bytes('\r\n', 'utf-8'))
76     client_socket.send(bytes(html_error, 'utf-8'))
77     print('SENT HTML_ERROR\r\n\r\n') # Print confirmation
78
79 # Main server loop
80 while True:
81     clientSocket, addr = serverSocket.accept() # Accept incoming connection
82     ip = addr[0] # Client IP
83     port = addr[1] # Client port
84     print('Got Connection from, IP: ' + ip + ', Port: ' + str(port)) # Print client info
85
86     full_request = clientSocket.recv(1024).decode() # Receive request from client
87     print(full_request) # Print request
88
89     # Process request
90     if len(full_request) != 0:
91         lines = full_request.splitlines() # Split request into lines
92         first_line = lines[0] # First line of request
93         method, path = first_line.split()[0], first_line.split()[1] # Extract method and path
94
95         # Handle POST request to fetch image
96         if method == 'POST' and path == '/find-image':
97             body = full_request.split('\r\n\r\n')[1] # Extract body of POST request
98             image_name = body.split('=')[1] # Extract image name from POST data
99             image_data, ext = find_image(image_name) # Find image data and extension
100             if image_data:
101                 # Embed image in HTML and send response
102                 html_content = f'<html><body><h1>Image Display</h1></body></html>'
103                 send_response(clientSocket, html_content, content_type=f'text/html')
104             else:
105                 handle_not_found(clientSocket) # Handle 404 error
106
107         # Handle GET requests
108         elif path in ['/', '/en', '/index.html', '/main_en.html']:
109             # Send main HTML content for English
110             with open('main_en.html', 'rb') as file:
111                 file_content = file.read()
112             clientSocket.send(bytes('HTTP/1.1 200 OK\r\n', 'utf-8'))
113             clientSocket.send(bytes('Content-Type: text/html\r\n', 'utf-8'))
114             clientSocket.send(bytes('\r\n', 'utf-8'))
115             clientSocket.send(file_content)
116             print('SENT HTML_EN\r\n\r\n')
117
118         elif path in ['/ar', '/main_ar.html']:
119             # Send main HTML content for Arabic
120             with open('main_ar.html', 'rb') as file:
```

```
server.py x  myform.html  < main_ar.html  < main_en.html  style.css
83 # Main server loop
84 while True:
85     clientSocket, addr = serverSocket.accept() # Accept incoming connection
86     ip = addr[0] # Client IP
87     port = addr[1] # Client port
88     print('Got Connection from, IP: ' + ip + ', Port: ' + str(port)) # Print client info
89
90     full_request = clientSocket.recv(1024).decode() # Receive request from client
91     print(full_request) # Print request
92
93     # Process request
94     if len(full_request) != 0:
95         lines = full_request.splitlines() # Split request into lines
96         first_line = lines[0] # First line of request
97         method, path = first_line.split()[0], first_line.split()[1] # Extract method and path
98
99         # Handle POST request to fetch image
100         if method == 'POST' and path == '/find-image':
101             body = full_request.split('\r\n\r\n')[1] # Extract body of POST request
102             image_name = body.split('=')[1] # Extract image name from POST data
103             image_data, ext = find_image(image_name) # Find image data and extension
104             if image_data:
105                 # Embed image in HTML and send response
106                 html_content = f'<html><body><h1>Image Display</h1></body></html>'
107                 send_response(clientSocket, html_content, content_type=f'text/html')
108             else:
109                 handle_not_found(clientSocket) # Handle 404 error
110
111         # Handle GET requests
112         elif path in ['/', '/en', '/index.html', '/main_en.html']:
113             # Send main HTML content for English
114             with open('main_en.html', 'rb') as file:
115                 file_content = file.read()
116             clientSocket.send(bytes('HTTP/1.1 200 OK\r\n', 'utf-8'))
117             clientSocket.send(bytes('Content-Type: text/html\r\n', 'utf-8'))
118             clientSocket.send(bytes('\r\n', 'utf-8'))
119             clientSocket.send(file_content)
120             print('SENT HTML_EN\r\n\r\n')
121
122         elif path in ['/ar', '/main_ar.html']:
123             # Send main HTML content for Arabic
124             with open('main_ar.html', 'rb') as file:
```



```
server.py <- myform.html <- main_ar.html <- main_en.html <- style.css
132 elif path in ["/ar", "/main_ar.html"]:
133     # Send main HTML content for Arabic
134     with open("main_ar.html", "rb") as file:
135         file_content = file.read()
136     clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
137     clientSocket.send(bytes("Content-Type: text/html\r\n", "utf-8"))
138     clientSocket.send(bytes("\r\n", "utf-8"))
139     clientSocket.send(file_content)
140     print("SENT HTML_AR\r\n\r\n")
141
142 # Handle CSS file request
143 elif path in ["/style.css", "/.css"]:
144     with open("style.css", "r") as file:
145         file_content = file.read()
146     # Send CSS file content
147     clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
148     clientSocket.send(bytes("Content-Type: text/css\r\n", "utf-8"))
149     clientSocket.send(bytes("\r\n", "utf-8"))
150     clientSocket.send(file_content)
151     print("SENT STYLE_CSS\r\n\r\n")
152
153 # Handle PNG image request
154 elif path in ["/bzu-logo.png", "/.png"]:
155     with open("bzu-logo.png", "rb") as file:
156         file_content = file.read()
157     # Send PNG image content
158     clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
159     clientSocket.send(bytes("Content-Type: image/png\r\n", "utf-8"))
160     clientSocket.send(bytes("\r\n", "utf-8"))
161     clientSocket.send(file_content)
162     print("SENT BZU_CIRCLE_PNG\r\n\r\n")
163
164 # Handle JPEG image request
165 elif path in ["/Birzeit-University-campus.jpg", "/.jpg"]:
166     with open("Birzeit-University-campus.jpg", "rb") as file:
167         file_content = file.read()
168     # Send JPEG image content
169     clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
170     clientSocket.send(bytes("Content-Type: image/jpeg\r\n", "utf-8"))
171     clientSocket.send(bytes("\r\n", "utf-8"))
172     clientSocket.send(file_content)
173
174 # Handle redirection to external URLs
175 elif path == "/itc":
176     clientSocket.send(bytes("HTTP/1.1 307 temporary Redirect\r\n", "utf-8"))
177     clientSocket.send(bytes("Content-Type:\r\n", "utf-8"))
178     clientSocket.send(bytes("Location: https://itc.birzeit.edu\r\n", "utf-8"))
179     print("REDIRECT TO ITC\r\n\r\n")
180
181 elif path == "/so":
182     clientSocket.send(bytes("HTTP/1.1 307 temporary Redirect\r\n", "utf-8"))
183     clientSocket.send(bytes("Content-Type:\r\n", "utf-8"))
184     clientSocket.send(bytes("Location: https://stackoverflow.com\r\n", "utf-8"))
185     print("REDIRECT TO STACKOVERFLOW\r\n\r\n")
186
187 # Handle request for form HTML file
188 elif path in ["/myform.html", "/myform", "/.html"]:
189     with open("myform.html", "rb") as file:
190         file_content = file.read()
191     # Send form HTML content
192     clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
193     clientSocket.send(bytes("Content-Type: text/html\r\n", "utf-8"))
194     clientSocket.send(bytes("\r\n", "utf-8"))
195     clientSocket.send(file_content)
196     print("SENT HTML_MYFORM\r\n\r\n")
197
198 else:
199     handle_not_found(clientSocket) # Handle 404 error for unknown paths
200 clientSocket.close() # Close client socket after handling request
201
202 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

*And here a screenshot of the HTTP request printed on the command line.:*

```
"C:\Users\Tech Zone\Networks\Project\pythonProject\venv\Scripts\python.exe" "C:\Users\Tech Zone\Networks\Project\pythonProject\server.py"
The server is ready to receive
('8.8.8.8', 4040)
Got Connection from, IP: 127.0.0.1, Port: 55815
GET / HTTP/1.1
Host: localhost:4040
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
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/124.0.0.0 Safari/537.36
Sec-Purpose: prefetch;prerender
Host: localhost:4040
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58-0a84abe4-2095-4b62-8ef8-57b0f05f63b7

SENT HTML_IN

Got Connection from, IP: 127.0.0.1, Port: 55819
GET /style.css HTTP/1.1
Host: localhost:4040
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
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/124.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: text/css,*/*;q=0.1
Purpose: prefetch
Host: localhost:4040
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://localhost:4040/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58-0a84abe4-2095-4b62-8ef8-57b0f05f63b7

SENT STYLE_CSS

Got Connection from, IP: 127.0.0.1, Port: 55820
GET /bzu-logo.png HTTP/1.1
Host: localhost:4040
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
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/124.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Purpose: prefetch
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:4040/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58-0a84abe4-2095-4b62-8ef8-57b0f05f63b7

SENT BZU_CIRCLE_PNG

Got Connection from, IP: 127.0.0.1, Port: 55821
GET /Birzeit-University-campus.jpg HTTP/1.1
Host: localhost:4040
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
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/124.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Purpose: prefetch
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:4040/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58-0a84abe4-2095-4b62-8ef8-57b0f05f63b7

SENT BZU_UNI_JPEG

Got Connection from, IP: 127.0.0.1, Port: 55822
I
```



```
|
Got Connection from, IP: 127.0.0.1, Port: 55845
GET /ar HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
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/124.0.0.0 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,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58=6a84abe4-2b95-4b62-8ef8-57bbf65f63b7

SENT HTML_AR
```

```
Got Connection from, IP: 127.0.0.1, Port: 55867
GET /nn HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
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/124.0.0.0 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,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58=6a84abe4-2b95-4b62-8ef8-57bbf65f63b7

SENT HTML_ERROR
```

```
Got Connection from, IP: 127.0.0.1, Port: 55883
GET /myform.html HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
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/124.0.0.0 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: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:6060/ar
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58=6a84abe4-2b95-4b62-8ef8-57bbf65f63b7

SENT HTML_MYFORM
```

```
Got Connection from, IP: 127.0.0.1, Port: 55882
GET /.css HTTP/1.1
```

```
Got Connection from, IP: 127.0.0.1, Port: 55897
GET /.html HTTP/1.1
```

```
Got Connection from, IP: 127.0.0.1, Port: 55898
GET /.png HTTP/1.1
```

```
Got Connection from, IP: 127.0.0.1, Port: 55916
GET /.jpg HTTP/1.1
```

```
Got Connection from, IP: 127.0.0.1, Port: 55938
GET /itc HTTP/1.1
Host: localhost:6060
```

```
Got Connection from, IP: 127.0.0.1, Port: 55941
GET /so HTTP/1.1
Host: localhost:6060
```