Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Computer Networks ENCS3320

Project Report

Prepared by:

Klarein Wassaya 1210279

Masa Jalamneh 1212145

Instructor: Dr. Abdalkarim Awad

Section: 1

Date: 27-11-2023

Part 1

Part 1.1

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

1. **Ping:** a tool that sends small data packet to a specific IP address, and measure the round-trip time it takes for the packet to reach the destination and come back.

2. **Tracert:** a tool that traces the route taken by packets from the source device to the destination device across an IP network.

3. **Nslookup:** a command-line tool used to query Domain Name System (DNS) to obtain domain names, IP addresses or any other related DNS records.

4. **Telnet:** a protocol and command-line tool that enables the user to establish a connection to a distant server or device.

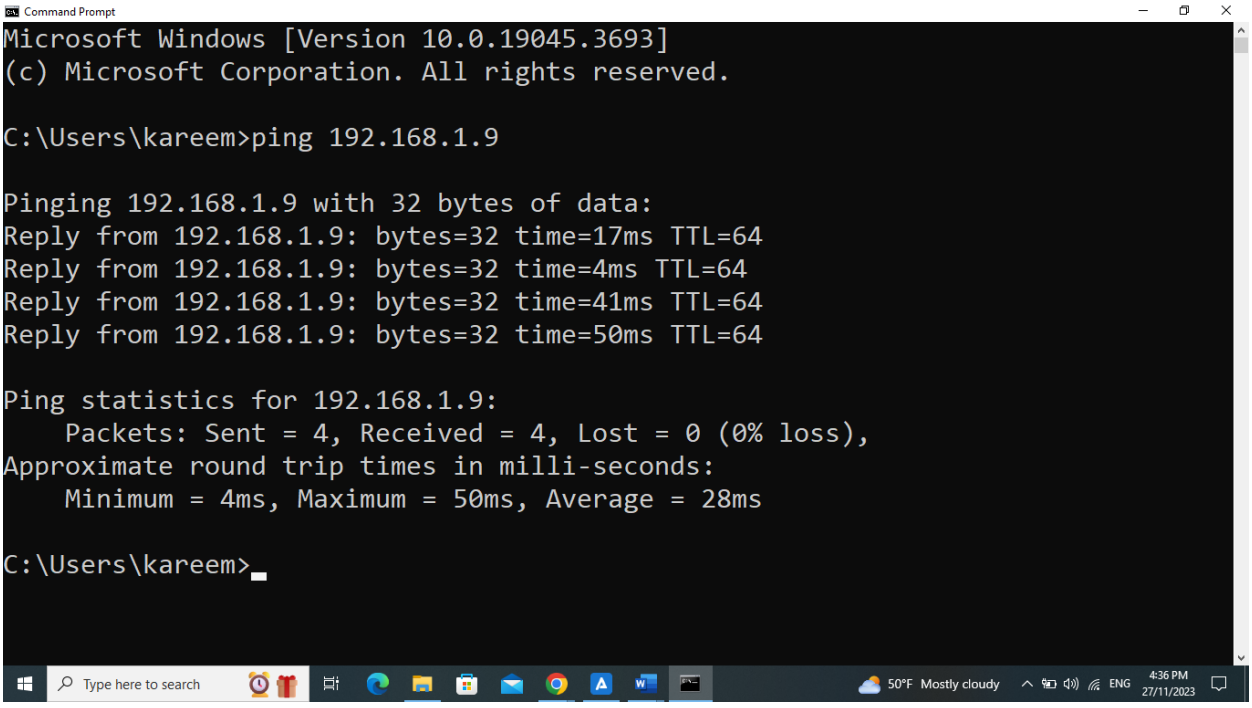
Part 1.2

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

1- Ping a device in the same network, e.g., from a laptop to a smartphone
2- ping www.cornell.edu
3- tracert www.cornell.edu
4- nslookup www.cornell.edu

Part 1.2.1

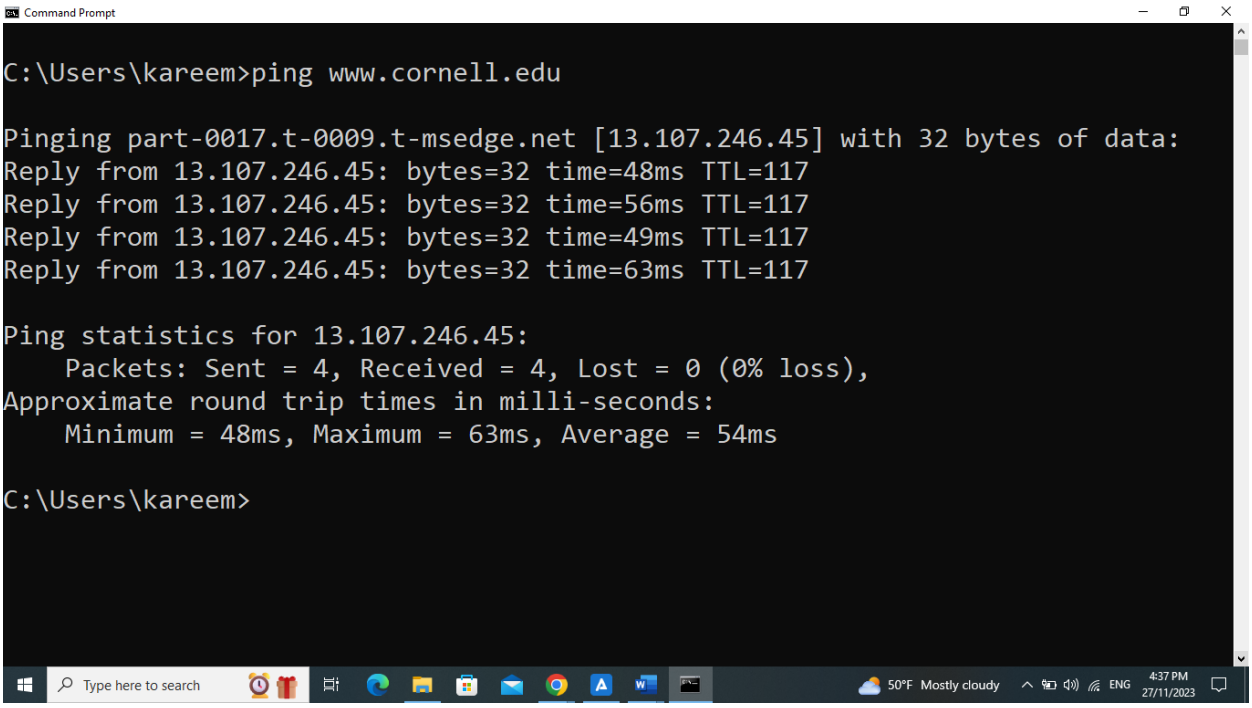
Ping a device in the same network, e.g., from a laptop to a smartphone



From the output in the picture, we see that there are four requests sent to the IP address 192.168.1.9, we see the response to each of these requests, including the time taken for the device to respond in milliseconds. It also shows the TTL (Time To live) value which refers to the number of hops that a packet is allowed to pass through before it is discarded. The last section shows a summary of the ping results, it shows that all the packets were sent and received successfully. Some statistics on the round-trip time of the packets are also shown, it is shown that four packets response were received from the IP address 192.168.1.9 with an average of 28ms delay

Part 1.2.2

ping www.cornell.edu



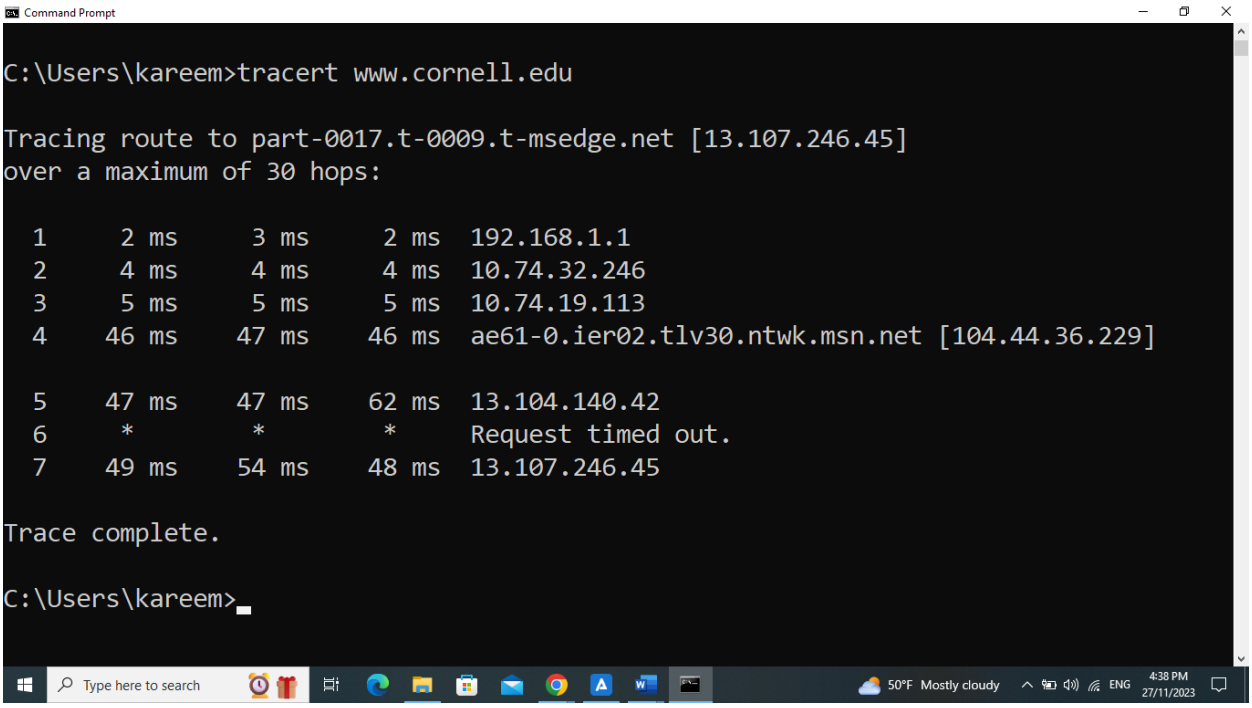
This command sent a series of "Echo Request" messages to the cornell.edu website's server and display the results. A successful reply from www.cornell.edu is received, four packets response were sent by 13.107.246.45 with their TTL with an average of 54ms delay, the four packets were sent and received successfully without any loss.

Output discussion:

The average round trip time of the response is 54ms, which indicates a reasonably fast connection to the www.cornell.edu website, however there are more things to be taken into consideration about determining the location of the IP address we get responses from. Due to a general estimation for pinging a server in the USA from PAESTINE, the average round trip time should be in the range between 100ms to 200ms (approximation - may be not accurate), since the RTT can vary based on the specific locations, network conditions, and other things. So according to this estimation, it doesn't seem that we are getting responses from a server in the USA.

Part 1.2.3

tracert www.cornell.edu

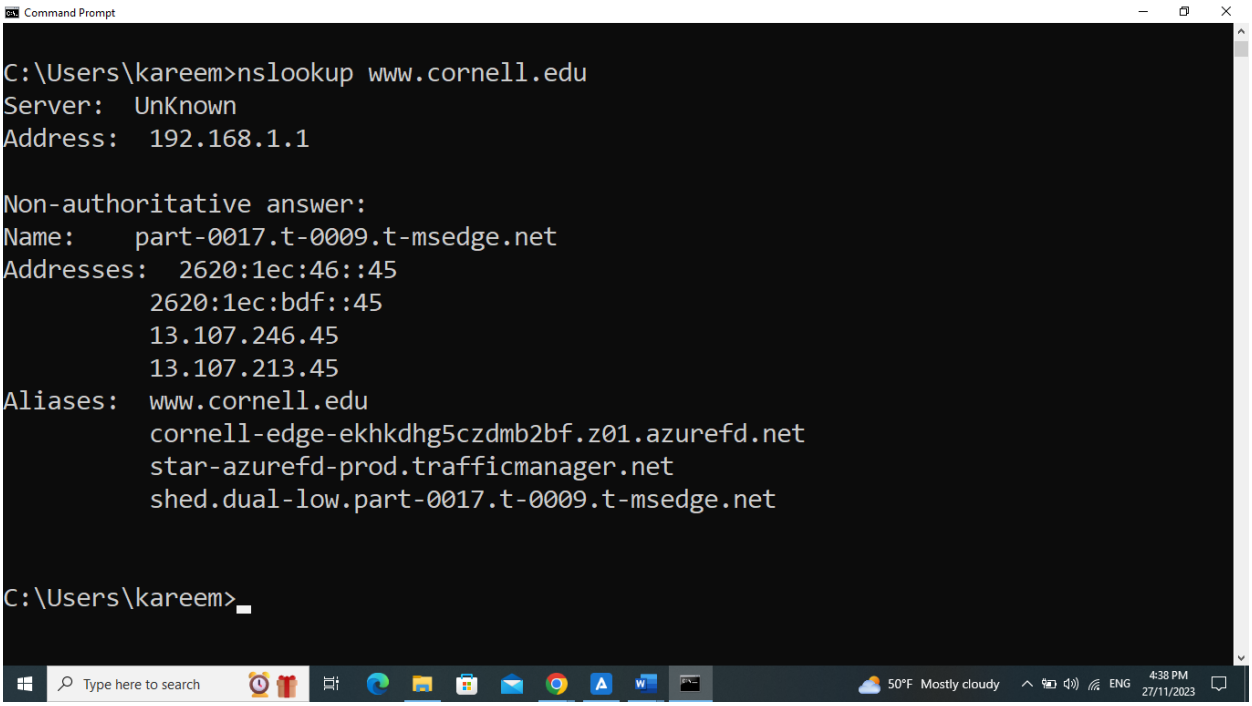


The output shows the path taken by the packets to reach the destination, including the IP addresses of the average hops. Each line represents a hop, each hop sends 3 packets, in each line there is information about the hop's number, the response time of each one of the three packets and the IP address of the device.

The sixth hop didn't respond within the specified time so it is marked "Request Timed Out" but the rest of the hops responded and the traceroute reached the target device successfully.

Part 1.2.4

nslookup www.cornell.edu



Nslookup lets users enter a host name and find out the corresponding IP address or domain name system (DNS) record.

So, the output shows the DNS server used for the query is specified as "Unknown" with the IP address 192.168.1.1, label "Non-authoritative," means that the DNS server providing the information is not the primary source for the domain but has cached the information, while name was out as: part-0017.t-0009.t-msedge.net.

And the corresponding addresses for this domain are also provided:

- Addresses: 2620:1ec:bdf::45, 2620:1ec:46::45
- Addresses: 13.107.246.43, 13.107.213.43

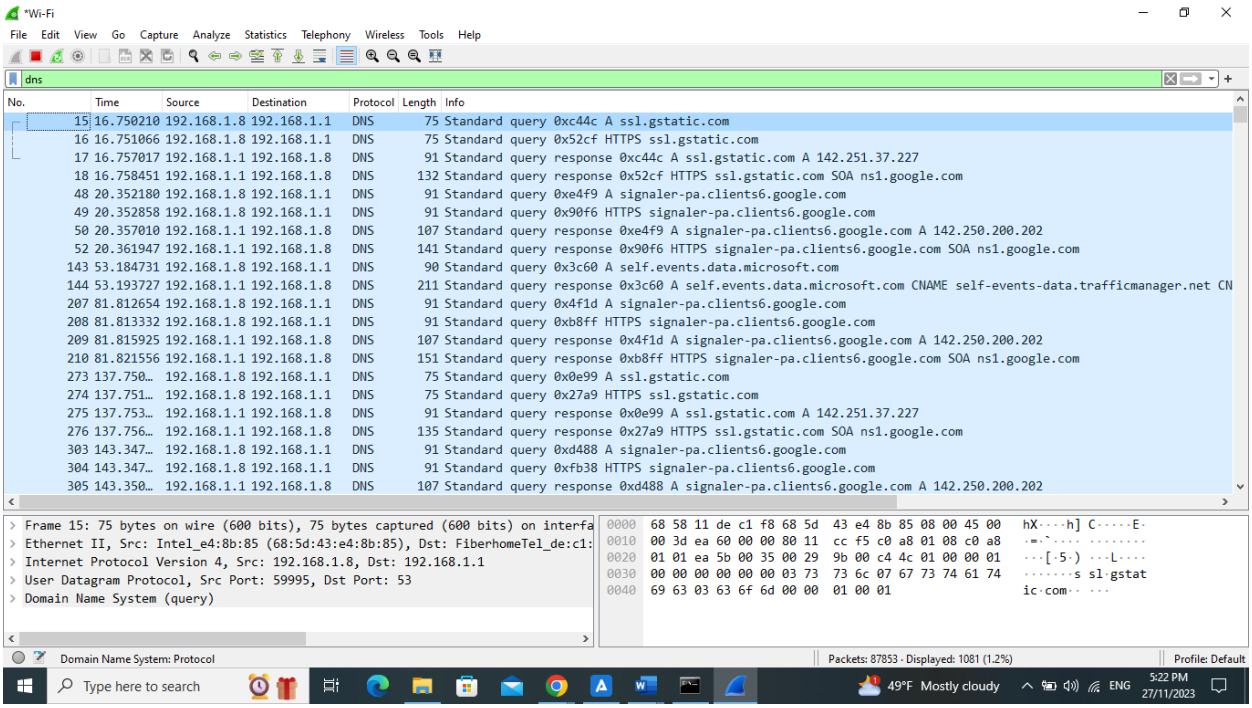
Aliases are also known as ‘alternative names’ are listed:

- www.cornell.edu
- cornell-edge-ekhkdhg5czdmb2bf.z01.azurefd.net
- star-azurefd-prod.trafficmanager.net
- shed.dual-low.part-0015.t-0009.t-msedge.net

The DNS resolution for www.cornell.edu as shown and explained previously is pointing to the IP addresses mentioned, and there are several aliases associated with this domain.

Part 1.3

Use wireshark to capture some DNS messages.



Using Wireshark to capture DNS (Domain Name System) messages, we can see a series of packets that contain information related to DNS queries and responses.

For DNS request/response (Query) Packets:

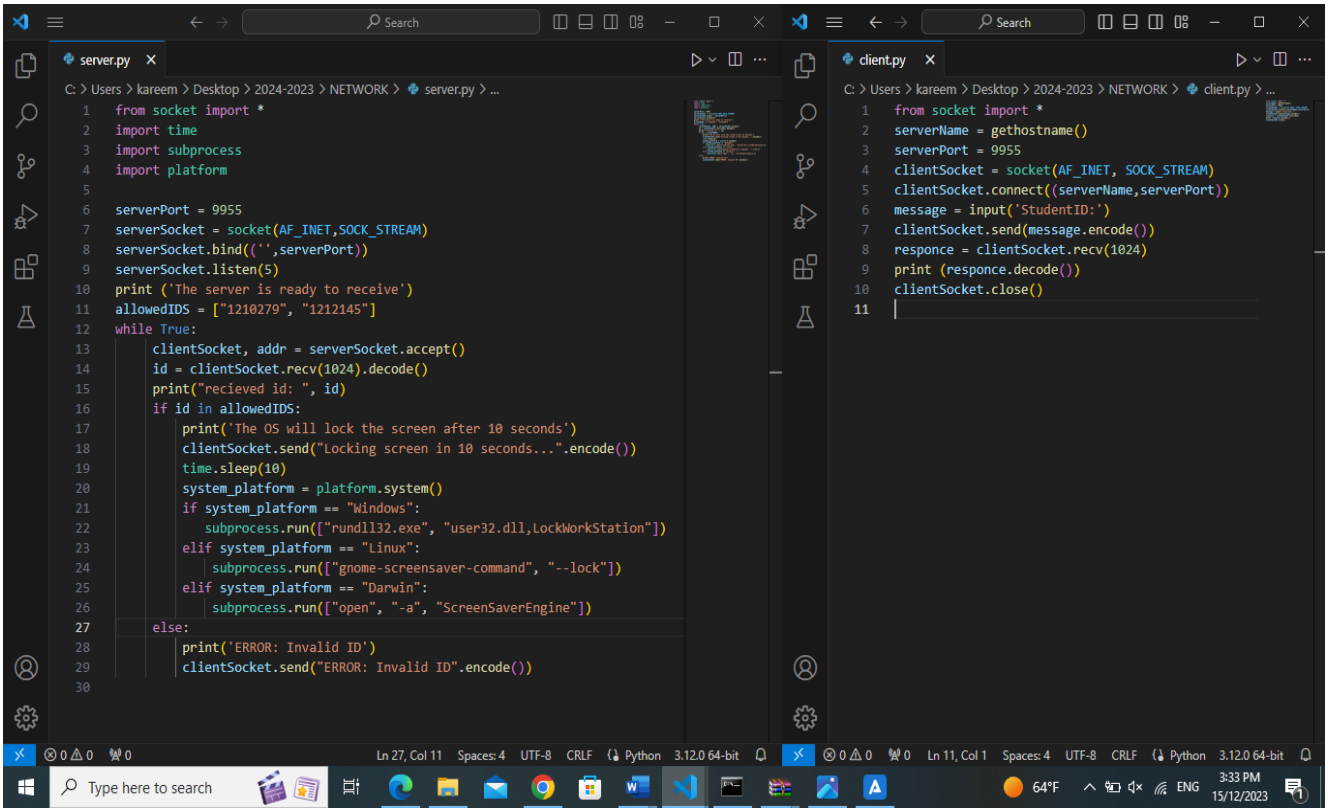
- Number.
- The source IP is my machine, and the destination IP is the DNS server.
- Protocol information: we can see the DNS protocol and the source and destination ports.
- Timestamps for each packet.
- Packet Length.
- Provide information.

For example, the first packet (NO. 15) is a DNS query from my machine to a DNS server asking for the IP address .The packet (NO. 17) is the DNS server's response, providing the resolved IP address. The actual details may vary based on the specific DNS query and response captured.

Part 2

Part 2.1:

Code:

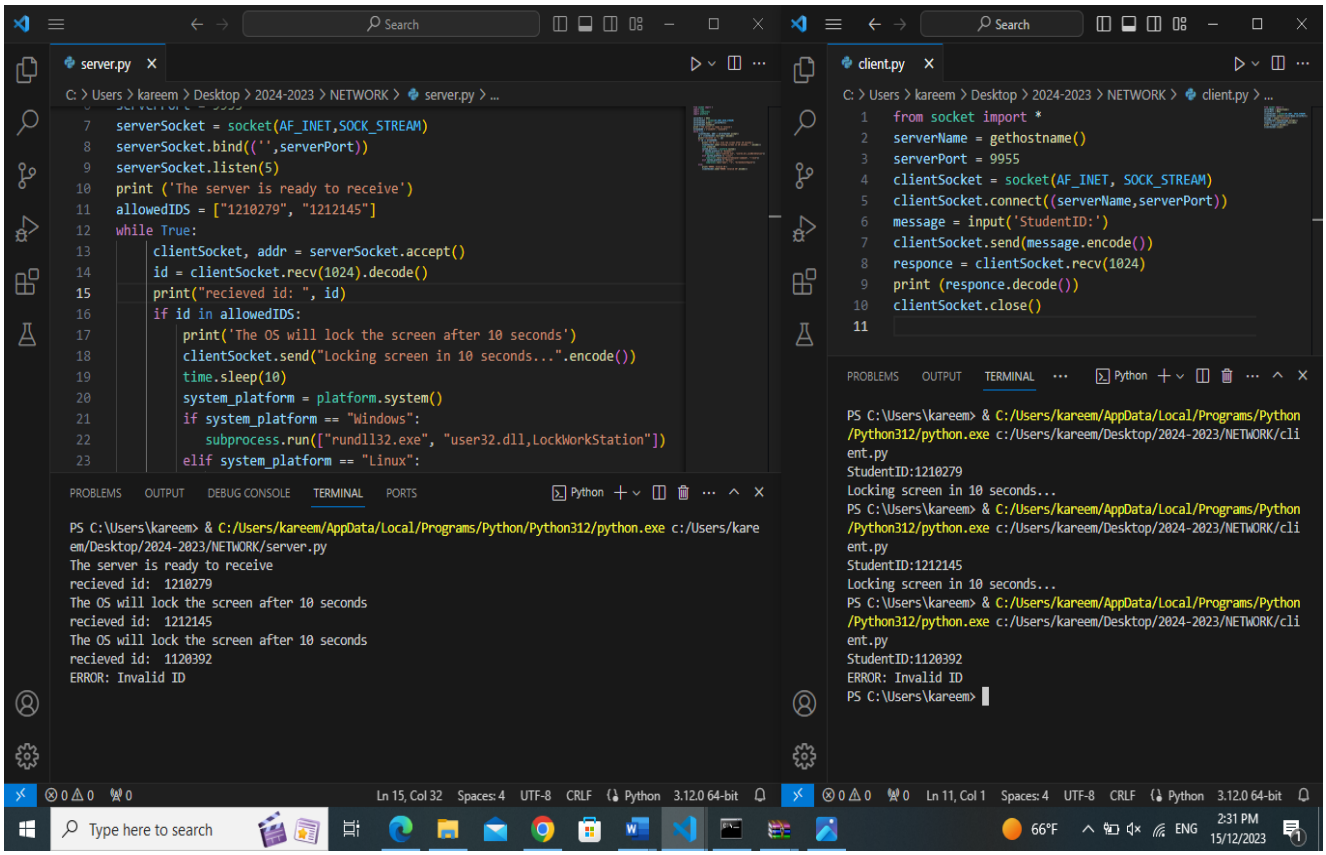


Explanation:

The figure above shows the python code for the client and the code for the server, in the client code we used the socket library to set a connection to the server, the client is connected to a server listening on port 9955, then the client asked the user to enter an id to be sent to the server and waited for a response from the server to display it for the user. In the server code, we used the socket library to set the port number, so that the client could connect to, our server is listening on port 9955. The server waits for a message from the client and then it prints the message on the terminal, our server has two valid IDs that it can responds to, otherwise the server send an error message to the client that the received ID was invalid and it does nothing else, if the ID is valid then the server sends a response to the client that the OS will lock the screen after 10 seconds, and then it waits for 10 seconds then the screen will be locked depending on the OS system we are running the server on.

Part 2.2:

Run:



Explanation:

From the terminal above in the figure, we see three messages sent from the client to the server, the first time the client sent a valid id so the OS locked the screen successfully, as so in the second time, the id was valid and the OS locked the screen successfully, while in the third time the client sent an invalid id so the response was an error message from the client and the OS did nothing.

Part 3

Part 3.0: content_types

From rfce2616 we found out that:

14.17 Content-Type:

The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET. Content-Type = "Content-Type" ":" media-type

Media types are defined in section 3.7. An example of the field is Content-Type: text/html; charset=ISO-8859-4 Further discussion of methods for identifying the media type of an entity is provided in section 7.2.1.

7.2.1 Type**:**

When an entity-body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding. These define a two-layer, ordered encoding model: entity-body : = Content-Encoding(Content-Type(data))

Content-Type specifies the media type of the underlying data. Content-Encoding may be used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the requested resource. There is no default encoding.

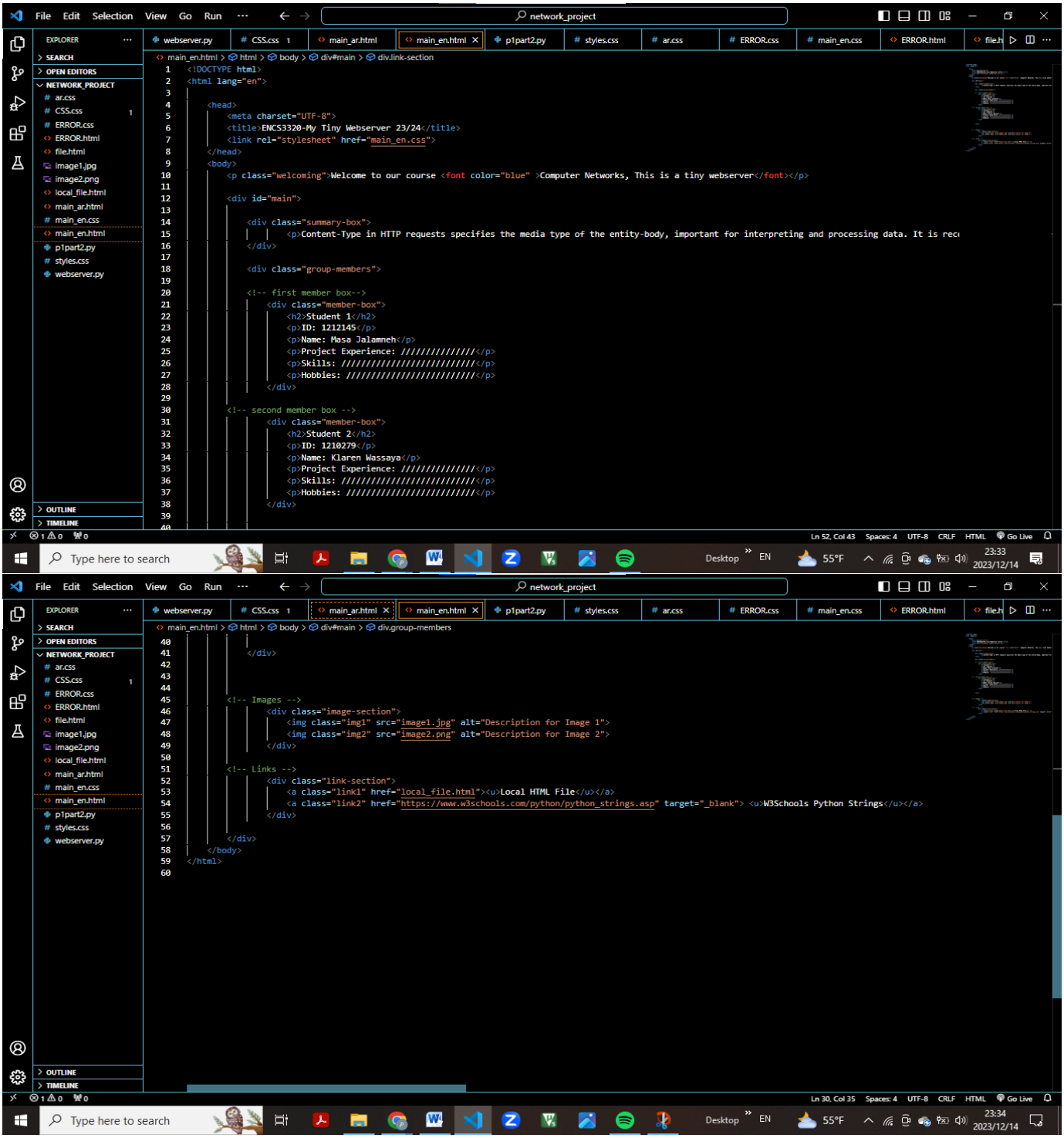
Any HTTP/1.1 message containing an entity-body SHOULD include a Content-Type header field defining the media type of that body. If and only if the media type is not given by a Content-Type field, the recipient MAY attempt to guess the media type via inspection of its content and/or the name extension(s) of the URI used to identify the resource. If the media type remains unknown, the recipient SHOULD treat it as type "application/octet-stream".

So, we can answer the question by saying that:

Content-Type in HTTP requests specifies the media type of the entity-body, important for interpreting and processing data. It is recommended in messages to define the media type, helping in type identification and negotiation. In the absence of explicit information, the recipient may deduce the media type. Proper handling ensures mutual understanding of data format between the sender and recipient.

Part 3.1: main_en.html / en [html and css codes + explanation]

HTML code:



```
<!DOCTYPE html>
<html lang="en">

    <head>
        <meta charset="UTF-8">
        <title>ENCS3320-My Tiny Webserver 23/24</title>
        <link rel="stylesheet" href="main_en.css">
    </head>
    <body>
        <p class="welcoming">Welcome to our course <font color="blue" >Computer Networks, This is a tiny webserver</font></p>

        <div id="main">

            <div class="summary-box">
                <p>Content-Type in HTTP requests specifies the media type of the entity-body, important for interpreting and processing data. It is rec
            </div>

            <div class="group-members">

            <!-- first member box-->
                <div class="member-box">
                    <h2>Student 1</h2>
                    <p>ID: 1212145</p>
                    <p>Name: Masa Jalamneh</p>
                    <p>Project Experience: ////////////////</p>
                    <p>Skills: ////////////////////////////</p>
                    <p>Hobbies: ////////////////////////////</p>
                </div>

            <!-- second member box -->
                <div class="member-box">
                    <h2>Student 2</h2>
                    <p>ID: 1210279</p>
                    <p>Name: Klaren Wassaya</p>
                    <p>Project Experience: ////////////////</p>
                    <p>Skills: ////////////////////////////</p>
                    <p>Hobbies: ////////////////////////////</p>
                </div>

            </div>

            <!-- Images -->
                <div class="image-section">
                    <img class="img1" src="image1.jpg" alt="Description for Image 1">
                    <img class="img2" src="image2.png" alt="Description for Image 2">
                </div>

            <!-- Links -->
                <div class="link-section">
                    <a class="link1" href="local_file.html"><u>Local HTML File</u></a>
                    <a class="link2" href="https://www.w3schools.com/python/python_strings.asp" target="_blank"> <u>W3Schools Python Strings</u></a>
                </div>

            </div>
        </body>
</html>
```

Explanation:

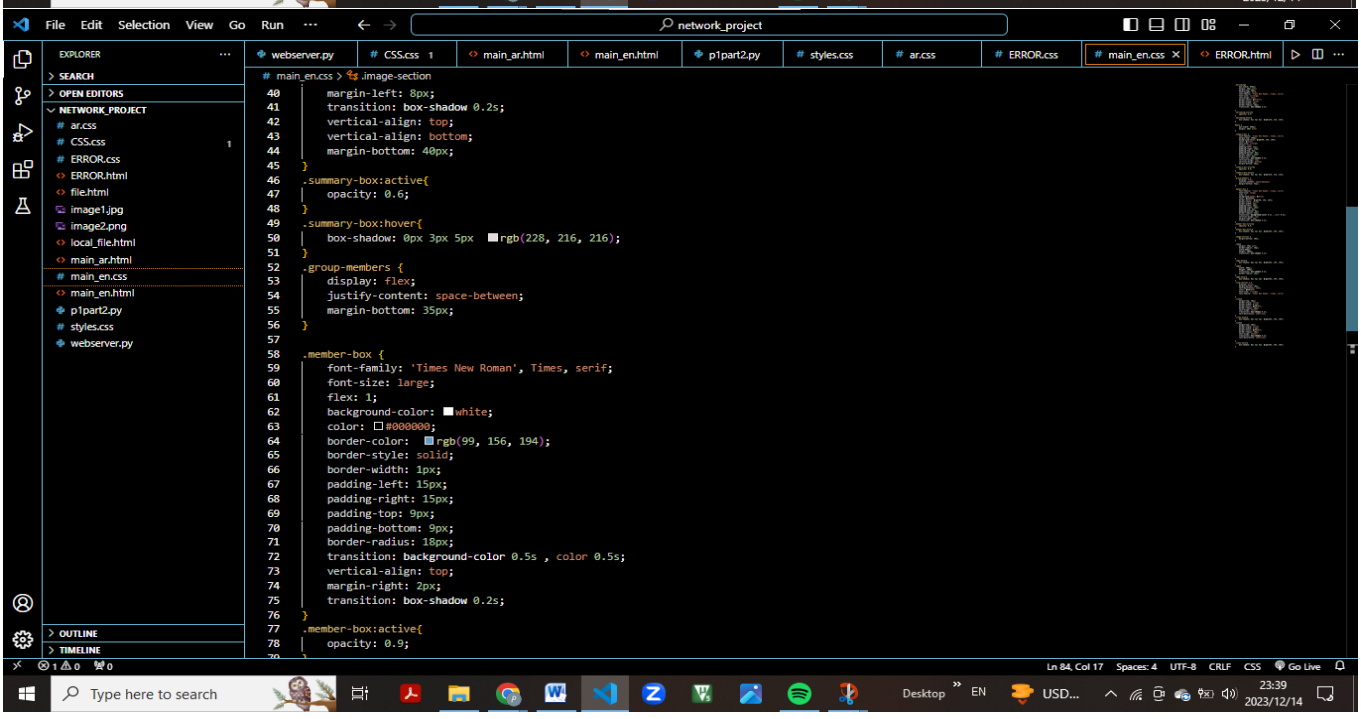
This is an html web page that contains:

- 'ENCS3320-My Tiny Webserver 23/24' as a tab title
- welcoming message with some styling (page styling will be in a CSS file)
- main html tag ( ) to divide the page (we used '**class**' to each part so we can style then in the CSS file)

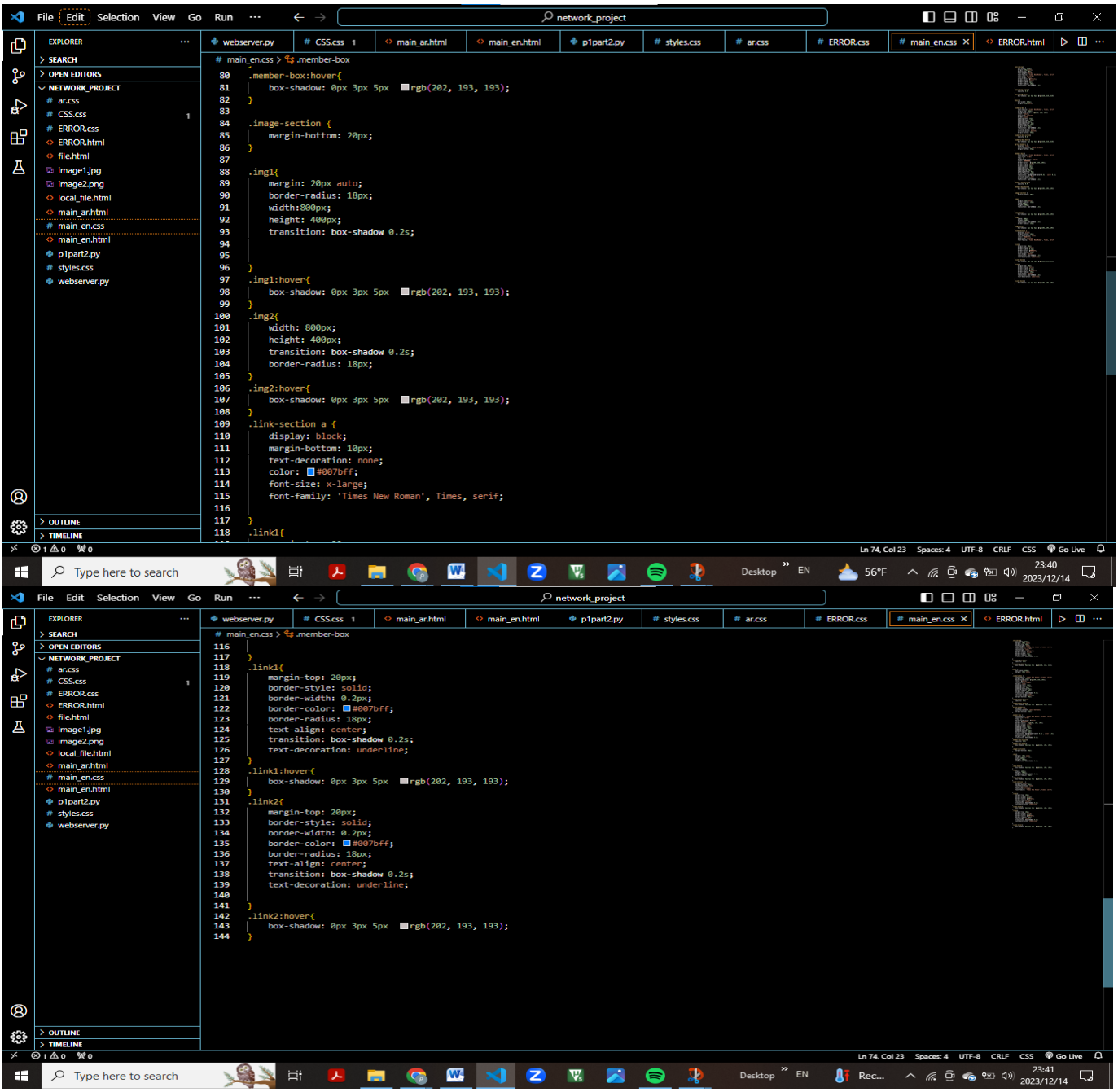
- boxes contain students information (each one in a box) divided by html tag (sub under the main)
- a box that contain the answer in part 3.0 also by (sub under the main)
- 2 images (one with extension .jpg, and the other one with .jpg), they are also divided by (sub under the main)
-2 links (the first one to a local html file, the second one to W3schools web page) divided by (sub)

CSS code:

```
.welcoming{
    max-width: 950px;
    margin: 30px auto;
    margin-top: 60px;
    font-weight: bold;
    font-family: 'Times New Roman', Times, serif;
    font-size: xx-large;
    text-align: left;
    border-color: #fffcfc;
    border-style: solid;
    border-width: 1px;
    border-radius: 18px;
    transition: box-shadow 0.2s;

}
.welcoming:active{
    opacity: 0.5;
}
.welcoming:hover{
    box-shadow: 0px 3px 5px rgb(228, 216, 216);
}

#main {
    max-width: 850px;
    margin: 30px auto;
}

.summary-box {
    font-family: 'Times New Roman', Times, serif;
    margin-top: 100px;
    background-color: rgb(99, 156, 194);
    color: white;
    font-size: x-large;
    border: none;
    padding-left: 17px;
    padding-right: 17px;
    padding-top: 9px;
    padding-bottom: 9px;
    border-radius: 18px;
```



```
    margin-left: 8px;
    transition: box-shadow 0.2s;
    vertical-align: top;
    vertical-align: bottom;
    margin-bottom: 40px;
}
.summary-box:active{
    opacity: 0.6;
}
.summary-box:hover{
    box-shadow: 0px 3px 5px rgb(228, 216, 216);
}
.group-members {
    display: flex;
    justify-content: space-between;
    margin-bottom: 35px;
}

.member-box {
    font-family: 'Times New Roman', Times, serif;
    font-size: large;
    flex: 1;
    background-color: white;
    color: #000000;
    border-color: rgb(99, 156, 194);
    border-style: solid;
    border-width: 1px;
    padding-left: 15px;
    padding-right: 15px;
    padding-top: 9px;
    padding-bottom: 9px;
    border-radius: 18px;
    transition: background-color 0.5s , color 0.5s;
    vertical-align: top;
    margin-right: 2px;
    transition: box-shadow 0.2s;
}
.member-box:active{
    opacity: 0.9;
```



Explanation:

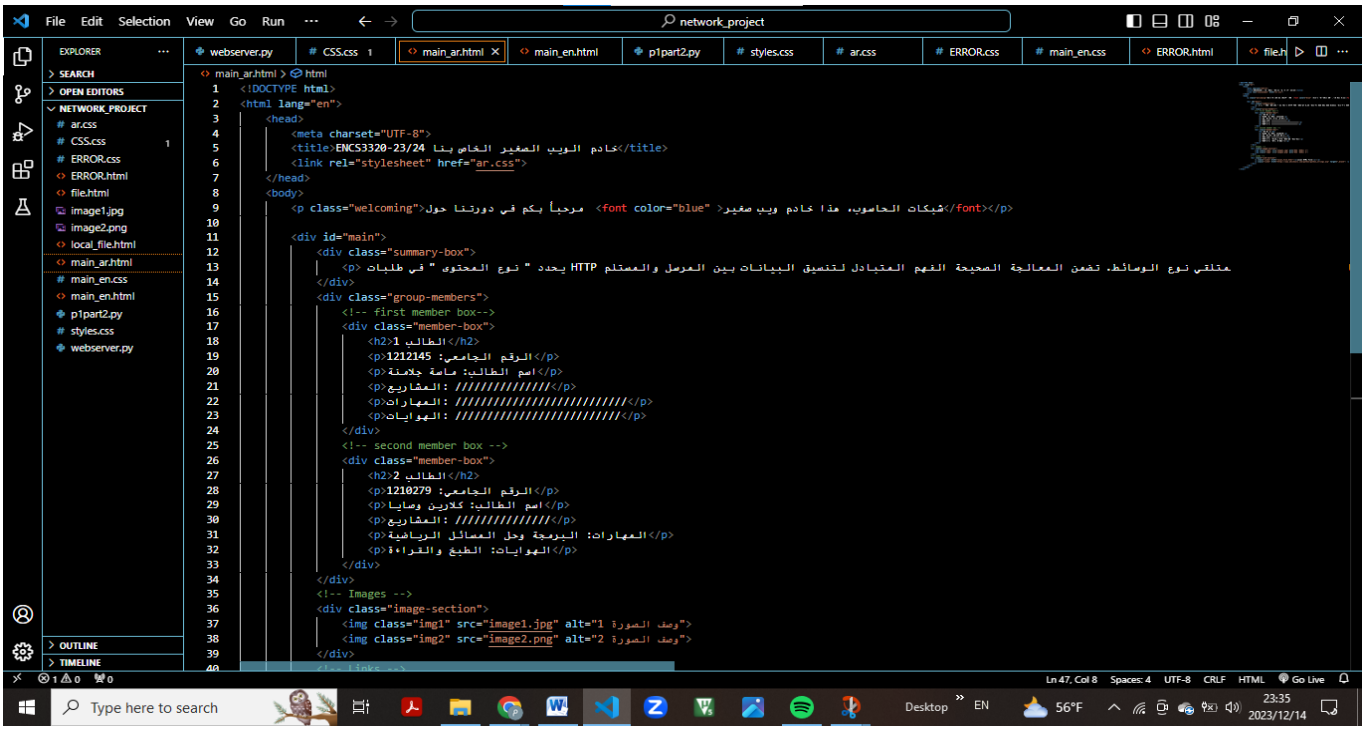
This CSS file to style the main_en html file (previously showed), we used classes to each section to target it here (in the css styling file).

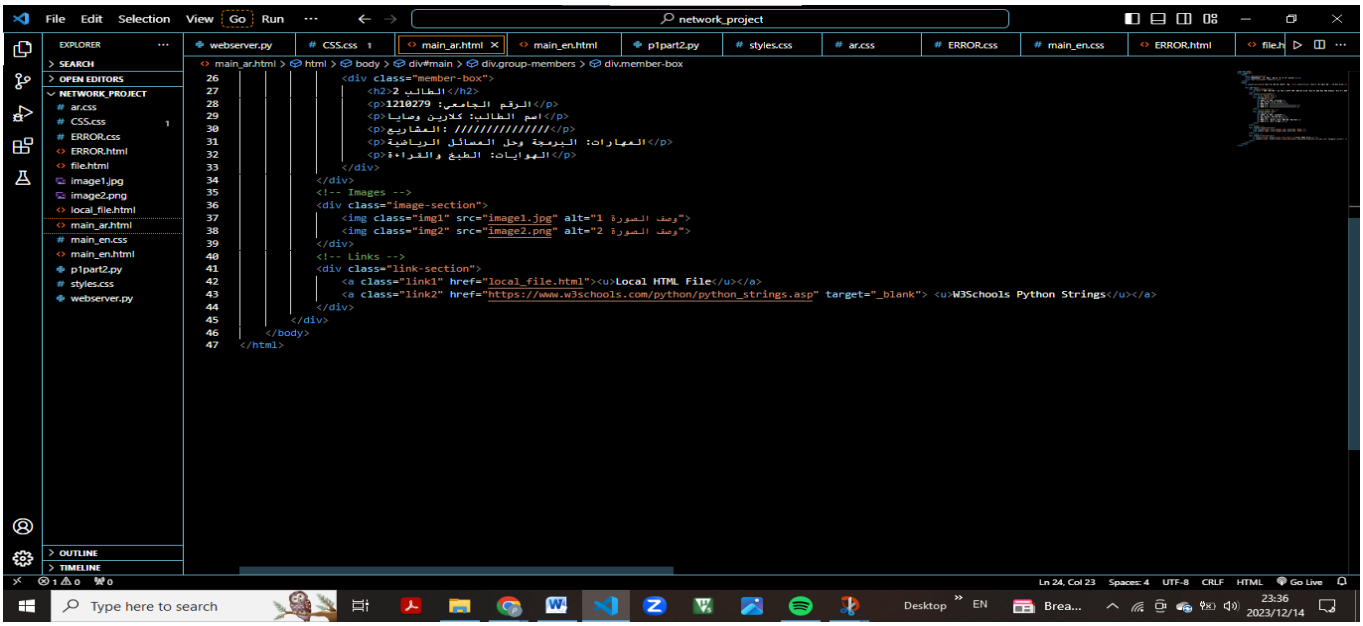
- Started by (welcoming class): set the font styling (font-family, font-size, font weight, and text align), set the border styling (border-style, border-color, border-width and border-radius), add some effects (transition, shadow and hover), and finally set the width with margin and top margin to place the welcoming phrase in a certain place in the web page.
- Then (summary class (point 0 answer)): set the font styling (font-family, font-size, color, padding), set the border styling (border(none), border-radius), add some effects (transition, shadow, hover), and finally set the width with margin, top margin, bottom margin and vertical-align to place the summary box in a certain place in the web page.

- Group member and member box classes: set the justify-content to have a space between the boxes, set the font styling (font-family, font-size, color, padding), set the border styling (border-style, border-width, border-radius), add some effects (transition, shadow, hover), and finally set the width with margin, right margin and vertical-align to place the group member and member boxes in a certain place in the web page.
  Images classes: for both of them, set the border-radius, set width and height, add some effects (transition, shadow, hover), and finally margin, bottom margin to place the images in a certain place in the web page.
- Finally the links classes: set the font styling (font-family, font-size, color, text-align, text-decoration (underline)), set the border styling (border-style, border-width, border-color, border-radius), add some effects (transition, shadow, hover), and finally set the width with margin, top margin to place the links in a certain place in the web page.

Part 3.2: main_ar.html / ar [html and css codes + explanation]

HTML code:

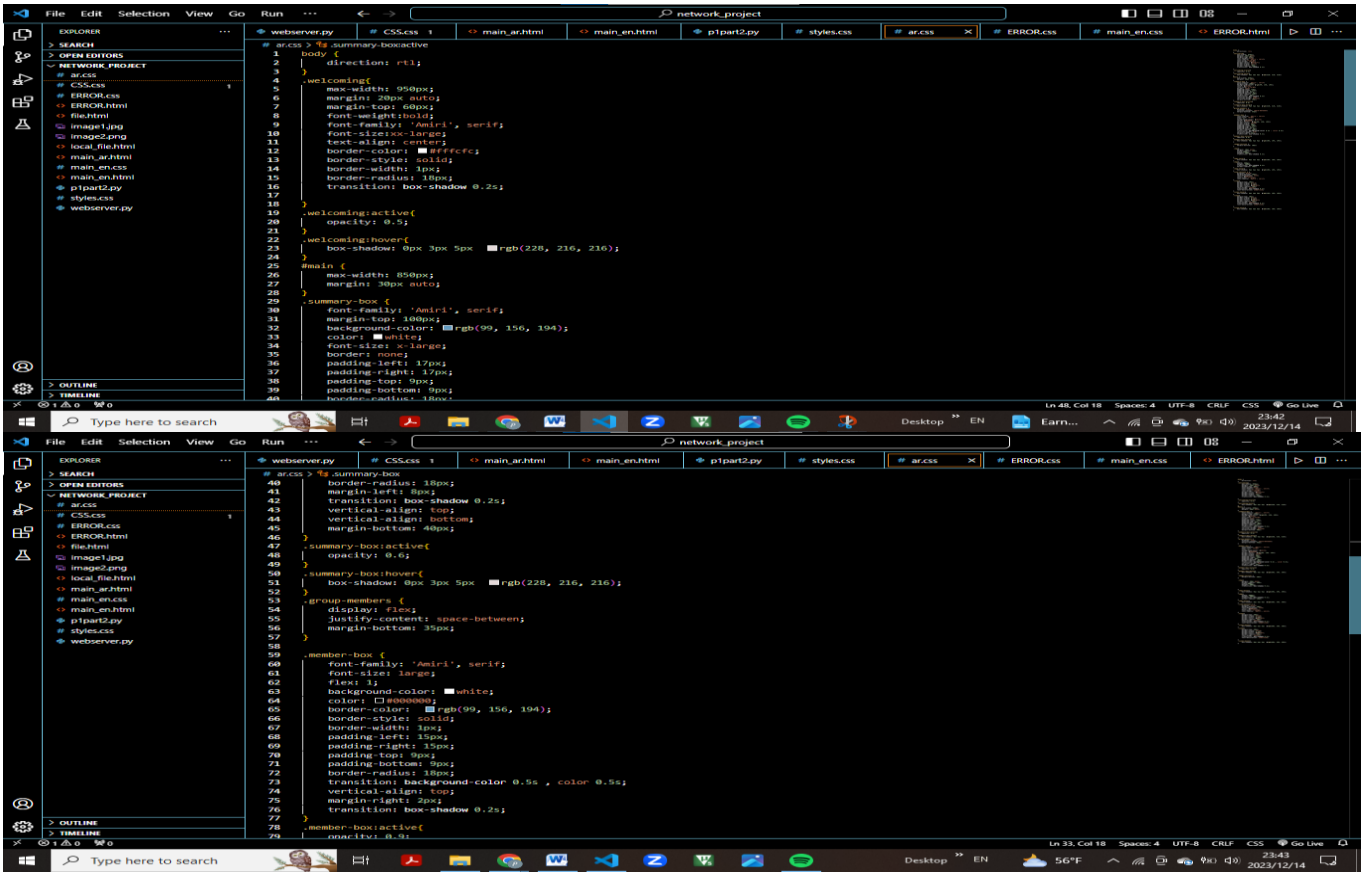


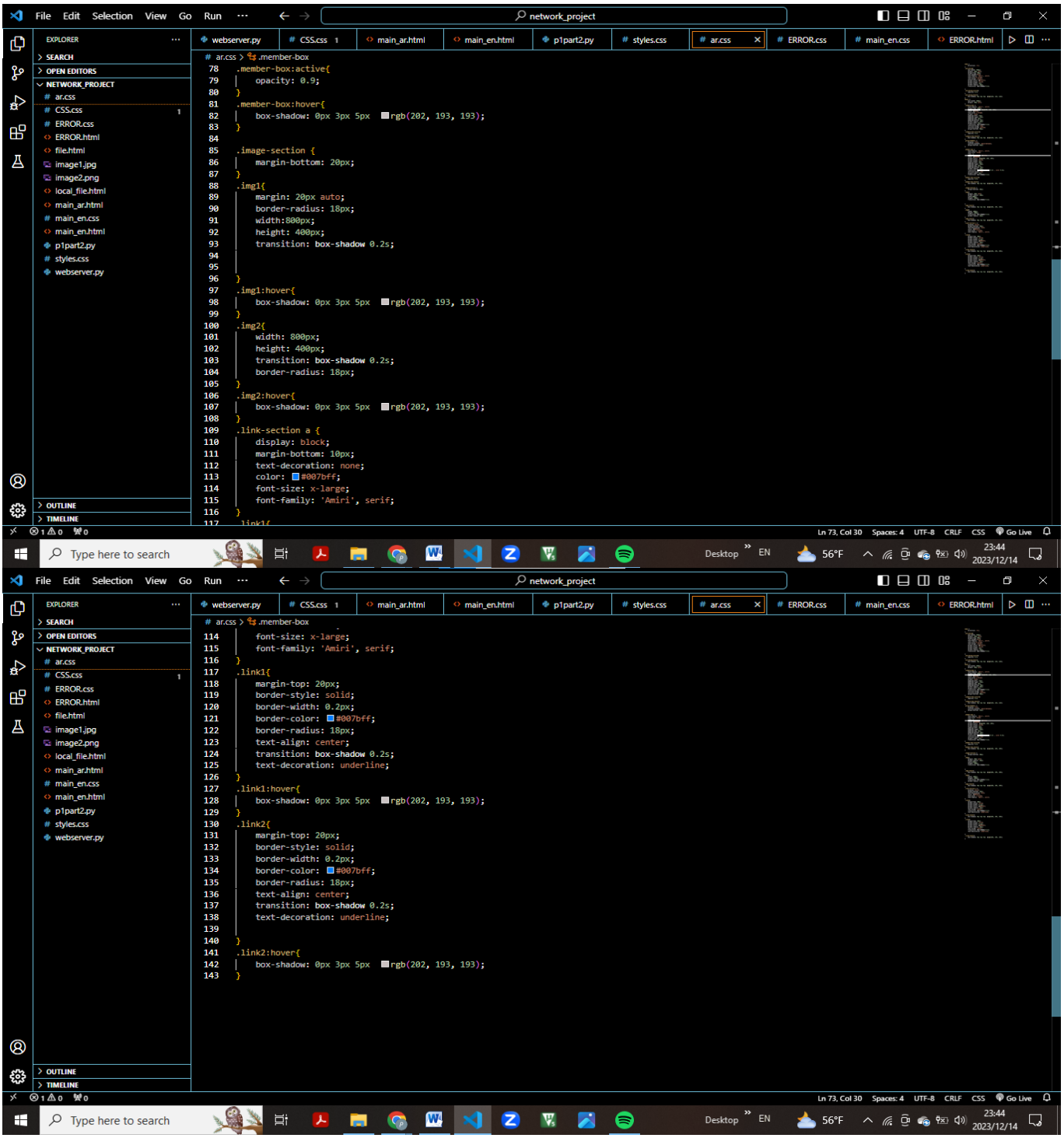


Explanation:

This file is the **Arabic** version of the previous file (main_en.html), so here we have the same content with the same page division and objects but texts are in **Arabic** instead of English with charset="UTF-8".

CSS code:

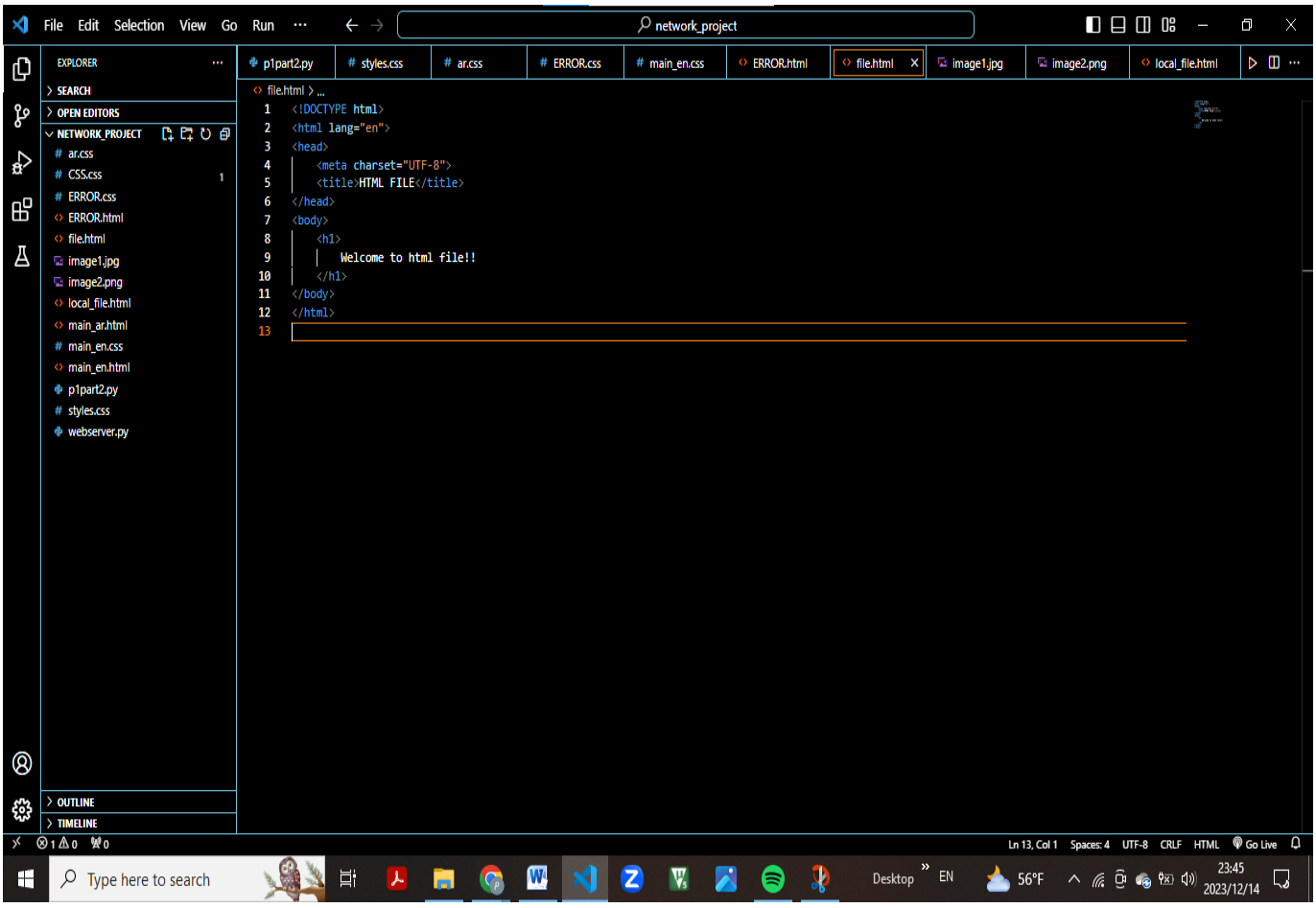




Explanation:
Also here we have the same CSS file as the previous one (main_en.css), same styling for all the section (classes), but the only different since this is a css file for html file that has Arabic text and words in it, So we added this to the whole body: **direction: rtl** so the direction of the page and texts are from right to left (Arabic).

Part 3.3: .html [code]

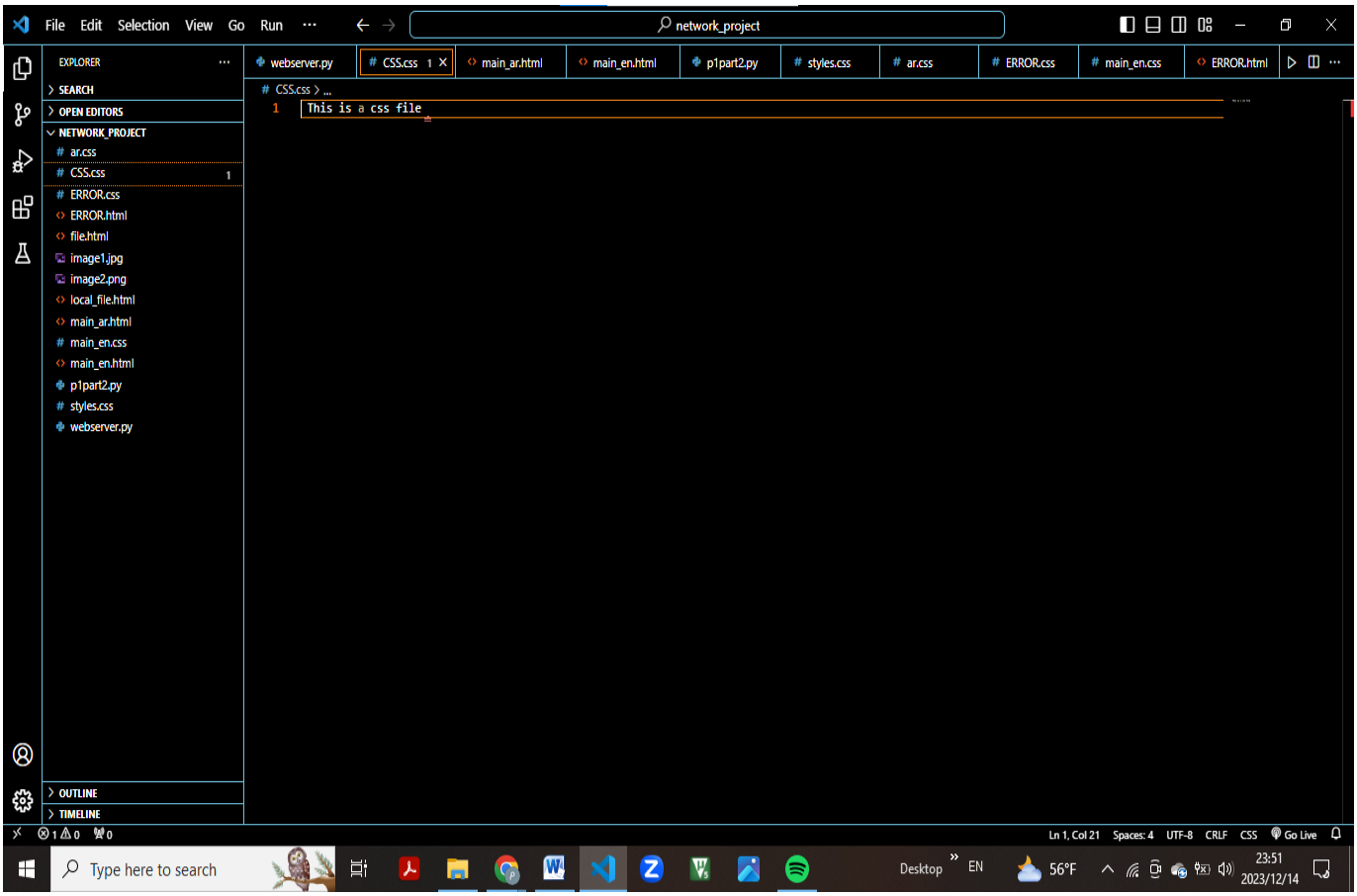
HTML code:



Explanation:

In this part we were asked to create a simple html file with Content-Type: text/html, so we created the file with a simple paragraph in the body section to be shown in the webpage.

Part 3.4: .css [code]



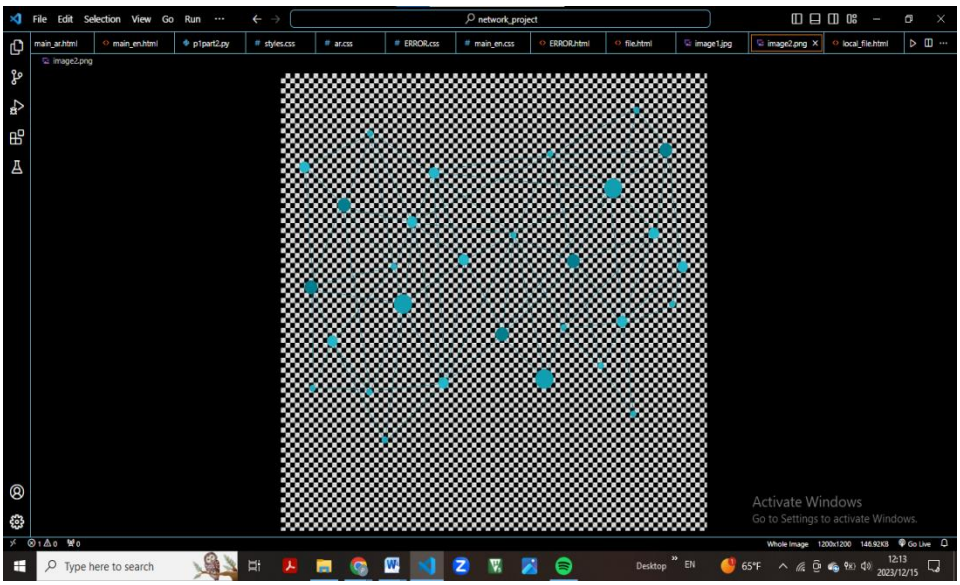
Explanation:
And here we were asked to create a simple css file with Content-Type: text/html, so we created the file with a simple paragraph to be shown in the webpage.

Part 3.5: .jpg /image1.jpg [explanation]

Explanation:

A jpg image with Content-Type: image/jpeg to be used in main_en.html, main_ar.html and as an explicit request later.

Part 3.6: .png / image2.png [explanation]



Explanation:

A png image with Content-Type: image/png to be used in main_en.html, main_ar.html and as an explicit request later.

Part 3.7: 307 Temporary Redirect [explanation]

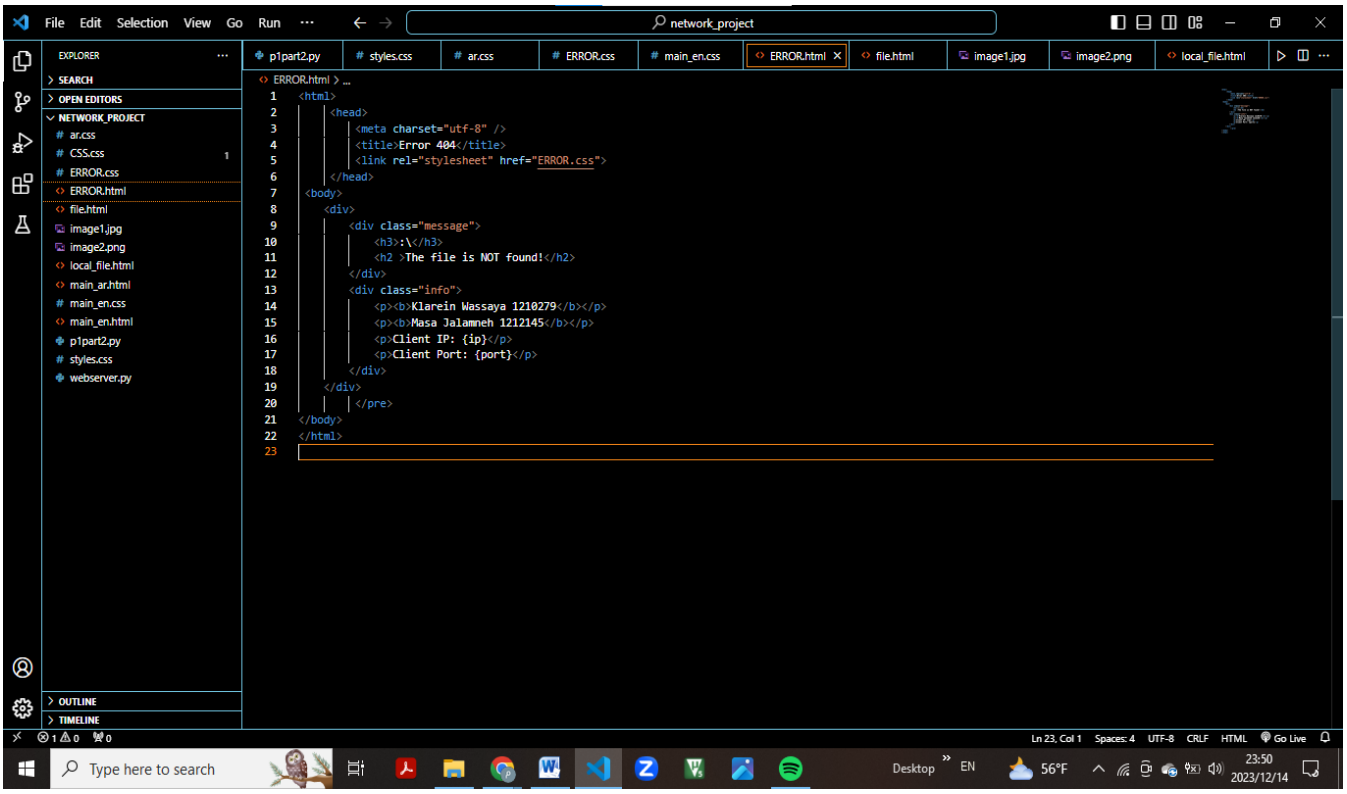
status code 307 Temporary Redirect to redirect the following

a. If the request is /cr then redirect to cornell.edu website
b. If the request is /so then redirect to stackoverflow.com website
c. If the request is /rt then redirect to ritaj website

This part was made in webserver.py code based on the request (/cr, /so, /rt).

Part 3.8: Error [html and css codes + explanation]

HTML code:



Explanation:

A simple HTML webpage to be shown if requested path is wrong or the file doesn't exist.

This html file contain:

1- "HTTP/1.1 404 Not Found" in the response status

2- "Error 404" in the title

3- "The file is not found" in the body in red

4- Names and IDs in Bold

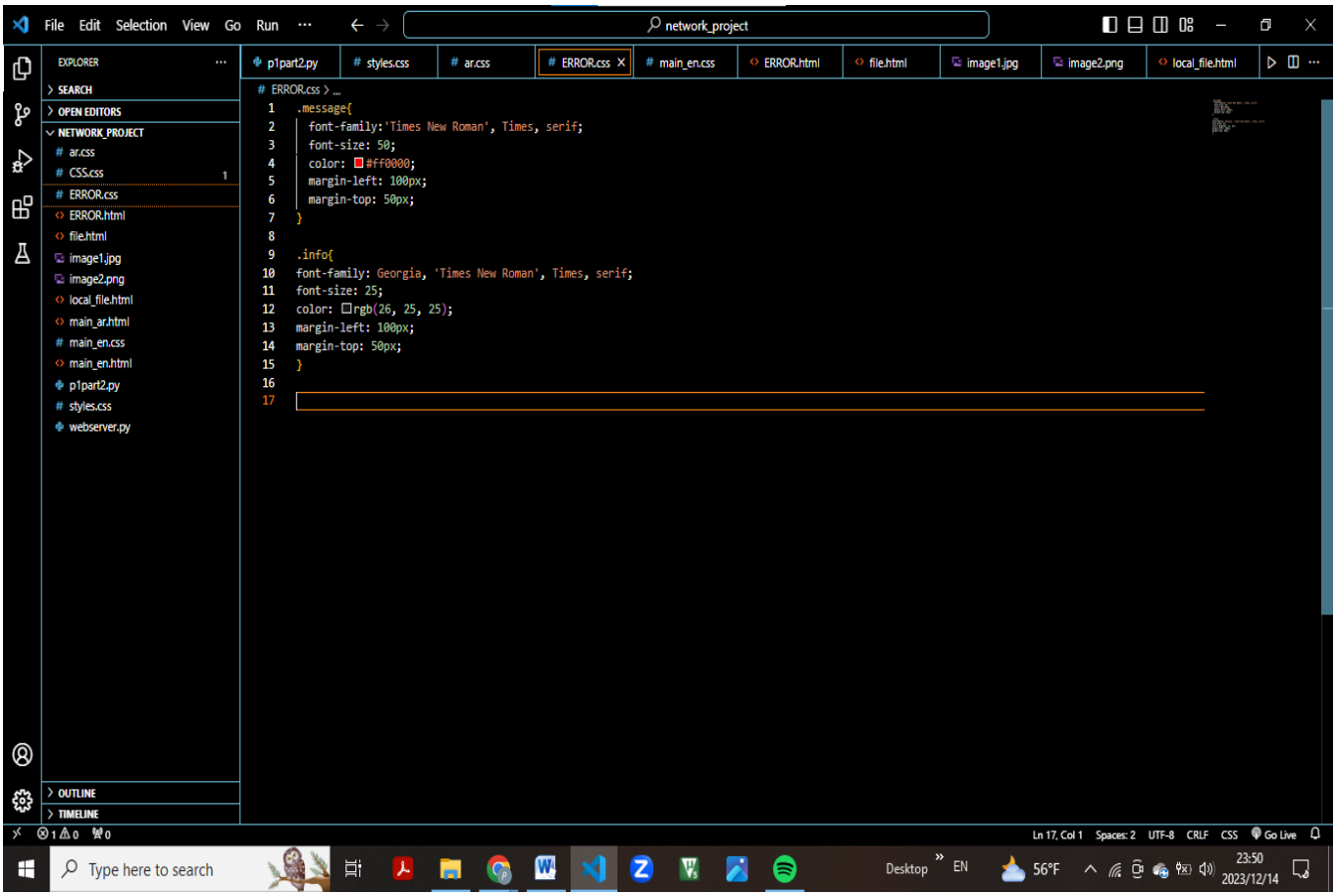
5- The IP and port number of the client

we divided the page main

then sub for the Error message. (with a class to style it with css)

and another sub for the information: names, ids, port number and number of the client (also with a class to style it with css)

CSS code:



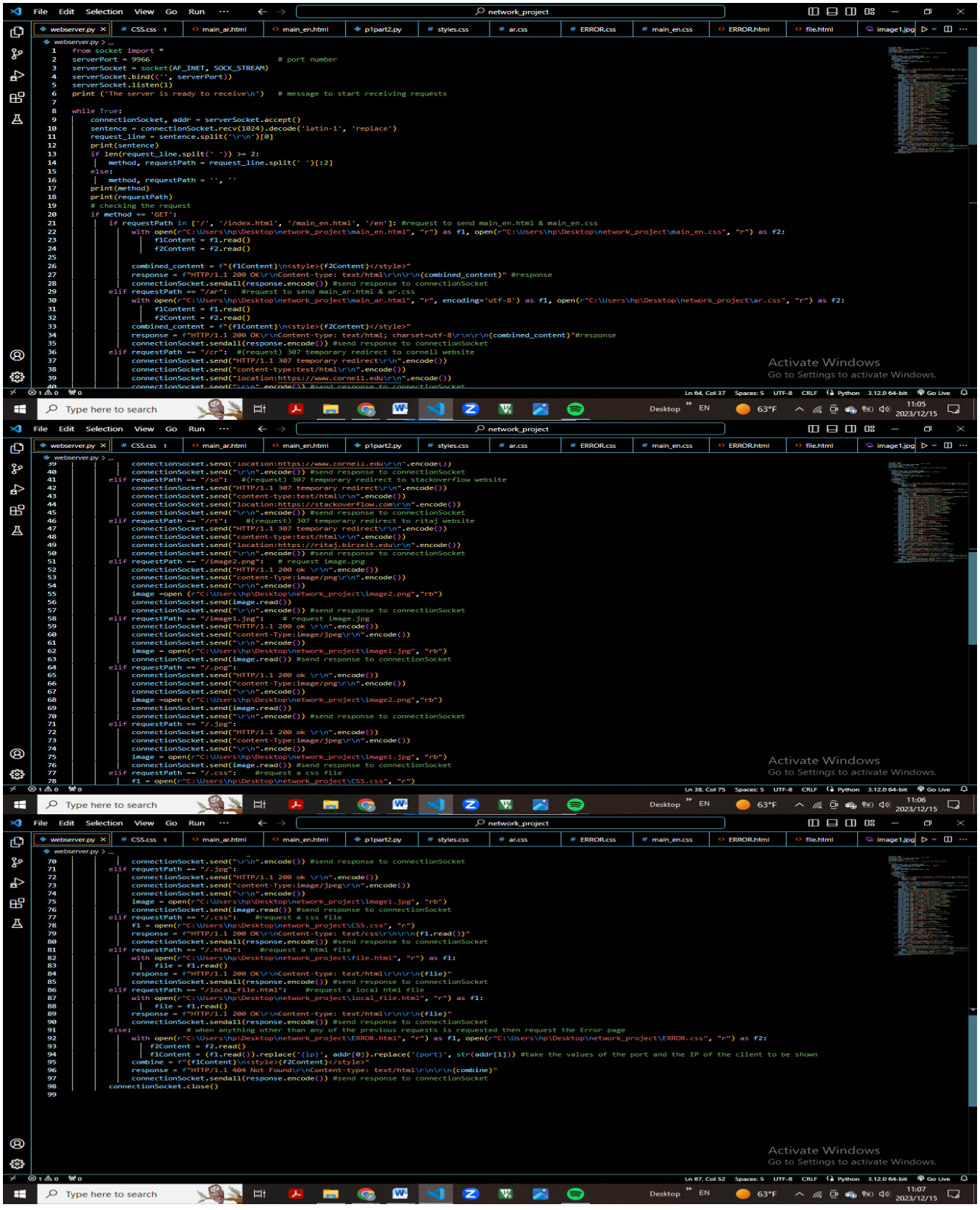
Explanation:

Styling the Error html file, starting with (message class): set the color (red), set font styling (font-size, font-family), then set margin and top margin to place in a certain spot in the webpage.

Moving to style the (information class): set font styling (font-family, font-size, color), then set left margin and top margin to place in a certain spot in the webpage.

Part 3.9: webserver.py [code + explanation]

Python code:



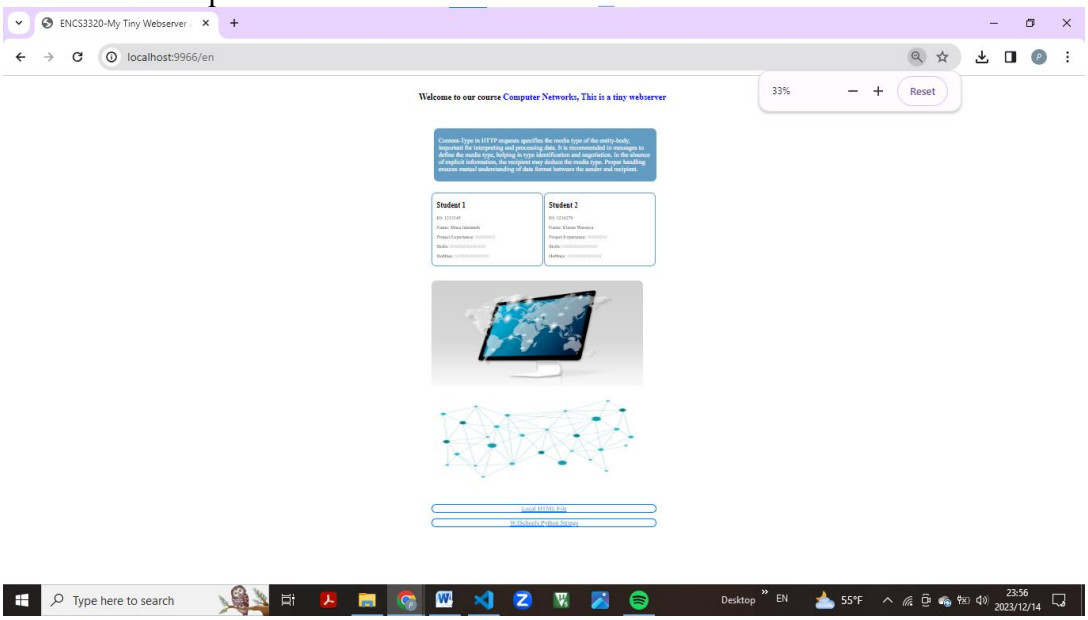
```
from socket import *
serverPort = 9966                       # port number
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1)
print ('The server is ready to receive\n')   # message to start receiving requests

while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode('latin-1', 'replace')
    request_line = sentence.split('\r\n')[0]
    print(sentence)
    if len(request_line.split(' ')) >= 2:
        method, requestPath = request_line.split(' ')[:2]
    else:
        method, requestPath = '', ''
    print(method)
    print(requestPath)
    # checking the request
    if method == 'GET':
        if requestPath in ['/', '/index.html', '/main_en.html', '/en']: #request to send main_en.html & main_en.css
            with open(r"C:\Users\hp\Desktop\network_project\main_en.html", "r") as f1, open(r"C:\Users\hp\Desktop\network_project\main_en.css", "r") as f2:
                f1Content = f1.read()
                f2Content = f2.read()

            combined_content = f"{f1Content}\n<style>{f2Content}</style>"
            response = f"HTTP/1.1 200 OK\r\nContent-type: text/html\r\n\r\n{combined_content}" #response
            connectionSocket.sendall(response.encode()) #send response to connectionSocket
        elif requestPath == "/ar":  #request to send main_ar.html & ar.css
            with open(r"C:\Users\hp\Desktop\network_project\main_ar.html", "r", encoding='utf-8') as f1, open(r"C:\Users\hp\Desktop\network_project\ar.css", "r") as f2:
                f1Content = f1.read()
                f2Content = f2.read()
            combined_content = f"{f1Content}\n<style>{f2Content}</style>"
            response = f"HTTP/1.1 200 OK\r\nContent-type: text/html; charset=utf-8\r\n\r\n{combined_content}"#response
            connectionSocket.sendall(response.encode()) #send response to connectionSocket
        elif requestPath == "/cr":  #(request) 307 temporary redirect to cornell website
            connectionSocket.send("HTTP/1.1 307 temporary redirect\r\n".encode())
            connectionSocket.send("content-type:test/html\r\n".encode())
            connectionSocket.send("location:https://www.cornell.edu\r\n".encode())
            connectionSocket.send("\r\n".encode()) #send response to connectionSocket
            connectionSocket.send("location:https://www.cornell.edu\r\n".encode())
            connectionSocket.send("\r\n".encode()) #send response to connectionSocket
        elif requestPath == "/so":   #(request) 307 temporary redirect to stackoverflow website
            connectionSocket.send("HTTP/1.1 307 temporary redirect\r\n".encode())
            connectionSocket.send("content-type:test/html\r\n".encode())
            connectionSocket.send("location:https://stackoverflow.com\r\n".encode())
            connectionSocket.send("\r\n".encode()) #send response to connectionSocket
        elif requestPath == "/rit":    #(request) 307 temporary redirect to ritaj website
            connectionSocket.send("HTTP/1.1 307 temporary redirect\r\n".encode())
            connectionSocket.send("content-type:test/html\r\n".encode())
            connectionSocket.send("location:https://ritaj.birzeit.edu\r\n".encode())
            connectionSocket.send("\r\n".encode()) #send response to connectionSocket
        elif requestPath == "/image2.png":   # request image.png
            connectionSocket.send("HTTP/1.1 200 ok \r\n".encode())
            connectionSocket.send("content-Type:image/png\r\n".encode())
            connectionSocket.send("\r\n".encode())
            image =open (r"C:\Users\hp\Desktop\network_project\image2.png","rb")
            connectionSocket.send(image.read())
            connectionSocket.send("\r\n".encode()) #send response to connectionSocket
        elif requestPath == "/image1.jpg":    # request image.jpg
            connectionSocket.send("HTTP/1.1 200 ok \r\n".encode())
            connectionSocket.send("content-Type:image/jpeg\r\n".encode())
            connectionSocket.send("\r\n".encode())
            image = open(r"C:\Users\hp\Desktop\network_project\image1.jpg", "rb")
            connectionSocket.send(image.read()) #send response to connectionSocket
        elif requestPath == "/.png":
            connectionSocket.send("HTTP/1.1 200 ok \r\n".encode())
            connectionSocket.send("content-Type:image/png\r\n".encode())
            connectionSocket.send("\r\n".encode())
            image =open (r"C:\Users\hp\Desktop\network_project\image2.png","rb")
            connectionSocket.send(image.read())
            connectionSocket.send("\r\n".encode()) #send response to connectionSocket
        elif requestPath == "/.jpg":
            connectionSocket.send("HTTP/1.1 200 ok \r\n".encode())
            connectionSocket.send("content-Type:image/jpeg\r\n".encode())
            connectionSocket.send("\r\n".encode())
            image = open(r"C:\Users\hp\Desktop\network_project\image1.jpg", "rb")
            connectionSocket.send(image.read()) #send response to connectionSocket
        elif requestPath == "/.css":     #request a css file
            f1 = open(r"C:\Users\hp\Desktop\network_project\CSS.css", "r")
            response = f"HTTP/1.1 200 OK\r\nContent-type: text/css\r\n\r\n{f1.read()}"
            connectionSocket.sendall(response.encode()) #send response to connectionSocket
        elif requestPath == "/.html":    #request a html file
            with open(r"C:\Users\hp\Desktop\network_project\file.html", "r") as f1:
                file = f1.read()
            response = f"HTTP/1.1 200 OK\r\nContent-type: text/html\r\n\r\n{file}"
            connectionSocket.sendall(response.encode()) #send response to connectionSocket
        elif requestPath == "/local_file.html":    #request a local html file
            with open(r"C:\Users\hp\Desktop\network_project\local_file.html", "r") as f1:
                file = f1.read()
            response = f"HTTP/1.1 200 OK\r\nContent-type: text/html\r\n\r\n{file}"
            connectionSocket.sendall(response.encode()) #send response to connectionSocket
        else:           # when anything other than any of the previous requests is requested then request the Error page
            with open(r"C:\Users\hp\Desktop\network_project\ERROR.html", "r") as f1, open(r"C:\Users\hp\Desktop\network_project\ERROR.css", "r") as f2:
                f2Content = f2.read()
                f1Content = (f1.read()).replace('{ip}', addr[0]).replace('{port}', str(addr[1])) #take the values of the port and the IP of the client to be shown
            combine = f"{f1Content}\n<style>{f2Content}</style>"
            response = f"HTTP/1.1 404 Not Found\r\nContent-type: text/html\r\n\r\n{combine}"
            connectionSocket.sendall(response.encode()) #send response to connectionSocket
        connectionSocket.close()

```

Explanation:

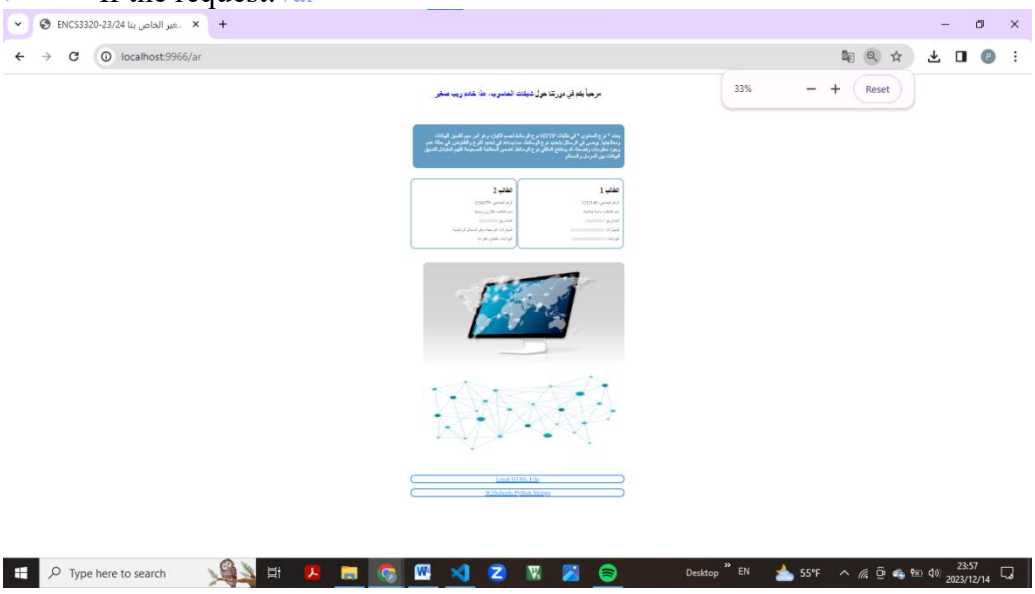
This is the code for our tiny webserver written in python language, the server listens on port 9966. This server is designed to open specific webpages depending on the path of the received request message, it receives the message and prints it on the terminal, take the first line of it, which includes the method (that should be GET) and the path, the server again prints the method and the path on the terminal, then depending on the path, some actions are taken. If the server receives /en path, it should open the html file and its css file and sends a response that indicates that the request was successful, then the webpage is opened and shown to the user who requested it, same for paths /ar, /.png, /.jpg, /.css and /.html. For paths like /cr, /rt, /so, the server sends a temporary redirect response to the client to go to specific locations, for /cr the www.cornell.edu website will open, for /rt the ritaj.birzeit.edu will open and for /so the stackoverflow.com website will open. If an unexpected path is received the server sends an error response to the client and opens an error webpage (which is designed by us).

Part 3.9: run (1-7) [screenshots]

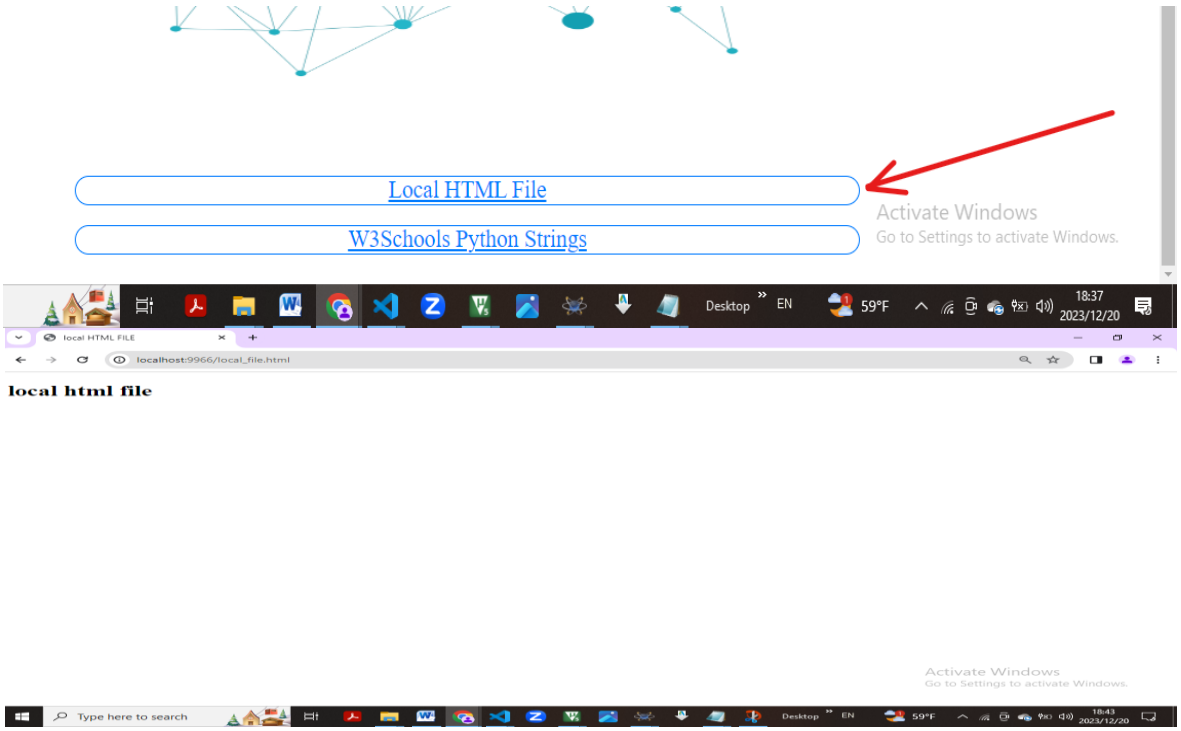
- If the request: / or /index.html or /main_en.html or /en



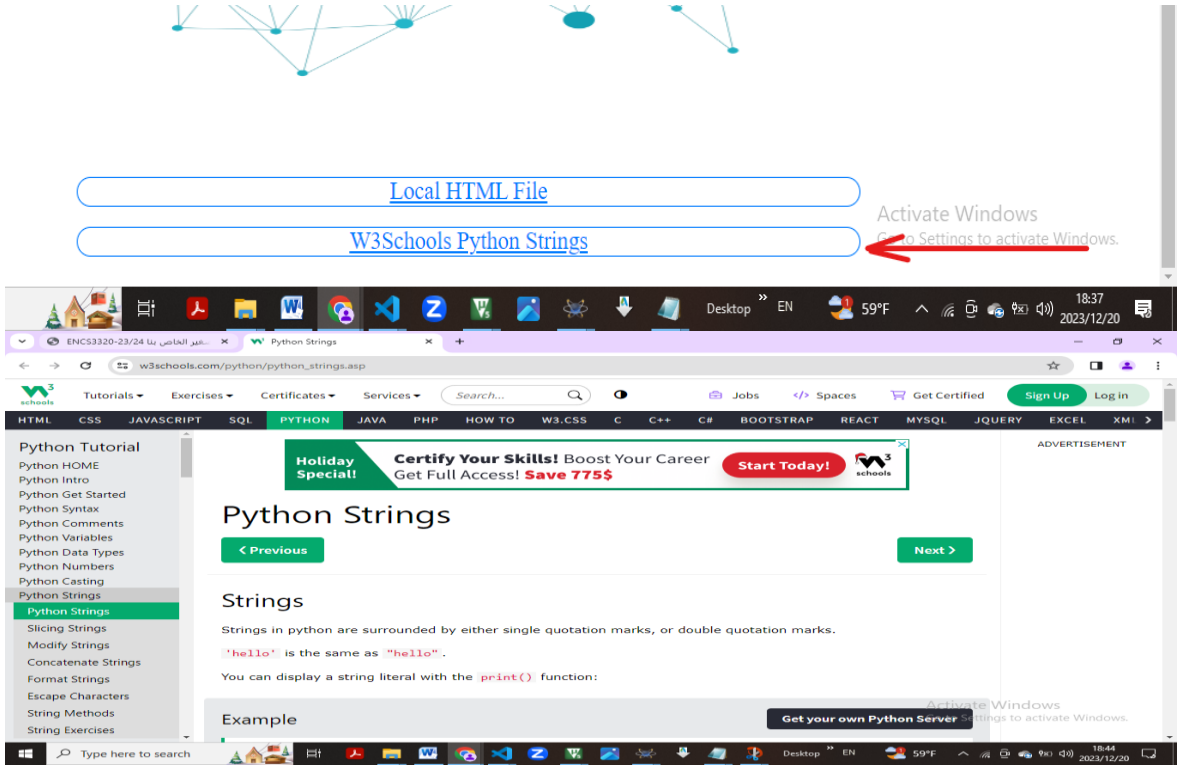
- If the request: /ar



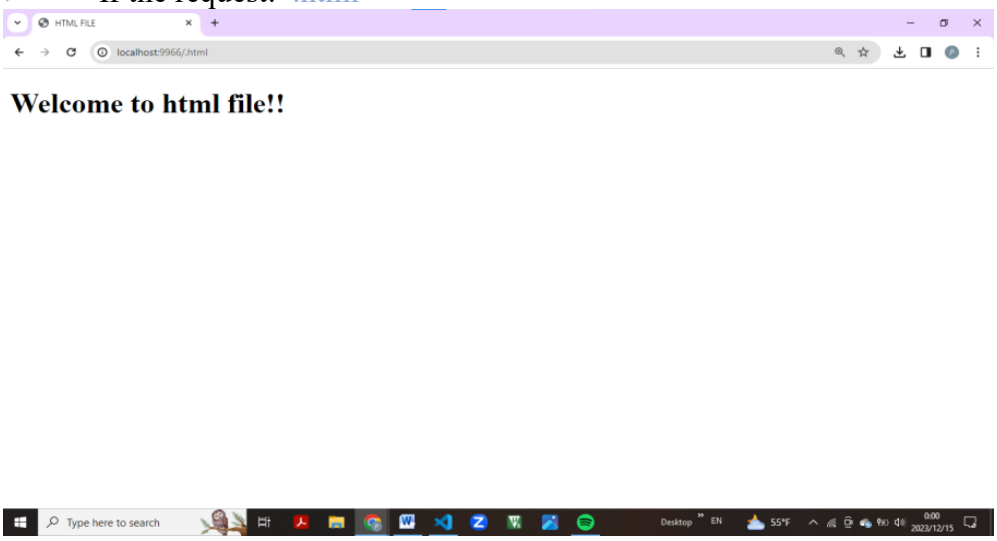
- in (main_en and main_ar): link => local html file



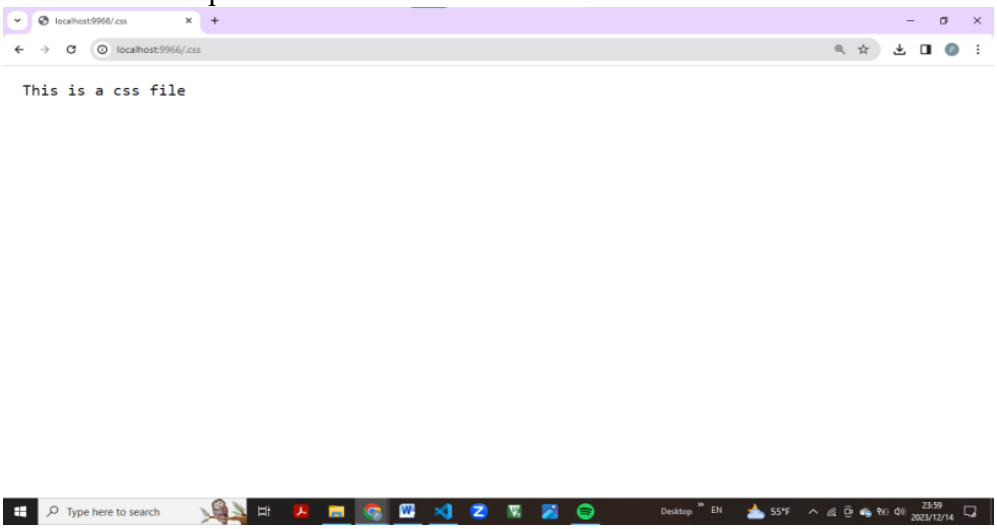
- in (main_en and main_ar): link => W3Schools Python Strings



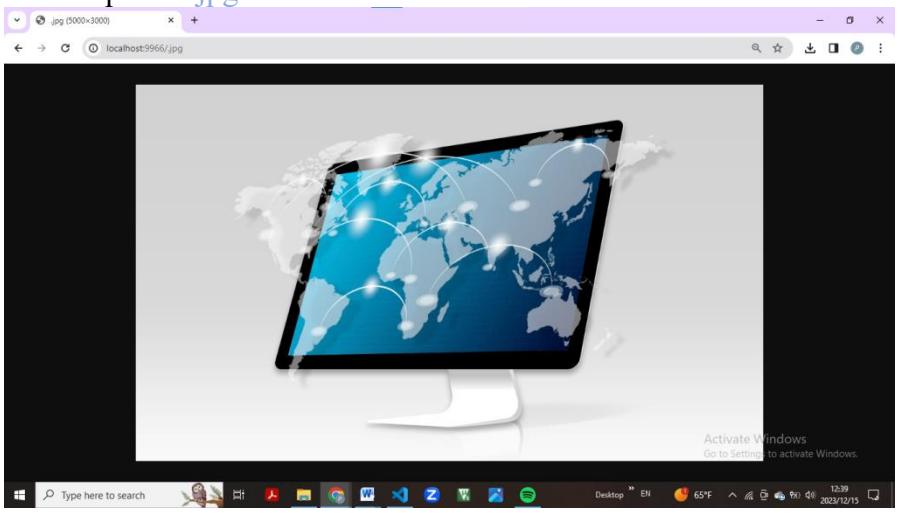
- If the request: .html



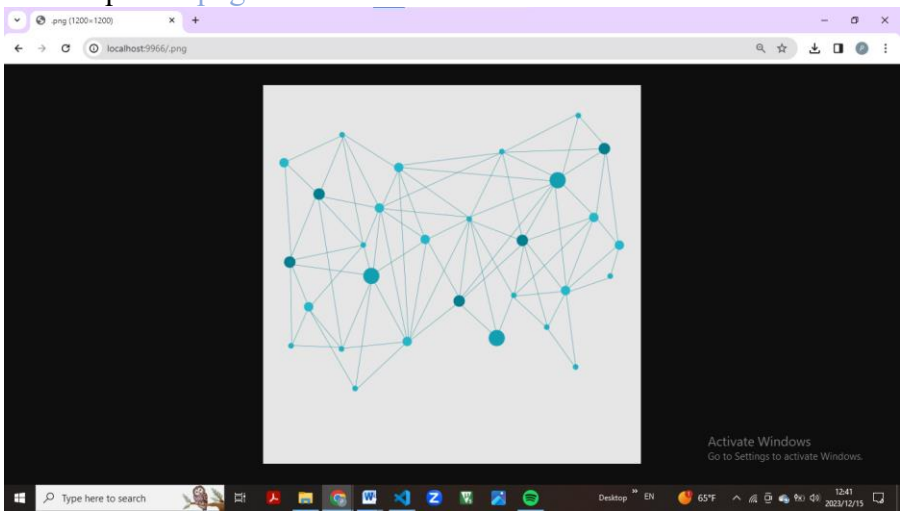
- If the request: .css



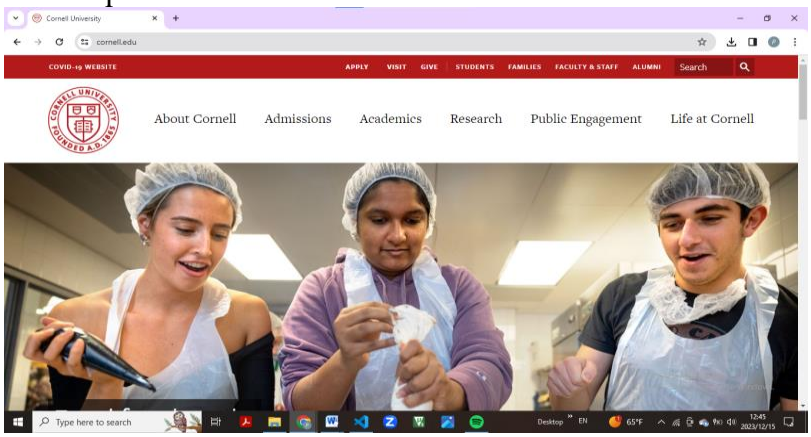
- If the request: .jpg



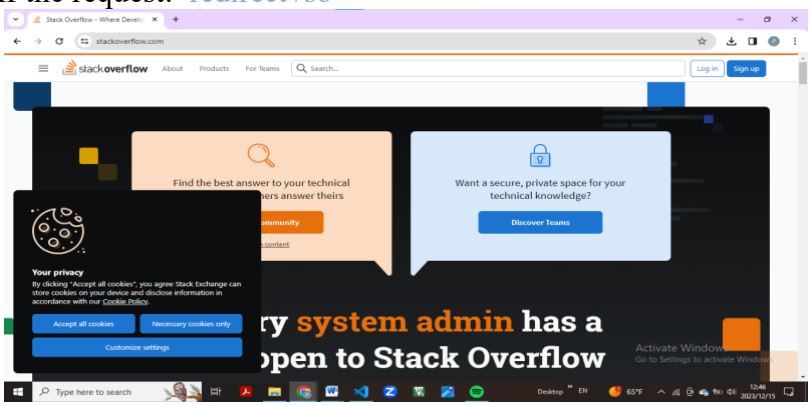
- If the request: .png



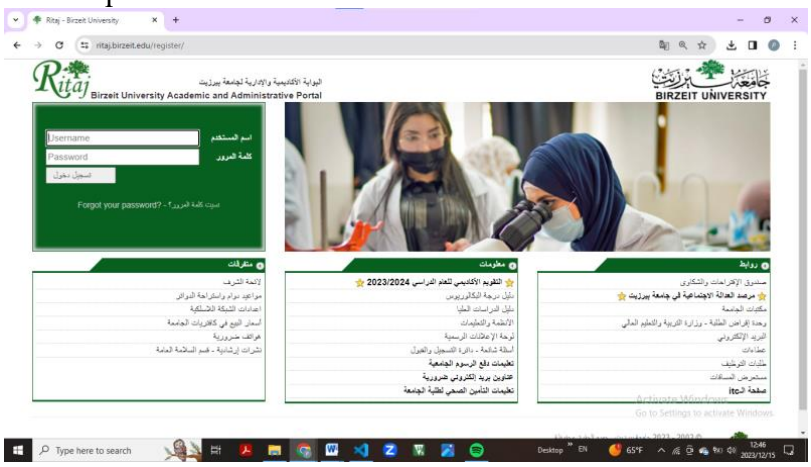
- If the request: redirect /cr



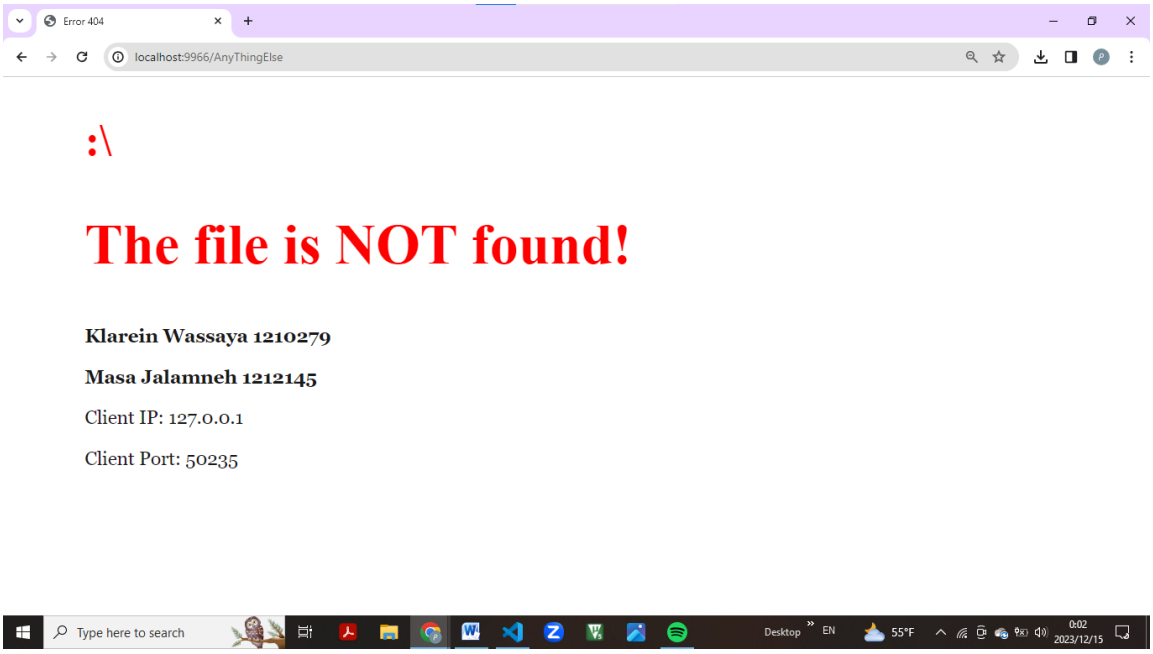
- If the request: redirect /so



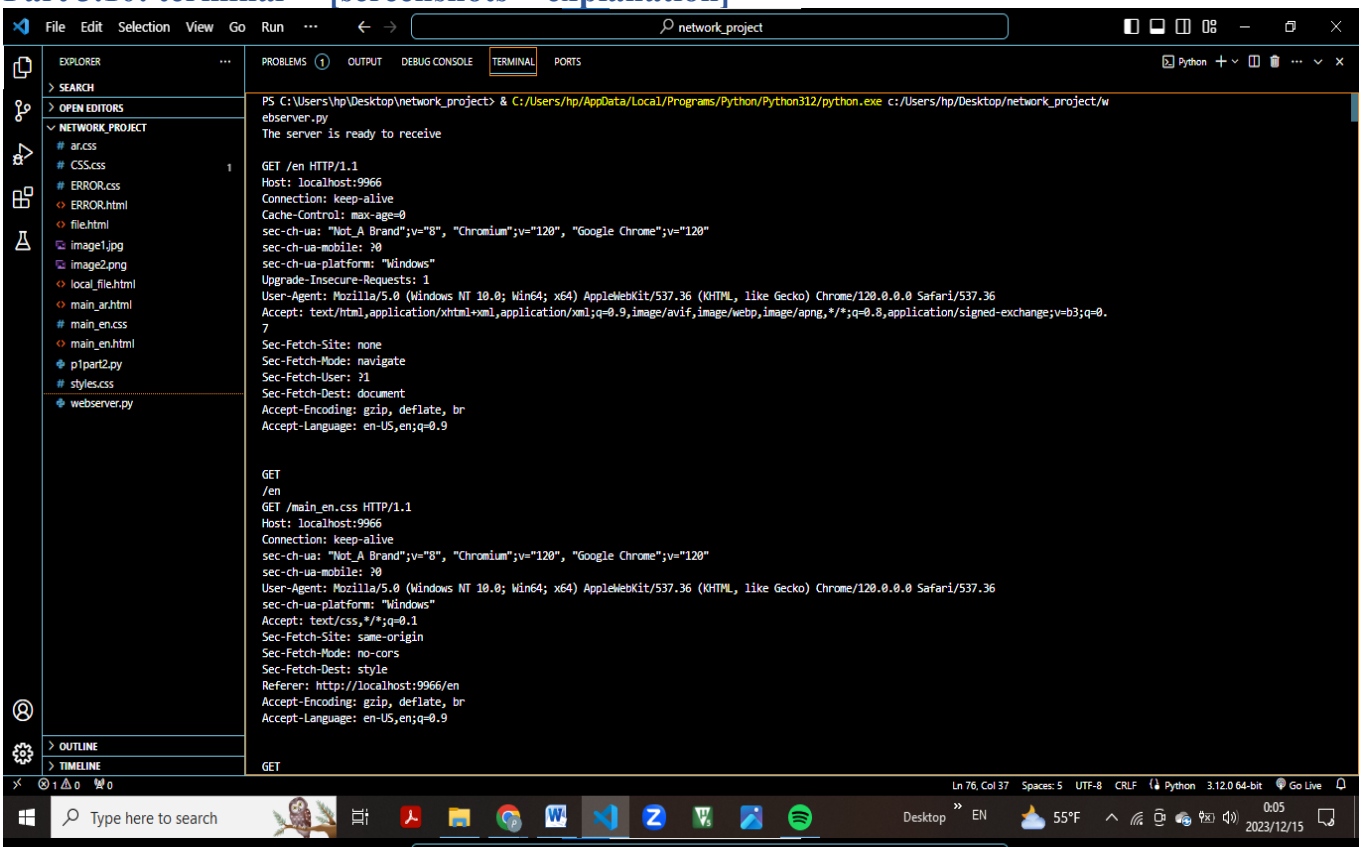
- If the request: redirect /rt

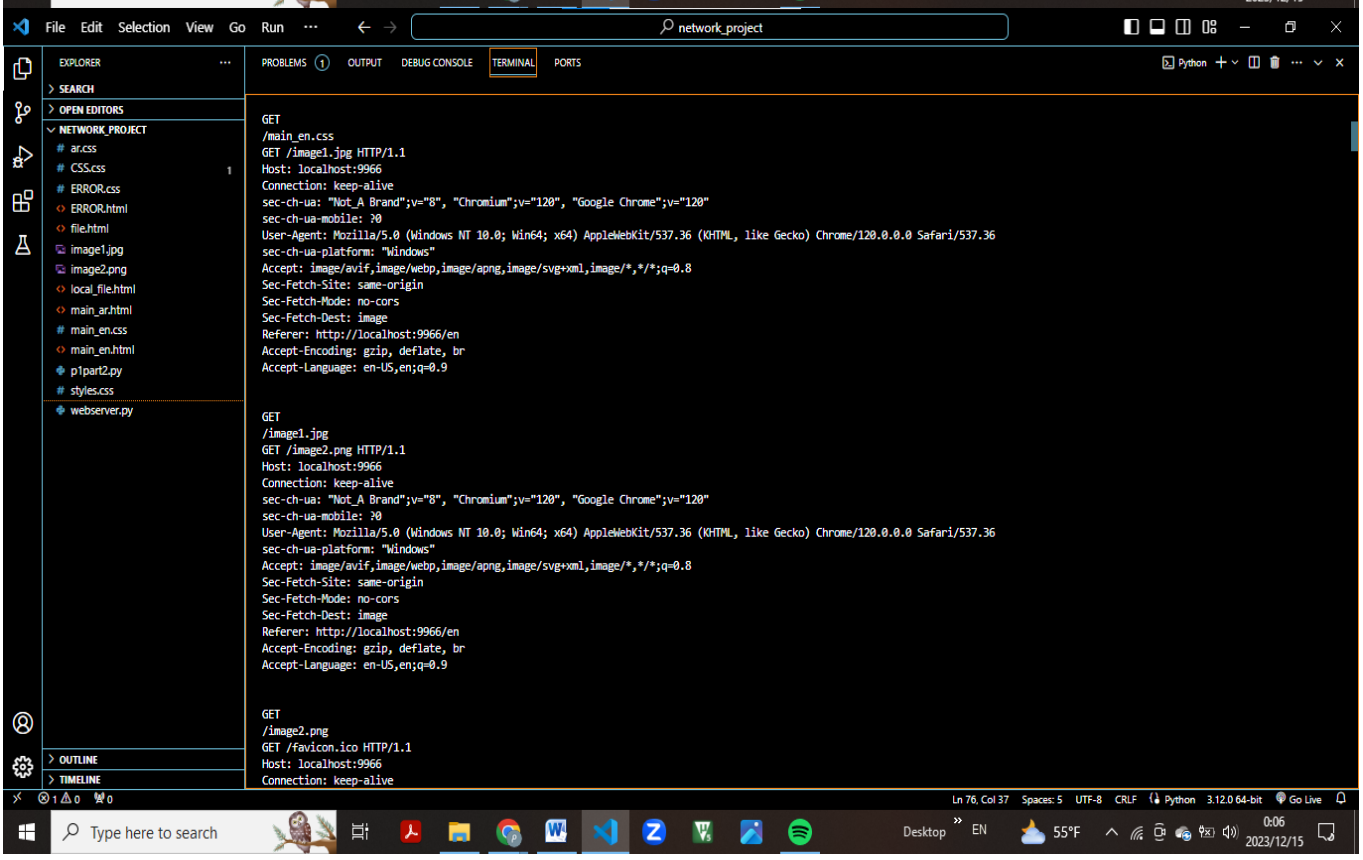


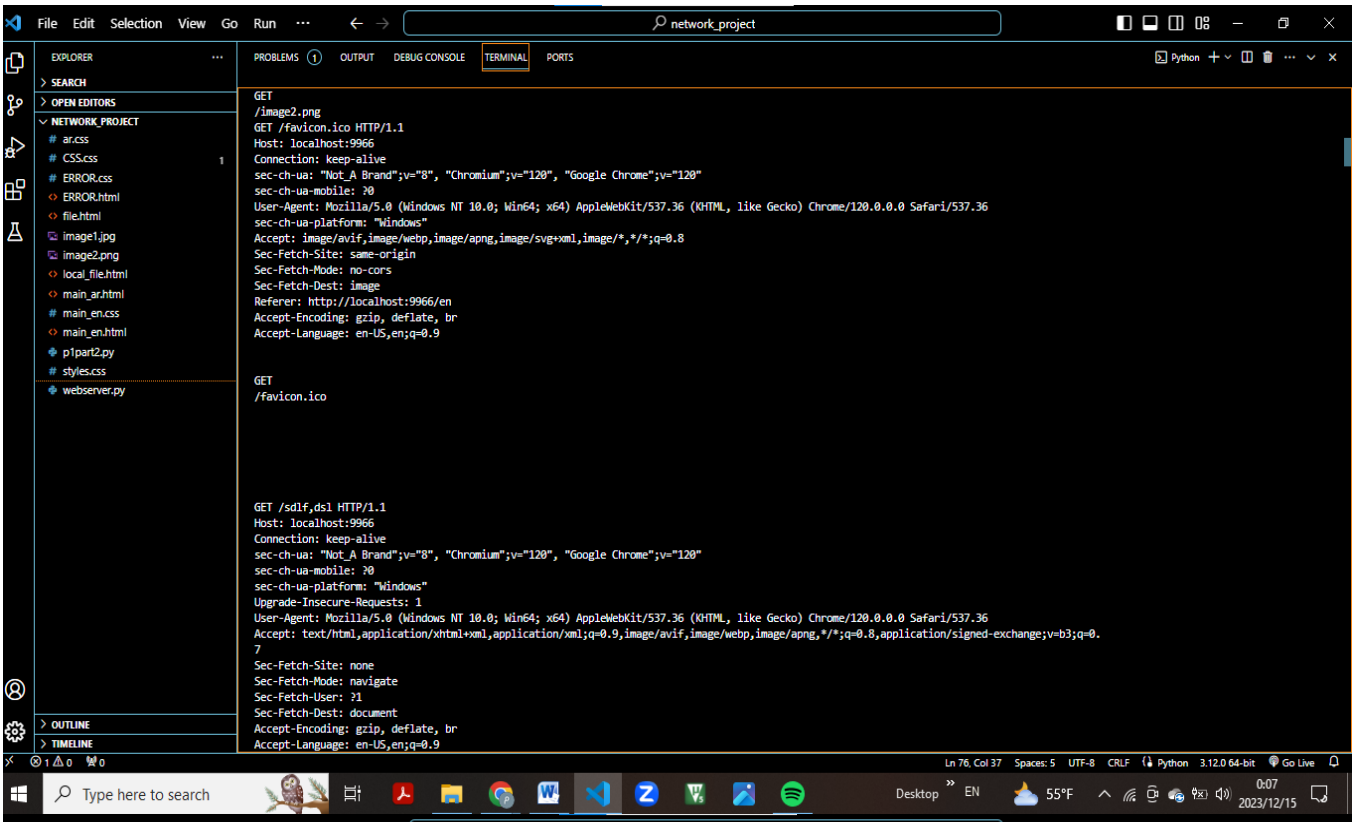
➢ If the request: /anything else (error)

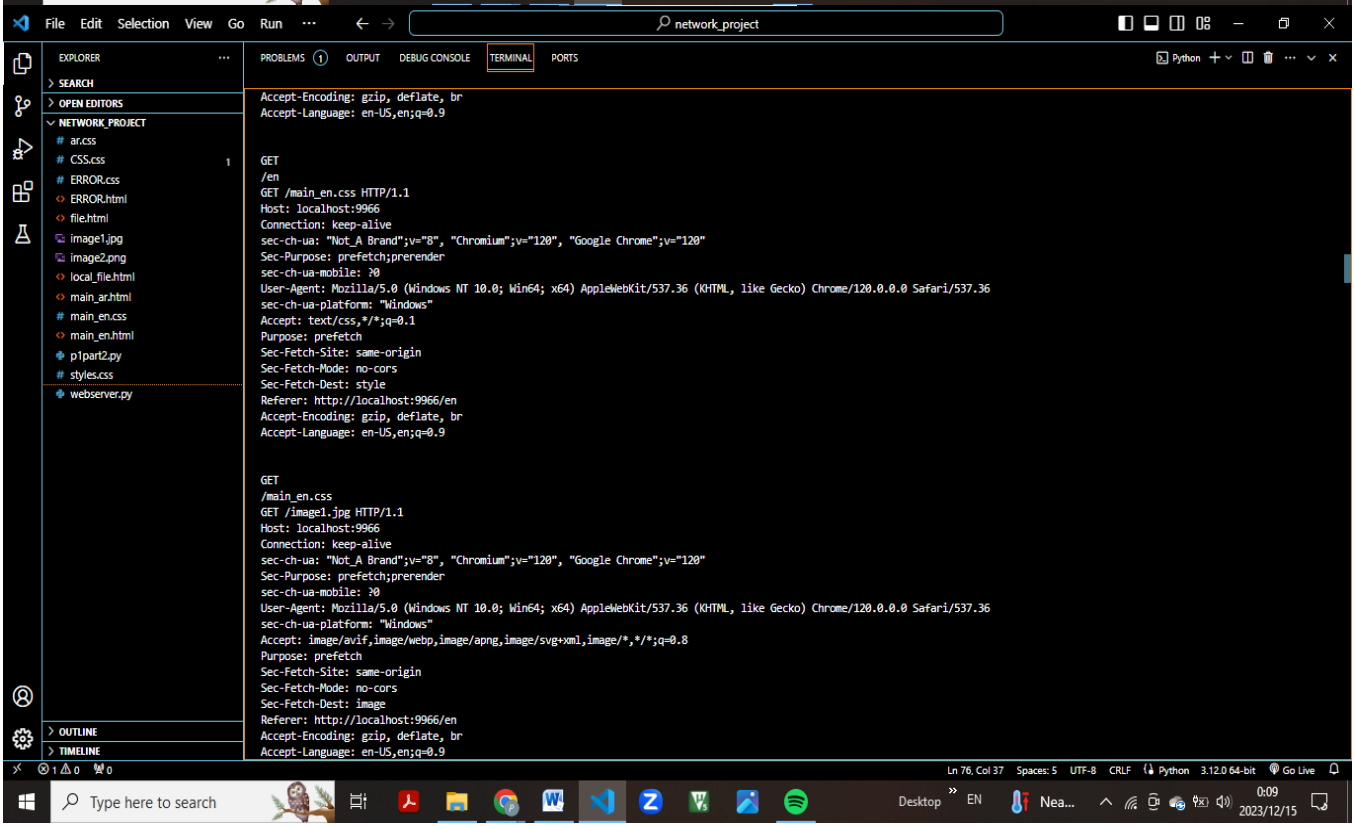


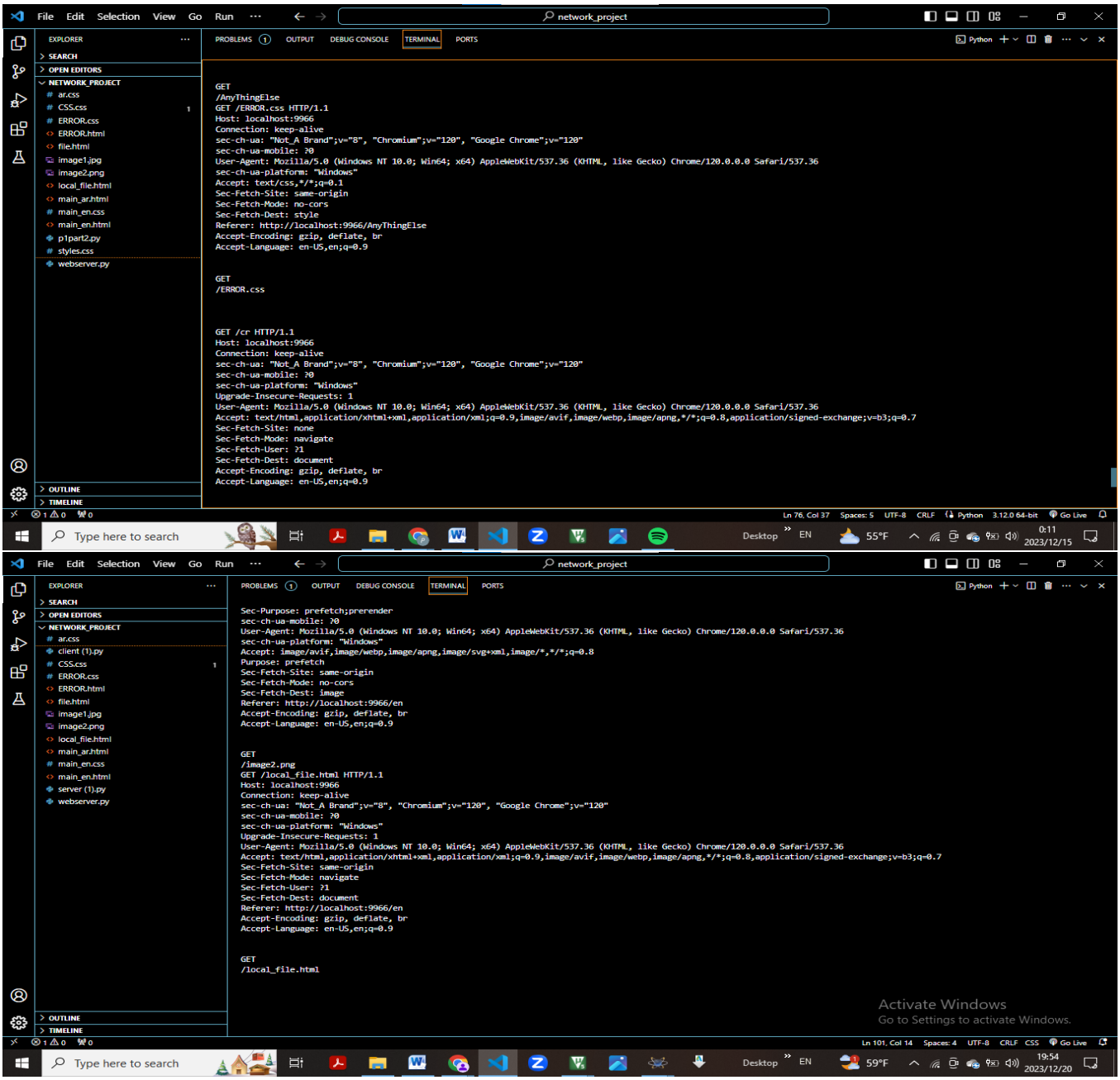
Part 3.10: terminal [screenshots + explanation]






File Edit Selection View Go Run ···
network_project
EXPLORER
SEARCH
OPEN EDITORS
NETWORK_PROJECT
ar.css
CSS.css
ERROR.css
ERROR.html
file.html
image1.jpg
image2.png
local_file.html
main_ar.html
main_en.css
main_en.html
p1part2.py
styles.css
webserver.py
OUTLINE
TIMELINE
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
GET
/image2.png
GET /favicon.ico HTTP/1.1
Host: localhost:9966
Connection: keep-alive
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:9966/en
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
GET
/favicon.ico
GET /sdlf,dsl HTTP/1.1
Host: localhost:9966
Connection: keep-alive
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
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/120.0.0.0 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
Accept-Language: en-US,en;q=0.9
Ln 76, Col 37 Spaces: 5 UTF-8 CRLF Python 3.12.0 64-bit Go Live
Type here to search
Desktop EN 55°F
0:07
2023/12/15


File Edit Selection View Go Run ···
network_project
EXPLORER
SEARCH
OPEN EDITORS
NETWORK_PROJECT
ar.css
CSS.css
ERROR.css
ERROR.html
file.html
image1.jpg
image2.png
local_file.html
main_ar.html
main_en.css
main_en.html
p1part2.py
styles.css
webserver.py
OUTLINE
TIMELINE
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
GET
/en
GET /main_en.css HTTP/1.1
Host: localhost:9966
Connection: keep-alive
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
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/120.0.0.0 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:9966/en
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
GET
/main_en.css
GET /image1.jpg HTTP/1.1
Host: localhost:9966
Connection: keep-alive
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
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/120.0.0.0 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:9966/en
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Ln 76, Col 37 Spaces: 5 UTF-8 CRLF Python 3.12.0 64-bit Go Live
Type here to search
Desktop EN Nea...
0:09
2023/12/15



Explanation:

We notice in the terminal that there are internal requests that happen when the server receives a request to open an html file like /en, /ar, this happens because opening these files requires opening css files , png files and jpg files, so the server should do that. Same for the ERROR.html file, the server should open the ERROR.css file to combine the style with the webpage, the webpage shown to the user will be styled.

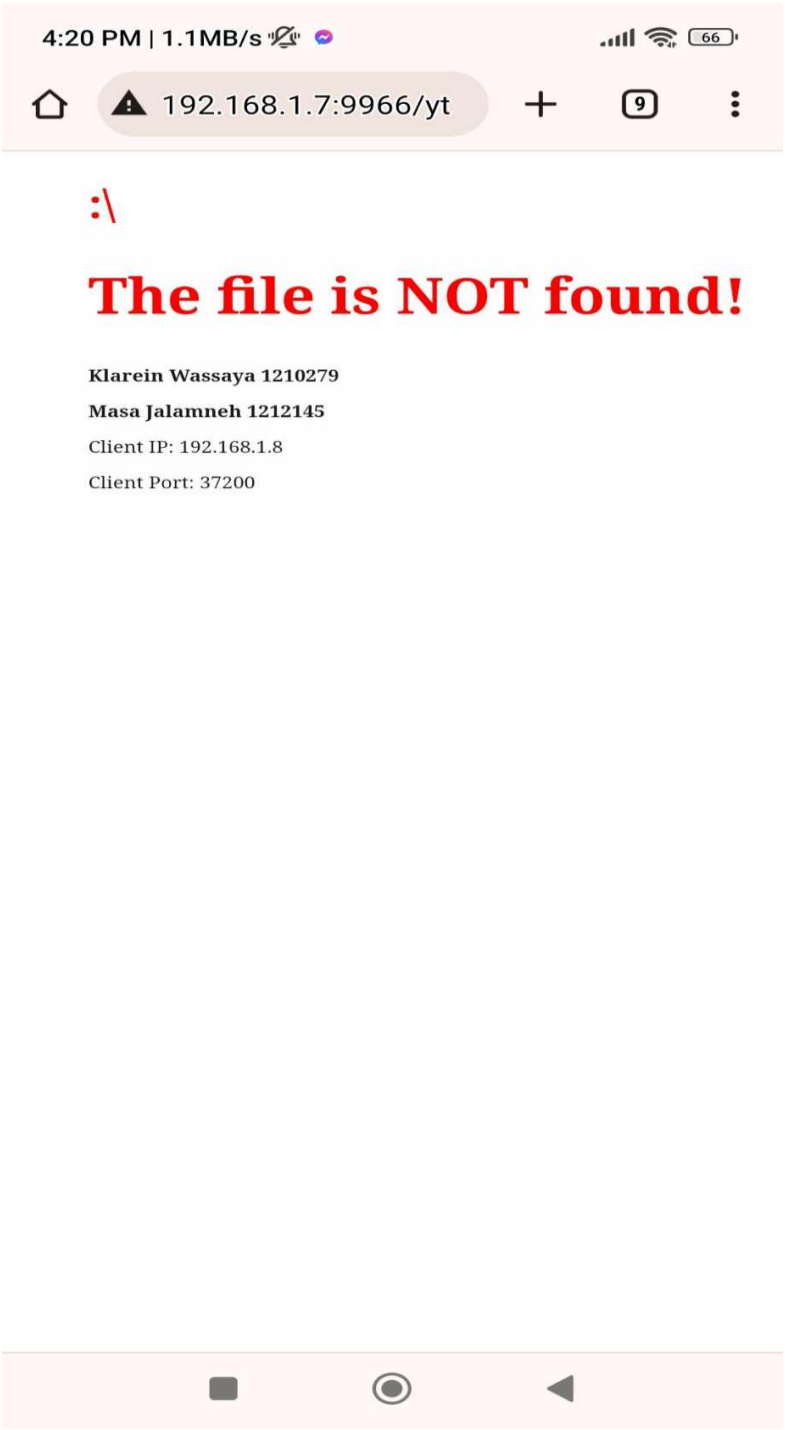
For the /.jpg request that indicates that the r, the terminal sends a response request was successful, then it opens the image to be shown to the user who sent the request, same for /.png request.

The last picture is for the /local_file.html request as a link in /en, /ar, html files, the terminal sends a response request was successful, then it opens the html file to be shown to the user who clicked on the link.

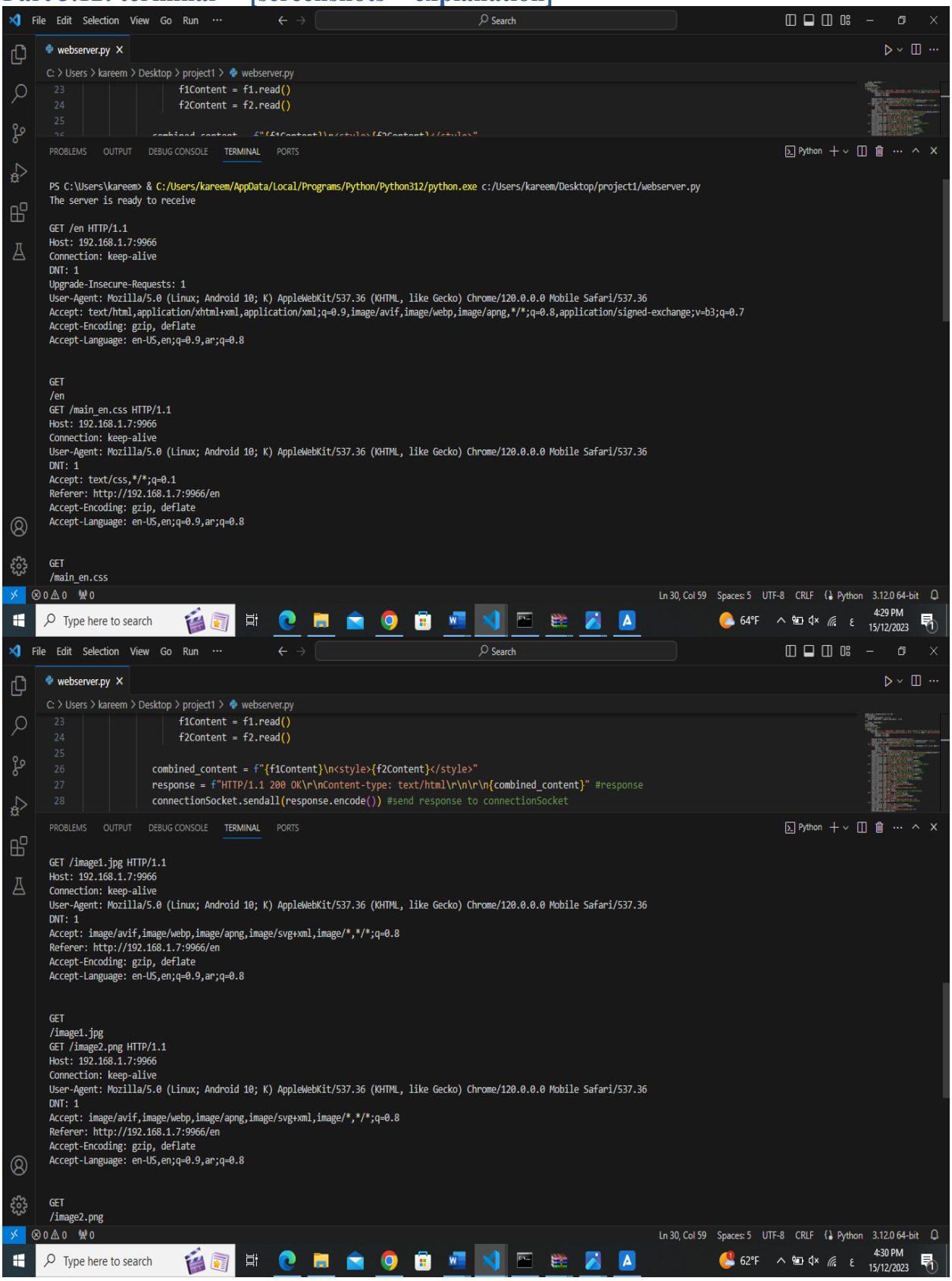
Part 3.11: run from a different device [screenshots]

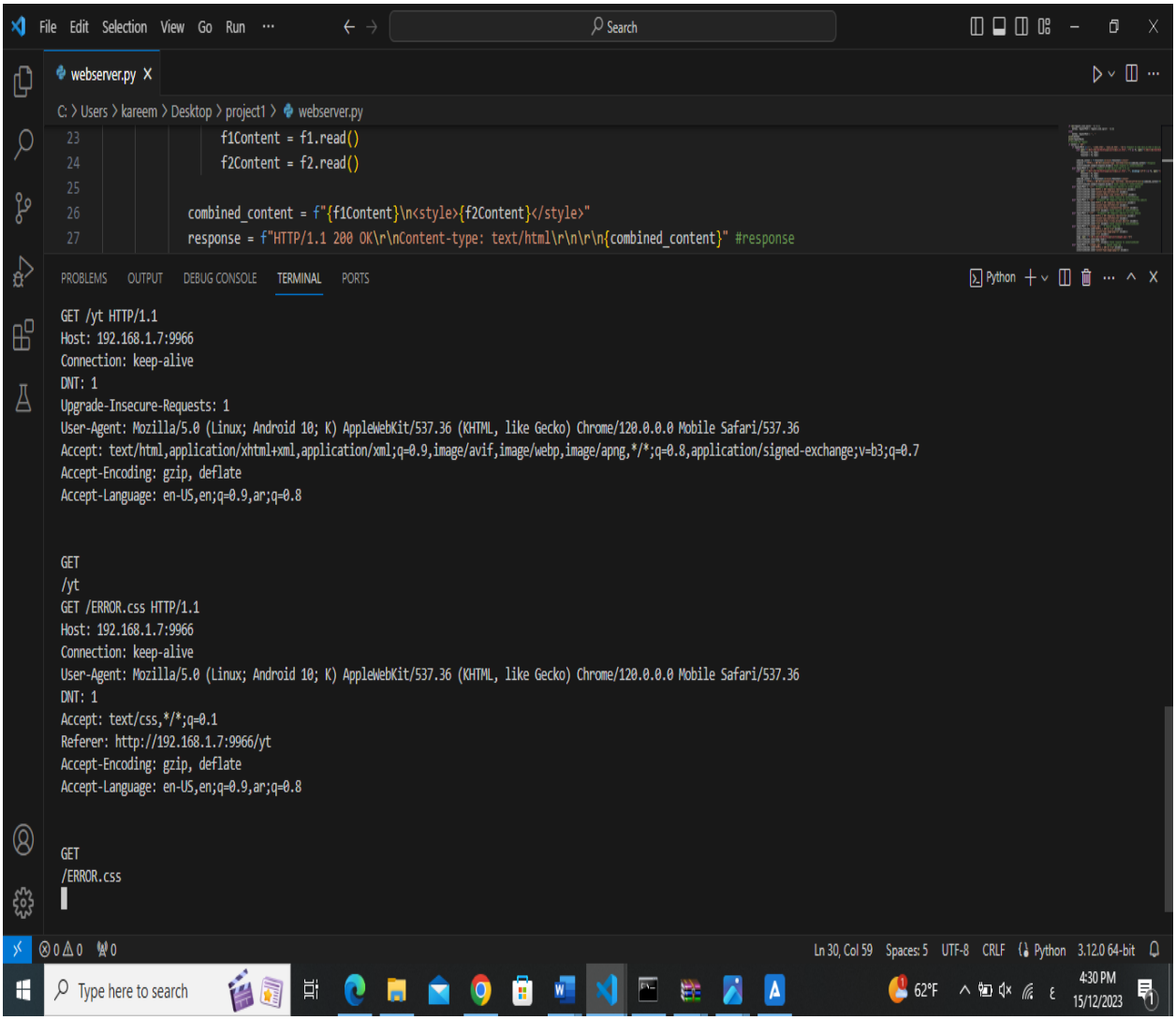
- If the request: / or /index.html or /main_en.html or /en

- If the request: /anything else (error)



Part 3.12: terminal [screenshots + explanation]





Explanation:

For running this program from a different device, we need to search for http://(local_IP_address):(port_number)/(path) from the other device. The local_IP_address of my device is 192.168.1.7, and the port number is 9966 (which is the port that our webserver is listening on), then we put the path, we didn't use all the paths again, but as a test we used path /en, and /yt (which resulted in an error webpage)

We notice from the terminal, when we request for the path /en, our request is followed by another internal requests, these requests happen due to our html file design, our file has a css file for the styles, a png file and a jpg file, all these files should be opened by the server.