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.

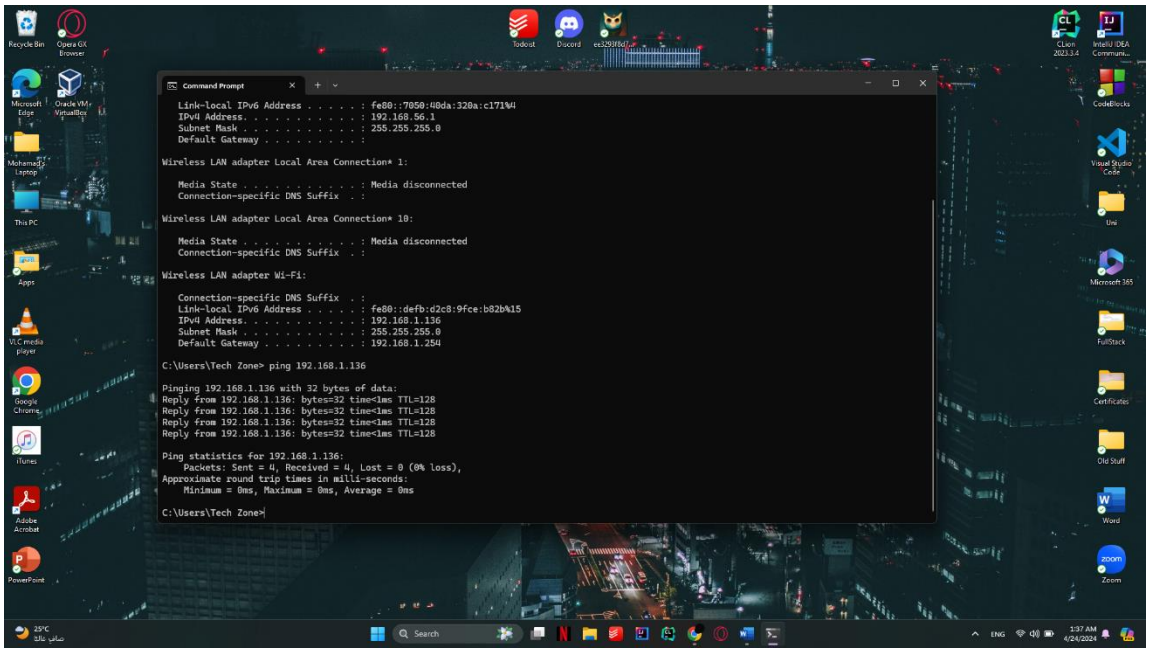
b) Tracert (Trace Route): Tracert is a command-line tool that traces the path packets take to a network host, revealing the routers they pass through along the way.

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is a protocol for remotely accessing another computer, primarily through command-line tasks, although it's considered outdated and less secure compared to SSH for such connections.

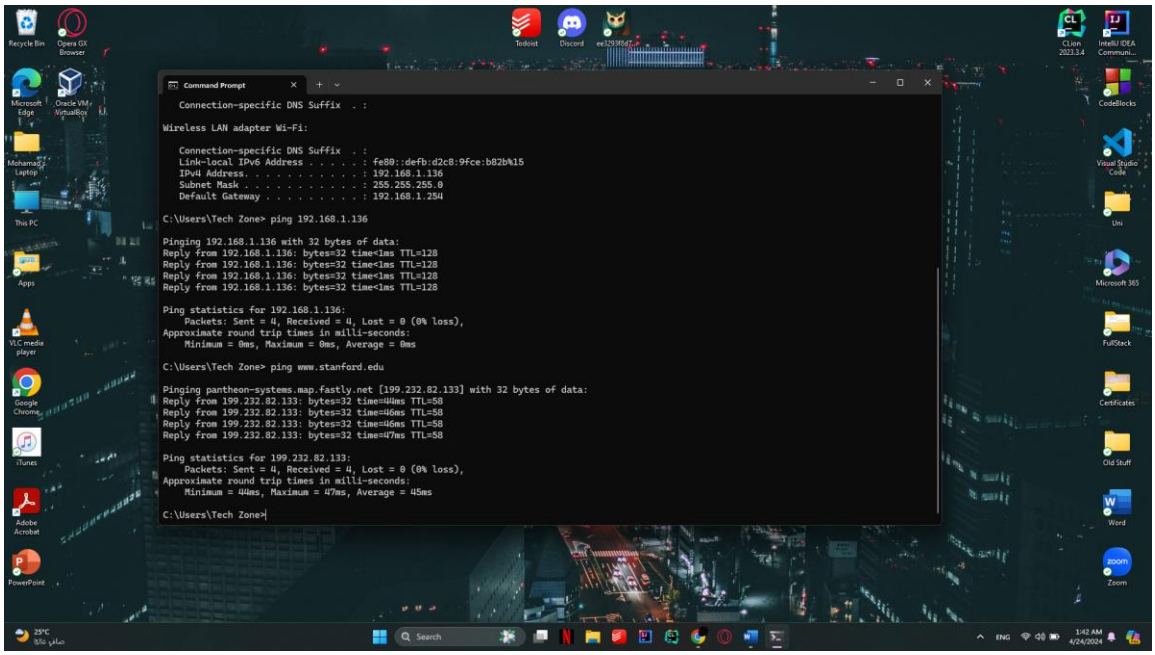
2. Make sure that your computer is connected to the internet and then run the following commands:

a) Ping a device in the same network:



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

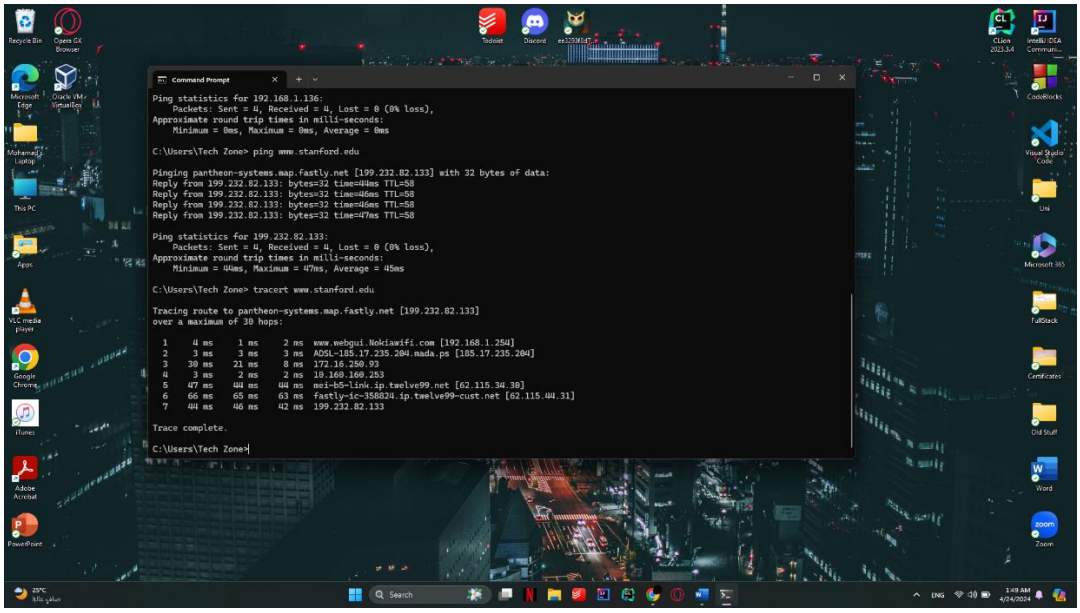


Here's a screenshot of a Windows Command Prompt after running a "ping" command to check the network response for www.stanford.edu. The command actually resolved to the domain pantheon-systems.map.fastly.net, which likely means that Stanford's website is using Fastly a content delivery network (CDN) The 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.

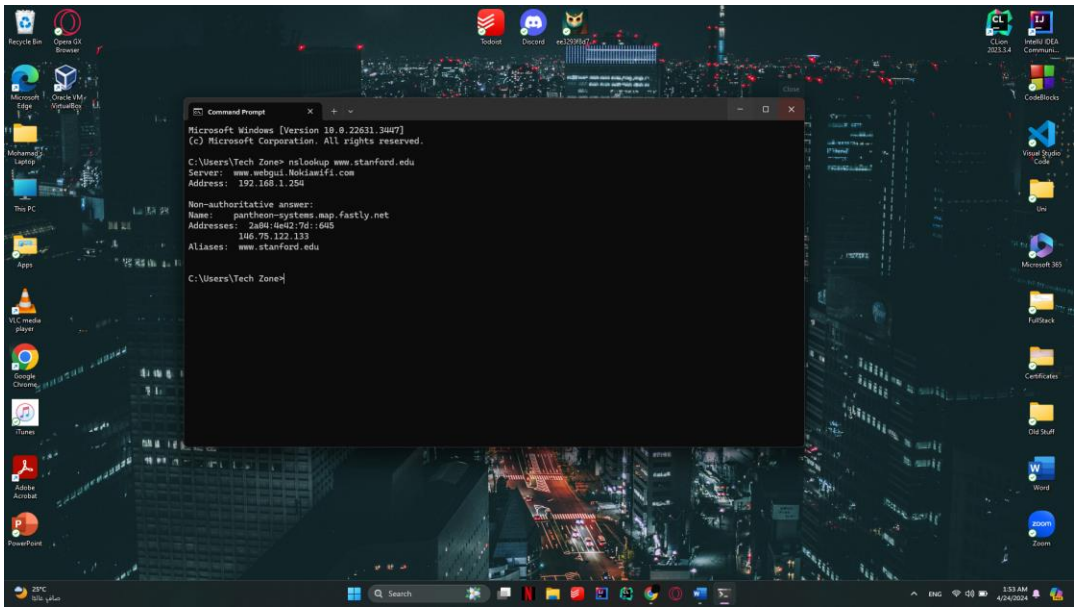
The ping command results show that `www.stanford.edu` resolved to the IP address `199.232.82.133` and the responses were consistently around 44-47 milliseconds. The "TTL" value is 58 which stands for Time to Live and indicates the number of hops a packet can take before it is discarded. However from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



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.

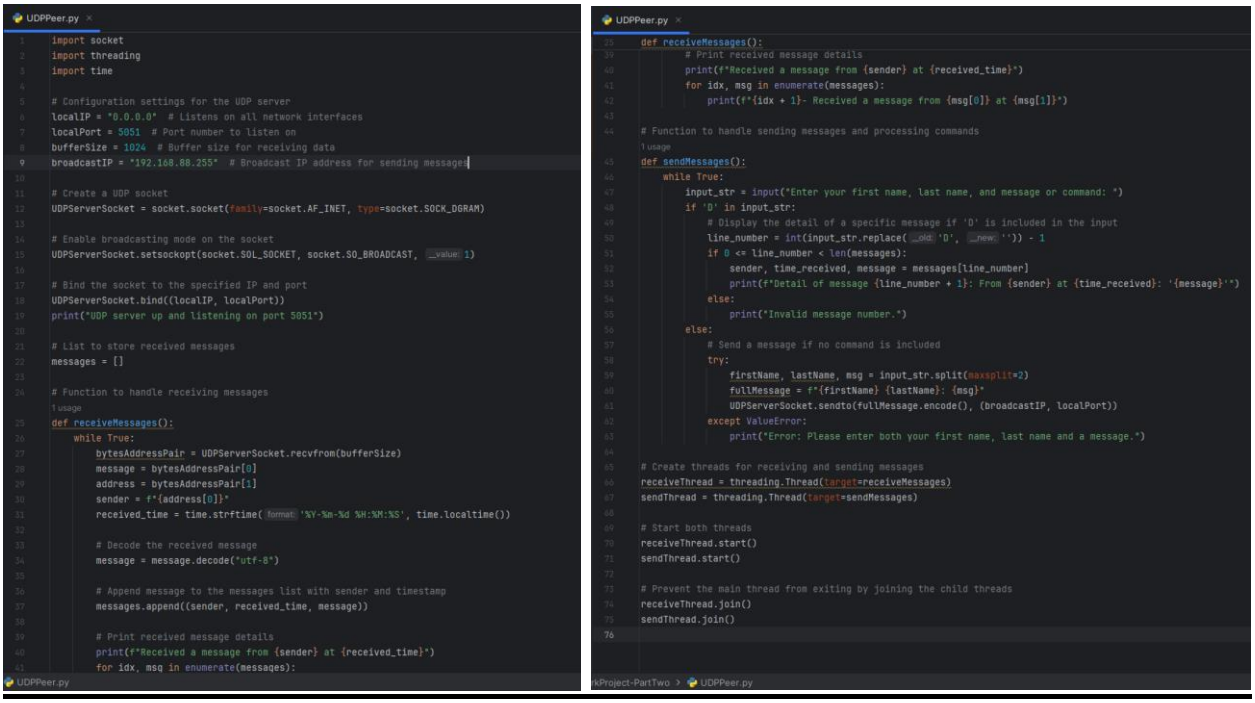
e) nslookup www.stanford.edu



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.



```python
import socket
import threading
import time

# Configuration settings for the UDP server
localIP = "0.0.0.0"  # Listens on all network interfaces
localPort = 5051  # Port number to listen on
bufferSize = 1024  # Buffer size for receiving data
broadcastIP = "192.168.88.255"  # Broadcast IP address for sending messages

# Create a UDP socket
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Enable broadcasting mode on the socket
UDPServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, _value: 1)

# Bind the socket to the specified IP and port
UDPServerSocket.bind((localIP, localPort))
print("UDP server up and listening on port 5051")

# List to store received messages
messages = []

# Function to handle receiving messages
1 usage
def receiveMessages():
    while True:
        bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
        message = bytesAddressPair[0]
        address = bytesAddressPair[1]
        sender = f"{address[0]}"
        received_time = time.strftime( format: '%Y-%m-%d %H:%M:%S', time.localtime())

        # Decode the received message
        message = message.decode("utf-8")

        # Append message to the messages list with sender and timestamp
        messages.append((sender, received_time, message))

        # Print received message details
        print(f"Received a message from {sender} at {received_time}")
        for idx, msg in enumerate(messages):
```

```python
    def receiveMessages():
        # Print received message details
        print(f"Received a message from {sender} at {received_time}")
        for idx, msg in enumerate(messages):
            print(f"{idx + 1}- Received a message from {msg[0]} at {msg[1]}")

# Function to handle sending messages and processing commands
1 usage
def sendMessages():
    while True:
        input_str = input("Enter your first name, last name, and message or command: ")
        if 'D' in input_str:
            # Display the detail of a specific message if 'D' is included in the input
            line_number = int(input_str.replace( _old: 'D', _new: '')) - 1
            if 0 <= line_number < len(messages):
                sender, time_received, message = messages[line_number]
                print(f"Detail of message {line_number + 1}: From {sender} at {time_received}: '{message}'")
            else:
                print("Invalid message number.")
        else:
            # Send a message if no command is included
            try:
                firstName, lastName, msg = input_str.split(maxsplit=2)
                fullMessage = f"{firstName} {lastName}: {msg}"
                UDPServerSocket.sendto(fullMessage.encode(), (broadcastIP, localPort))
            except ValueError:
                print("Error: Please enter both your first name, last name and a message.")

# Create threads for receiving and sending messages
receiveThread = threading.Thread(target=receiveMessages)
sendThread = threading.Thread(target=sendMessages)

# Start both threads
receiveThread.start()
sendThread.start()

# Prevent the main thread from exiting by joining the child threads
receiveThread.join()
sendThread.join()
```

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.

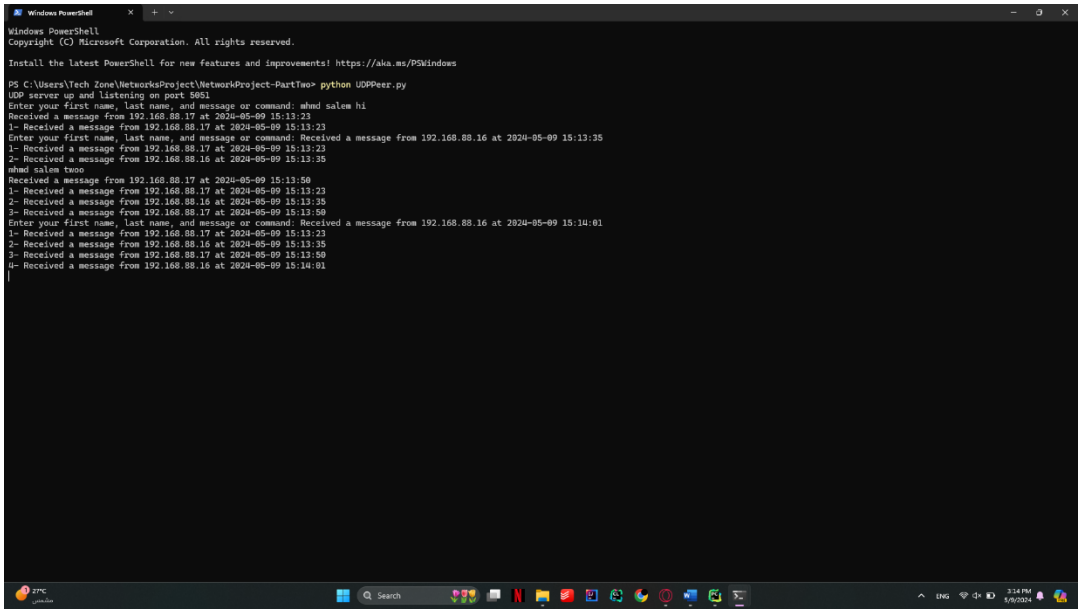
Messages are handled through two main functions: one for sending and one for receiving. Users can send messages by inputting their first and last names followed by their message. This input is then broadcast to all peers. Concurrently, the script listens for incoming messages, which, upon receipt, are 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:

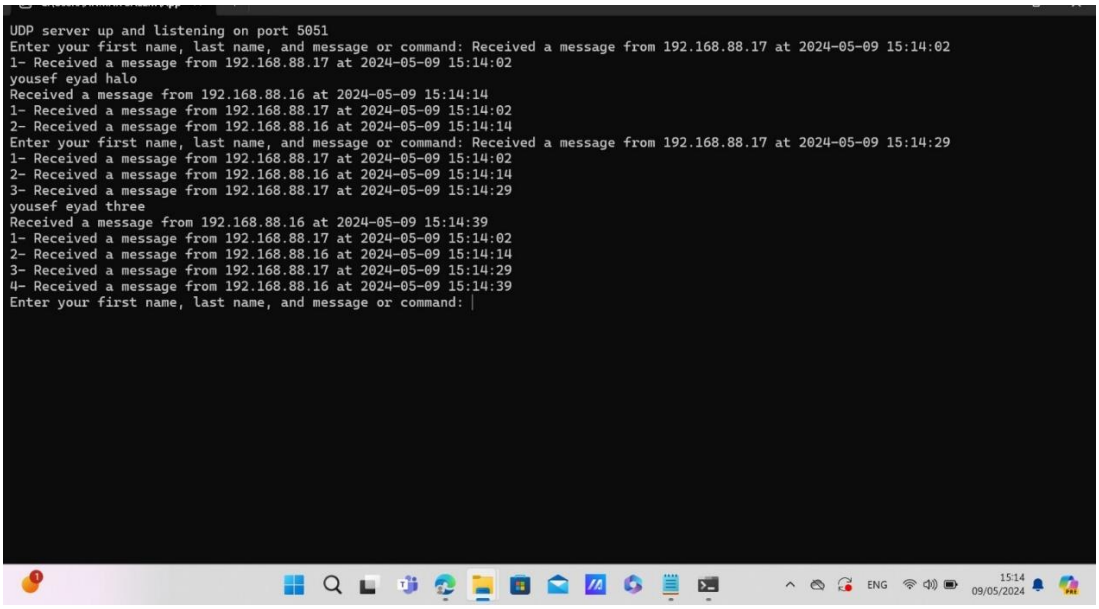
To enhance interactivity, the script includes threading capabilities using Python's `**threading**` library. This allows the script to handle sending and receiving messages simultaneously, providing a seamless user experience. Users can interact with the script through a command-line interface where 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.



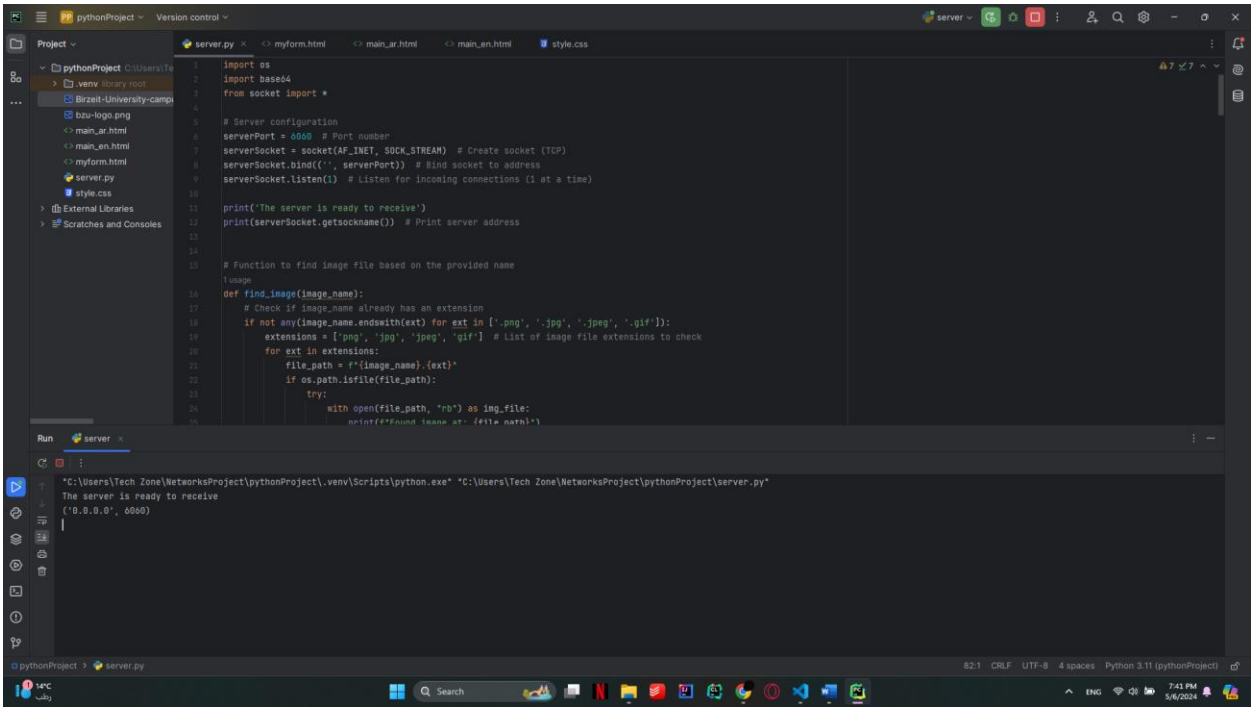
The image shows a Windows PowerShell window from the first user “Mohammed Salem” where I was running a Python script called "UDPPeer.py". The script sets up a UDP server on port 5051 and displays interactions between users on the network. In the session, we can see that the user "mhmd salem" sending 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.



This image shows a Windows PowerShell window from the second user “Youse 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:**

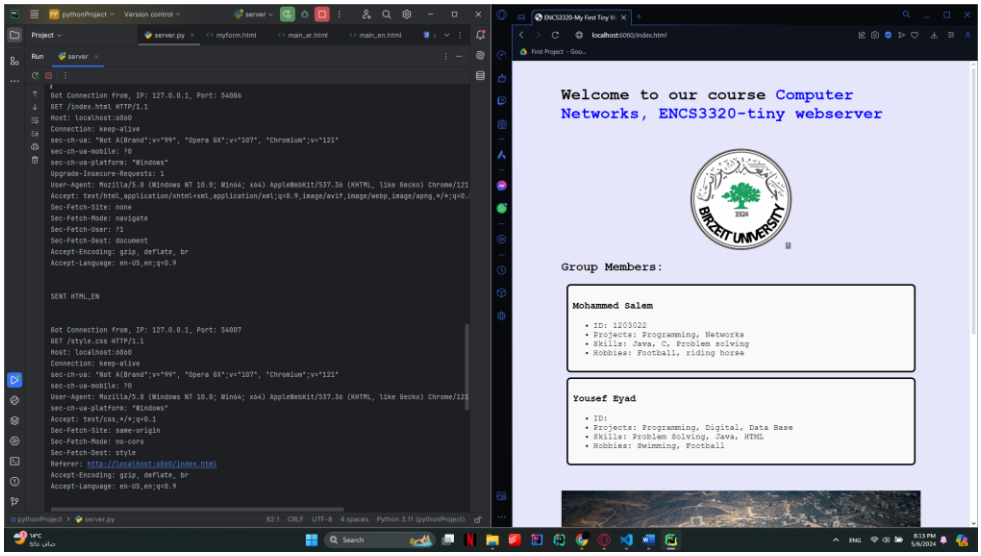
Using socket programming, implement a simple but a complete web server in go, python, java or C that is listening on port 6060.



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.

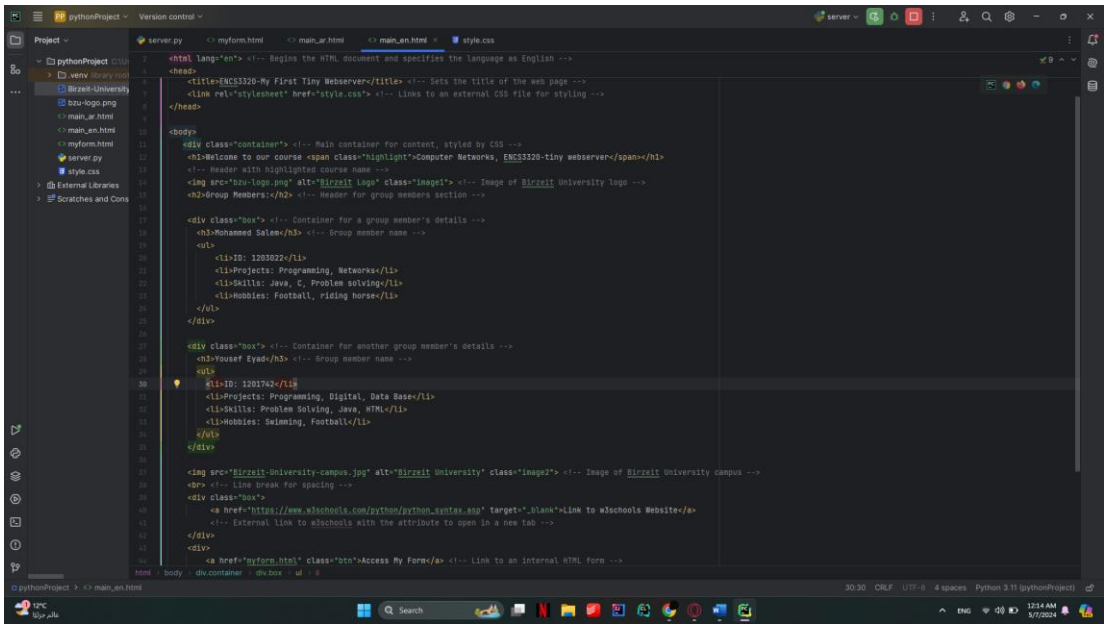
1. If the request is **/ or /index.html or /main_en.html or /en (for example localhost:6060/ or localhost:6060/en)** then the server should send **main_en.html** file with Content-Type: text/html; charset=UTF-8

```
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:



2. The **main_en.html** file should contain HTML webpage, which contains the following:
   a. "ENCS3320-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.

Welcome to our course Computer Networks, ENCS3320-tiny webserver

Group Members:

Mohammed Salem

- ID: 1203022
- Projects: Programming, Networks
- Skills: Java, C, Problem solving
- Hobbies: Football, riding horse

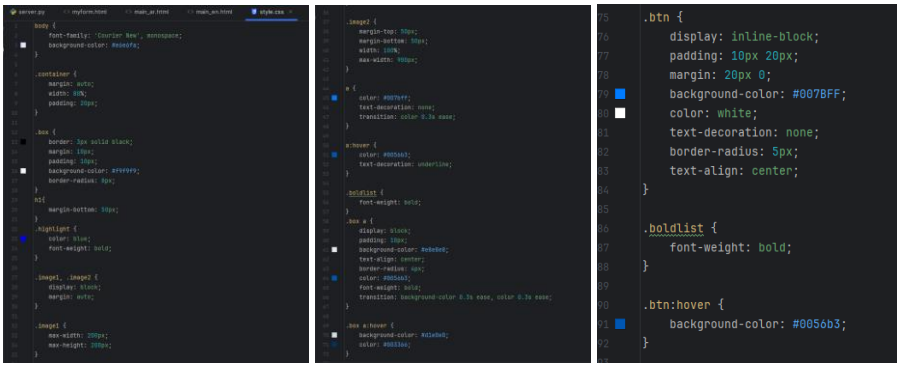
Yousef Eyad

- ID: 1201742
- Projects: Programming, Digital, Data Base
- Skills: Problem Solving, Java, HTML
- Hobbies: Swimming, Football

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.



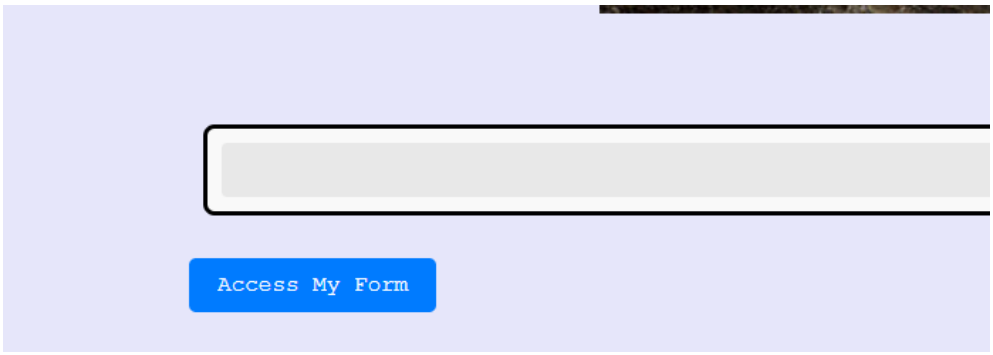
```
.btn {
    display: inline-block;
    padding: 10px 20px;
    margin: 20px 0;
    background-color: #007BFF;
    color: white;
    text-decoration: none;
    border-radius: 5px;
    text-align: center;
}

.boldlist {
    font-weight: bold;
}

.btn:hover {
    background-color: #0056b3;
}
```

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 extention.jpg and an image with extension .png:

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

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

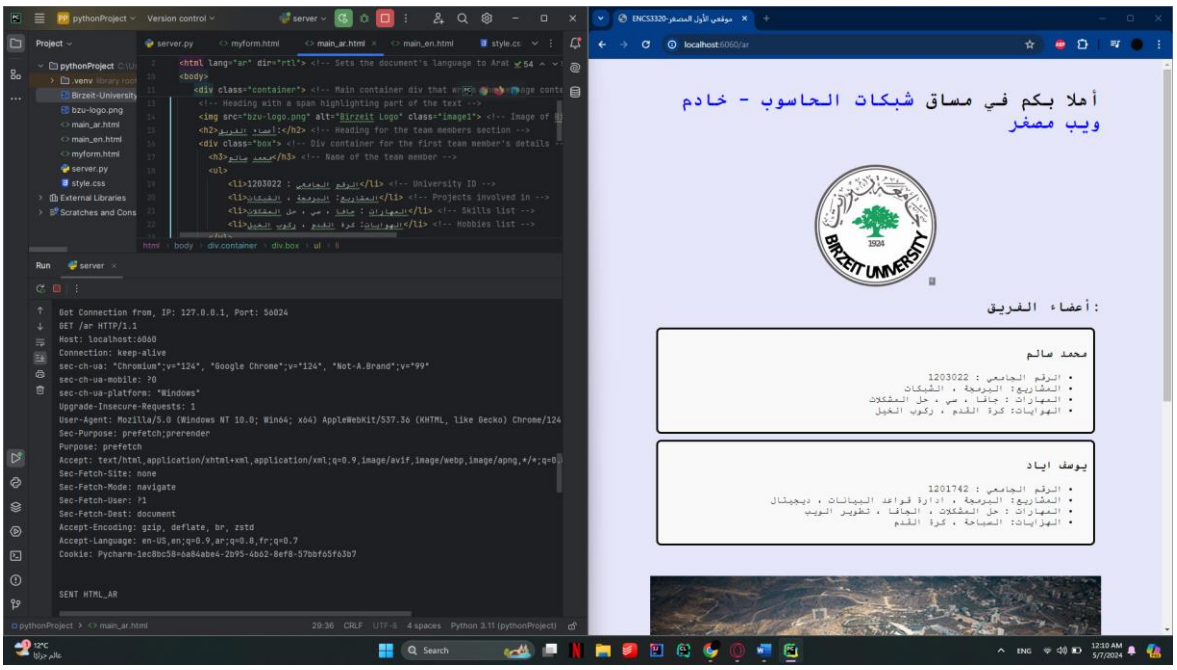


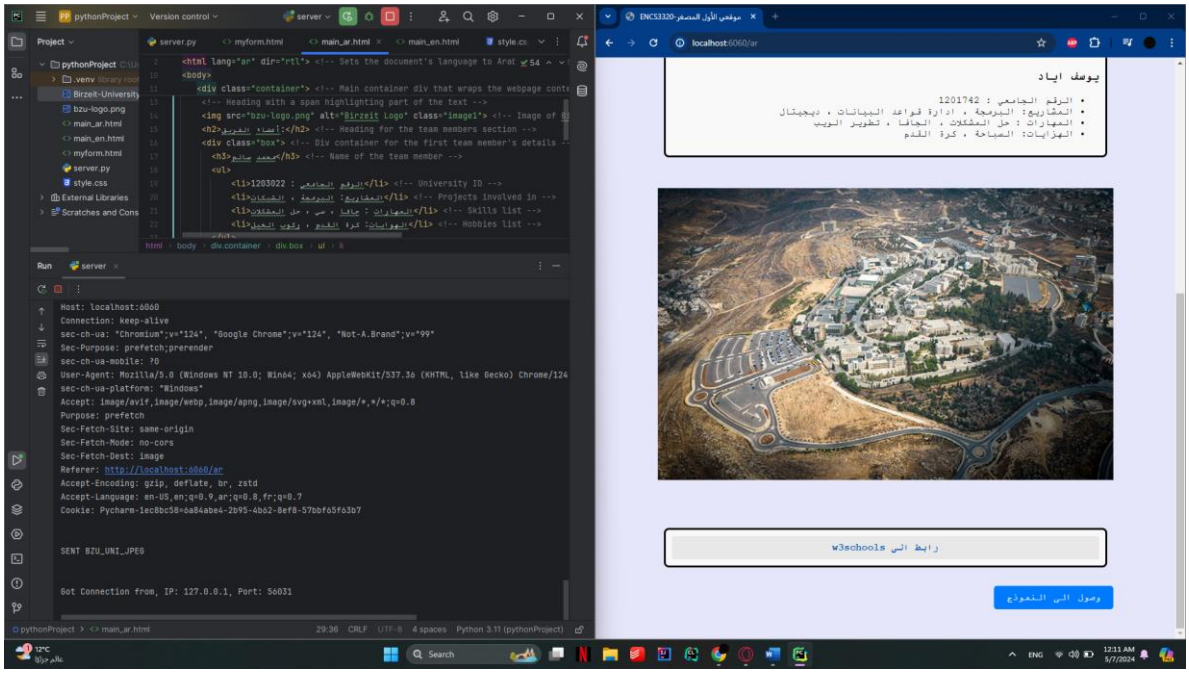
In our webpage, we've included a link that directs users to a local HTML file (myform.html) using the **<a>** HTML tag. We styled this link with CSS to resemble a button, enhancing its interactivity by changing its color when clicked. This creative touch improves user engagement.

j. a link to https://www.w3schools.com/python/python_syntax.asp

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

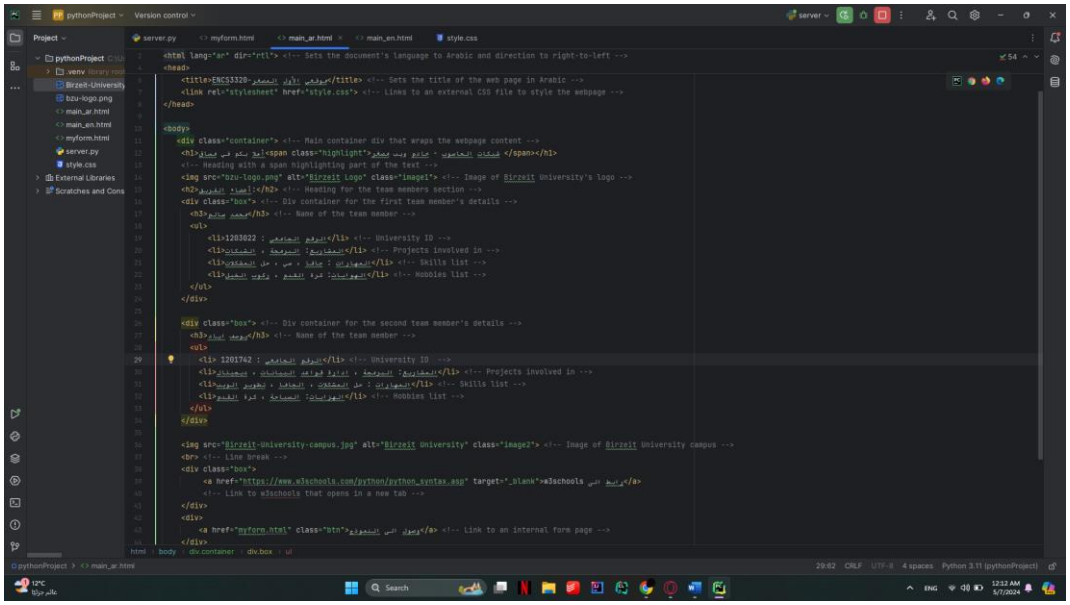
3. If the request is “/ar” then the server response with main_ar.html which is an Arabic version of main_en.html



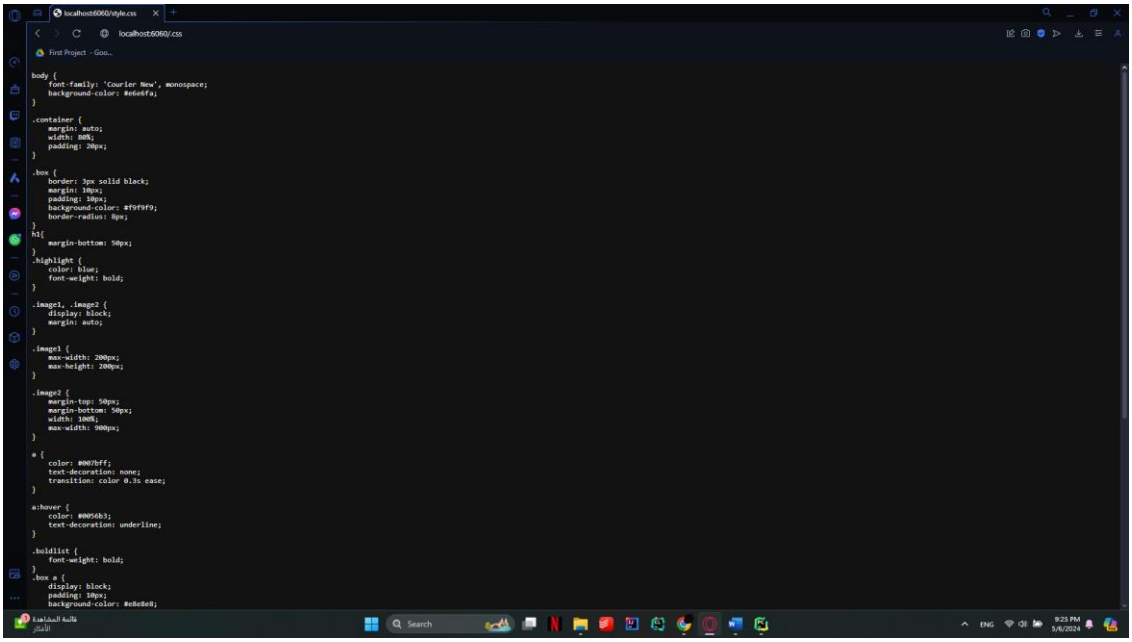


When you visit the link http://localhost:6060/ar, you'll be automatically redirected to the Arabic version of the website. It shares the same style and design as the English version, using the same CSS file for both.

This is the full code for main_ar.html file :

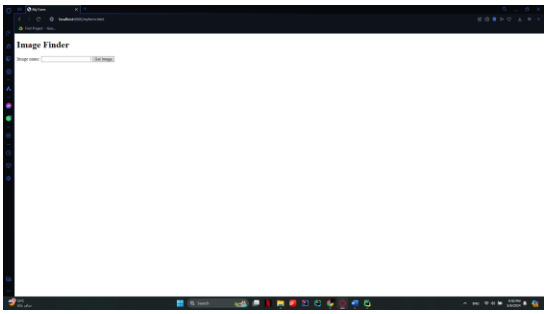


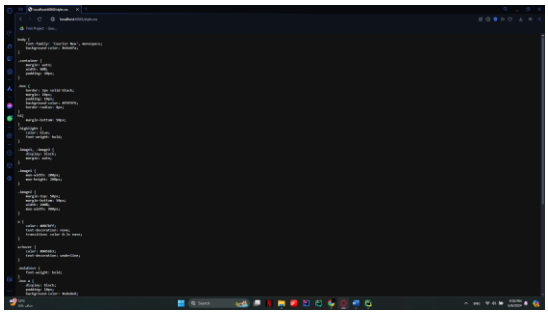
4. if the request is a **.css** file then the server should send the requested css file with Content-Type: text/css. You can use any CSS file. Make it general (not only for specific filename)

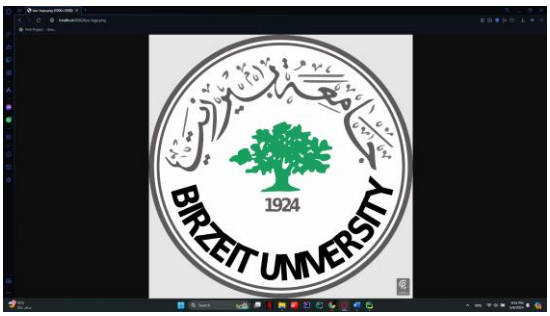


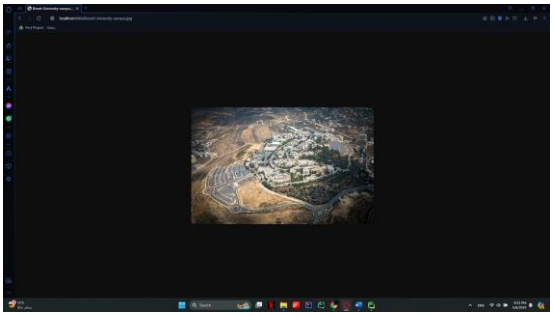
You can easily access any file in my project by entering its name in this format: **localhost:6060/filename.extension**. This includes HTML files, images, and the CSS file, making it straightforward to find whatever you're looking for.

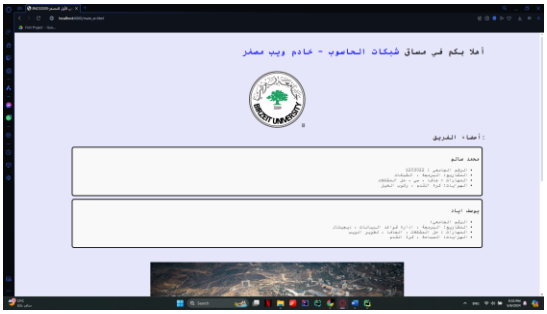
Here are some screenshots to illustrate what I'm talking about:

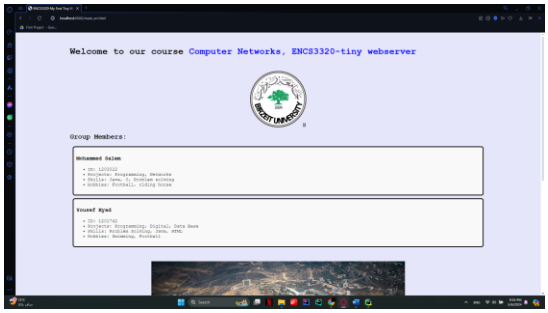




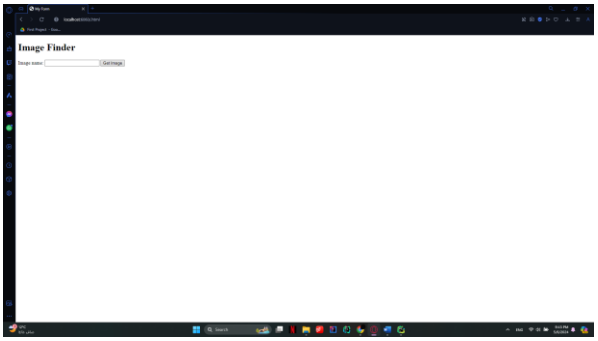




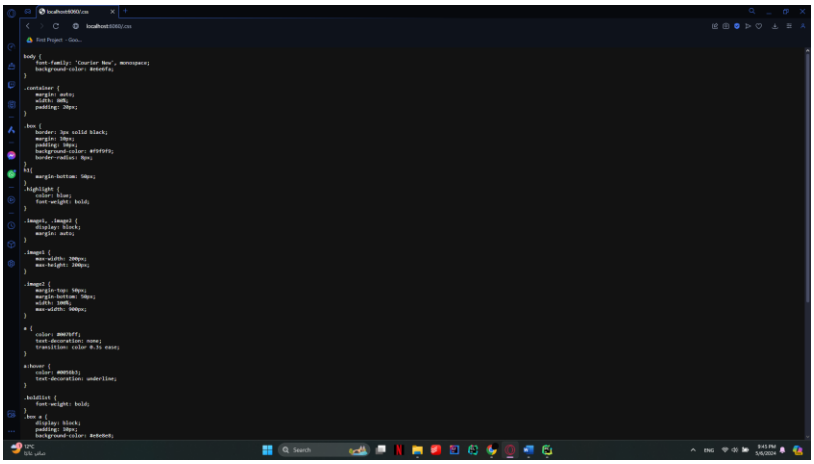




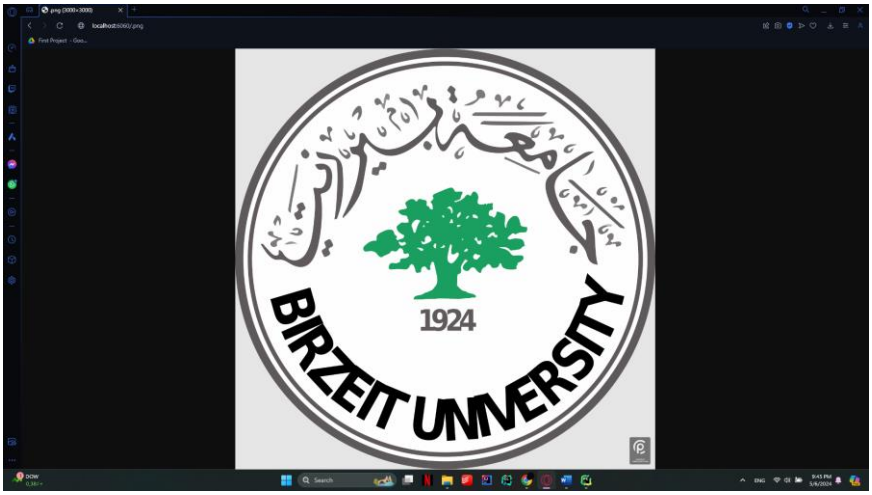
5. if the request is an .**html** file then the server should send the requested html file with Content-Type: text/html. You can use any html file. Make it general (not only for specific filename)



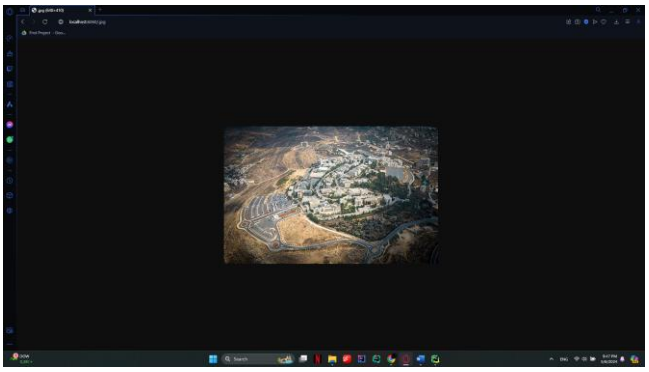
6. if the request is a **.css** file then the server should send the requested css file with Content-Type: text/css. You can use any CSS file. Make it general (not only for specific filename)



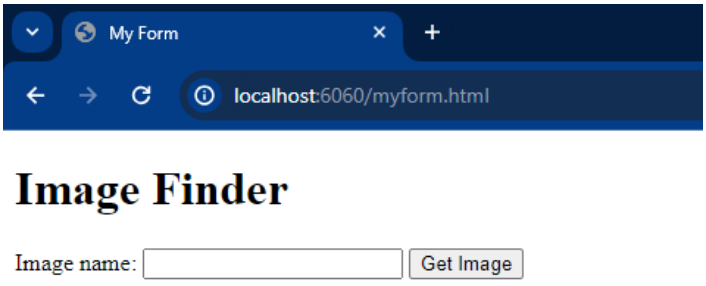
7. if the request is a .**png** then the server should send the png image with Content-Type: image/png. You can use any image. Make it general (not only for specific filename)



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)

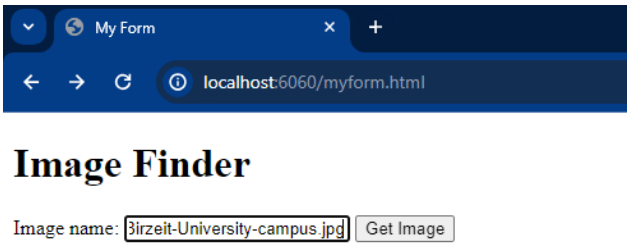


9. Use myform.html to get image by typing the name of the image in a box
   For instance:

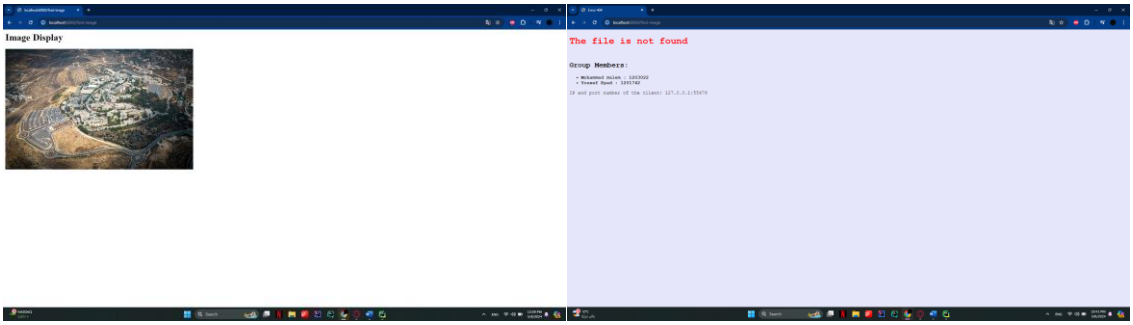


The **"myform.html"** on our web page allows users to enter the name of an image they want to find. When you submit the form by pressing on **“Get Image”** buuton, the name you entered is sent to the server. The server, set up by the **server.py** script, receives this name and looks for the image file 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.

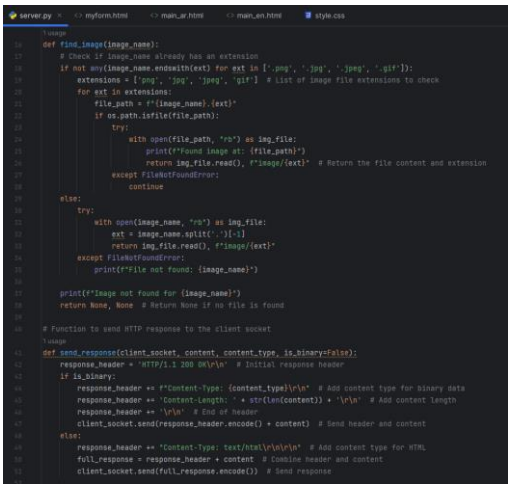
At first we entered the name of the image we want the press **Get Image.**

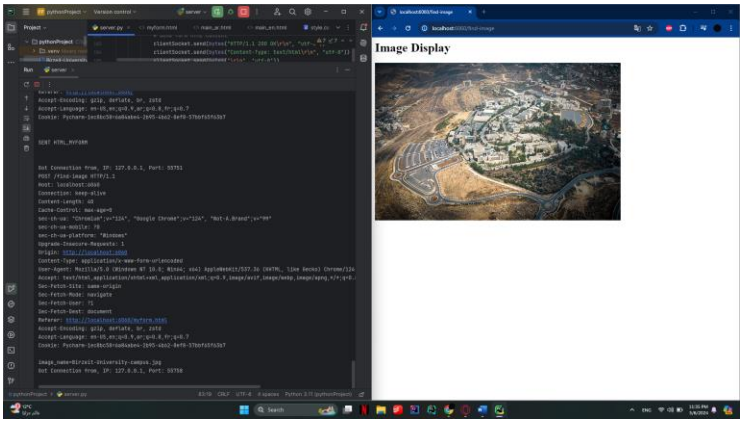


Then the server receives this name and looks for the image file in its storage. If the image exists,the server reads it and send it back to your web browser.The browser then displays the image on the screen. And if the server can't find it, it'll show an error message isted.



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.

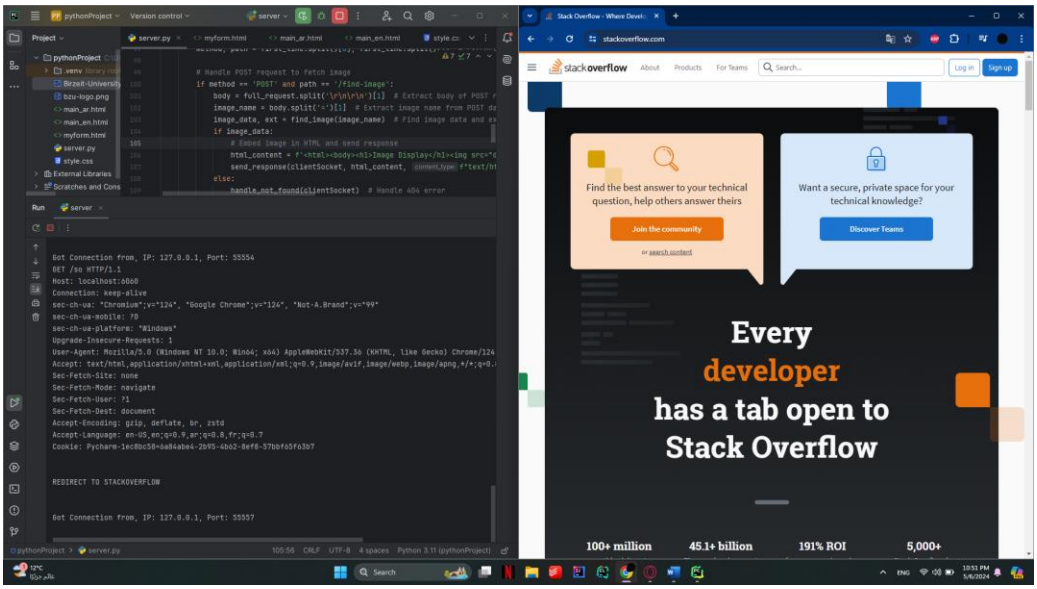


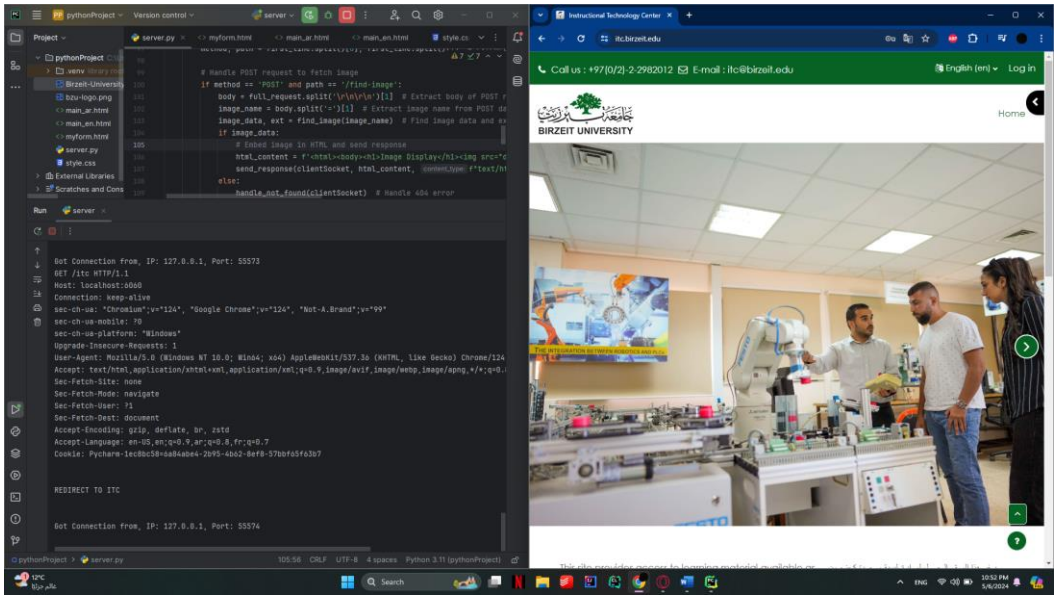



```
# Handle POST request to fetch image
if method == 'POST' and path == '/find-image':
    body = full_request.split('\r\n\r\n')[1]  # Extract body of POST request
    image_name = body.split('=')[1]  # Extract image name from POST data
    image_data, ext = find_image(image_name)  # Find image data and extension
    if image_data:
        # Embed image in HTML and send response
        html_content = f'<html><body><h1>Image Display</h1><img src="{base64.b64encode(image_data).decode()}" /></body></html>'
        send_response(clientSocket, html_content, content_type: f"text/html")
    else:
        handle_not_found(clientSocket)  # Handle 404 error
```

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

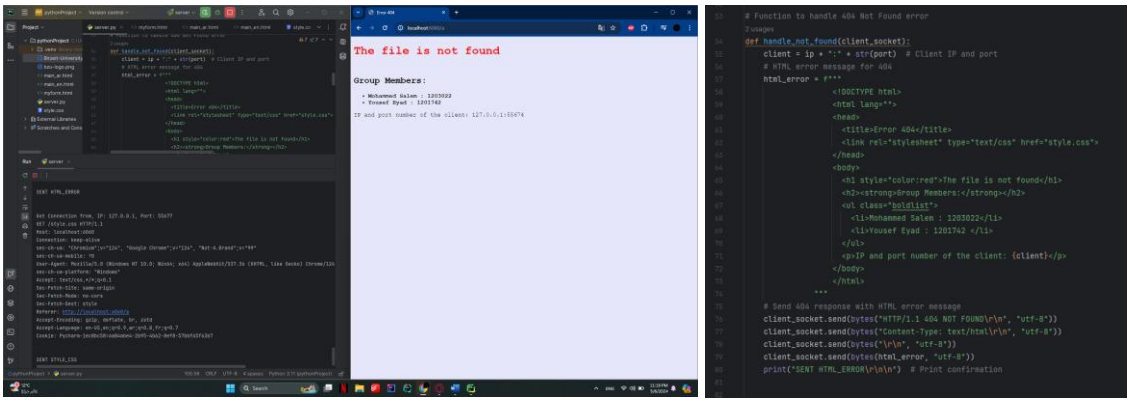
a. If the request is /so then redirect to stackoverflow.com website
b. If the request is /itc then redirect to itc.birzeit.edu website





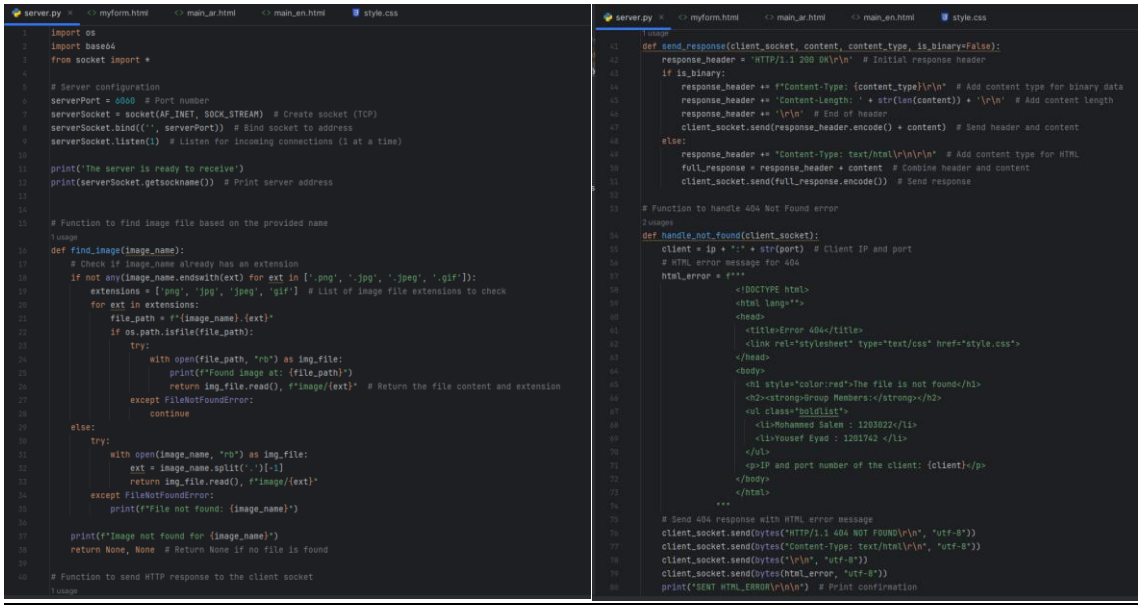
As appear in the screenshot, the server is configured to handle specific routing rules: a request to "localhost:6060/so" automatically redirects to https://stackoverflow.com/, and similarly, a request to "localhost:6060/itc" redirects to https://itc.birzeit.edu/. This configuration ensures efficient 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 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:*



```python
import os
import base64
from socket import *

# Server configuration
serverPort = 6060  # Port number
serverSocket = socket(AF_INET, SOCK_STREAM)  # Create socket (TCP)
serverSocket.bind(('', serverPort))  # Bind socket to address
serverSocket.listen(1)  # Listen for incoming connections (1 at a time)

print('The server is ready to receive')
print(serverSocket.getsockname())  # Print server address


# Function to find image file based on the provided name
1 usage
def find_image(image_name):
    # Check if image_name already has an extension
    if not any(image_name.endswith(ext) for ext in ['.png', '.jpg', '.jpeg', '.gif']):
        extensions = ['png', 'jpg', 'jpeg', 'gif']  # List of image file extensions to check
        for ext in extensions:
            file_path = f"{image_name}.{ext}"
            if os.path.isfile(file_path):
                try:
                    with open(file_path, "rb") as img_file:
                        print(f"Found image at: {file_path}")
                        return img_file.read(), f"image/{ext}"  # Return the file content and extension
                except FileNotFoundError:
                    continue
    else:
        try:
            with open(image_name, "rb") as img_file:
                ext = image_name.split('.')[-1]
                return img_file.read(), f"image/{ext}"
        except FileNotFoundError:
            print(f"File not found: {image_name}")

    print(f"Image not found for {image_name}")
    return None, None  # Return None if no file is found

# Function to send HTTP response to the client socket
1 usage
```

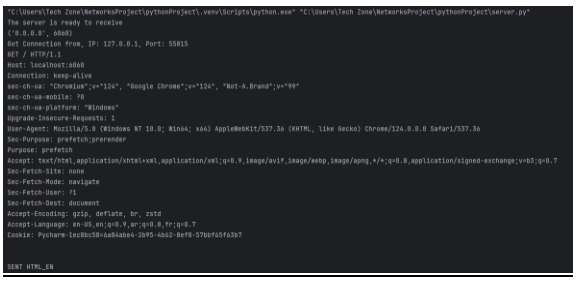
```python
1 usage
def send_response(client_socket, content, content_type, is_binary=False):
    response_header = 'HTTP/1.1 200 OK\r\n'  # Initial response header
    if is_binary:
        response_header += f"Content-Type: {content_type}\r\n"  # Add content type for binary data
        response_header += 'Content-Length: ' + str(len(content)) + '\r\n'  # Add content length
        response_header += '\r\n'  # End of header
        client_socket.send(response_header.encode() + content)  # Send header and content
    else:
        response_header += "Content-Type: text/html\r\n\r\n"  # Add content type for HTML
        full_response = response_header + content  # Combine header and content
        client_socket.send(full_response.encode())  # Send response

# Function to handle 404 Not Found error
2 usages
def handle_not_found(client_socket):
    client = ip + ":" + str(port)  # Client IP and port
    # HTML error message for 404
    html_error = f"""
                    <!DOCTYPE html>
                    <html lang="">
                    <head>
                      <title>Error 404</title>
                      <link rel="stylesheet" type="text/css" href="style.css">
                    </head>
                    <body>
                      <h1 style="color:red">The file is not found</h1>
                      <h2><strong>Group Members:</strong></h2>
                      <ul class="boldlist">
                        <li>Mohammed Salem : 1203022</li>
                        <li>Yousef Eyad : 1201742 </li>
                      </ul>
                      <p>IP and port number of the client: {client}</p>
                    </body>
                    </html>
                  """
    # Send 404 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\n")  # Print confirmation
```

```python
# Main server loop
while True:
    clientSocket, addr = serverSocket.accept()  # Accept incoming connection
    ip = addr[0]  # Client IP
    port = addr[1]  # Client port
    print('Got Connection from, IP: ' + ip + ", Port: " + str(port))  # Print client info

    full_request = clientSocket.recv(1024).decode()  # Receive request from client
    print(full_request)  # Print request

    # Process request
    if len(full_request) != 0:
        lines = full_request.splitlines()  # Split request into lines
        first_line = lines[0]  # First line of request
        method, path = first_line.split()[0], first_line.split()[1]  # Extract method and path

        # Handle POST request to fetch image
        if method == 'POST' and path == '/find-image':
            body = full_request.split('\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><h1>Image Display</h1><img src="{base64.b64encode(image_data).decode()}" /></body></html>'
                send_response(clientSocket, html_content, content_type: f"text/html")
            else:
                handle_not_found(clientSocket)  # Handle 404 error

        # Handle GET requests
        elif path in ['/', '/en', '/index.html', '/main_en.html']:
            # Send main HTML content for English
            with open("main_en.html", "rb") as file:
                file_content = file.read()
            clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type: text/html\r\n", "utf-8"))
            clientSocket.send(bytes("\r\n", "utf-8"))
            clientSocket.send(file_content)
            print("SENT HTML_EN\r\n\n")

        elif path in ['/ar', '/main_ar.html']:
            # Send main HTML content for Arabic
            with open("main_ar.html", "rb") as file:
```

```
server.py  myform.html  main_ar.html  main_en.html  style.css

        elif path in ['/ar', '/main_ar.html']:
            # Send main HTML content for Arabic
            with open("main_ar.html", "rb") as file:
                file_content = file.read()
            clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type: text/html\r\n", "utf-8"))
            clientSocket.send(bytes("\r\n", "utf-8"))
            clientSocket.send(file_content)
            print("SENT HTML_AR\r\n\n")

        # Handle CSS file request
        elif path in ['/style.css', '/.css']:
            with open("style.css", "r") as file:
                file_content = file.read()
            # Send CSS file content
            clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type: text/css\r\n", "utf-8"))
            clientSocket.send(bytes("\r\n", "utf-8"))
            clientSocket.send(bytes(file_content, "utf-8"))
            print("SENT STYLE_CSS\r\n\n")

        # Handle PNG image request
        elif path in ['/bzu-logo.png', '/.png']:
            with open("bzu-logo.png", "rb") as file:
                file_content = file.read()
            # Send PNG image content
            clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type: image/png\r\n", "utf-8"))
            clientSocket.send(bytes("\r\n", "utf-8"))
            clientSocket.send(file_content)
            print("SENT BZU_CIRCLE_PNG\r\n\n")

        # Handle JPEG image request
        elif path in ['/Birzeit-University-campus.jpg', '/.jpg']:
            with open("Birzeit-University-campus.jpg", "rb") as file:
                file_content = file.read()
            # Send JPEG image content
            clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type: image/jpeg\r\n", "utf-8"))
            clientSocket.send(bytes("\r\n", "utf-8"))
            clientSocket.send(file_content)
            print("SENT BZU_UNI_JPEG\r\n\n")
```

```
server.py  myform.html  main_ar.html  main_en.html  style.css

        # Handle JPEG image request
        elif path in ['/Birzeit-University-campus.jpg', '/.jpg']:
            with open("Birzeit-University-campus.jpg", "rb") as file:
                file_content = file.read()
            # Send JPEG image content
            clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type: image/jpeg\r\n", "utf-8"))
            clientSocket.send(bytes("\r\n", "utf-8"))
            clientSocket.send(file_content)
            print("SENT BZU_UNI_JPEG\r\n\n")

        # Handle redirection to external URLs
        elif path == '/itc':
            clientSocket.send(bytes("HTTP/1.1 307 temporary Redirect\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type:\r\n", "utf-8"))
            clientSocket.send(bytes("location: https://itc.birzeit.edu\r\n", "utf-8"))
            print("REDIRECT TO ITC \r\n\n")

        elif path == '/so':
            clientSocket.send(bytes("HTTP/1.1 307 temporary Redirect\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type:\r\n", "utf-8"))
            clientSocket.send(bytes("location: https://stackoverflow.com\r\n", "utf-8"))
            print("REDIRECT TO STACKOVERFLOW\r\n\n")

        # Handle request for form HTML file
        elif path in ['/myform.html', '/myform', '/form', '/.html']:
            with open("myform.html", "rb") as file:
                file_content = file.read()
            # Send form HTML content
            clientSocket.send(bytes("HTTP/1.1 200 OK\r\n", "utf-8"))
            clientSocket.send(bytes("Content-Type: text/html\r\n", "utf-8"))
            clientSocket.send(bytes("\r\n", "utf-8"))
            clientSocket.send(file_content)
            print("SENT HTML_MYFORM\r\n\n")

        else:
            handle_not_found(clientSocket)  # Handle 404 error for unknown paths
    clientSocket.close()  # Close client socket after handling request
```

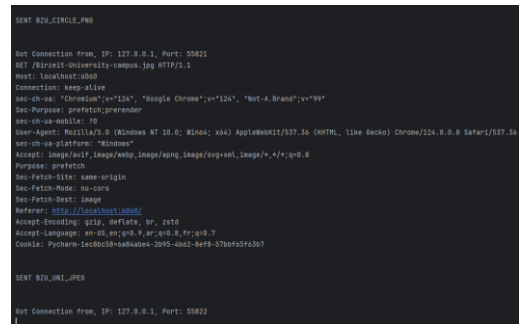
And here a screenshot of the HTTP request printed on the command line.:



```
"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)
Got Connection from, IP: 127.0.0.1, Port: 55815
GET / 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-1ec8bc58-0a84abe4-2b95-4b62-8ef8-57bbf65f63b7


SENT HTML_EN
```

```
Got Connection from, IP: 127.0.0.1, Port: 55819
GET /style.css 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-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
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://localhost:6060/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58-0a84abe4-2b95-4b62-8ef8-57bbf65f63b7


SENT STYLE_CSS
```

```
Got Connection from, IP: 127.0.0.1, Port: 55820
GET /bzu-logo.png 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-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:6060/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58-0a84abe4-2b95-4b62-8ef8-57bbf65f63b7


SENT BZU_CIRCLE_PNG
```



```
SENT BZU_CIRCLE_PNG


Got Connection from, IP: 127.0.0.1, Port: 55821
GET /Birzeit-University-campus.jpg 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-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:6060/
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8,fr;q=0.7
Cookie: Pycharm-1ec8bc58-0a84abe4-2b95-4b62-8ef8-57bbf65f63b7


SENT BZU_UNI_JPEG


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

```
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
```