



Faculty of Engineering & Technology
Electrical and Computer Engineering Department
ENCS3320 Computer Networks
Project Report

Prepared by:

Afaf Amwas	1203359
Dana Asfour	1211924
Mohammed Salem	1200651

Instructor: Abdalkarim Awad

Section: 1 and 3

Date: December 13, 20

Table of Contents

1) Part 1	
- 1.1 Defining The Words Ping, Tracert, Nslookup, and Telnet	1
- 1.2 Running Commands	2
- 1.3 Wireshark Capturing DNS messages	5
2) Part 2	
- 2.1 Socket Programming- TCP Client and Server Application	8
3) Part 3	12

Table of Figures

● Figure 1: Pinging a Device in Same Network	2
● Figure 2: Ping Command	3
● Figure 3: tracert Command	4
● Figure 4: nslookup Command	4
● Figure 5: Search For IP address of Device	5
● Figure 6: Wireshark Program	6
● Figure 7: Wireshark DNS query For Google	6
● Figure 8: Wireshark DNS query For W3Schools	7
● Figure 9: Run the server	9
● Figure 10: Client	10
● Figure 11: Server console	11
● Figure 12: Code for main_en.html Request	12
● Figure 13.1: Html Webpage	13
● Figure 13.2: Html Webpage	13
● Figure 13.3: Html Webpage	14
● Figure 13.4: Request Html File	14
● Figure 13.5: Request master1 CSS file	15
● Figure 13.6: Request Image	15
● Figure 13.7: Request Birzeit Image	16
● Figure 13.8: Request Background Image	16
● Figure 14: Code for main_ar.html Request	17
● Figure 15.1: Arabic Html Webpage	18
● Figure 15.2: Arabic Html Webpage	18
● Figure 15.3: Arabic Html Webpage	19
● Figure 15.4: Request Arabic Html File	19
● Figure 15.5: Request master2 CSS File	20
● Figure 15.6: Request Image 2	20
● Figure 15.7: Request Al-Quds image	21
● Figure 15.8 Request Background Image 2	21
● Figure 16: Code for .html Request	22

● Figure 16.1: Request Html File 2	23
● Figure 16.2: Request Html File 2	23
● Figure 16.3: Request Html File 2	24
● Figure 18: Code for .css Request	25
● Figure 18.1 CSS Code	26
● Figure 19: Code for .png Request	27
● Figure 20: Code for .jpg Request	28
● Figure 21: Code for Temporary Redirect	29
● Figure 21.1: Firefox Redirect	29
● Figure 21.2: Cornell Redirect	30
● Figure 21.3: StackOverflow Redirect	30
● Figure 21.4: Ritaj Redirect	31
● Figure 22.1: Error Webpage	32
● Figure 22.2: Error Webpage	32
● Figure 22.3: Error Webpage	33
● Figure 22.4: Error Webpage	33
● Figure 22.5: Error Webpage	34

Part 1

1.1 Defining the Words Ping, Tracert, Nslookup, and Telnet

- Ping: (Packet internet groper) A command line used to check if a server is reachable by sending a request and waiting for a response, while measuring the round-trip time (RTT) if a response is received, it is also used to test host/server are reachable, internet connection, and adapter functional correctly.
- Tracert: Traceroute is a command line that shows the IP addresses of the routers that are on the path that packets take from source to destination.
- Nslookup: A command line used to get information about the DNS (Domain Name Server) server, either by translating the DNS to its corresponding IP address or vice versa, it is usually used to determine the location of the server.
- Telnet: A command line that allows the user to connect to another computer or server over a network and execute commands remotely, sending data in clear text.

1.2 Running Commands

Figure 1 shows the pinging of a device in the same network. First, an Ubuntu terminal is opened to get the IP address of the Ubuntu computer using the command line ‘ifconfig’. As seen in the figure the IP address is 192.168.56.101. Then, a windows terminal is opened, and the command ‘ping 192.168.56.101’ is used to check if the device with IP address ‘192.168.56.101’ (Ubuntu machine) is reachable or not. From the figure below, it is clear that the device is reachable because a reply was obtained about the packets, such as the round-trip time for each packet, in this case (1ms).

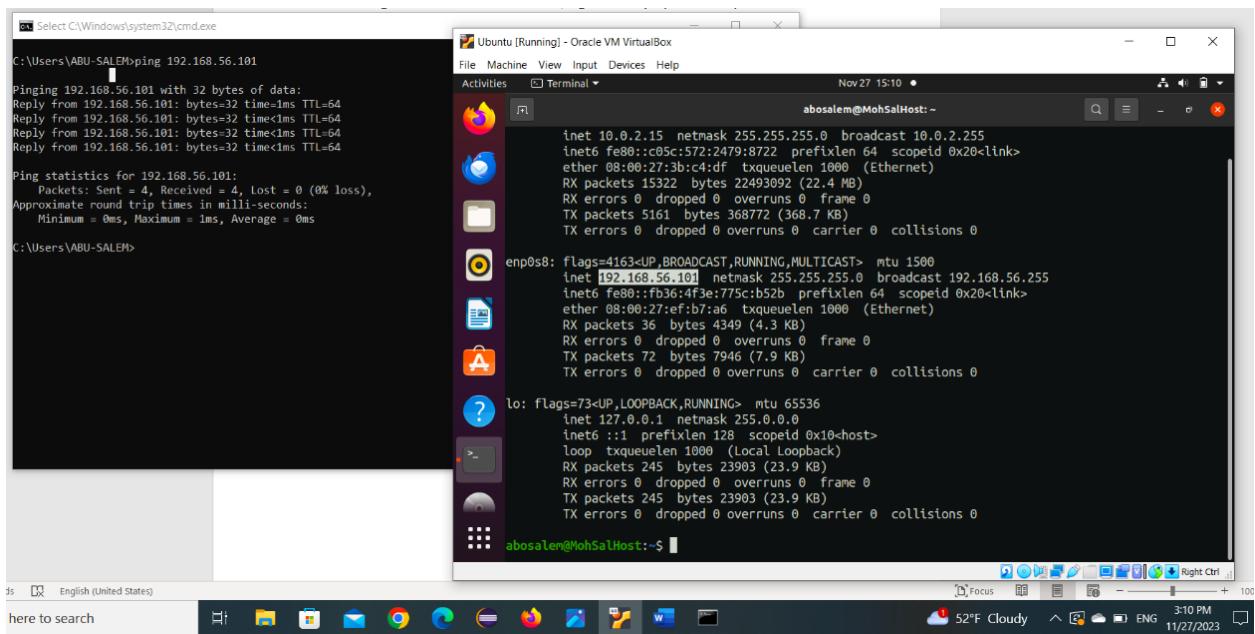
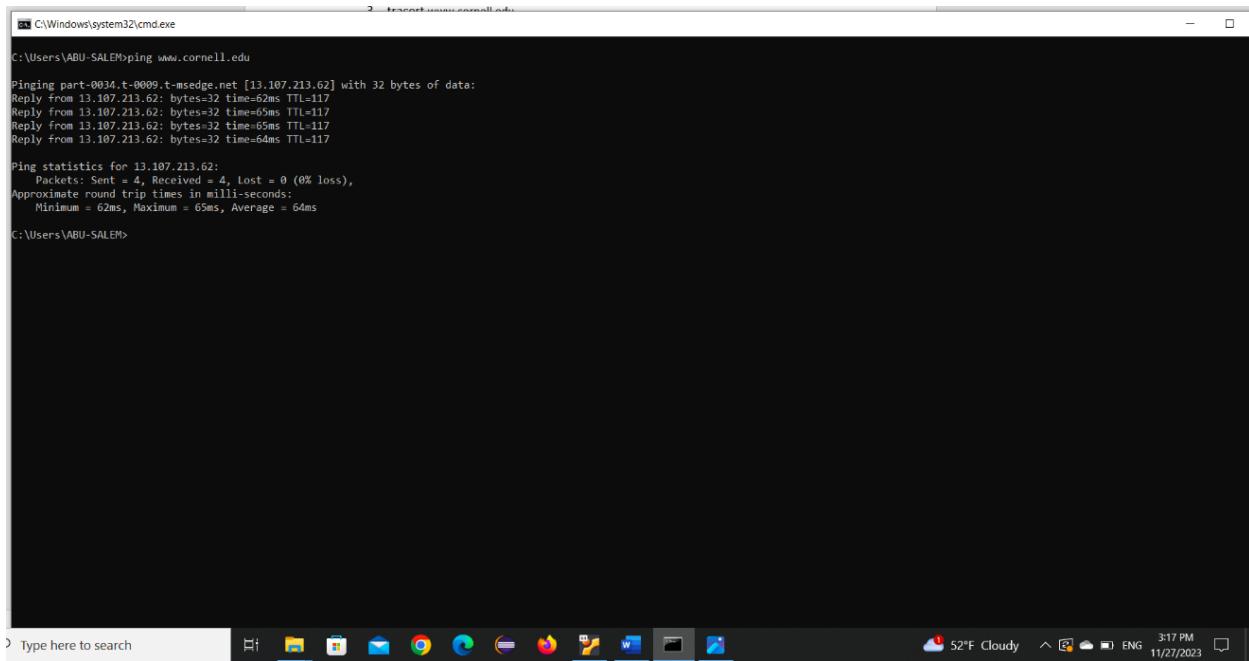


Figure 1: Pinging a Device in Same Network

In figure 2, the command line ‘ping www.cornell.edu’ will send a request to the domain www.cornell.edu. Since a response was received, the host “www.cornell.edu” is reachable. This number “13.107.213.62” is the IP address of the server that is responding back. The size of the request packet is 32 bytes that is sent from the computer source to the server. The server responds back with each packet being 32 bytes. Each response shows the time to live (TTL) which is the maximum number of routers the packet travels on, in this case 117. Moreover, the round-trip time (RTT) is also shown, which is the time it takes for a packet to travel from the

source to the destination and back, in this case 65ms (in maximum). Ping statistics are shown that present the number of packets being sent (4 packets) and received (4 packets). As seen in the figure, the percentage of the packets lost is 0% and the minimum, maximum and average round trip time is also shown.



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command entered is 'ping www.cornell.edu'. The output shows four successful replies from the target IP address 13.107.213.62, each with a TTL of 117 and a round-trip time between 62ms and 65ms. The statistics summary indicates 0% loss and an average round-trip time of 64ms. The taskbar at the bottom shows various application icons and the system clock indicating 3:17 PM on 11/27/2023.

```
C:\Windows\system32\cmd.exe
C:\Users\ABU-SALEM>ping www.cornell.edu

Pinging part-0034.t-0009.t-msedge.net [13.107.213.62] with 32 bytes of data:
Reply from 13.107.213.62: bytes=32 time=62ms TTL=117
Reply from 13.107.213.62: bytes=32 time=65ms TTL=117
Reply from 13.107.213.62: bytes=32 time=65ms TTL=117
Reply from 13.107.213.62: bytes=32 time=64ms TTL=117

Ping statistics for 13.107.213.62:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 62ms, Maximum = 65ms, Average = 64ms

C:\Users\ABU-SALEM>
```

Figure 2: Ping Command

In figure 3, the traceroute command (`tracert`) will trace the route that packets take from the source to destination, measuring the number of hops or routers that packets travel on. However, there is a maximum number of hops on the route as a limit for packets to reach their destination. Each row represents a hop with its IP address and the round-trip time. For Example, at the 4th hop, the router has an IP address of 10.74.42.234, and 3 packets with round trip times 23ms, 20ms and 19ms. As can be seen in the figure, there are some rows that have asterisks instead of the intended information. This means the router is not responding to the request within time. At the end of the command result, a message is displayed “trace complete” that indicates the route was traced successfully.

```
C:\Windows\system32\cmd.exe
C:\Users\ABU-SALEM>tracert www.cornell.edu
Tracing route to part-0034.t-0009.t-msedge.net [13.107.246.62]
over a maximum of 30 hops:
  1   3 ms    7 ms    8 ms  superfast [192.168.2.1]
  2   *         *         * Request timed out.
  3   23 ms   20 ms   20 ms  10.74.22.1
  4   23 ms   20 ms   19 ms  10.74.42.234
  5   64 ms   62 ms   62 ms  ae61-0.iec92.tlv30.ntwk.msn.net [104.44.36.229]
  6   74 ms   62 ms   63 ms  13.104.140.42
  7   *         *         * Request timed out.
  8   66 ms   62 ms   62 ms  13.107.246.62

Trace complete.

C:\Users\ABU-SALEM>
```

Figure 3: tracert Command

In figure 4, the nslookup command is shown in both Ubuntu and Windows terminal. This command gives more details on the Domain Name System (DNS). First, information about the server is given such as the server and address. In the Ubuntu terminal there were many canonical names for the server www.cornell.edu which are alias names for the domain name. In the Windows terminal this can be seen under the section Aliases. In both terminals, different IP addresses are shown. This can occur if the domain is using clustered servers.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ABU-SALEM>nslookup www.cornell.edu
Server:  superfast
Address: 192.168.2.1

Non-authoritative answer:
Name:  part-0034.t-0009.t-msedge.net
Addresses:  2620:1ec:bdf:62
           13.107.246.62
           13.107.213.62
Aliases:  www.cornell.edu
          cornell-edge-ekhkdhg5czdnib2bf.z01.azurefd.net
          star-azurefd-prod.trafficmanager.net
          shed.dual-low.part-0034.t-0009.t-msedge.net

C:\Users\ABU-SALEM>

Ubuntu (Running) - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal abosalem@MohSalHost: ~
Nov 27 15:20
abosalem@MohSalHost: ~ nslookup www.cornell.edu
Server:  127.0.0.53#53
Address: 127.0.0.53#53

Non-authoritative answer:
www.cornell.edu canonical name = cornell-edge-ekhkdhg5czdnib2bf.z01.azurefd.net.
cornell-edge-ekhkdhg5czdnib2bf.z01.azurefd.net canonical name = star-azurefd-prod.trafficmanager.net.
star-azurefd-prod.trafficmanager.net canonical name = shed.dual-low.part-0034.t-0009.t-msedge.net.
shed.dual-low.part-0034.t-0009.t-msedge.net canonical name = part-0034.t-0009.t-msedge.net.
Name:  part-0034.t-0009.t-msedge.net
Address: 13.107.213.62
Name:  part-0034.t-0009.t-msedge.net
Address: 13.107.246.62
Name:  part-0034.t-0009.t-msedge.net
Address: 2620:1ec:bdf:62
Name:  part-0034.t-0009.t-msedge.net
Address: 2620:1ec:46::62

abosalem@MohSalHost: ~
```

Figure 4: nslookup Command

1.3 Wireshark Capturing DNS messages

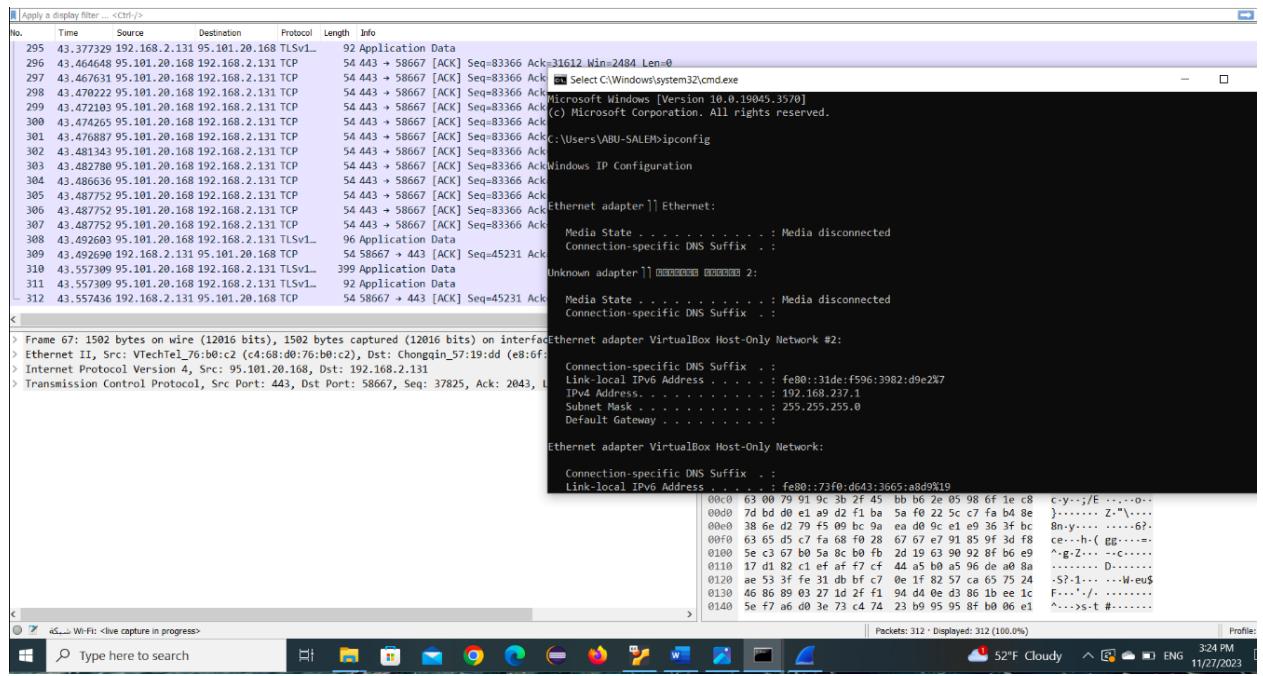


Figure 5: Search for IP address of Device

Wireshark is like an eavesdropper that listens to the messages being sent and received over the network. It helps us understand how devices communicate with each other by showing information about the packets being sent and received in real time. This information includes the packet number, time, IP address of the source and destination, the protocol type, length of the packet in bytes, and finally the info column that provides a description of the packet.

As seen in figure 5, a search for the IP address of the device being used is done on the terminal using ipconfig command. In figure 6, the filter bar on the wireshark program is being used to display only the packets that involve DNS communication, such as DNS queries and responses. As seen in figure 6, the IP address of the device is shown which is 192.168.2.131. In the wireshark program, a packet is being sent to this IP address shown in the destination column of the highlighted line.

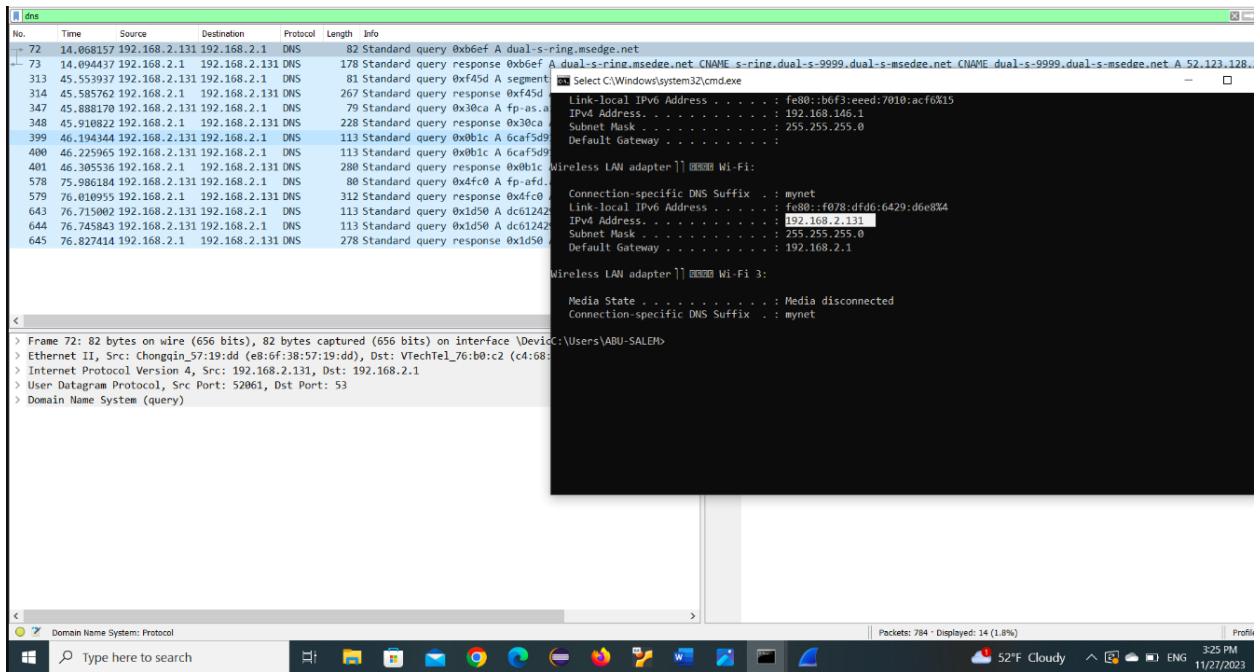


Figure 6: Wireshark Program

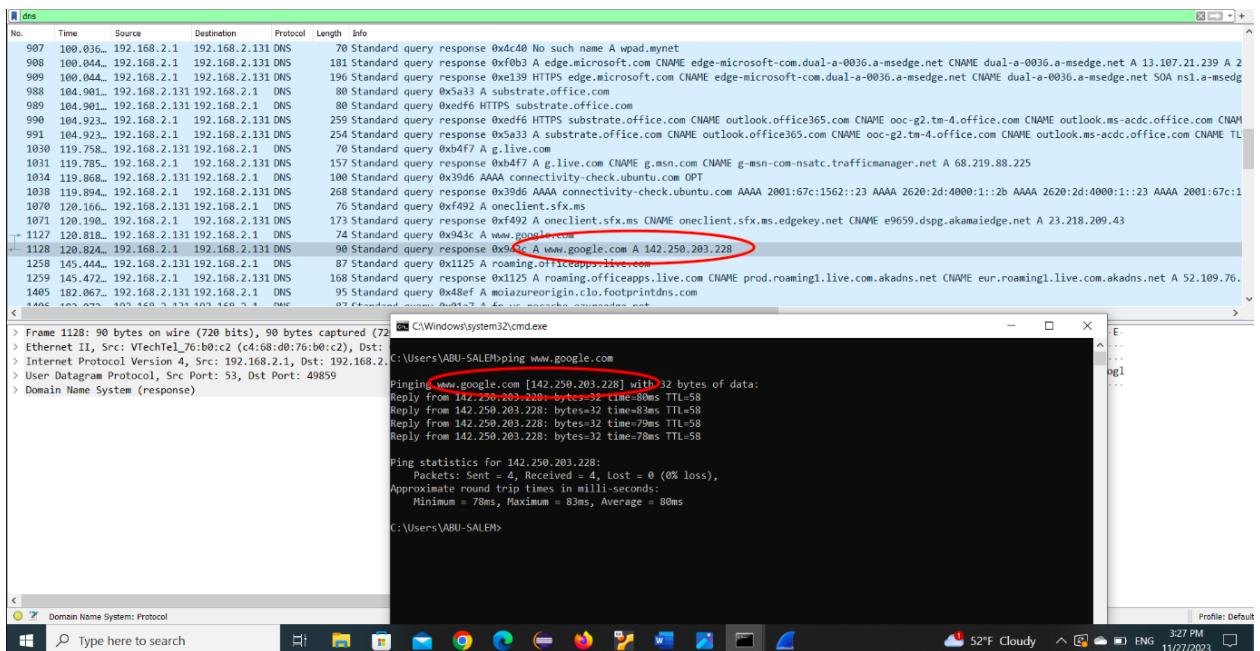


Figure 7: Wireshark DNS query For Google

The function of a DNS server is to translate the domain name to its corresponding IP address and vice versa. This is crucial because humans remember domain names easier than a big number (IP address), and devices on the network operate using IP addresses. It's important to

note that DNS services are not only implemented in proxy servers, but also can be found in home routers. As seen in figure 7, wireshark is capturing DNS messages in real time. Here visiting the Google website (google server IP address of 142.250.203.228) is captured in the wireshark program, shown in the info column. Figure 8 shows another instance where the wireshark program captures DNS messages, in this case visiting W3schools.

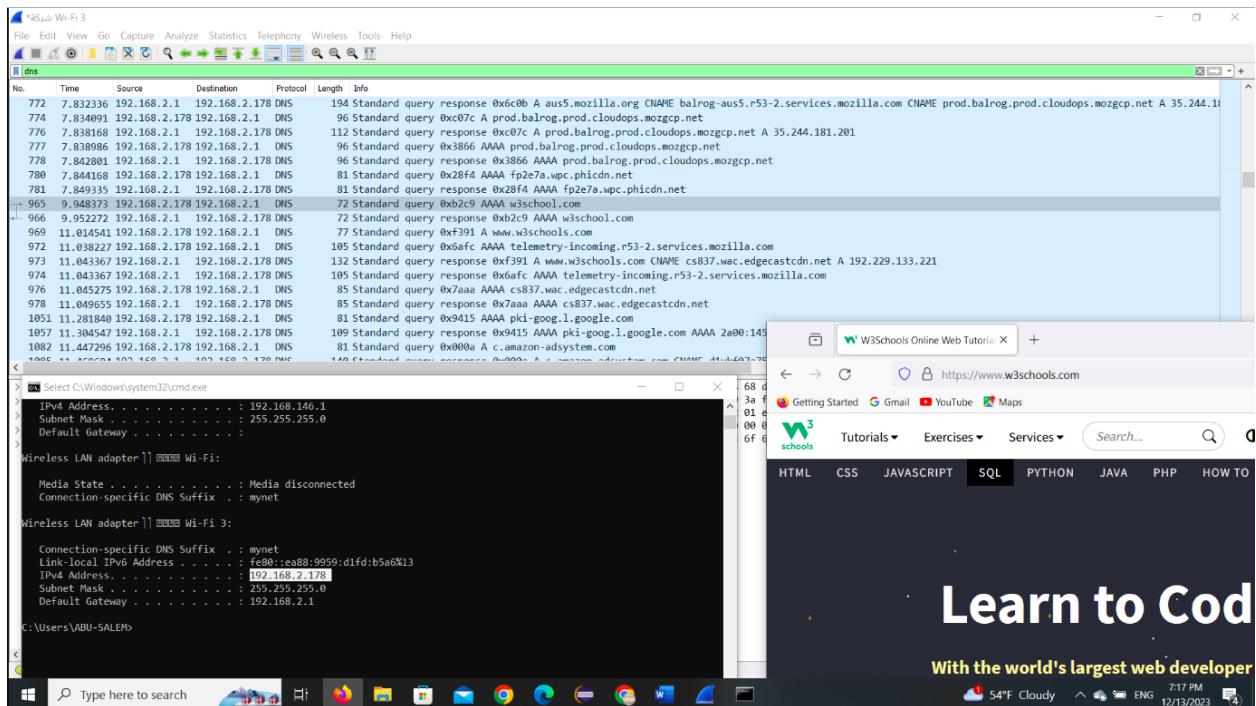


Figure 8: Wireshark DNS query For W3Schools

From the ping results, do you think the response you have got is from the USA? explain your answer briefly

As seen in figure 2, pinging to cornell.edu website takes 117 TTL (Time to Live) which corresponds to 117 routers, whereas pinging to a device locally takes 64 TTL as seen in figure 1. Locally, there are fewer hops (64 TTL) which is a much shorter path than the path when pinging cornell.edu (117 TTL). This suggests that the response from the ping results in figure 2 is in fact from the USA.

Part 2

2.1 Socket Programming- TCP Client and Server Application

Using socket programming, we have implemented a TCP client and server application in Python. The server is listening on port 9955 and the server waits for a message from the client. If the message is with one of our student's ID's, then it will do a series of steps. It will display a message on the server side that the OS will lock the screen in 10 seconds. Then, it sends a message to the client that the server will lock the screen after 10 seconds. It will wait 10 seconds and lock the screen of the operating system.

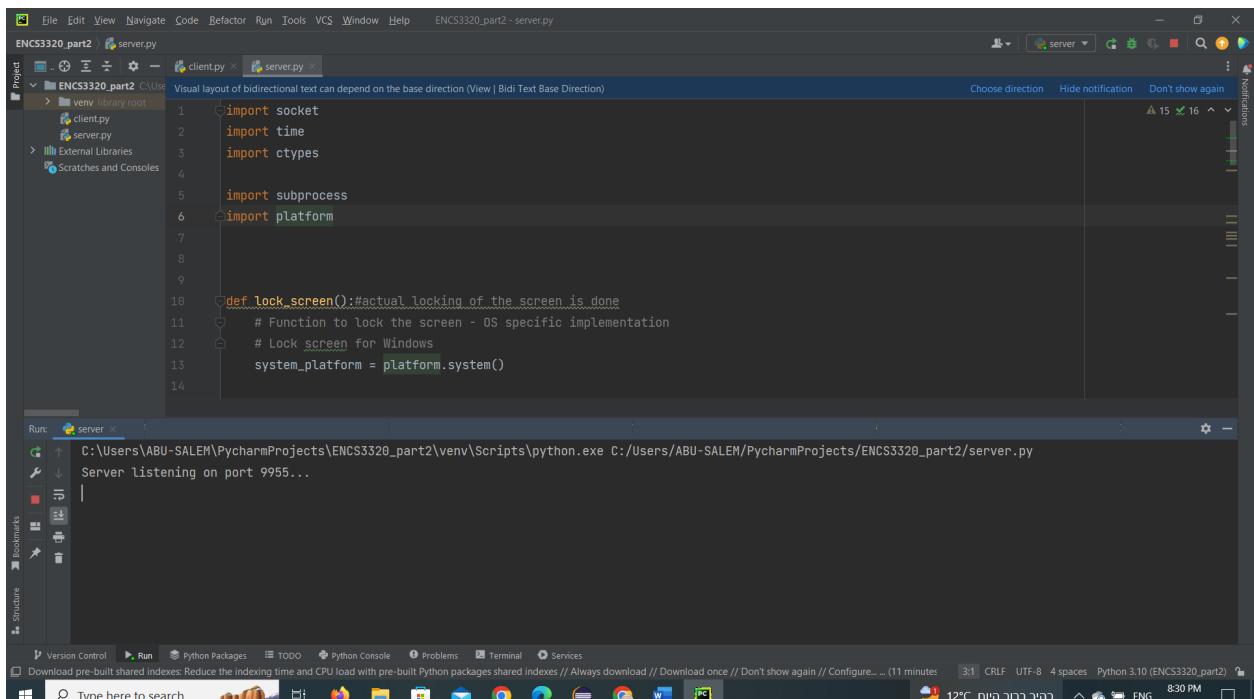
Here's an explanation of what the code does:

- Imports: It imports the socket module, which provides a low-level interface to network sockets.
- Server Address: It defines the server's IP address ('127.0.0.1') and port number (9955) in the server_address tuple.
- Student IDs: It defines three student IDs (student_id1_Dana, student_id2_Mohammad, student_id3_Masa) that will be sent to the server. In this example, only student_id1_Dana is used.
- Socket Creation and Connection: It creates a new socket using `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`.
It connects to the server using `client_socket.connect(server_address)`.
- Sending Student ID: It sends the student ID (student_id1_Dana) to the server using `client_socket.send(student_id1_Dana.encode())`.
The encode() method is used to convert the string to bytes, as sockets transmit data in bytes.
- Receiving Response: It receives the server's response using `response = client_socket.recv(1024).decode()`.

- The `recv(1024)` method reads up to 1024 bytes from the socket, and `decode()` converts the received bytes to a string.
- Printing Response: It prints the received response to the console using `print(response)`.

In the context of a larger client-server application, the server would be designed to handle incoming connections, receive and process the student ID, and send an appropriate response back to the client. The response could be a confirmation, an error message, or any other relevant information.

First, we run the server (in figure 9), then in the specific code snippet, when run, will send the student ID '1200651' to the server, receive a response, and print that response to the console, as shown in figure 10.



The screenshot shows the PyCharm IDE interface. The project structure on the left includes a 'client.py' file and a 'server.py' file under the 'ENC3320_part2' directory. The 'server.py' file is open in the editor, displaying the following code:

```

1 import socket
2 import time
3 import ctypes
4
5 import subprocess
6 import platform
7
8
9
10 def lock_screen():#actual locking of the screen is done
11     # Function to lock the screen - OS specific implementation
12     # Lock screen for Windows
13     system_platform = platform.system()
14

```

The 'Run' tab at the bottom shows the command: `C:/Users/ABU-SALEM/PycharmProjects/ENC3320_part2/venv/Scripts/python.exe C:/Users/ABU-SALEM/PycharmProjects/ENC3320_part2/server.py`. The output window shows: "Server listening on port 9955...". The status bar at the bottom right indicates: 3:1 CRLF UTF-8 4 spaces Python 3.10 (ENC3320_part2) 12°C ENG 8:30 PM.

Figure 9: Run the server

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and ENCS3320_part2 - client.py. The Project tool window on the left shows a file tree for 'ENCS3320_part2' with 'client.py' and 'server.py' selected. The main editor window contains the following Python code:

```

import socket

server_address = ('127.0.0.1', 9955)

# Replace 'YOUR_STUDENT_ID' with a valid student ID
student_id1_Dana = '12111924'
student_id2_Mohammad = '1200651'
# student_id3_Nasa = '1203359'

#test_str = "\r\n"

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    client_socket.connect((server_address))

```

The Run tool window at the bottom shows the command: C:\Users\ABU-SALEM\PycharmProjects\ENCS3320_part2\venv\Scripts\python.exe C:/Users/ABU-SALEM/PycharmProjects/ENCS3320_part2/client.py. The output pane displays: Server: Locking screen in 10 seconds... and Process finished with exit code 0.

Figure 10: Client

Figure 10 shows the code of the client side creating a socket using ‘socket.socket()’ function’. We also defined the three acceptable student IDs which are 12111924, 1200651, and 1203359. When creating a socket, we used IPv4 (internet protocol version 4) addressing and a TCP connection indicated in the code by `socket.AF_INET` and `socket.SOCK_STREAM` respectively.

Here's an explanation of what the server.py code does:

- Ports: The code starts by importing necessary modules such as `socket`, `time`, `ctypes`, `subprocess`, and `platform`.
- Lock Screen Function: The `lock_screen()` function locks the screen based on the operating system. It uses platform-specific commands for Windows, Linux, and macOS.
- Handle Client Function: The `handle_client()` function takes a client socket and data as parameters. It checks if the received data (presumably a student ID) matches the valid

student ID ('1200651'). If it's a valid ID, it prints a message on the server side, sends a message to the client, waits for 10 seconds, locks the screen, and notifies the client that the screen is locked. If the ID is invalid, it prints an error message.

- Server Setup: The main server code sets up a socket, binds it to address ('127.0.0.1', 9955), and starts listening for incoming connections. It enters an infinite loop where it accepts a client connection, receives data from the client, and calls handle_client() to process the data. The client socket is closed after processing each connection.
- Server Execution: When you run this script, the server will continuously listen for client connections on port 9955. When a client connects and sends data (presumably a student ID), the server will process the data based on the conditions outlined in the handle_client() function.

```

ENC3320_part2 > client.py
Project  Ele Edit View Navigate Code Refactor Run Tools VCS Window Help ENC3320_part2 - client.py
ENC3320_part2 > client.py  server.py
client.py
1 import socket
2
3 server_address = ('127.0.0.1', 9955)
4
5 # Replace 'YOUR_STUDENT_ID' with a valid student ID
6 student_id1_Dana = '1211924'
7 student_id2_Mohammad = '1200651'
8 # student_id3_Nasa = '1203359'
9
10 #test_str = "\r\n"
11
12 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
13
14     client_socket.connect((server_address))
15
server > client
C:\Users\ABU-SALEM\PycharmProjects\ENCS3320_part2\venv\Scripts\python.exe C:/Users/ABU-SALEM/PycharmProjects/ENCS3320_part2/server.py
Server listening on port 9955...
Received data: 1200651
Invalid student ID or text received.
Received student ID: 1200651
Locking screen...
Invalid student ID or text received.

Run:  Version Control  Python Packages  TODO  Python Console  Problems  Terminal  Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // Download once // Don't show again // Configure... (12 minute)
7:18  CRLF  UTF-8  4 spaces  Python 3.10 (ENCS3320_part2)

```

Figure 11: Server console

As can be seen in figure 11, the output shows that the server is listening on port 9955. The received data is the number 1200651 which is also the received student ID. As mentioned above after 10 seconds the screen will print a message saying “locking screen” in which the screen locks.

Part 3

We have created a web server listening to port 9966 using python. In our code we have created a socket using the AF_INET address family and SOCK_STREAM that specifies a TCP connection. Then we used the bind () method to connect the serverSocket with the port 9966. We used the empty string ‘’ to let the server accept any available network. Then we used the listen () method to let the server listen to incoming messages.

When the user types in the browser http://localhost:9966/main_en.html or <http://localhost:9966/en> the program will check if the request is / or /index.html or /main_en.html or /en then the server sends the main_en.html file with content type: text/html. This code is shown in figure 12.

```
# part3 branch 1
if requestedFile == '/' or requestedFile == '/index.html' or requestedFile == '/main_en.html' or \
    requestedFile == '/en':
    try:
        #print(requestedFile)
        main_en_path = "main_en.html"
        main_en_file = open(main_en_path, "r")
        html_msg = main_en_file.read()

        response = "HTTP/1.1 200 OK\r\n" # http response
        response += "Content-Type: text/html\r\n" # the kind of message is html message or text/plain or image/png
        response += "\r\n"
        response += html_msg
        connectionSocket.send(response.encode())
        connectionSocket.close() # to close the sentence => (close Connection)

    except FileNotFoundError:
        print(f"Error: '{main_en_path}' not found. Please check the file path.")
    except IOError as e:
        print(f"Error reading file: {e}")
#####
```

Figure 12: Code for main_en.html Request

Figure 12 shows the html file that contains “ENCS3320-My Tiny Webserver 23/24” in the title. It also contains different boxes explaining who we are as students. The decorations on this page are done inside a separate CSS file. As can be seen in the figure, the content type in the

HTTP request is summarized in a box. There are also two links available, one is a local html file, and another is w3schools website.

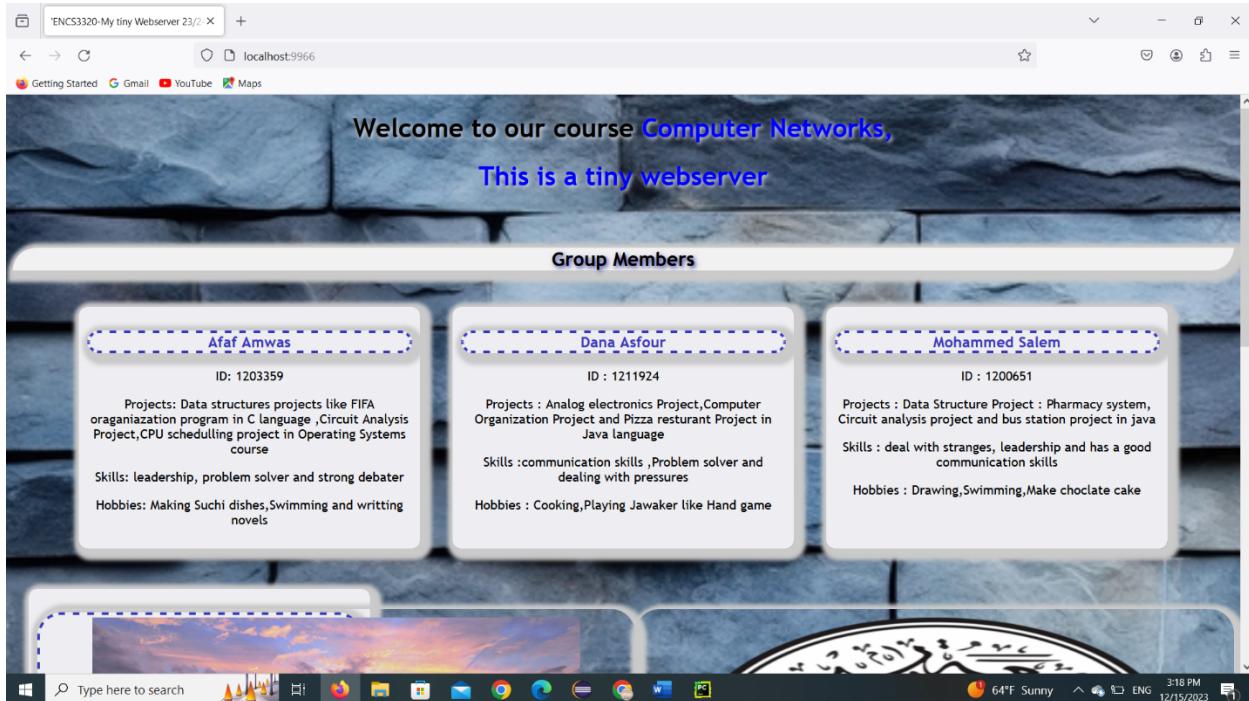


Figure 13.1: Html Webpage

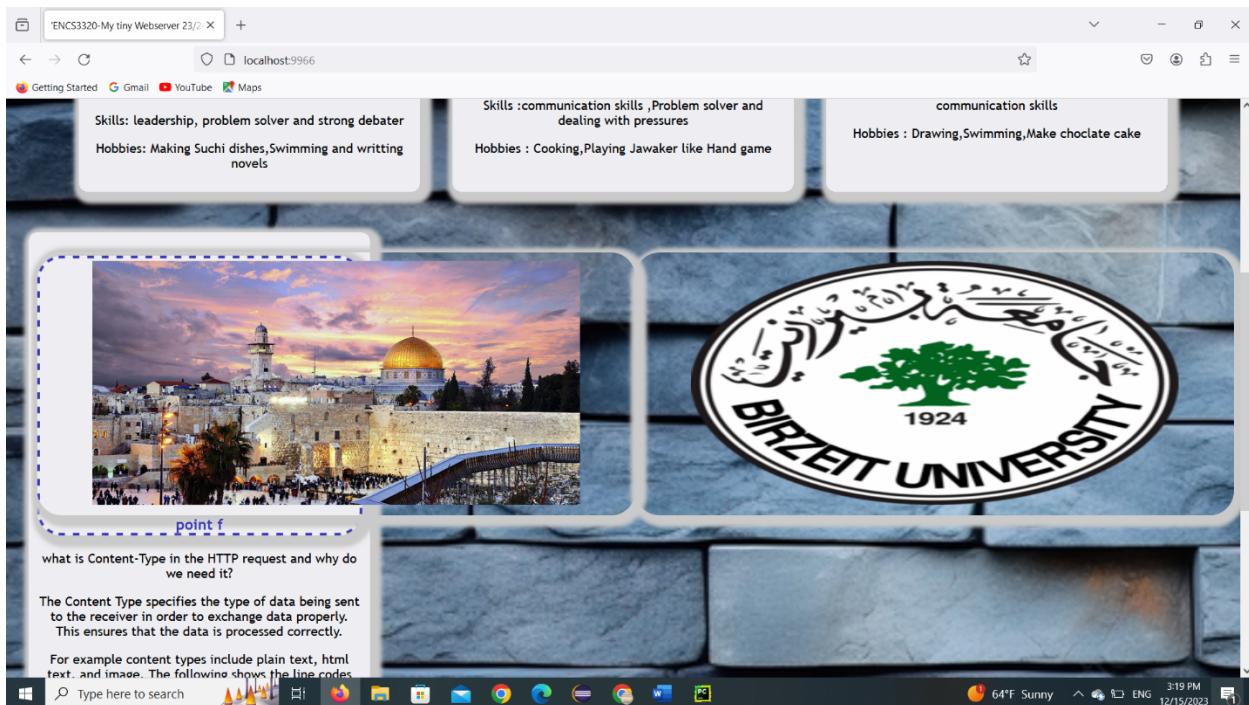


Figure: 13.2: Html Webpage

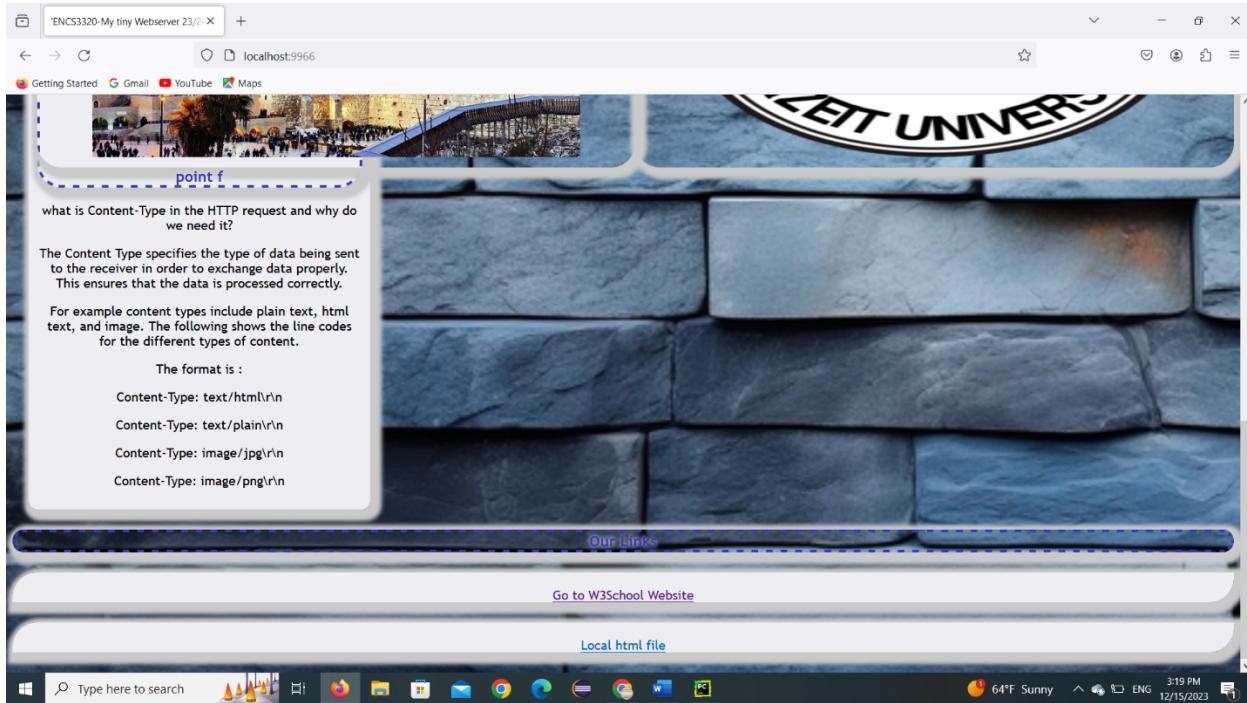


Figure 13.3: Html Webpage

Here http request appears in python console, Firefox request the main_en.html file

The screenshot shows a PyCharm IDE interface with a terminal window. The terminal output displays an incoming HTTP request from a Firefox browser. The request details include the client IP address (127.0.0.1), port number (62019), and the full HTTP header with various fields like Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Connection, Cookie, Upgrade-Insecure-Requests, Sec-Fetch-Dest, Sec-Fetch-Mode, Sec-Fetch-Site, and Sec-Fetch-User.

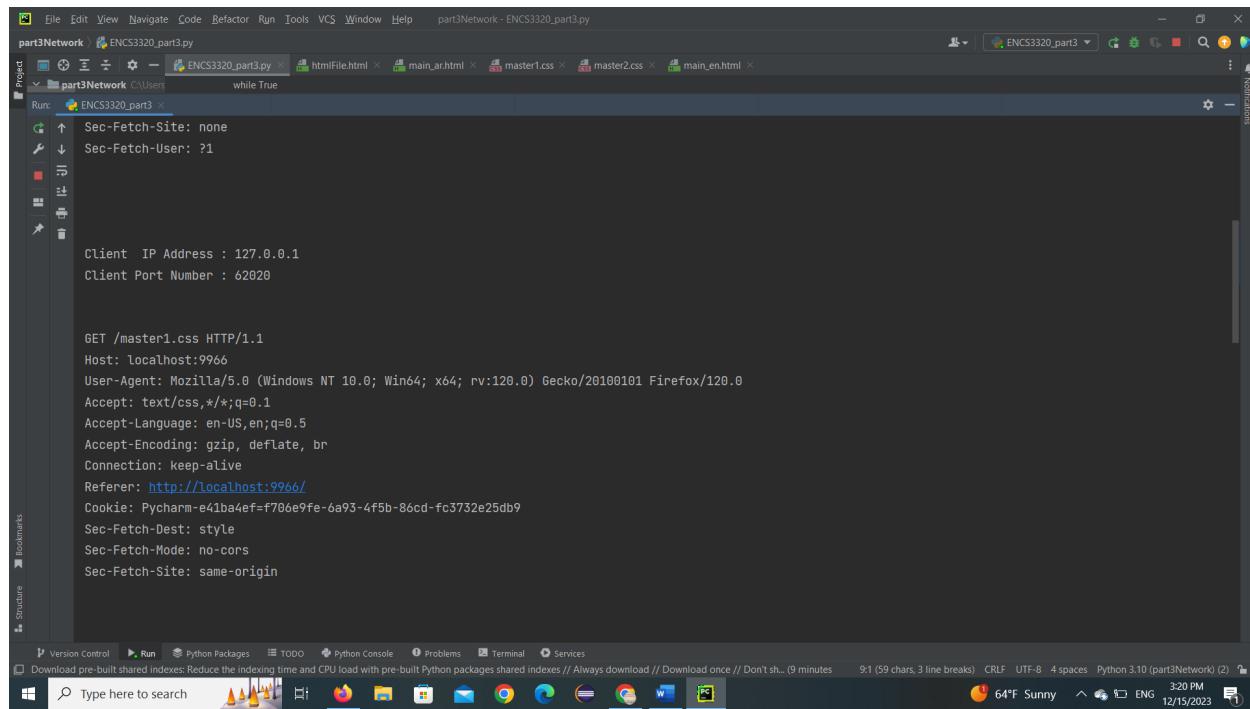
```

Client IP Address : 127.0.0.1
Client Port Number : 62019

GET / HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
  
```

Figure 13.4: Request Html File

Here CSS file request appears in python console, Firefox request the master1.css file

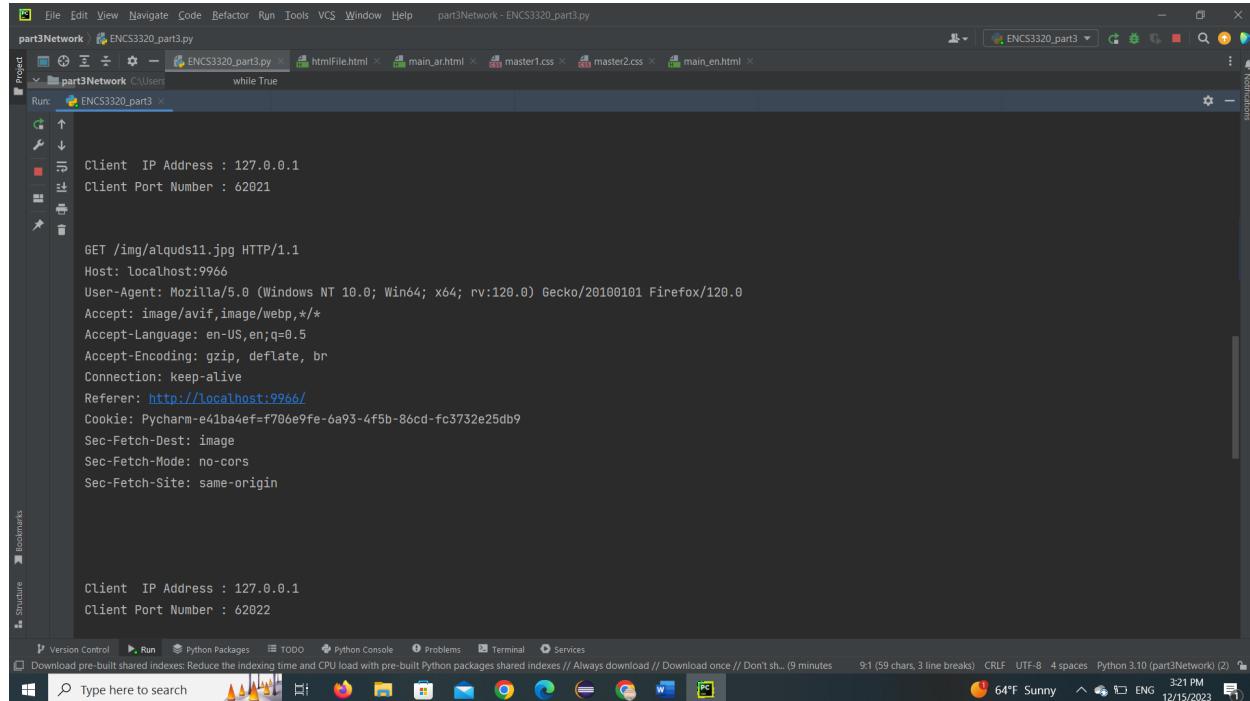


```
Client IP Address : 127.0.0.1
Client Port Number : 62020

GET /master1.css HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: text/css,*/*;q=0.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: style
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Figure 13.5: Request master1 CSS file

Here jpg image request appears in python console, Firefox request the alquds11.jpg

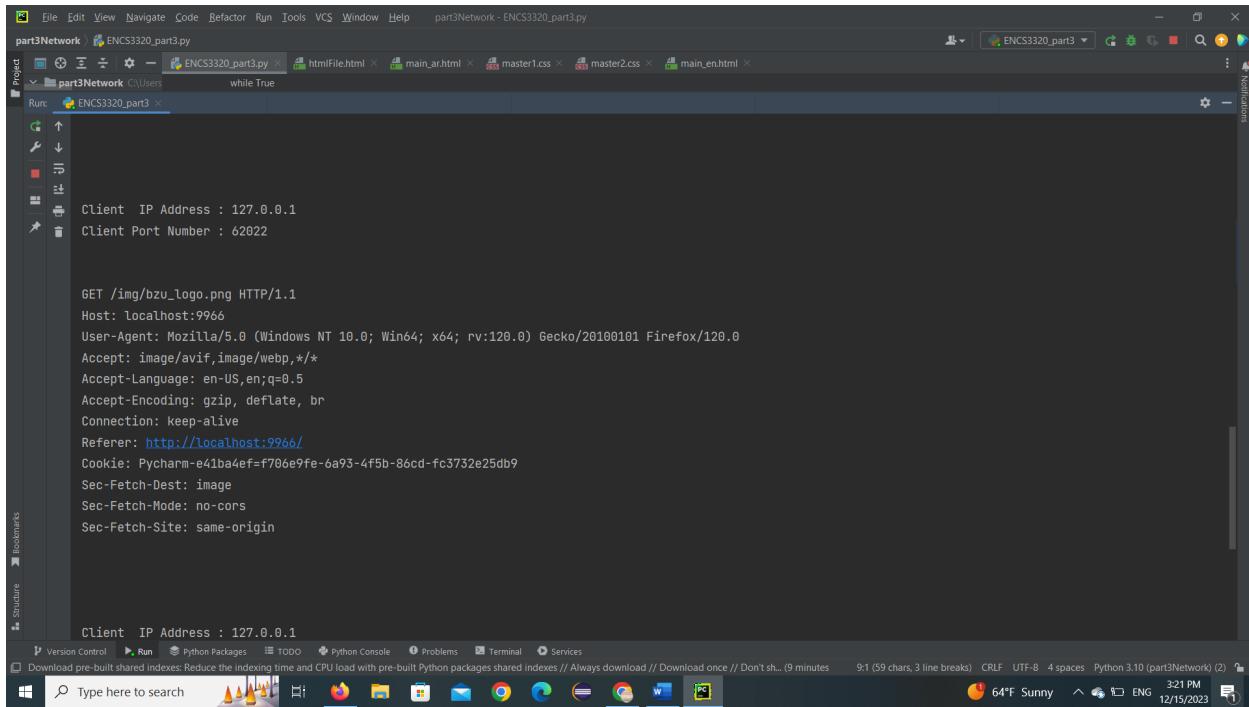


```
Client IP Address : 127.0.0.1
Client Port Number : 62021

GET /img/alquds11.jpg HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Figure 13.6: Request Image

Here png image request appears in python console, Firefox request the bzu_logo.png



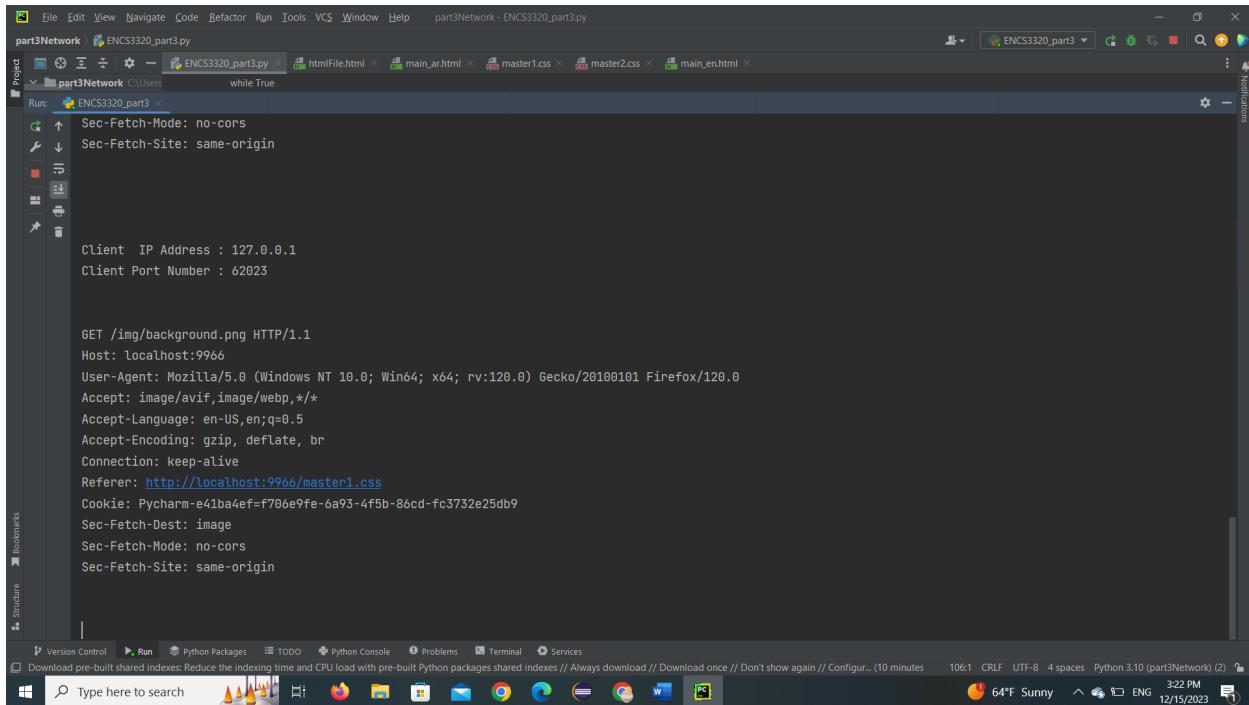
Client IP Address : 127.0.0.1
Client Port Number : 62022

```
GET /img/bzu_logo.png HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/
Cookie: Pycharm-e41ba4ef-f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Client IP Address : 127.0.0.1

Figure 13.7: Request Birzeit Image

Here image request appears in python console, Firefox request the background.png



Client IP Address : 127.0.0.1
Client Port Number : 62023

```
GET /img/background.png HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/master1.css
Cookie: Pycharm-e41ba4ef-f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Figure 13.8: Request Background Image

When the user types in the browser http://localhost:9966/main_ar.html or <http://localhost:9966/ar> the program will check if the request is /main_ar.html or /ar then the server sends the main_ar.html file with content type: text/html. This code is shown in figure 14.

Figure 14: Code for main_ar.html Request

Figure 14 shows the code for the Arabic version of the html file. First, we opened the file using the `open()` method with the path and the `utf-8` encoding as parameters to this method. The variable `html_msg` holds the contents that are read from the html file. The `http response` header initializes the response with the `200 OK` status and then the content type which is `text/html`. Then the `http response` is sent to the client using the `send()` method, and the connection closes.

If the request is /ar then the server responds with a main_ar.html file which is an Arabic version of the previous html file that contains information about us as students. This can be seen in Figure 15.



Figure 15.1: Arabic Html Webpage



Figure 15.2: Arabic Html Webpage

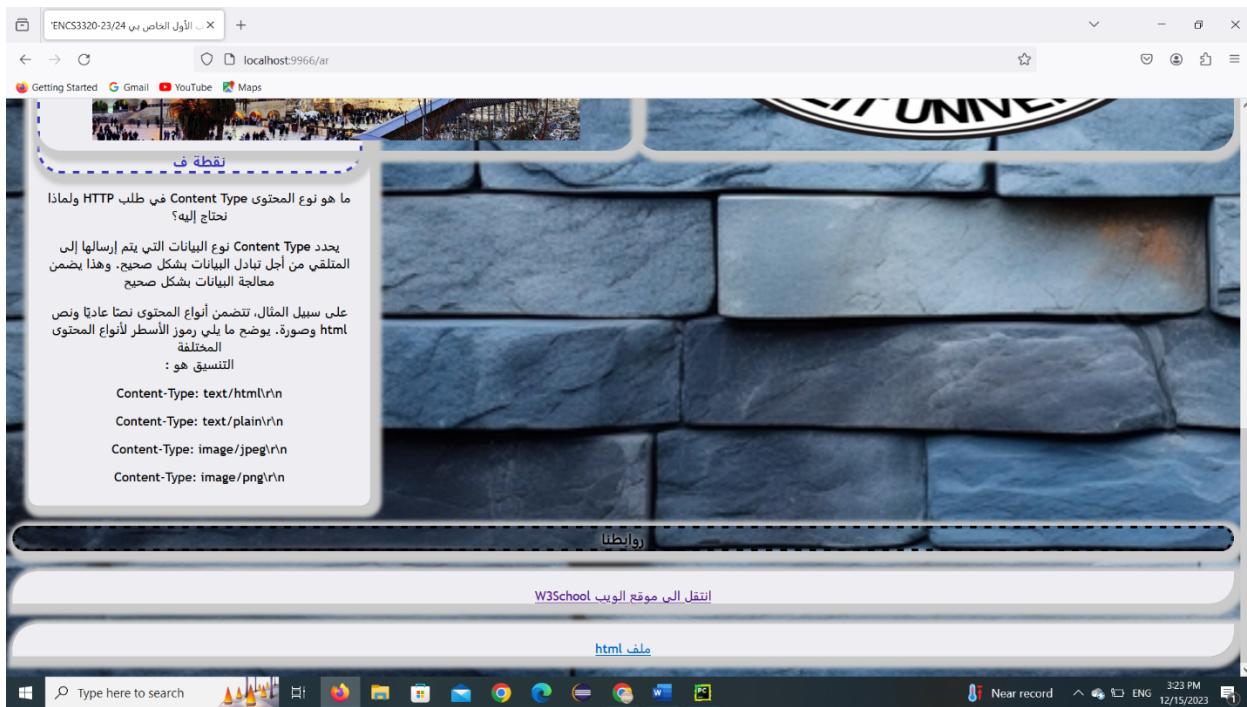


Figure 15.3: Arabic Html Webpage

Here http request appears in python console, Firefox request the main_ar.html file

```
requestedFile = request_list[1]
f_name = requestedFile.split('?')[0] # effectively removes any query parameters or additional information #This is presumably a string variable containing a file name
# print(requestedFile)
except IndexError:
    while True:
        try:
            f_name = f_name.lstrip('/')
            # after the first question mark
            #print(f_name)
            break
        except IndexError:
            pass
# print(f_name)
htmlFile = open(f_name, 'rb')
htmlFile.read()
htmlFile.close()
```

Client IP Address : 127.0.0.1
Client Port Number : 62061

GET /ar HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none

Figure 15.4: Request Arabic Html File

Here CSS file request appears in python console, Firefox request the master2.css file

The screenshot shows the PyCharm IDE interface. In the top navigation bar, the project is named 'part3Network' and the file is 'ENC53320_part3.py'. The code editor shows a snippet of Python code that handles file requests. The 'Run' tool window at the bottom displays the following log entry:

```
Client IP Address : 127.0.0.1
Client Port Number : 62065

GET /master2.css HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: text/css,*/*;q=0.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/ar
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: style
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Figure 15.5: Request master2 CSS File

Here image request appears in python console, Firefox request the bzu_logo.png

The screenshot shows the PyCharm IDE interface. In the top navigation bar, the project is named 'part3Network' and the file is 'ENC53320_part3.py'. The code editor shows a snippet of Python code that handles file requests. The 'Run' tool window at the bottom displays the following log entry:

```
Client IP Address : 127.0.0.1
Client Port Number : 62064

GET /img/bzu_logo.png HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/ar
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Figure 15.6: Request Image 2

Here image request appears in python console, Firefox request the alquds11.jpg

The screenshot shows the PyCharm IDE interface. In the top right, there's a terminal window titled 'ENC53320_part3' displaying a client request for an image:

```
Client IP Address : 127.0.0.1
Client Port Number : 62066

GET /img/alquds11.jpg HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/ar
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Below the terminal, a browser screenshot shows the image 'alquds11.jpg' displayed.

Figure 15.7: Request Al-Quds image

Here image request appears in python console, Firefox request the background.png

The screenshot shows the PyCharm IDE interface. In the top right, there's a terminal window titled 'ENC53320_part3' displaying a client request for an image:

```
Client IP Address : 127.0.0.1
Client Port Number : 62069

GET /img/background.png HTTP/1.1
Host: localhost:9966
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://localhost:9966/master2.css
Cookie: Pycharm-e41ba4ef=f706e9fe-6a93-4f5b-86cd-fc3732e25db9
Sec-Fetch-Dest: image
Sec-Fetch-Mode: no-cors
Sec-Fetch-Site: same-origin
```

Below the terminal, a browser screenshot shows the image 'background.png' displayed.

Figure 15.8 Request Background Image 2

If the request is an html file, then the server sends the requested html file with content type: text/html. Figure 16 shows the code for this part. A printing statement “Error: not found” is included to let the user know if the html file was not found, as well as, an html code to display an error message, our names and IDs. Figure 16 shows the html file with this content.

```
##### part3 branch_3 #####
elif requestedFile.endswith(".html"):
    try:
        print("hiiii in .html case")
        #print(requestedFile)

        htmlf_path = str(f_name)
        with open(htmlf_path, "r") as html_file:
            html_msg = html_file.read()

        response = "HTTP/1.1 200 OK\r\n" # http response
        response += "Content-Type: text/html\r\n" # the kind of masseg is html masseg or text/plain or
        response += html_msg
        response += "\r\n"
        connectionSocket.send(response.encode())
        connectionSocket.close() # to close the sentence => (close Connection)

    except FileNotFoundError:
        print(f"Error: '{htmlf_path}' not found. Please check the file path.")
        connectionSocket.send(f"HTTP/1.1 404 Not Found\r\n".encode())
        responseMessage = (
            '<html><title>Error 404</title><body><center><h1 style = "color : red ;">The file is not '
            'found => Error 404 <= </h1><hr><p style= '
            '"font-weight: '
            'bold;">Afaf Amwas: 1203359 </p><p style="font-weight: bold;"> Mohammed Salem: 1200651 '
            '</p><p style="font-weight: bold;"> Dana Asfour: 1211924 '
            '</p><hr><h2>IP: ' + str(IPAddress) + ', Port: ' + str(PortNumber) +
            '</h2></center></body></html>').encode('utf-8')
        connectionSocket.send(f"\r\n".encode())
        connectionSocket.send(responseMessage)
        connectionSocket.close()
```

Figure 16: Code for .html Request

Here html request appears in python console, Firefox request the htmlFile.html

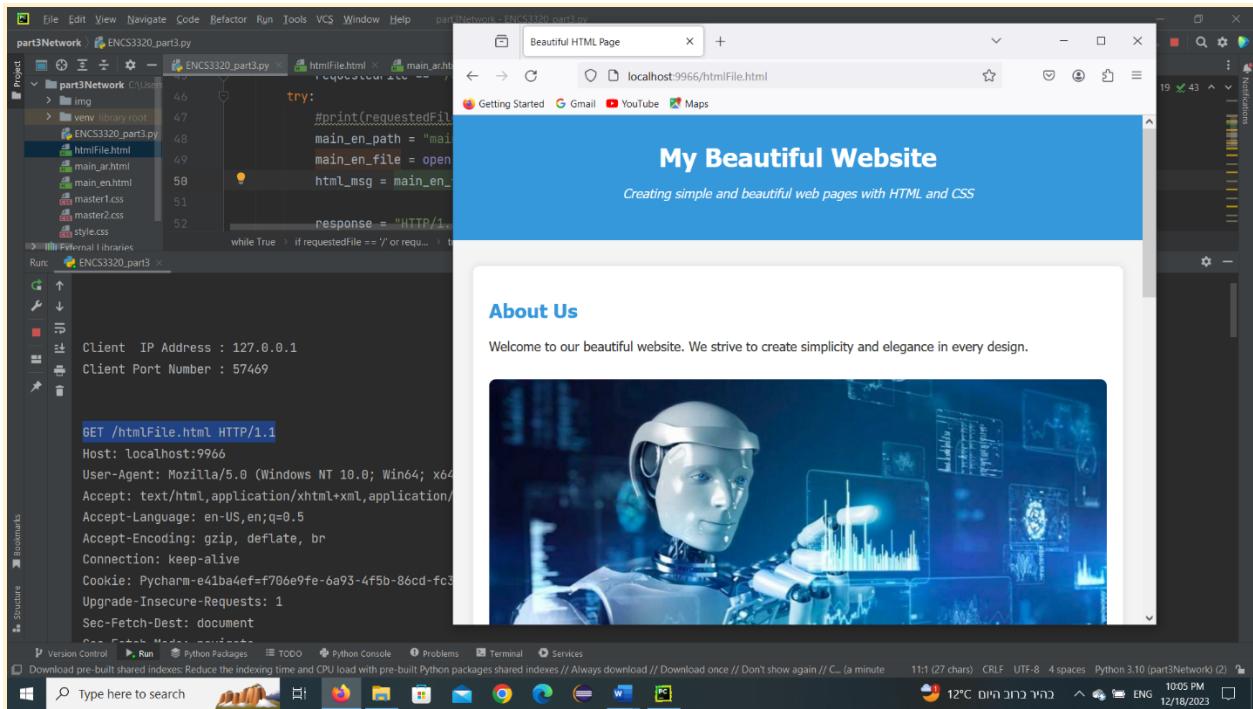


Figure 16.1: Request Html File 2

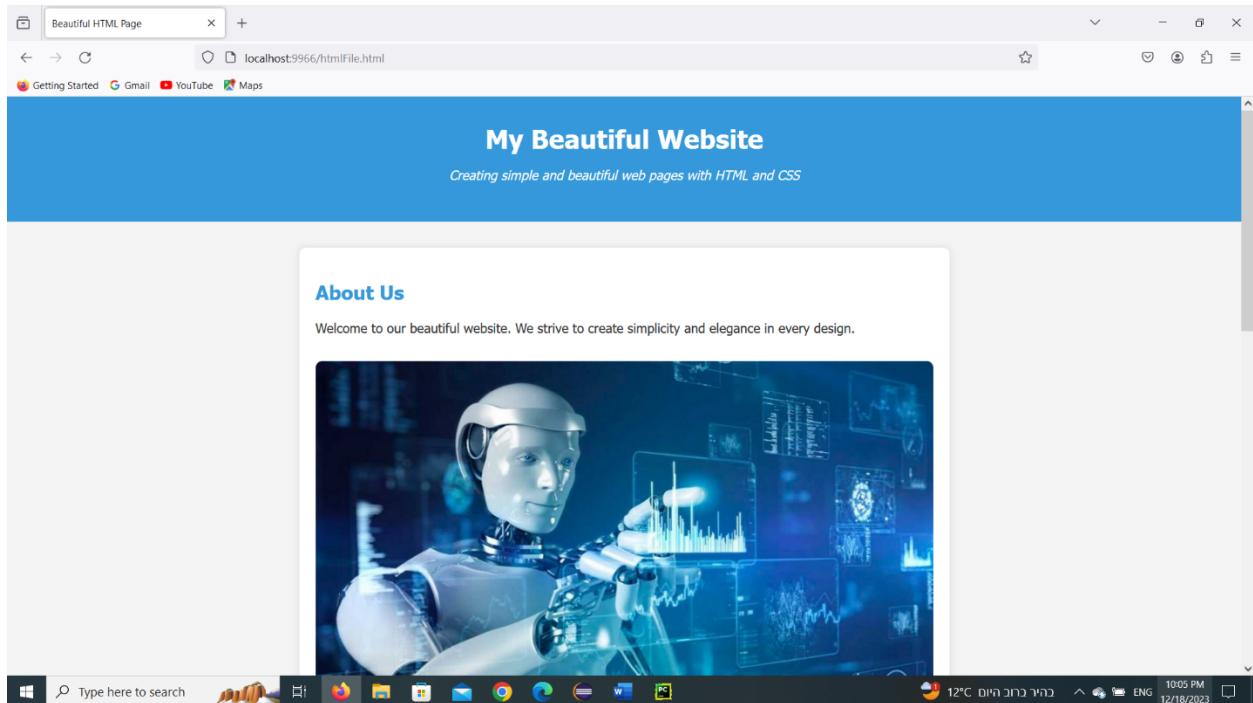


Figure 16.2: Request Html File 2

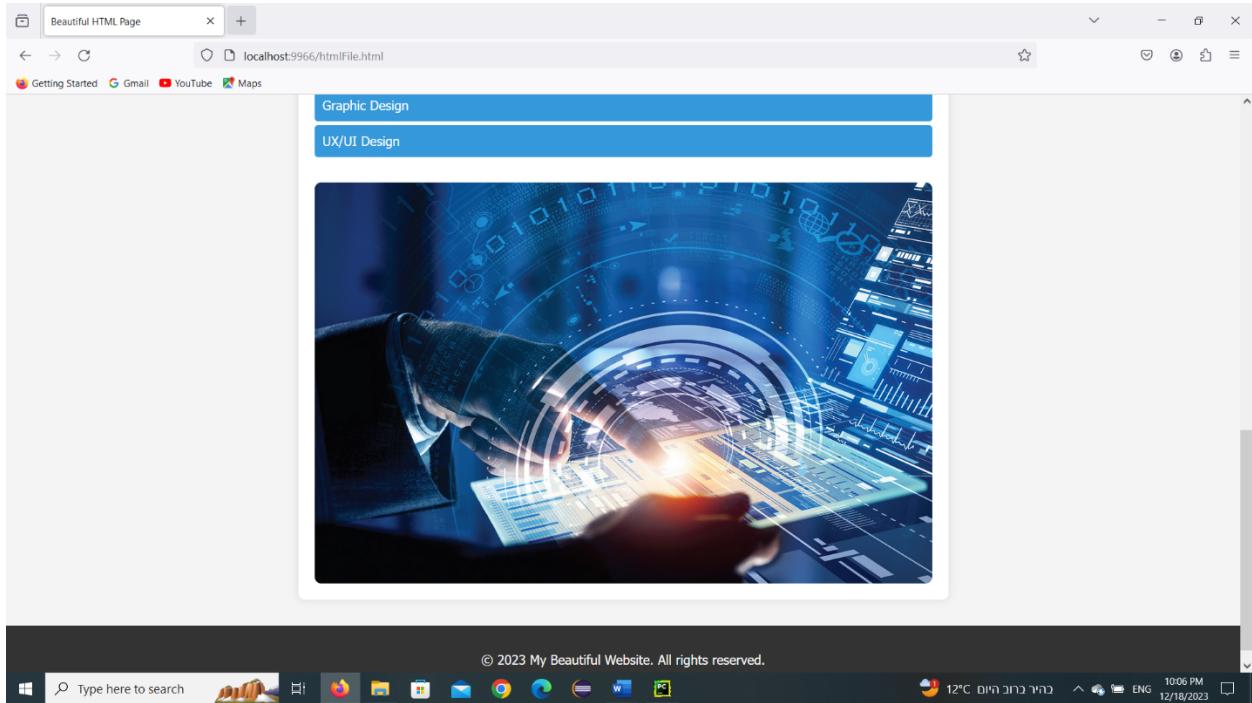


Figure 16.3: Request Html File 2

If the request is a .css file then the server sends that css file with content type: text/css as seen in figure18. The Figure below shows the css file for this part.

```

#####
part3 branch 4 #####
elif requestedFile.endswith(".css"):
    try:
        # print(requestedFile)
        style_path = str(f_name)
        with open(style_path, "r") as style_file:
            style_msg = style_file.read()

        response = "HTTP/1.1 200 OK\r\n" # http response
        response += "Content-Type: text/css\r\n" # the kind of masseg is html masseg or text/plain or
        response += "\r\n"
        response += style_msg
        connectionSocket.send(response.encode())
        connectionSocket.close() # to close the sentance => (close Connection)

    except FileNotFoundError:
        print(f"Error: '{style_path}' not found. Please check the file path.")
        connectionSocket.send(f"HTTP/1.1 404 Not Found\r\n".encode())
        responseMessage = (
            '<html><title>Error 404</title><body><center><h1 style = "color : red ;">The file is not '
            'found => Error 404 <= </h1><hr><p style= '
            '"font-weight: '
            'bold;">Afaf Amwas: 1203359 </p><p style="font-weight: bold;"> Mohammed Salem: 1200651 '
            '</p><p style="font-weight: bold;"> Dana Asfour: 1211924 '
            '</p><hr><h2>IP: ' + str(IPAddress) + ', Port: ' + str(PortNumber) +
            '</h2></center></body></html>').encode('utf-8')
        connectionSocket.send(f"\r\n".encode())
        connectionSocket.send(responseMessage)
        connectionSocket.close()
    except IOError as e:
        print(f"Error reading file: {e}")

```

Figure 18: Code for .css Request

The screenshot shows a PyCharm IDE interface. On the left, the project structure for 'part3Network' is visible, containing files like 'ENC53320_part3.py', 'htmlFile.html', 'main_ar.html', 'main_en.html', 'master1.css', 'master2.css', and 'style.css'. The main code editor window displays 'ENC53320_part3.py' with Python code for handling HTTP requests. Below the code editor, the terminal window shows the server is ready to receive connections. A browser window on the right shows the local host at port 9966, displaying the 'master1.css' file which includes CSS for styling the page.

```

response = "HTTP/1.1 200 OK\r\n"
response += "Content-Type: text/html\r\n"
response += "\r\n"
response += html_msg
connectionSocket.send(response)
connectionSocket.close() # to free connection
    
```

```

body {
    background-image: url(img/background.png);
    font-family: Trebuchet MS;
    text-align: center;
    background-size: cover;
}

h1 {
    /* text-shadow: 6px 6px rgb(211, 211, 211); */
    text-shadow: 2px 2px 5px rgb(216, 216, 222);
}

h3 {
    border-style: dashed;
    box-shadow: 4px 4px 5px 10px #cacaca;
    text-shadow: 2px 2px 5px rgb(223, 223, 231);
    border-radius: 30px;
    color: rgb(56, 56, 186);
    text-shadow: 0 0 5px rgb(240, 236, 236);
}

h2 {
    /* border-style: solid; */
    margin-top: 70px;
    border-radius: 30px 0 30px 0;
    text-shadow: 2px 2px 5px rgb(63, 63, 157);
    box-shadow: 4px 4px 5px 10px #cacaca;
    background-color: #f1f1f1;
}

.container {
    display: flex;
    justify-content: center;
}

```

Figure 18.1 CSS Code

If the request is a .png image then the server sends that png image with content type: image/png as seen in figure 19.

```

#####
# part3 branch 5 #####
elif requestedFile.endswith(".png"):
    try:
        png_path = str(f_name)
        with open(png_path, "rb") as png_file:
            png_msg = png_file.read()

        response = "HTTP/1.1 200 OK\r\n" # http response
        response += "Content-Type: image/png\r\n" # the kind of masseg is html masseg or text/plain or imag/
        response += "\r\n"
        connectionSocket.send(response.encode())
        connectionSocket.send(png_msg)
        connectionSocket.close() # to close the sentance => (close Connection)

    except FileNotFoundError:
        print(f"Error: '{png_path}' not found. Please check the file path.")
        connectionSocket.send(f"HTTP/1.1 404 Not Found\r\n".encode())
        responseMessage = (
            '<html><title>Error 404</title><body><center><h1 style = "color : red ;">The file is not '
            'found => Error 404 <= </h1><hr><p style= '
            '"font-weight: '
            "'bold;">Afaf Anwas: 1203359 </p><p style="font-weight: bold;"> Mohammed Salem: 1200651 '
            '</p><p style="font-weight: bold;"> Dana Asfour: 1211924 '
            '</p><hr><h2>IP: ' + str(IPAddress) + ', Port: ' + str(PortNumber) +
            '</h2></center></body></html>').encode('utf-8')
        connectionSocket.send(f"\r\n".encode())
        connectionSocket.send(responseMessage)
        connectionSocket.close()
    except IOError as e:
        print(f"Error reading file: {e}")
#####

```

Figure 19: Code for .png Request

If the request is a .jpg image then the server sends that png image with content type: image/jpg as seen in figure 20.

```

#####
#_part3_branch_6 #####
#####

elif requestedFile.endswith(".jpg"):
    try:
        jpg_path = str(f_name)
        with open(jpg_path, "rb") as jpg_file:
            jpg_msg = jpg_file.read()

        response = "HTTP/1.1 200 OK\r\n" # http response
        response += "Content-Type: image/jpg\r\n" # the kind of message is html message or text/plain or image
        response += "\r\n"
        connectionSocket.send(response.encode())
        connectionSocket.send(jpg_msg)
        connectionSocket.close() # to close the sentence => (close Connection)

    except FileNotFoundError:
        print(f"Error: '{jpg_path}' not found. Please check the file path.")
        connectionSocket.send(f"HTTP/1.1 404 Not Found\r\n".encode())
        responseMessage = (
            '<html><title>Error 404</title><body><center><h1 style = "color : red ;">The file is not '
            'found => Error 404 <= </h1><hr><p style= '
            '"font-weight: '
            'bold;">Afaf Amwas: 1203359 </p><p style="font-weight: bold;"> Mohammed Salem: 1200651 '
            '</p><p style="font-weight: bold;"> Dana Asfour: 1211924 '
            '</p><hr><h2>IP: ' + str(IPAddress) + ', Port: ' + str(PortNumber) +
            '</h2></center></body></html>').encode('utf-8')
        connectionSocket.send(f"\r\n".encode())
        connectionSocket.send(responseMessage)
        connectionSocket.close()
    except IOError as e:
        print(f"Error reading file: {e}")

```

Figure 20: Code for .jpg Request

The below figure shows the code for how we used the status code 307 Temporary Redirect to redirect to cornell.edu, stackoverflow.com, and ritaj website depending on the type of request which are /cr, /so, and /rt respectively.

```

#####
# part 3 branch 7 #####
#####

elif requestedFile == "/cr":
    connectionSocket.send(f"HTTP/1.1 307 Temporary Redirect\r\n".encode())
    connectionSocket.send(f"Location: https://www.cornell.edu/ \r\n".encode())
elif requestedFile == "/so":
    connectionSocket.send(f"HTTP/1.1 307 Temporary Redirect\r\n".encode())
    connectionSocket.send(f"Location: https://stackoverflow.com/ \r\n".encode())
elif requestedFile == "/rt":
    connectionSocket.send(f"HTTP/1.1 307 Temporary Redirect\r\n".encode())
    connectionSocket.send(f"Location: https://ritaj.birzeit.edu/register/ \r\n".encode())
#####
# part 3 branch 8 #####
#####

else:
    connectionSocket.send(f"HTTP/1.1 404 Not Found\r\n".encode())
    responseMessage = (
        '<html><title>Error 404</title><body><center><h1 style = "color : red ;">The file is not '
        'found => Error 404 <= </h1><hr><p style= '
        '"font-weight: '
        'bold;">Afaf Amwas: 1203359 </p><p style="font-weight: bold;"> Mohammed Salem: 1200651 '
        '</p><p style="font-weight: bold;"> Dana Asfour: 1211924 '
        '</p><hr><h2>IP: ' + str(IPAddress) + ', Port: ' + str(PortNumber) +
        '</h2></center></body></html>')
    connectionSocket.send(f"\r\n".encode())
    connectionSocket.send(responseMessage)
    connectionSocket.close()

```

Figure 21: Code for Temporary Redirect

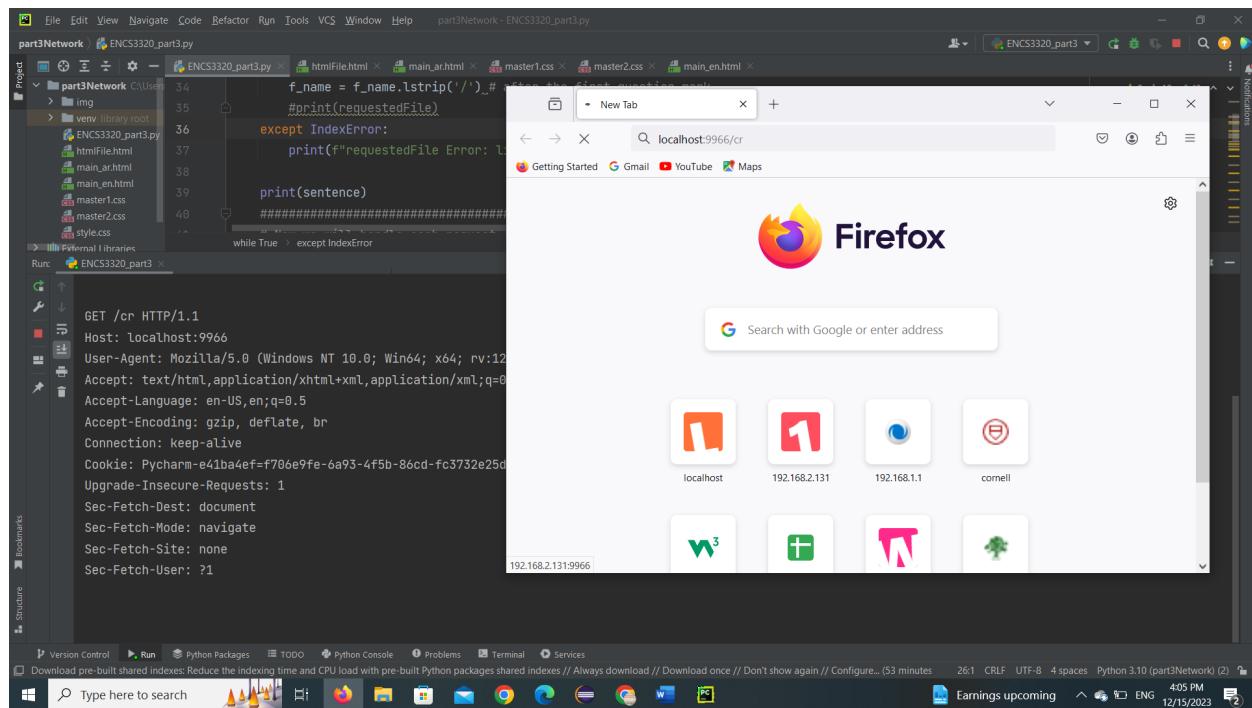


Figure 21.1: Firefox Redirect

Figure 21.2 shows a screenshot of a browser window. The URL bar shows <https://www.cornell.edu>. The page content is a photograph of three students in a kitchen, wearing hairnets and aprons, churning ice cream. Overlaid text on the image says "Food for thought" and "Students churn 'Freedom of Expression' ice creams". The browser interface includes a navigation bar with tabs like "Cornell University", a search bar, and various icons.

Figure 21.2: Cornell Redirect

Figure 21.3 shows a screenshot of a browser window. The URL bar shows <https://stackoverflow.com>. The page content is the StackOverflow homepage, featuring a search bar, a "Discover Teams" button, and a "Your privacy" cookie consent banner at the bottom. The browser interface includes a navigation bar with tabs like "stackoverflow", a search bar, and various icons.

Figure 21.3: StackOverflow Redirect

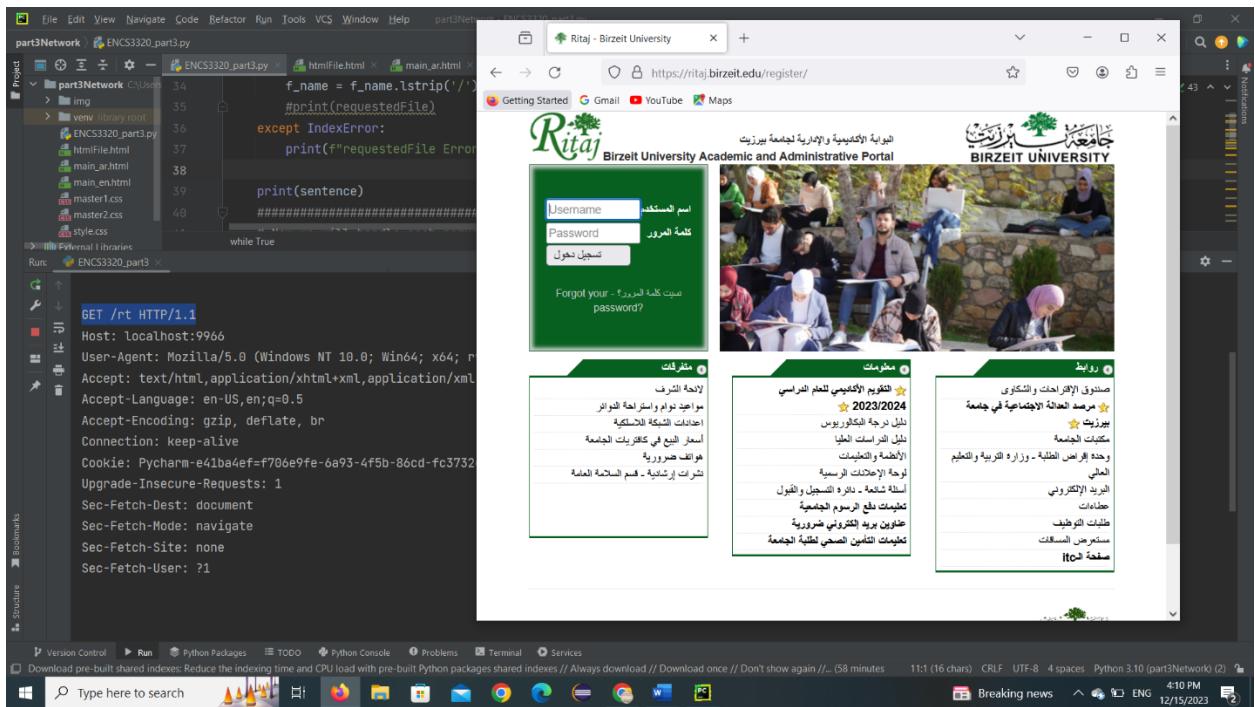


Figure 21.4: Ritaj Redirect

As seen in all these requests, if the request is wrong or the file doesn't exist the server returns a simple html web page that contains content type: text/html. This webpage shown in figure 22 shows the response status (404 Not Found), an error message in the title, a message in the body, our names and IDs, and finally the IP and port number of the client.

The screenshot shows the PyCharm IDE interface. On the left, the Project tool window displays a Python project named 'part3Network' containing files like 'ENC53320_part3.py', 'htmlFile.html', 'main_ar.html', 'main_en.html', 'master1.css', 'master2.css', and 'style.css'. The code editor shows a portion of 'ENC53320_part3.py' with a syntax error highlighted. To the right, a web browser window titled 'Error 404' shows the URL 'localhost:9966/anyvaluehihi'. The page content is red text: 'The file is not found => Error 404 <=' followed by three names and their IDs: Afaf Amwas: 1203359, Mohammed Salem: 1200651, and Dana Asfour: 1211924. Below the browser window, the system tray shows various icons and the status bar indicates 'IP: 127.0.0.1, Port: 52181'.

Figure 22.1: Error Webpage

This screenshot is nearly identical to Figure 22.1, showing the same PyCharm environment and browser output. The only difference is the URL in the browser's address bar, which has changed from 'localhost:9966/anyvaluehihi' to 'localhost:9966/anyvaluehihi.html'. The error message and names remain the same.

Figure 22.2: Error Webpage

The screenshot shows the PyCharm IDE interface. On the left, the project structure for 'part3Network' is visible, containing files like 'ENC53320_part3.py', 'htmlFile.html', 'main_ar.html', 'master1.css', 'master2.css', and 'style.css'. The code editor window displays a portion of 'ENC53320_part3.py' with a syntax error highlighted. To the right, a web browser window titled 'Error 404' shows the URL 'localhost:9966/mohammadSalem'. The page content includes the text 'The file is not found => Error 404 <=' and a list of names and IDs: Afaf Amwas: 1203359, Mohammed Salem: 1200651, Dana Asfour: 1211924. Below the browser window, the system tray shows various icons and the date/time: 402 PM 12/15/2023.

Figure 22.3: Error Webpage

This screenshot is similar to Figure 22.3, showing the PyCharm IDE and a browser error page. The project structure and code editor are identical. The browser window shows the same 'Error 404' page for 'localhost:9966/mohammadSalem.css'. The page content includes the text 'The file is not found => Error 404 <=' and the same list of names and IDs. The system tray at the bottom indicates 901 CRLF, UTF-8, 4 spaces, Python 3.10 (part3Network) (2), and the date/time 402 PM 12/15/2023.

Figure 22.4: Error Webpage

The screenshot shows a PyCharm IDE interface. On the left, the Project tool window displays a file structure for a project named 'part3Network'. Inside, there's a folder 'ENCSS3320_part3' containing files like 'ENCS3320_part3.py', 'htmlFile.html', 'main_ar.html', 'master1.css', 'master2.css', and 'style.css'. The 'ENCS3320_part3.py' file is open in the main editor area, showing Python code that handles file requests. The code includes logic to strip file paths and print requested files. A terminal window at the bottom shows a successful 'GET' request for 'mohammadSalem.png' on port 9966. To the right, a browser window titled 'Error 404' shows the message 'The file is not found => Error 404 <=' and lists three user details: Afaf Amwas (ID: 1203359), Mohammed Salem (ID: 1200651), and Dana Asfour (ID: 1211924). Below the browser is a status bar with system information.

Figure 22.5: Error Webpage