
**Faculty of Engineering and Technology**

**Electrical and Computer Engineering Department**

**Computer Networks**
**ENCS3320**

**Course Project**

---

# WEB SERVER IMPLAMENTATION

---

**Student's Name:** Lojain Abdalrazaq **ID:** 1190707, **Sec**: 4

**Student's Name:** Mariam Taweel **ID:** 1192099, **Sec**: 3

**Student's Name:** Aseel Deek **ID:** 1190587, **Sec**: 4

**Instructors:**

**Dr.**Abdalkarim Awad **Dr.**Imad Tartir

7th Dec 2021

## Abstract

In this project, three parts will be included, the first part represents several commands to run (including Ping, Tracert and Nslookup). The second part describes how to use the HEAD of the HTTP response to get the HTTP response from a specific webserver using Python. While the last part illustrates the implementation of a web server using socket programming, using Python. The program checks up to eight different requests and sends the results required.

# ➢ Table Of Contents

# 1. Theory

## 1.1. What are Root Servers?

Root servers are the authoritative name servers that serve the DNS root zone **[1],** commonly known as the **"root servers"** that are responsible for the functionality of the DNS as well as the entire Internet, they translate the name domain into IP addresses. The operation of mapping the domain names into IP addresses works in hierarchical order using DNS zones and the root servers serve the root zone **[2]**. In addition, the root servers are a network of hundreds of servers in many countries around the world and they are configured in the DNS root zone as 12 organizations held responsible, with VeriSign two of them (in total 13 named authoritative) as shown in the following

| Hostname | IP address IPv4 / IPv6 | Organization |
|---|---|---|
| a.root-servers.net | 198.41.0.4, 2001:503:ba3e::2:30 | VeriSign, Inc. |
| b.root-servers.net | 199.9.14.201, 2001:500:200::b | University of Southern California (ISI) |
| c.root-servers.net | 192.33.4.12, 2001:500:2::c | Cogent Communications |
| d.root-servers.net | 199.7.91.13, 2001:500:2d::d | University of Maryland |
| e.root-servers.net | 192.203.230.10, 2001:500:a8::e | NASA |
| f.root-servers.net | 192.5.5.241, 2001:500:2f::f | Internet Systems Consortium, Inc. |
| g.root-servers.net | 192.112.36.4, 2001:500:12::d0d | US Department of Defense (NIC) |
| h.root-servers.net | 198.97.190.53, 2001:500:1::53 | US Army (Research Lab) |
| i.root-servers.net | 192.36.148.17, 2001:7fe::53 | Netnod |
| j.root-servers.net | 192.58.128.30, 2001:503:c27::2:30 | VeriSign, Inc. |
| k.root-servers.net | 193.0.14.129, 2001:7fd::1 | RIPE NCC |
| l.root-servers.net | 199.7.83.42, 2001:500:9f::42 | ICANN |
| m.root-servers.net | 202.12.27.33, 2001:dc3::35 | WIDE Project |

**Figure (1.1): List of Root Servers**

# 2. Procedure

## 2.1. Part1: Running Commands:

### ➢ 2.1.1: Ping a Device in The Same Network:

The goal of pinging a device is to find out if a device is reachable at a particular IP address. Ping is a computer network administration utility **used to test the reachability of a host on an Internet Protocol (IP) network and to measure the round-trip time for messages sent from the originating** host to a destination computer **[3].** In this part, the laptop was connected to the internet, and a smartphone connected to the same network was pinged, using the IP address of that phone.

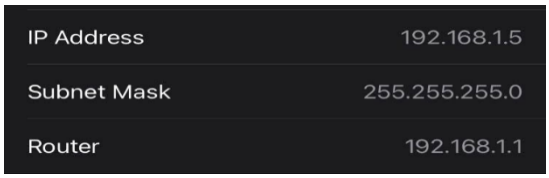



**Figure (2.1.1): IP address of the smartphone**

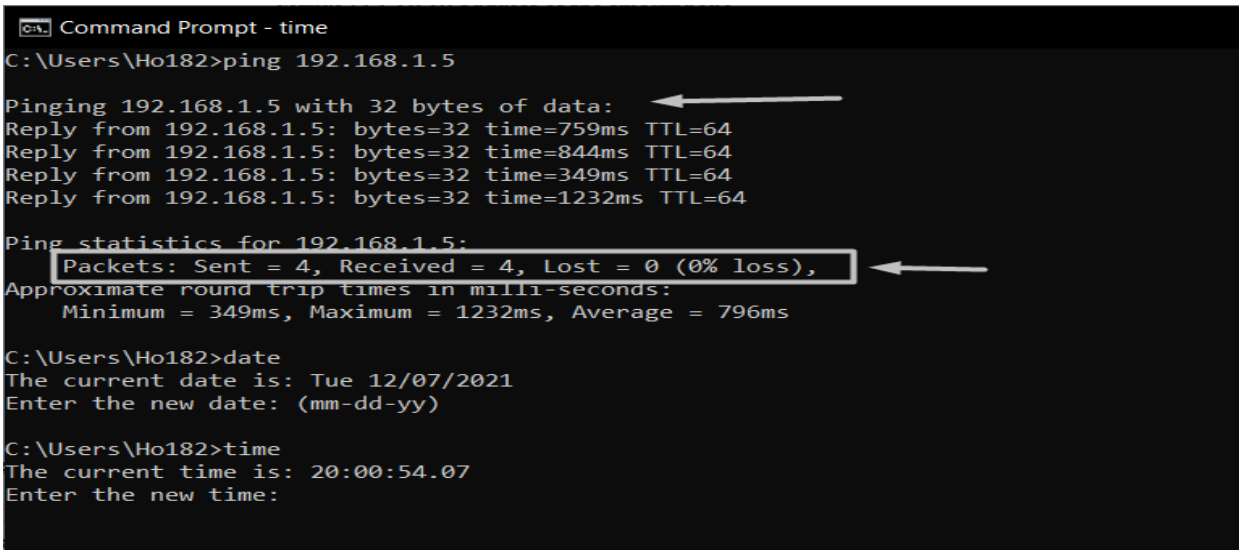



**Figure (2.1.2): the result of pinging the smartphone**

By pinging the mobile IP, we observe a set of observations, including: first, the time field that shows the round-trip time of the packet (reaching the host and returning response to the sender in milliseconds), secondly, the TTL value indicates the maximum number of routers the packet can go through before it's thrown away. The computer and the phone are connected on the same local network, so the TTL value is 64 for Windows. Finally, Statistics show the number of sent and received packets (in our case 4 packets were sent and 4 received, the packet loss is zero), the minimum, maximum and average response time.

A device on a computer network should respond to an echo request (ping). This reply should come within milliseconds. If a reply does not come it is said to have "timed out" because the predefined wait time for a reply has been exceeded. If no reply is seen we typically assume that no device is present at, or assigned with, that IP address **[3]** (figure 2.1.3).

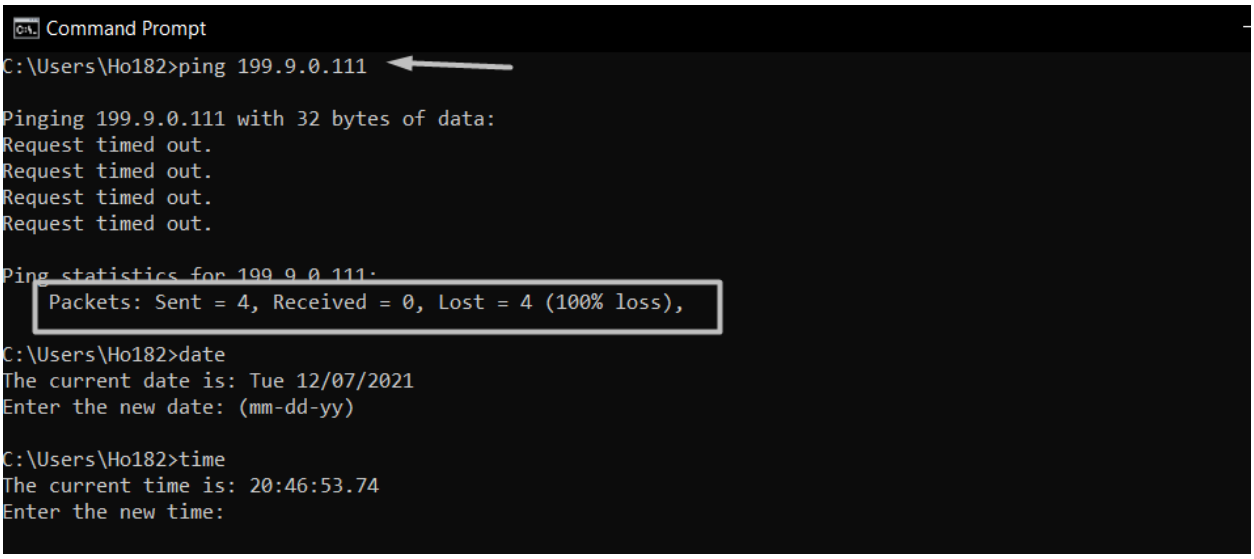



**Figure (2.1.3): the result of pinging wrong IP address**

### ➢ 2.1.2: Ping b.root-servers.net:

The B-Root Domain-Name-Server is operated the University of Southern California. B Root provides root DNS service at **199.9.14.201** (IPv4) and **2001:500:200::b** (IPv6), operating asb.root-servers.net**[4].**

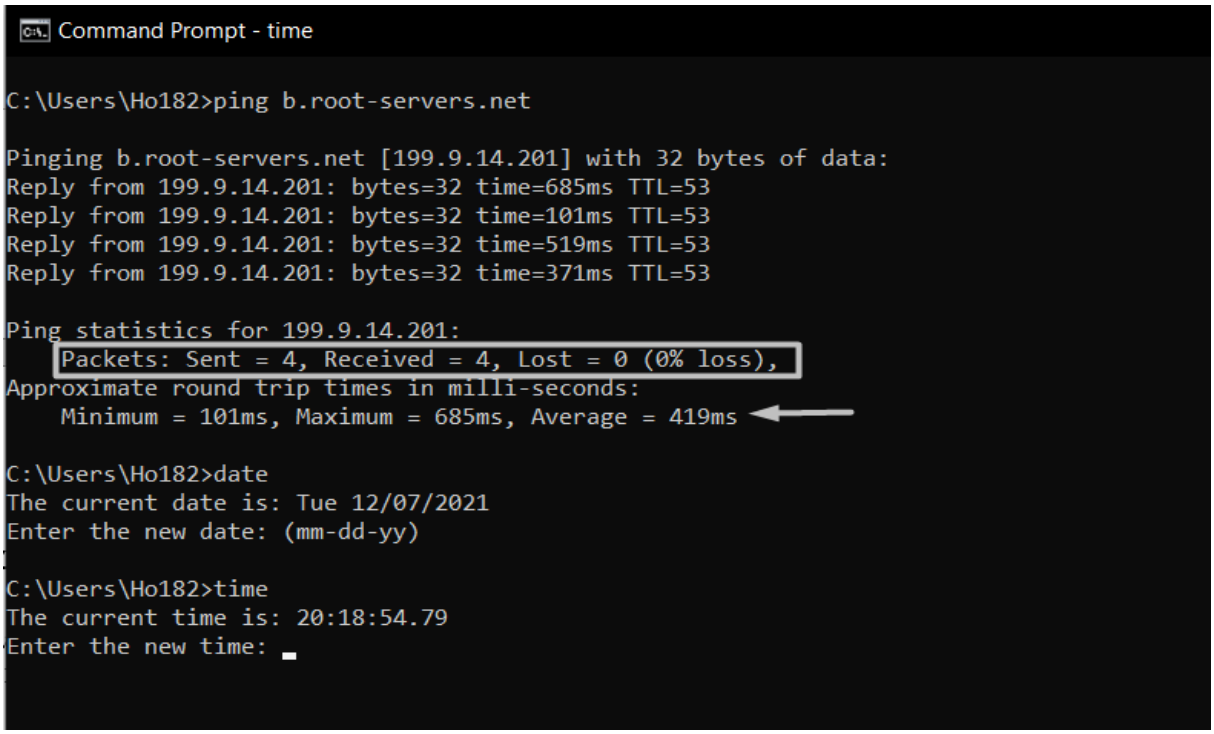



**Figure (2.1.4): the result Results of pinging b.root-servers.net**

The results obtained show the IP address related to that b.root-servers.net ,which is equal to199.9.14.201. It also shows the size of the package that was sent, which is 32. TTL is a remaining number of hops that a packet may still remain viable before being discarded if it does not yet reach its destination**[5].** TTL=53 That means that the ping request went through 11 routers to get to the destination address. The statistic at the end shows the number of packets sent and received (in our case, 4 packages were sent and 4 were received, and the loss in the package equals zero), and the minimum, maximum and average response time.

### ➢ 2.1.3: Tracert Command:

The traceroute command in Windows is tracert, a Command prompt command that is used to show **several details about the path** that a packet takes from the computer or the device to any specified destination **[6]**. This command specifies the path that the packets take from source to the destination, including the time for each stop station along the way which indicates the response times, and help to locate any points of failure encountered while end route to a certain destination. The following figure (2.1.5) shows the path that the packet takes from the computer to **b.root-servers.net.**

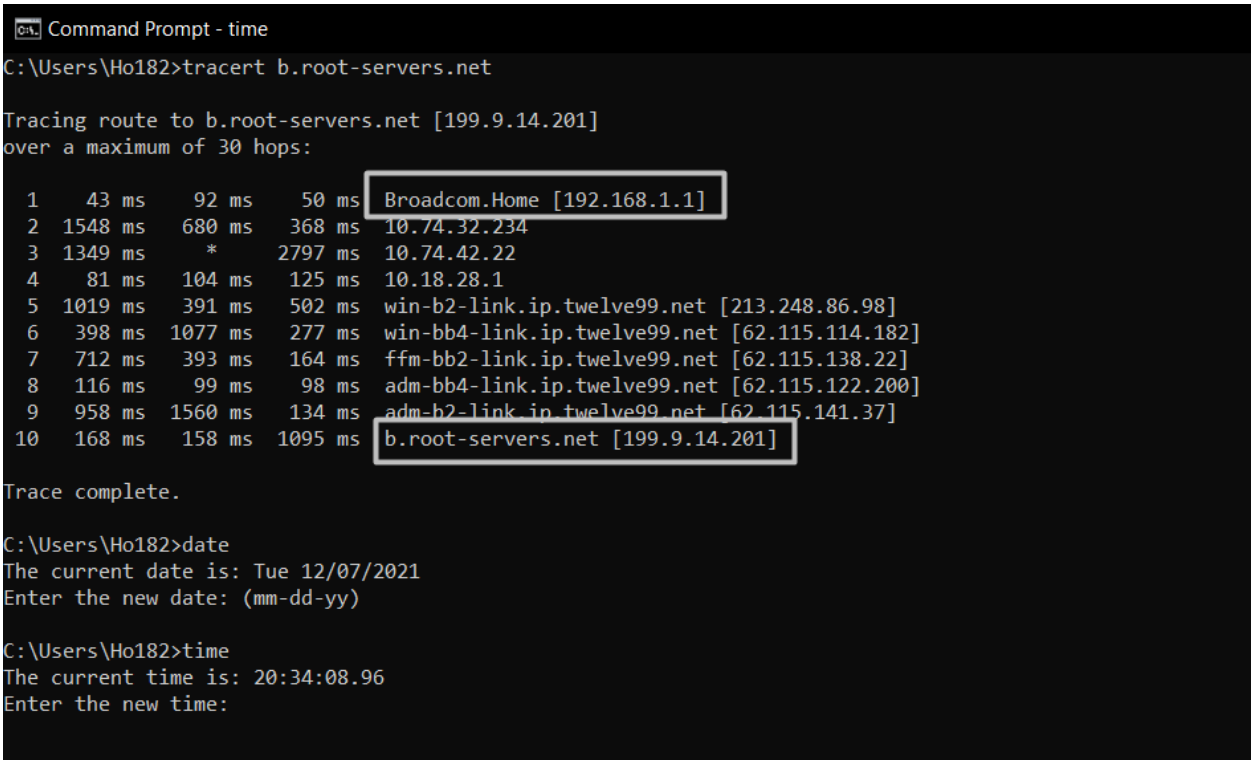
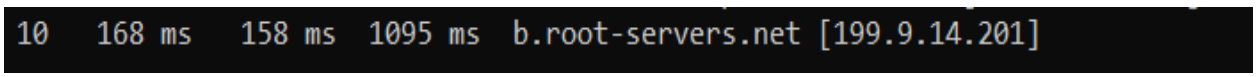



**Figure (2.1.5): the result Results of pinging b.root-servers.net**

Each row obtained in result above from (1 to 10) represents a "hop" across the route and you can see the comment "**over a maximum of 30 hops"** which means that the diameter of the internet is roughly 30. Therefore, many traceroute will only go that far out in trying to reach the destination. In addition, the first column is the hop number, and the next three columns contains the round-trip times (RTT) in milliseconds which shows the time needed for each packet to reach the destination point and return to the source computer. The last column is the host name and IP address of the responding system. Finally, we notice that the last row contains the destination host name **b.root-servers.net** and the IP address **[199.9.14.201].**



### ➢ 2.1.4: Nslookup Command:

The Nslookup command is very important network administration command-line tool. It can be used to perform DNS queries and receive the IP address or the domain names **[7].** Nslookup, askes the DNS server to give the IP address for the given hostname. This command has two modes, interactive and non-interactive mode. The non-interactive mode is used when searching for a single piece of data, the opposite is the interactive mood since it is used when searching for more than one piece of data **[8].** Figure (2.1.6) shows the IP address for the **b.root-servers.net.**

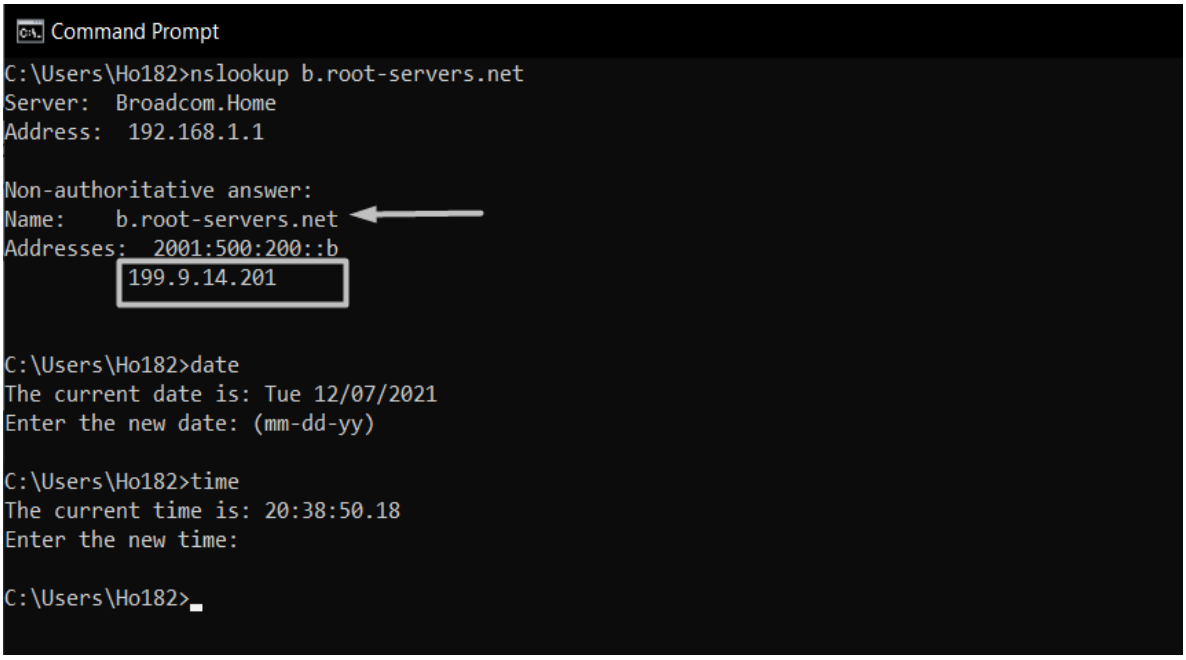



**Figure (2.1.6): the result Results Nslookup command**

## 2.2. Part2: HTTP Response and Elapsed Time:

In this part, it is required to design a Socket program that receives the URL and then finds the HTTP response time. Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. **[9]** Now let's take a look at the timeline of a usual HTTP Request [**10]:**


**Figure (2.2.1): timeline of a usual HTTP Request.**

**DNS Lookup:** Time spent performing the DNS lookup. DNS lookup resolves domain names to IP addresses. Every new domain requires a full round trip to do the DNS lookup. There is no DNS lookup when the destination is already an IP address.

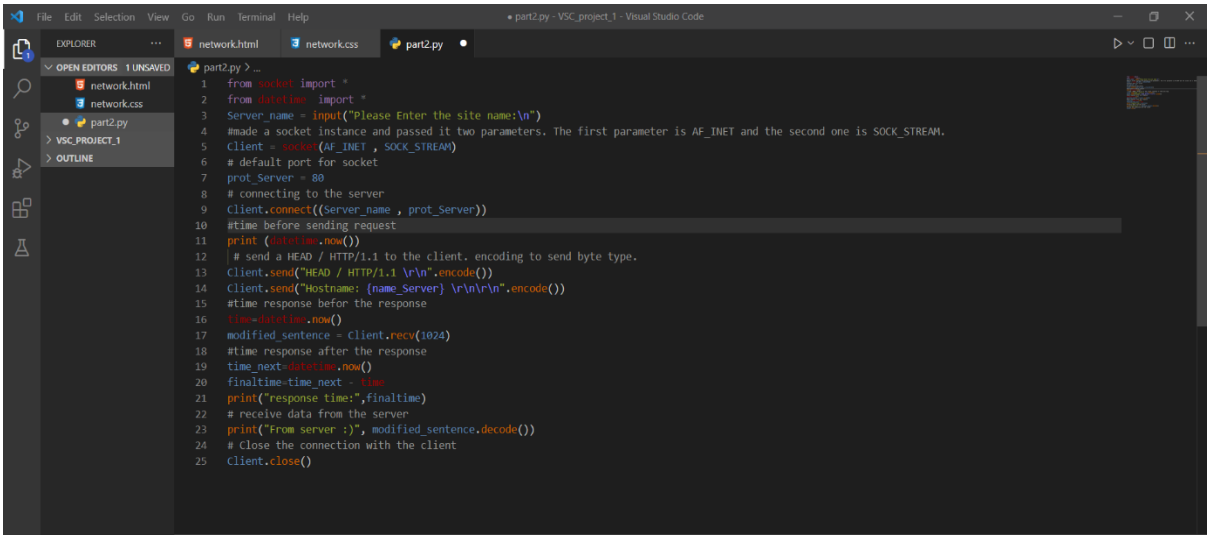
**TCP Connection:** Time it took to establish TCP connection between a source host and destination host.

**TLS handshake:** Time spent completing a TLS handshake. During the handshake process endpoints exchange authentication and keys to establish or resume secure sessions. There is no TLS handshake with a not HTTPS request.

**Time to First Byte (TTFB):** Time spent waiting for the initial response. This time captures the latency of a round trip to the server in addition to the time spent waiting for the server to process the request and deliver the response.

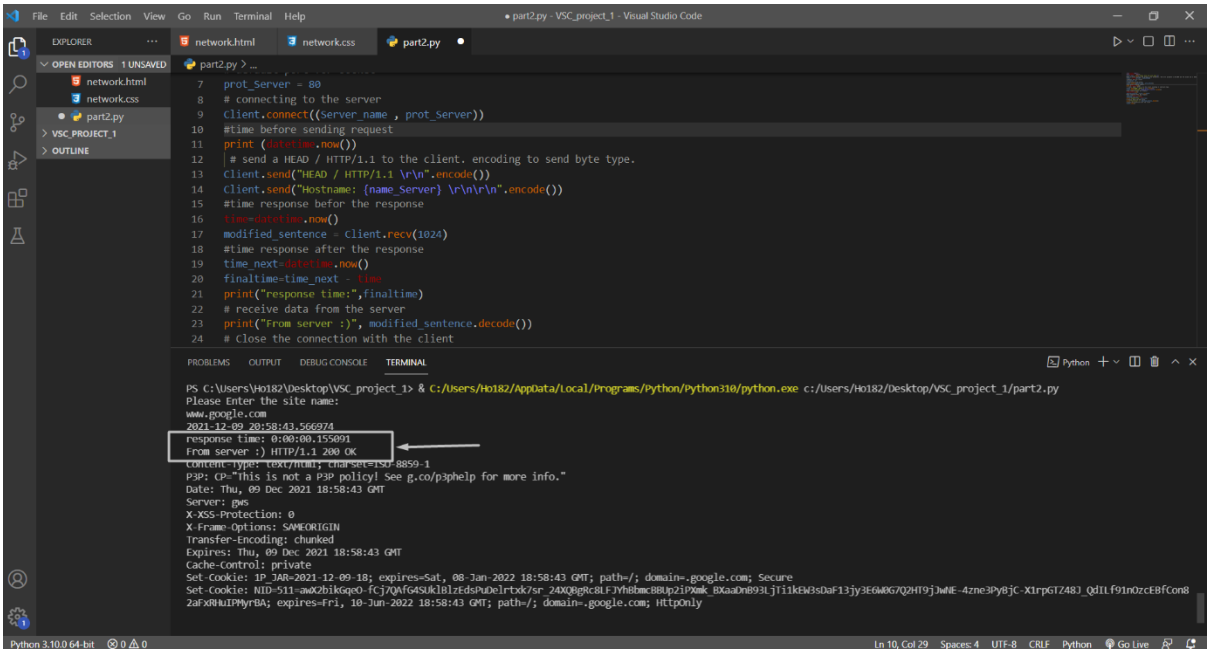
**Content Transfer:** Time spent receiving the response data. The size of the response data and the available network bandwidth determinates its duration.

In this part, the socket programming in Python language was applied to design a program that receives the URL from the user and then finding the HTTP response time from sending the HTTP request until it received the HTTP response. Through this socket programming, we were able to find the response time by finding the difference between the time before the response and time after the response.



```python
from socket import *
from datetime  import *
Server_name = input("Please Enter the site name:\n")
#made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM.
Client = socket(AF_INET , SOCK_STREAM)
# default port for socket
prot_Server = 80
# connecting to the server
Client.connect((Server_name , prot_Server))
#time before sending request
print (datetime.now())
 # send a HEAD / HTTP/1.1 to the client. encoding to send byte type.
Client.send("HEAD / HTTP/1.1 \r\n".encode())
Client.send("Hostname: {name_Server} \r\n\r\n".encode())
#time response befor the response
time=datetime.now()
modified_sentence = Client.recv(1024)
#time response after the response
time_next=datetime.now()
finaltime=time_next - time
print("response time:",finaltime)
# receive data from the server
print("From server :)", modified_sentence.decode())
# Close the connection with the client
Client.close()
```

**Figure (2.2.2): the required code for the requesting**



```
PS C:\Users\Ho182\Desktop\VSC_project_1> & C:/Users/Ho182/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Ho182/Desktop/VSC_project_1/part2.py
Please Enter the site name:
www.google.com
2021-12-09 20:58:43.566974
response time: 0:00:00.155091
From server :) HTTP/1.1 200 OK
Content-Type: text/html; charset=ISO-8859-1
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Date: Thu, 09 Dec 2021 18:58:43 GMT
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Transfer-Encoding: chunked
Expires: Thu, 09 Dec 2021 18:58:43 GMT
Cache-Control: private
Set-Cookie: 1P_JAR=2021-12-09-18; expires=Sat, 08-Jan-2022 18:58:43 GMT; path=/; domain=.google.com; Secure
Set-Cookie: NID=511=aw0Zbik6qeO-fCj7QAfG4SUklBlzEdsPuDelrtxk7sr_24XQBgRc8LFJYh8bmcB8Up2iPXmk_BXaaDnB93LjTi1kEW3sDaF13jy3E6W0G7Q2HT9jJwNE-4zne3Py8jC-X1rpGTZ48J_QdILf91nOzcE8fCon8
2aFxRHuIPMyr8A; expires=Fri, 10-Jun-2022 18:58:43 GMT; path=/; domain=.google.com; HttpOnly
```

**Figure (2.2.3): the response time for requesting www.googel.com**

## 2.3. Part3: Implanting Web Server:

In this part of the project, the socket programming in Python language was applied to carry out a simple web server that server eight different requests specified and shows the HTTP requests on the terminal window.

- **2.3.1: Request1: localhost:6500/ localhost: 6500/index.html**

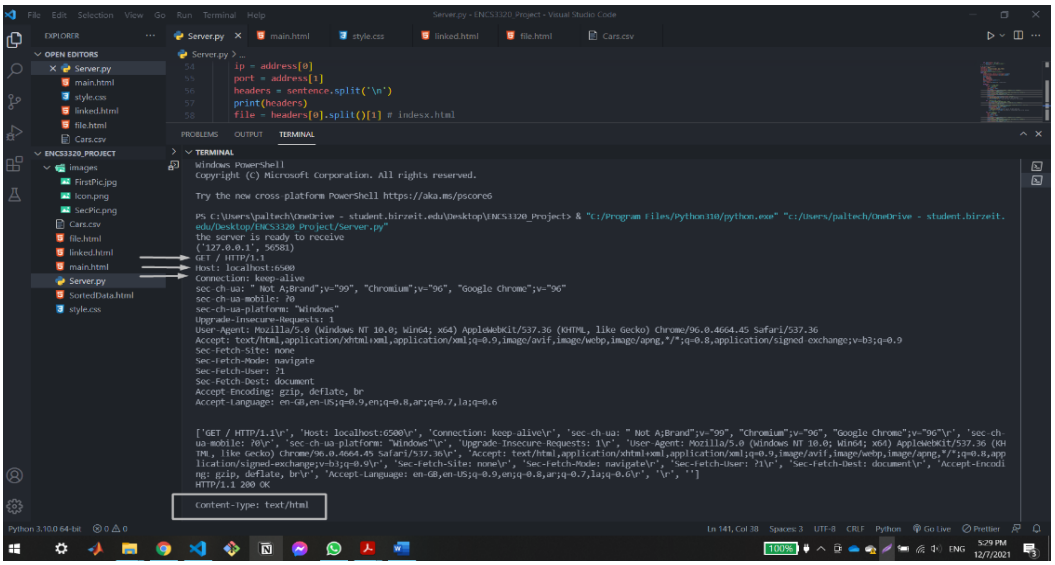



**Figure (2.3.1): HTTP request for the html file**

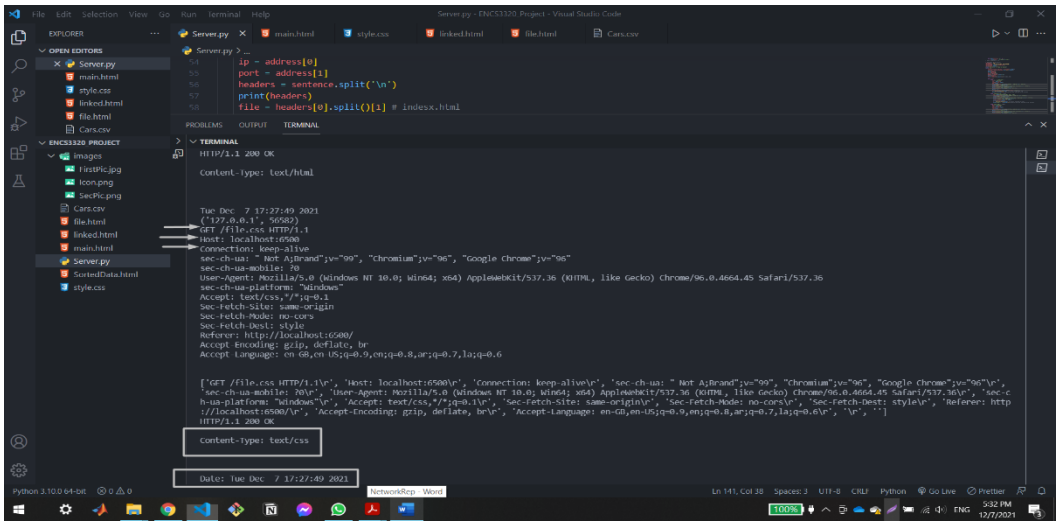
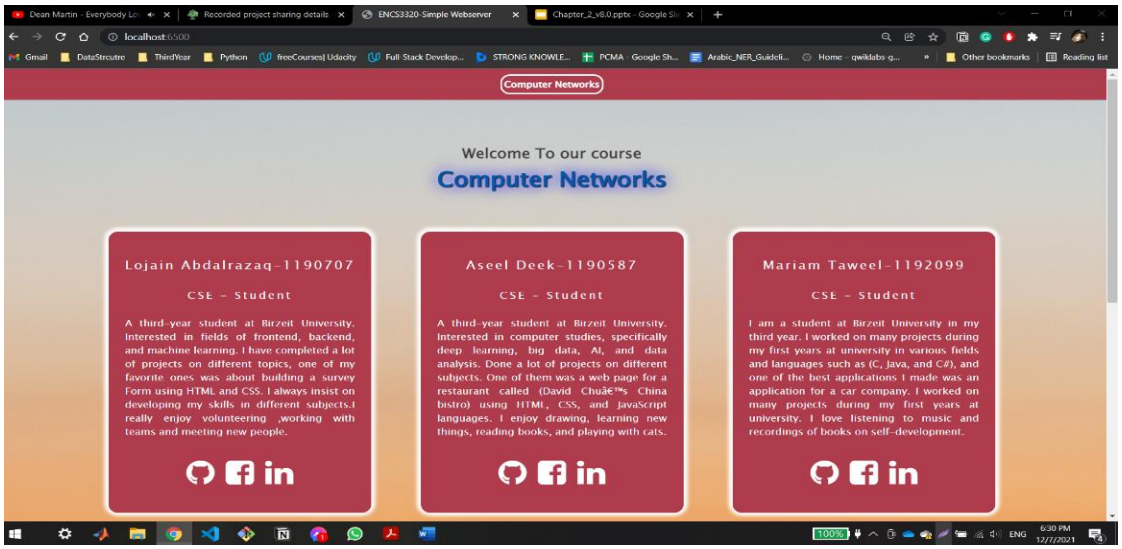



**Figure (2.3.2): HTTP request for the CSS file**

Figures (2.3.1) and (2.3.2), demonstrate the HTTP request messages when representing **localhost:6500/ or localhost: 6500/index.html** requests in the terminal. The first message is for the HTML file. The message contents tell that the data was sent to the server using the GET

method, the HTTP type is HTTP/1.1, the content type is text/HTML and the connection type for the request is persistent since that was told from the keep-alive statement as shown in figure (2.3.1). The other figure shows the request message for the second request. This request is for the style.css file, which was created as a separate file and linked to the HTML file for styling the page, the request message is the same, but the difference is the content type is text/CSS.

- The figure below displays the styled Html page required with the request **localhost:6500/,** which contains (course name/ the group members names and IDs and some simple information about them), as well as two images, one with a .**jpg** extension, another one with **.png** extension, a link to a local html file and a link for https://www.w3schools.com/tags/att_img_src.asp .



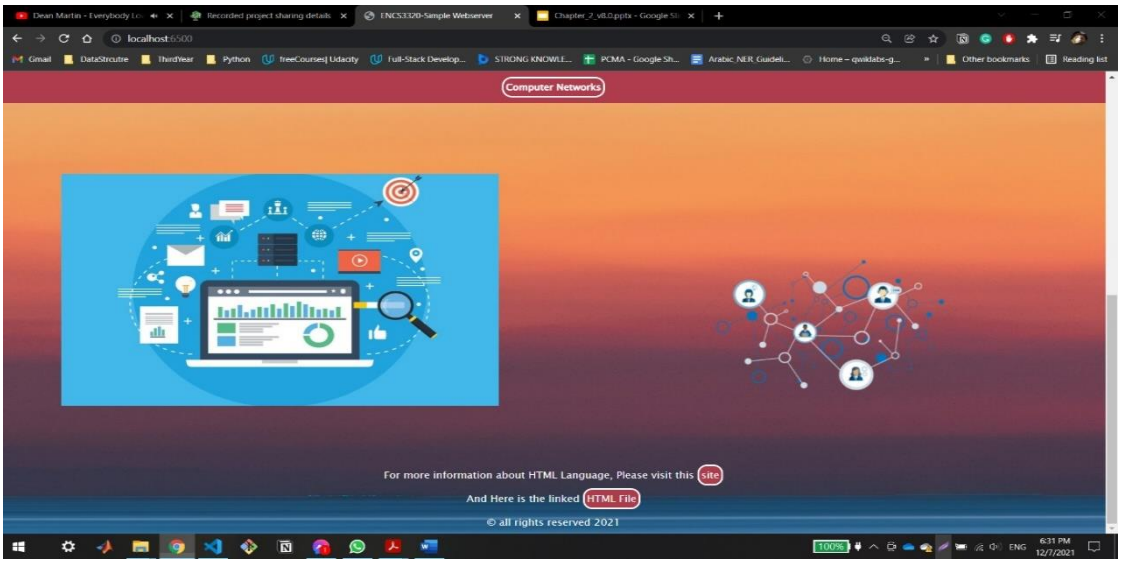




**Figure (2.3.3): the styled HTML file**

Figure (2.3.4) represents the **localhost:6500/index.html**, in this request the method GET is used to send the data to the sever too, and likewise, there are two requests represented, one for the /index.html, and another one for the related styles.css styling file.

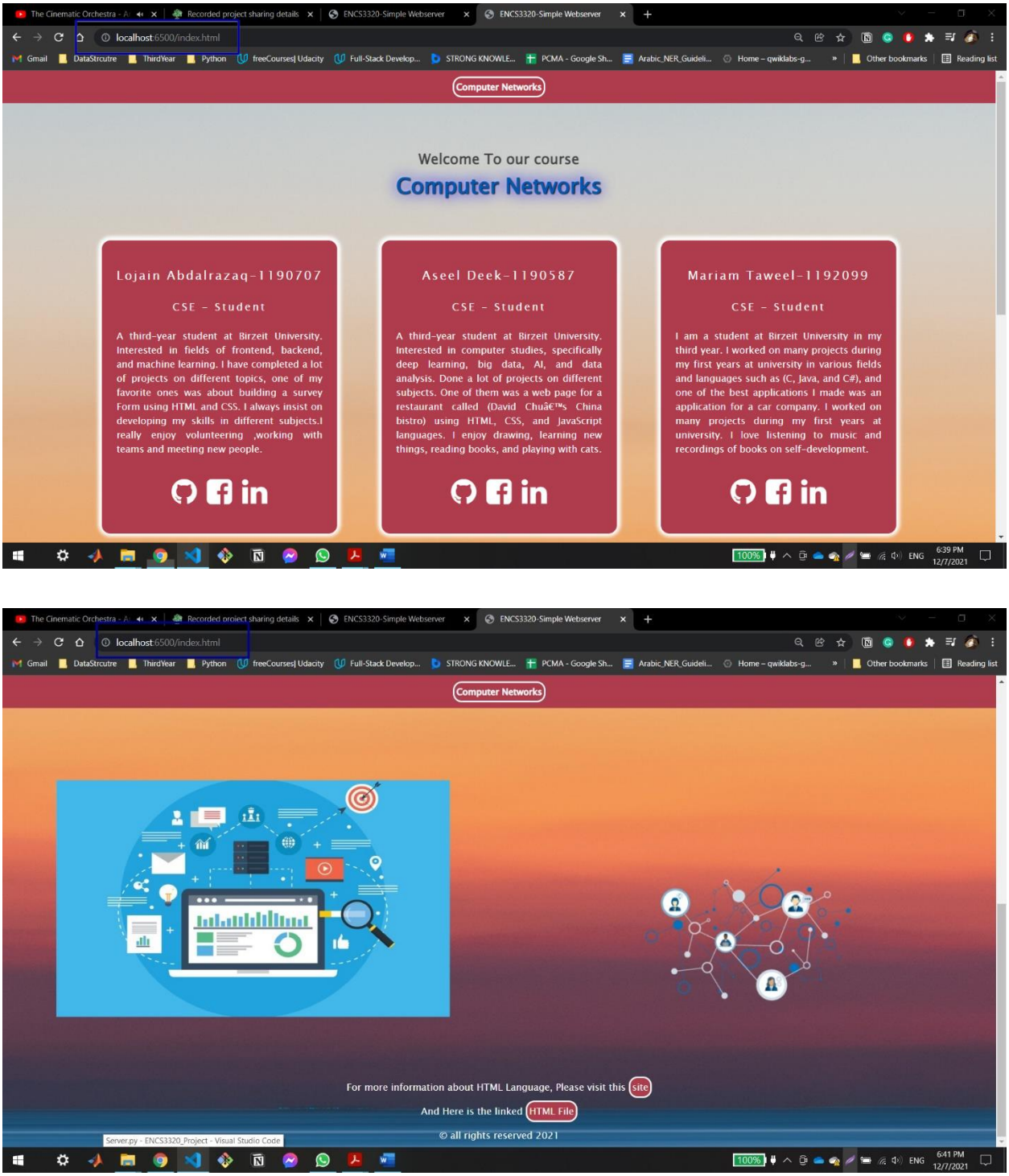



**Figure (2.3.4): the styled HTML file when /index.html**


**Figure (2.3.5): the required HTML page from a smartphone**

Figure (2.3.5) shown above, illustrates the taken result using the corresponding request **/index.html**, in another device which is a smart phone. Making a request from another device like smart phones demands the IP address for the network of the used computer. To do a request the IP address instead of **(localhost:6500/)** should be followed by the port number . In the above figure the request is **192.168.1.200: 6500/index.html**, since 192.168.1.200 is the IP address.

### ➢ 2.3.2: Request2: localhost:6500/file.css

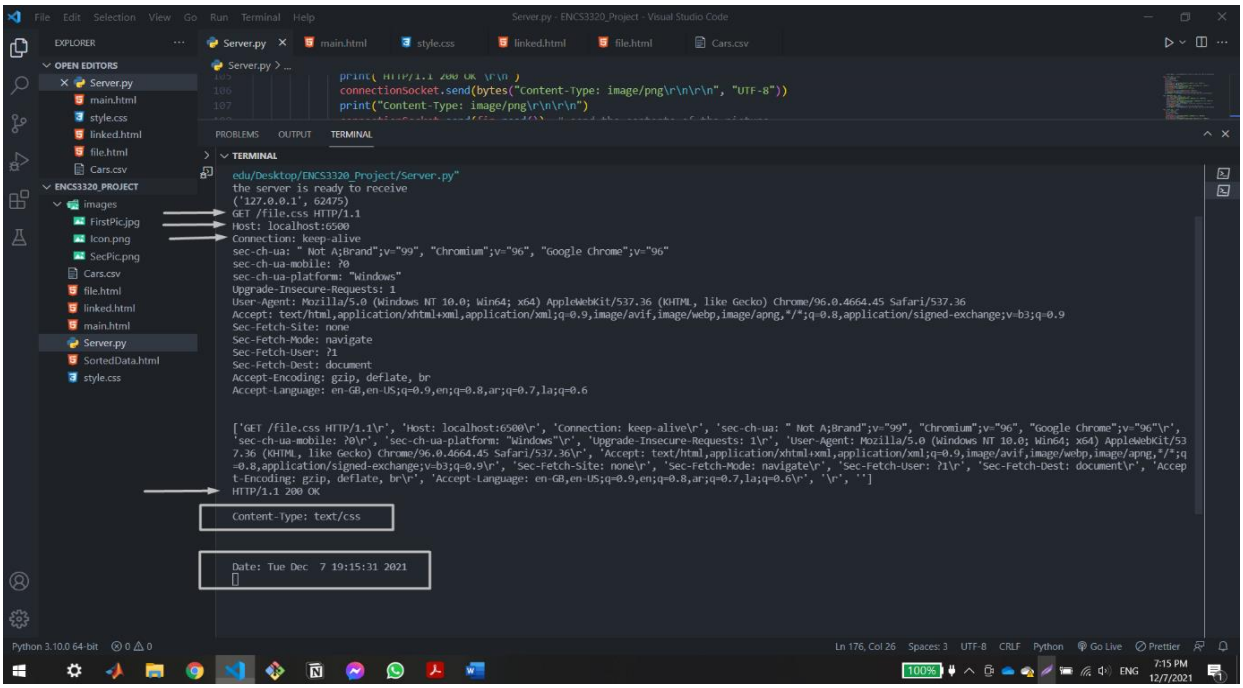



**Figure (2.3.6): the request message for localhost:6500/file request.**

Figures (2.3.6) demonstrate the HTTP request messages when representing **localhost:6500/file.css** request in the terminal. The message contents tell that the data was sent to the server using the GET method, the HTTP type is HTTP/1.1, the content type is text/css and the connection type for the request is persistent since that was told from the keep-alive statement.

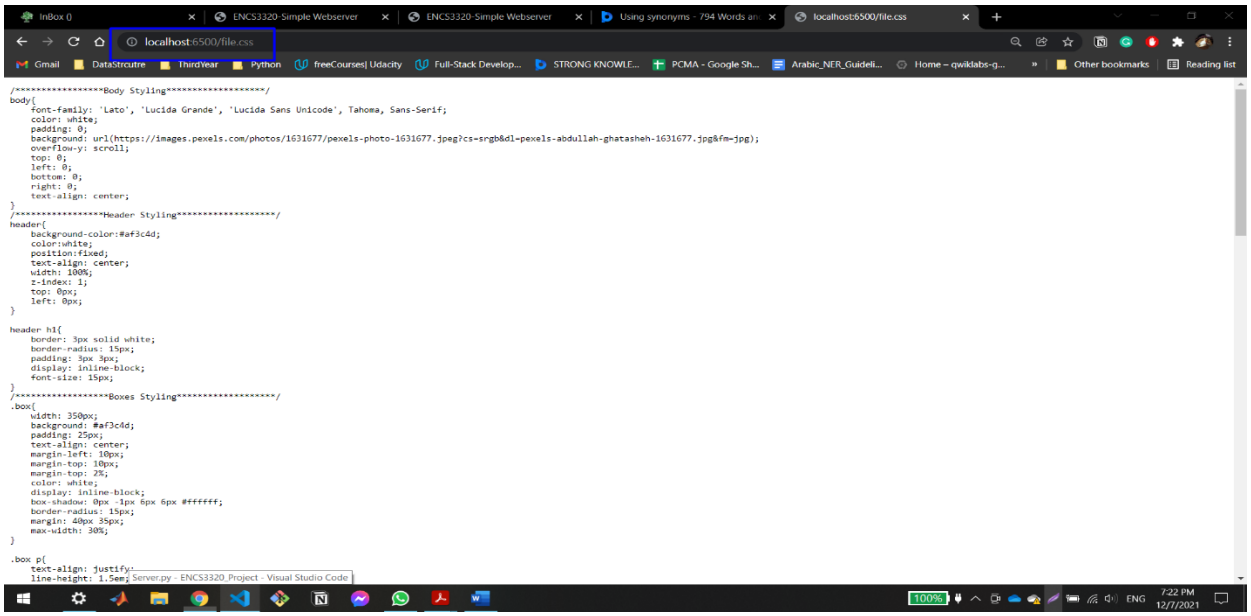



**Figure (2.3.7): the result of requesting localhost:6500/file**

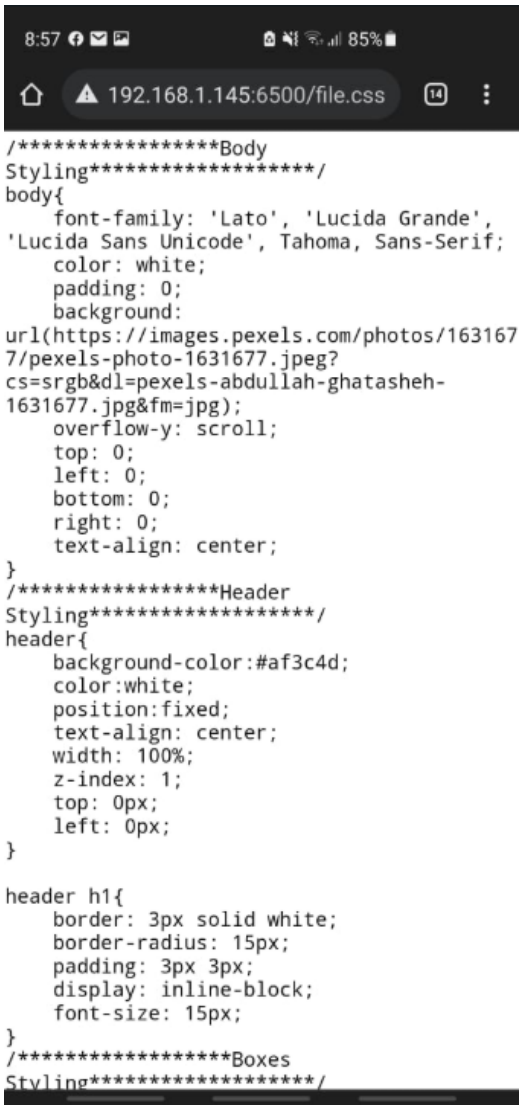



**Figure (2.3.8): the required file.css from a smartphone**

Figure (2.3.8) shown above, illustrates the taken result using the corresponding request **/file.css**, using another device which is a smart phone. As mentioned before to do a request using the other device the IP address of the used computer should be known, here the IP address is 192.168.1.145 and the request is **192.168.1.145:6500/file.css.**

### ➢ 2.3.3: Request3: localhost:6500/Network1.png

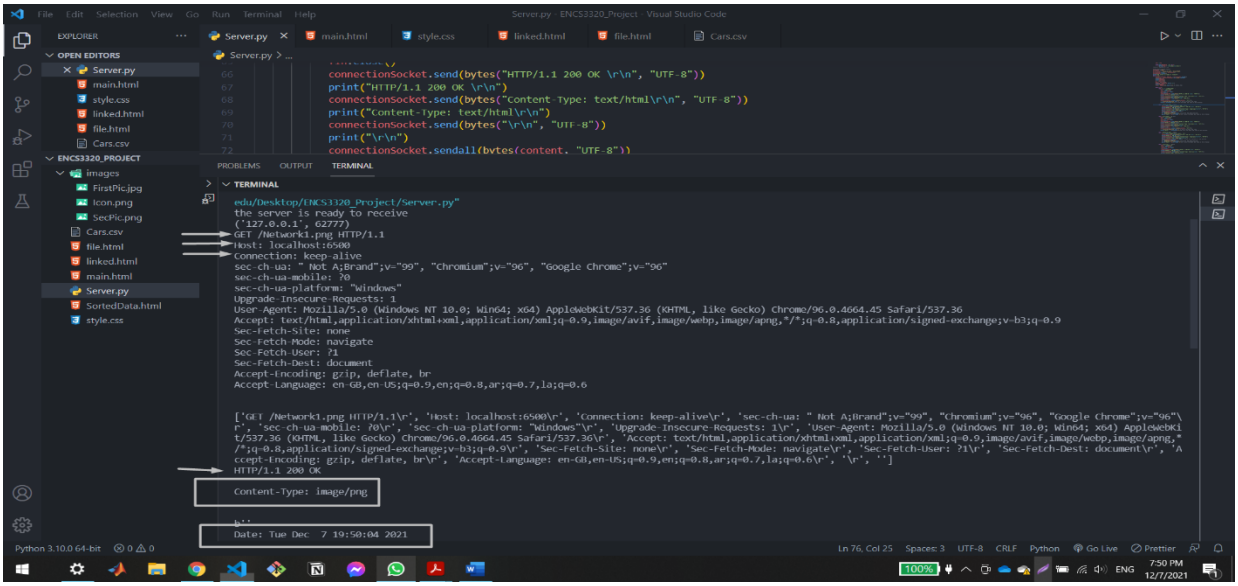



Figure (2.3.9): the request message for localhost:6500/Network1.png

Figures (2.3.9) demonstrate the HTTP request messages when representing **localhost:6500/Network1.png** request in the terminal. The message contents tell that the data was sent to the server using the GET method, the HTTP type is HTTP/1.1, the content type is image/png, the connection type for the request is persistent since that was told from the keep-alive statement, and it shows the status of the HTTP response ***HTTP/1.1 200 OK*** means that the request succeeded.

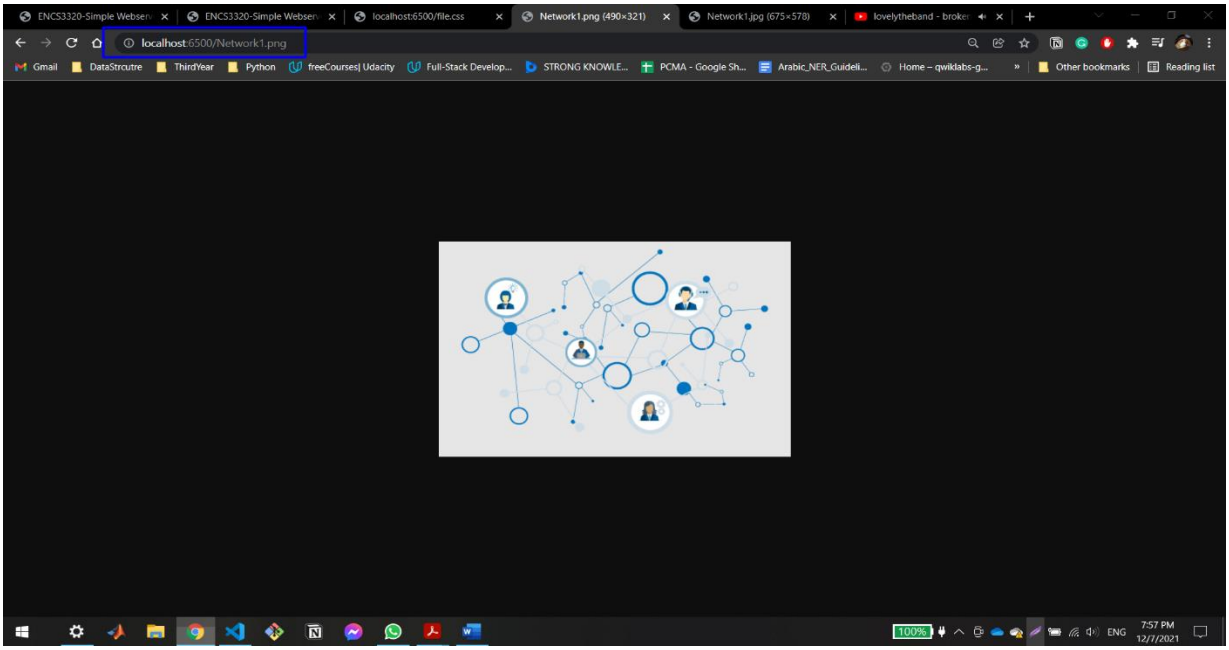



Figure (2.3.10): the result of localhost:6500/Network1.png request


**Figure (2.3.11): the required Network1.png from a smartphone**

Figure (2.3.11) shown above, illustrates the taken result using the corresponding request **/Network1.png**, using another device which is a smart phone. As mentioned before to do a request using the other device the IP address of the used computer should be known, here the IP address is 192.168.1.145 and the request is **192.168.1.145:6500/ Network1.png.** Using this method gives the same result achieved on the used computer.

### ➢ 2.3.4: Request3: localhost:6500/Network1.jpg

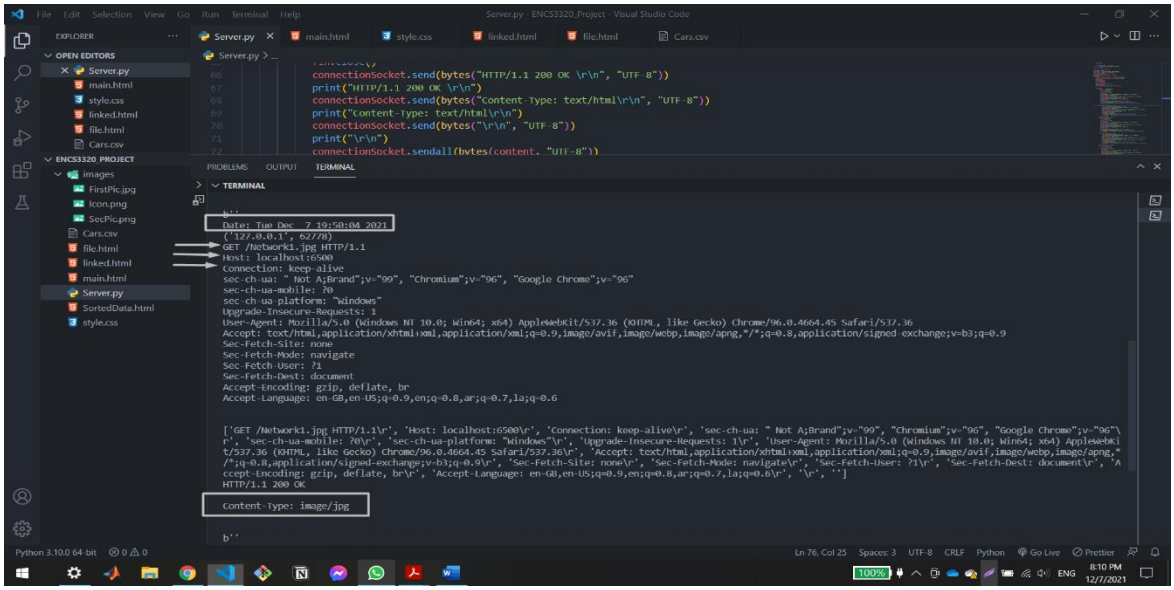



**Figure (2.3.12): the request message for localhost:6500/Network1.jpg**

Figures (2.3.12) demonstrate the HTTP request messages when representing **localhost:6500/Network1.jpg** request in the terminal. The message contents tell that the data was sent to the server using the GET method, the HTTP type is HTTP/1.1, the content type is image/png, the connection type for the request is persistent since that was told from the keep-alive statement and it shows the status of the HTTP response ***HTTP/1.1 200 OK*** means that the request succeeded.

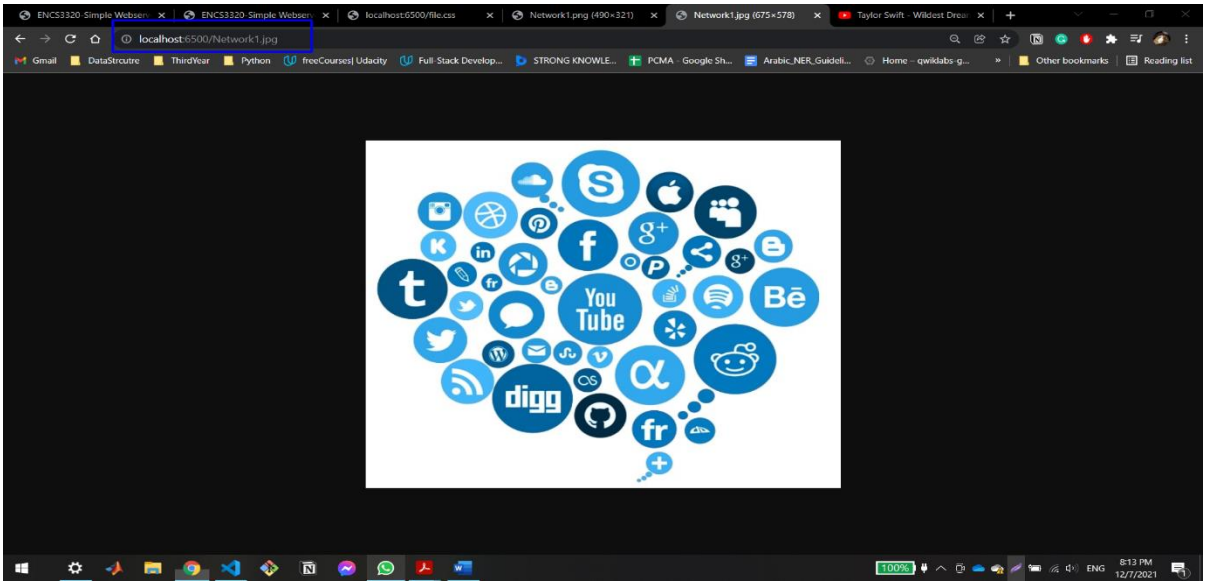



**Figure (2.3.13): the result of localhost:6500/Network1.jpg request**


**Figure (2.3.14): the required Network1.jpg from a smartphone**

Figure (2.3.14) shown above, illustrates the taken result using the corresponding request **/Network1.jpg**, using another device which is a smart phone. As mentioned before to do a request using the other device the IP address of the used computer should be known, here the IP address is 192.168.1.145 and the request is **192.168.1.145:6500/ Network1.jpg.** Using this method gives the same result achieved on the used computer.

### ➢ 2.3.5: Request4: localhost:6500/other.html

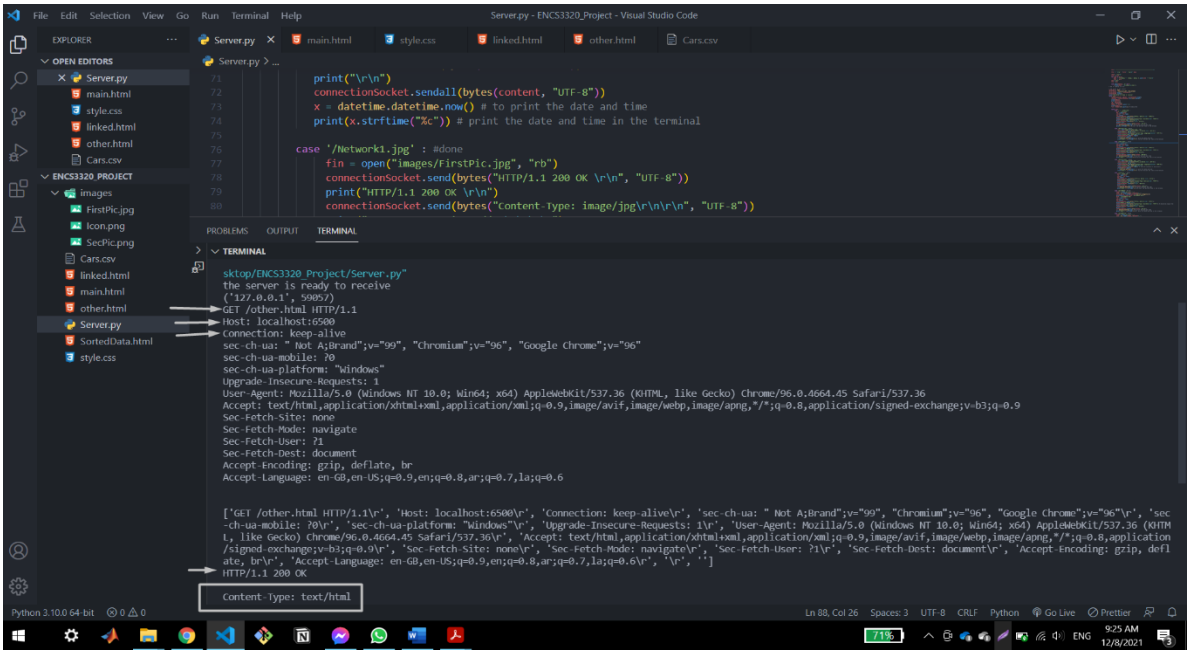



**Figure (2.3.15): the request message for localhost:6500/file.html**

Figures (2.3.15) demonstrate the HTTP request messages when representing **localhost:6500/other.html** request in the terminal. The message contents tell that the data was sent to the server using the GET method, the HTTP type is HTTP/1.1, the content type is text/html, the connection type for the request is persistent since that was told from the keep-alive statement, and it shows the status of the HTTP response ***HTTP/1.1 200 OK*** means that the request succeeded.

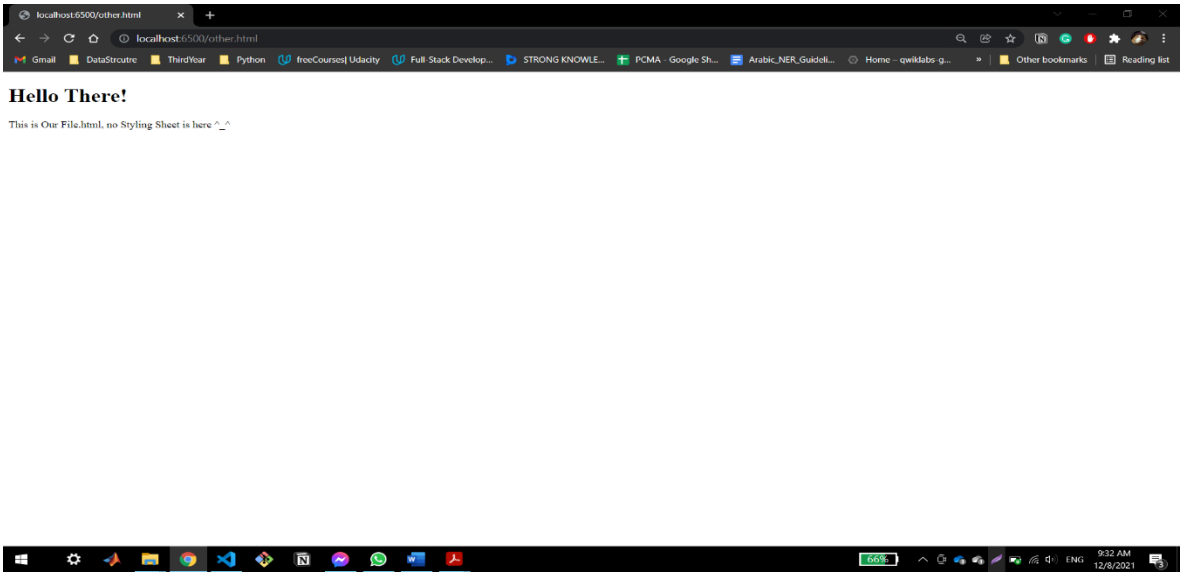



**Figure (2.3.16): the result of localhost:6500/other.html**


**Figure (2.3.17): the required other.html from a smartphone**

Figure (2.3.17) shown above, illustrates the taken result using the corresponding request **/other.html**, using another device which is a smart phone. As mentioned before to do a request using the other device the IP address of the used computer should be known, here the IP address is 172.19.31.152 and the request is **172.19.31.152:6500/ other.html.** Using this method gives the same result achieved on the used computer.

### ➢ 2.3.6: Request3: localhost:6500/ File does not find

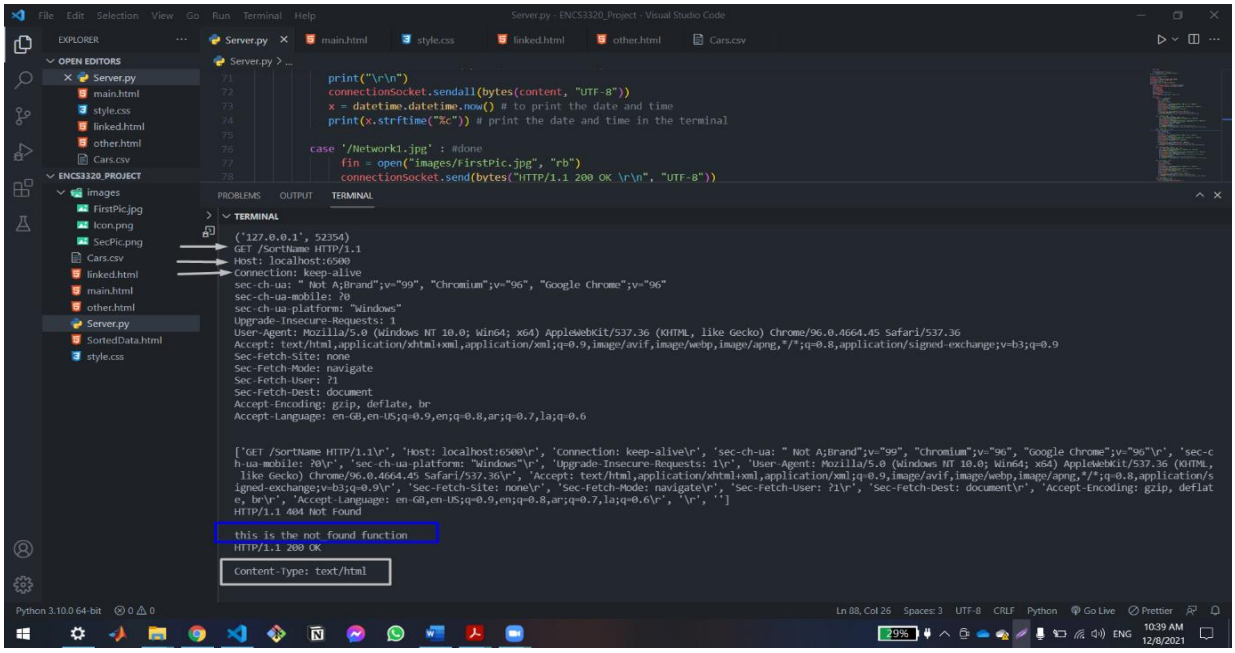



**Figure (2.3.18): ): the request message for localhost:6500/SortName**

This part is when the request is wrong, in other words when the request is unknown to the server. In this case a wrong request was send ( **localhost:6500/SortName**), the server proceeds an Html page with a content type: text/html. The Html page has ***"Error"*** as a title and contains the sentence (The file is not found in red), besides the names and IDs of the group members, the IP address and port number.

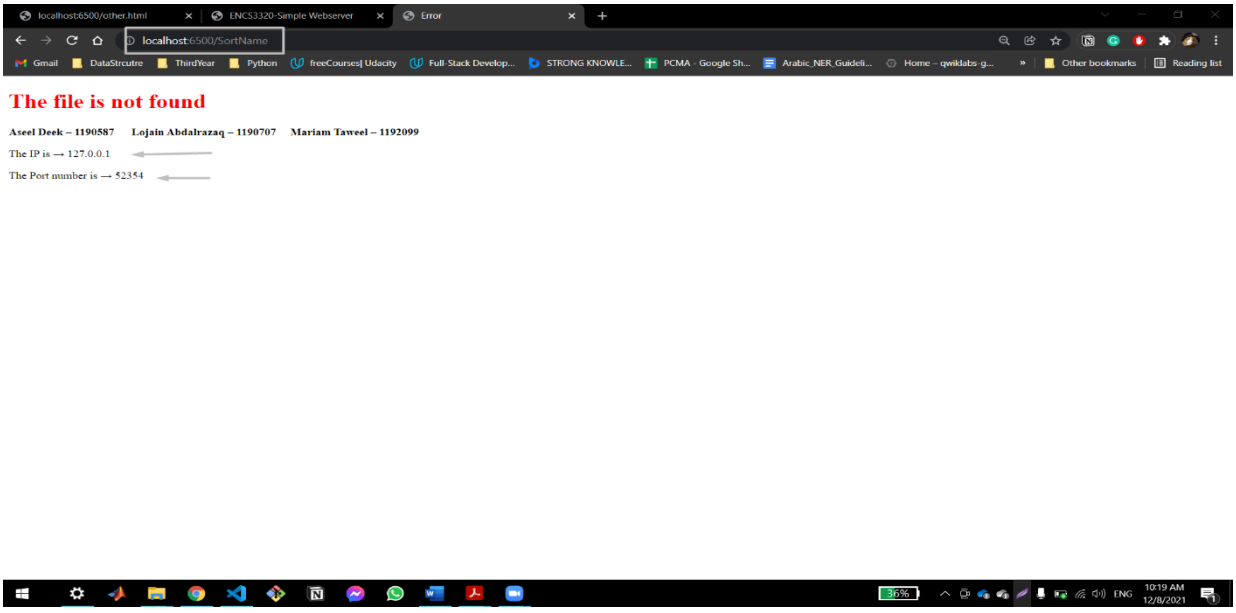



**Figure (2.3.19): the result of localhost:6500/ SortName**

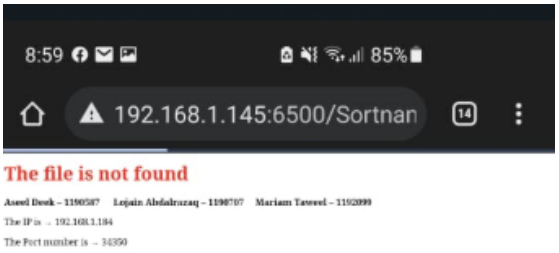



**Figure (2.3.20): the required SortName from a smartphone**

Figure (2.3.20) shown above, illustrates the taken result using the corresponding the wrong request **/ SortName**, using another device which is a smart phone.

### ➢ 2.3.7: Request3: localhost:6500/ SortByName

For this section, a file of type csv was made. The file contains the names and prices of ten different Cars. The requests exposed in part 2.3.7 and 2.3.8 are used to sort the content of that file either by name, or by price in descending order. figure (2.3.21) shows content of the csv file.

Cars.csv

```
1   Range Rover SE P360 , 580000
2   AUDI RS7, 114000
3   BMW M8 Convertible, 142500
4   Mercedes-Maybach S-class, 184900
5   G-Class SUV, 131750
6   Toyota Mirai 2021, 125000
7   Nissan GTR, 210740
8   CT5-V BLACKWING, 110150
9   Tesla Model S Plaid, 148990
10  Porsche 911 turbo S, 207000
```

**Figure (2.3.21): the content of Cars.csv**

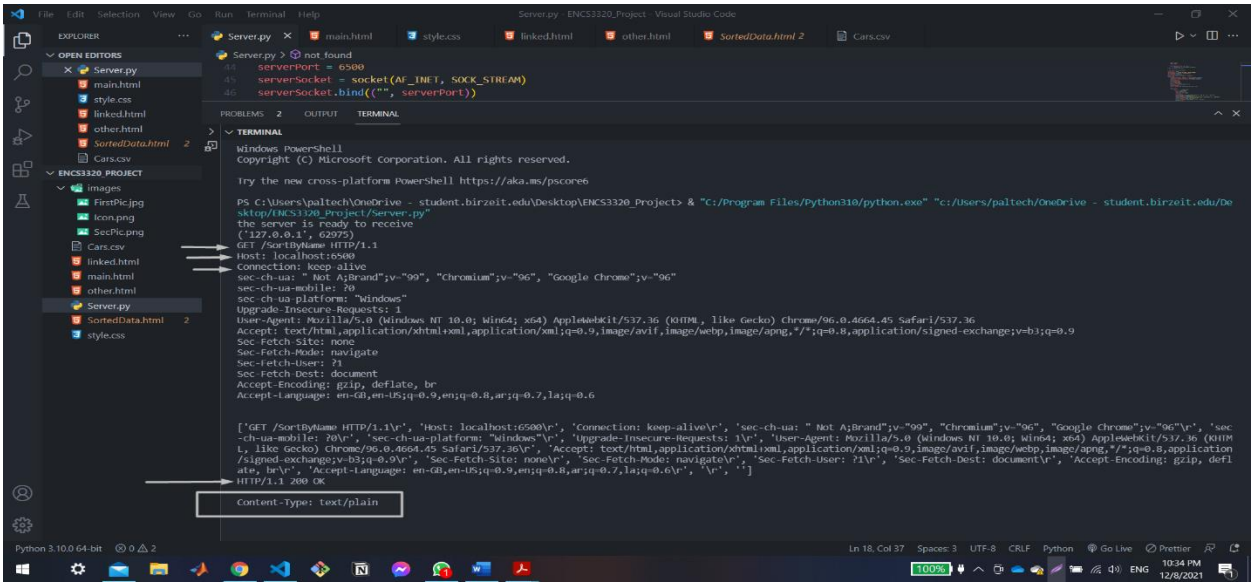



**Figure (2.3.22): the request message for localhost:6500/SortByName**

Figures (2.3.22) demonstrate the HTTP request messages when representing **localhost:6500/SortByName** request in the terminal. The message contents tell that the data was sent to the server using the GET method, the HTTP type is HTTP/1.1, the content type is text/plain, the connection type for the request is persistent since that was told from the keep-alive statement, and it shows the status of the HTTP response ***HTTP/1.1 200 OK*** means that the request succeeded.

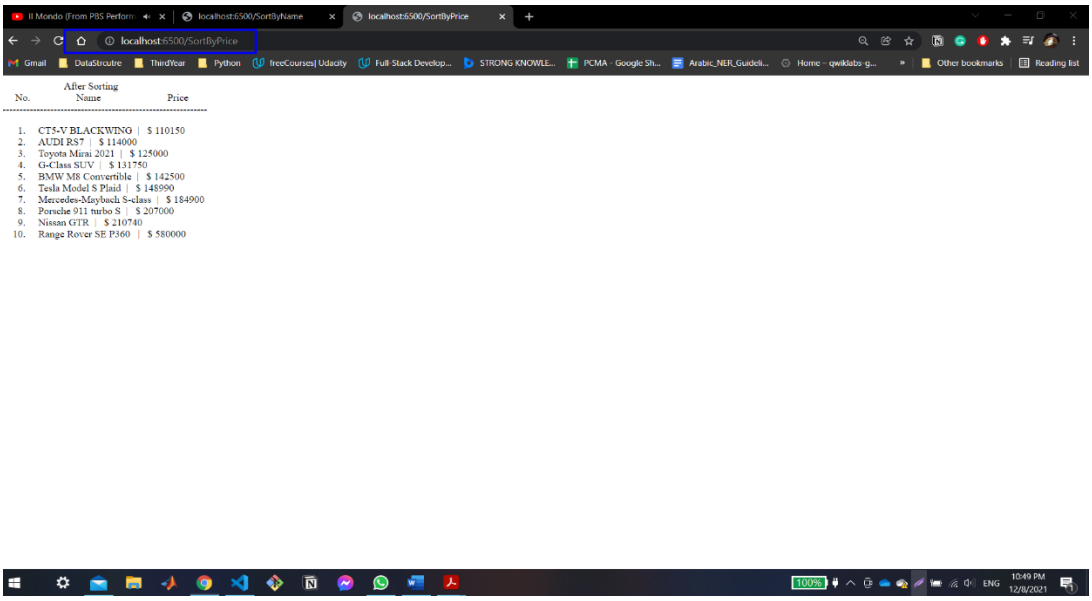



**Figure (2.3.23): the result of the localhost:6500/SortByName**

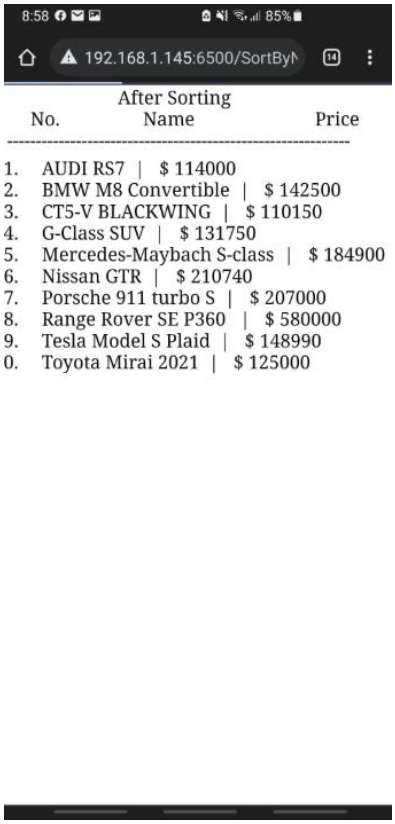



**Figure (2.3.24): the required SortByName from a smartphone**

### ➢ 2.3.8: Request3: localhost:6500/ SortByPrice

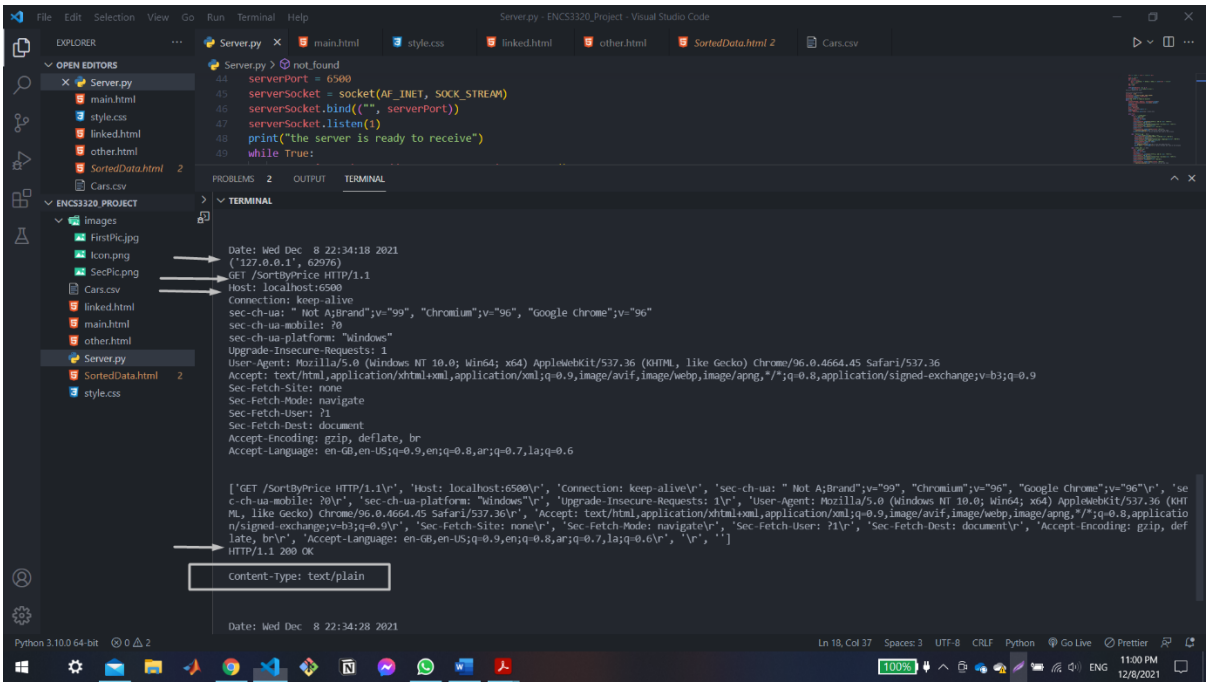



**Figure (2.3.25): the request message for localhost:6500/SortByPrice**

Figures (2.3.22) demonstrate the HTTP request messages when representing **localhost:6500/SortByPrice** request in the terminal. The message contents tell that the data was sent to the server using the GET method, the HTTP type is HTTP/1.1, the content type is text/plain, the connection type for the request is persistent since that was told from the keep-alive statement, and it shows the status of the HTTP response ***HTTP/1.1 200 OK*** means that the request succeeded.

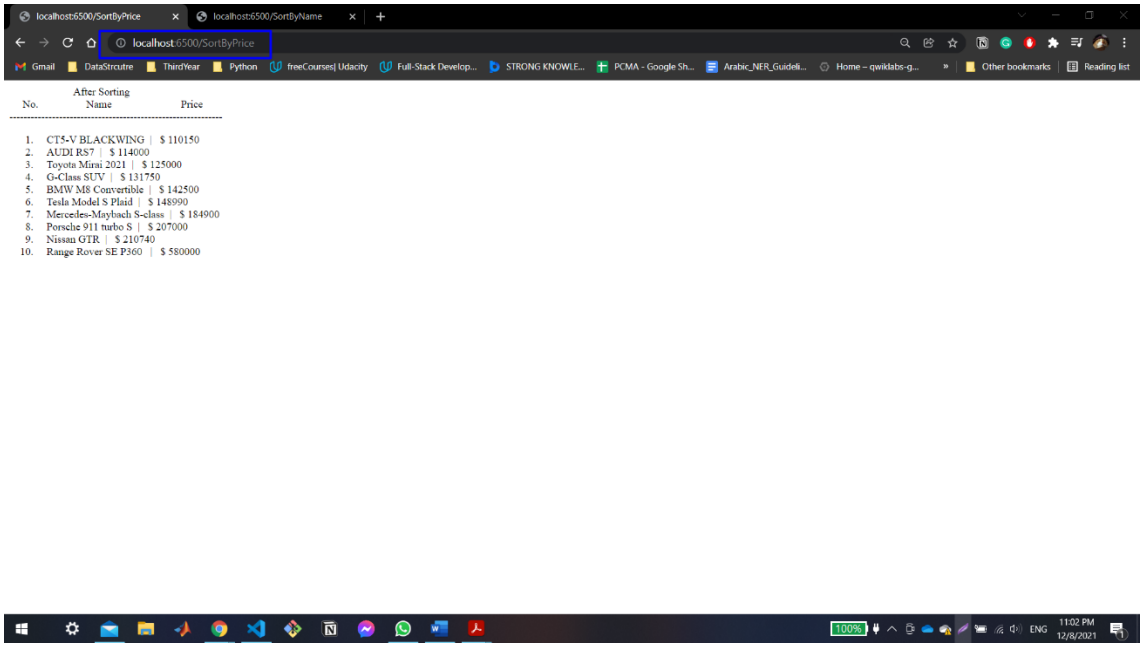



**Figure (2.3.26): the result of the localhost:6500/SortByPrice**


**Figure (2.3.27): the required SortByPrice from a smartphone**

# 3. Conclusion

The first part of the project includes several commands that illustrate the process of connecting a server to a specific destination, **ping** command, **tracert** command and **Nslookup** command were run using the CMD, and results were described in detail. In the Second part, the socket programming was used to write a program that uses HEAD method of HTTP to get HTTP response from a specific webserver. The concept of web server implementation was clearly proven in the last part. Several requests were tested, and the results were obtained successfully and as expected.

# 4. References


[1] Root Servers. (2018, August 11). Retrieved December 10, 2021, from https://www.iana.org/domains/root/servers.

[2] Jelen, S. (2021, July 30). *SecurityTrails: DNS root servers: What are they and are there really only 13*. SecurityTrails Index Page. Retrieved December 1, 2021, from https://securitytrails.com/blog/dns-root-servers.

[3] Biamp. (2019, August 5). *Pinging an IP address*. Biamp Cornerstone. Retrieved November 29, 2021, from https://support.biamp.com/General/Networking/Pinging_an_IP_address.

[4] *B root*. B Root. (2018, September 5). Retrieved August 26, 2021, from https://b.root-servers.org/.

[5] by Satoms, P., Satoms, Satoms Satoms are a leading online training course provider that specializes in technical & HSE courses., & Satoms are a leading online training course provider that specializes in technical & HSE courses. (2019, December 17). *Top 10 ping commands*. Satoms. Retrieved November 4, 2021, from https://satoms.com/ping-commands/.

[6] Fisher, T. (2020, September 11). *How to use the TRACERT command in windows*. Lifewire. Retrieved November 4, 2021, from https://www.lifewire.com/tracert-command-2618101.

[7] Pramatarov, M. (2021, July 21). *10 most used NSLOOKUP commands*. ClouDNS Blog. Retrieved November 4, 2021, from https://www.cloudns.net/blog/10-most-used-nslookup-commands/.

[8] JasonGerend. (2020, August 2). *NSLOOKUP*. Microsoft Docs. Retrieved November 4, 2021, from https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/nslookup.

[9] *Socket programming in python*. GeeksforGeeks. (2021, November 10). Retrieved December 5, 2021, from https://www.geeksforgeeks.org/socket-programming-python/.

[10] RisingStack Engineering. (2021, October 11). *Understanding & measuring HTTP timings with node.js*. RisingStack Engineering. Retrieved December 5, 2021, from https://blog.risingstack.com/measuring-http-timings-node-js/.

# 5. Appendixes

## ➢ 5.1: part3: **Html Code (Main.html) :**

```
<!DOCTYPE html>
<html>

    <!--*********************************-->
    <!--The head part of the HTML Webpage...-->
    <head>

        <meta charset="utf-8">
        <script src="https://use.fontawesome.com/d1341f9b7a.js"></script>
        <link rel="stylesheet" href=" http://localhost:6500/file.css"> #style.css http://localhost:6500/file.css
        <!--Putiing the icone of the page using the <link> tag-->
        <link rel = "icon" href ="https://img.icons8.com/external-wanicon-flat-wanicon/2x/external-network-computer-hardware-wanicon-flat-wanico
        <!--The title of the web page-->
        <title>ENCS3320-Simple Webserver</title>
    </head>
    <!--*********************************-->
    <!--The body part of the HTML Webpage...-->
    <body>
        <header>
         <!-- The main title of the Webpage-->
         <h1>Computer Networks</h1>
        </header>
        <!-- the section tag is for difining the sections of the page such as chaptrers or footers etc....-->

        <section class="banner">
         <!-- we gave it a class in order to style it later -->
         <div class="WelcomingSent">
             <br><br><br>

             <!--*********************************-->
             <!----------------------------------Welcoming Sentence------------------------------------------------->
             <!--*********************************-->
             <!-- Here is the welcoming message as required in the project-->
             <h2>Welcome To our course <br><span>Computer Networks</span></h2>
         </div>
        </section>

        <!--*********************************-->
        <main>
            <!--main content of the page which is the the group members name and the IDs-->
            <!--Some information about each students-->
            <article>
                <!--*********************************-->
                <!---------------------------------StudentS Information------------------------------------------------->
                <!--*********************************-->

                <!--The class box was given in order to style them like boxes using css language-->
                <div class="box">

                    <!-----------------------------------Lojain Abdalrazaq's part---------------------------------------------->
                    <!--The student name-->
                    <h3>Lojain Abdalrazaq-1190707</h3>
                    <h4>CSE - Student</h4>
                    <!--the information in the <p> tag-->
                    <p>
                        A third-year student at Birzeit University. Interested in fields of frontend, backend, and machine learning.
                        I have completed a lot of projects on different topics, one of my favorite ones was about building a survey Form
                        using HTML and CSS. I always insist on developing my skills in different subjects.I really enjoy volunteering
                        ,working with teams and meeting new people.
                    </p>
                    <ul>
                        <!--List of Lojain's accounts(Gihub, Facebook and Linkedin) using <ul> tag--->
                        <li><a href="https://github.com/Lojain-Abdalrazaq"><i class="fa fa-github" aria-hidden="true"></i></a></li>
                        <li><a href="https://www.facebook.com/lojain.jalal"><i class="fa fa-facebook-square" aria-hidden="true"></i></a></li>
                        <li><a href="http://linkedin.com/in/lojain-jalal-60a074206"><i class="fa fa-linkedin" aria-hidden="true"></i></a></li>
                    </ul>
                </div>

                 <!--The class box was given in order to style them like boxes using css language-->
                 <div class="box">
                    <!-----------------------------------Aseel Deek's part---------------------------------------------->
                    <!--The student name-->
                    <h3>Aseel Deek-1190587</h3>
                    <h4>CSE - Student</h4>
                    <!--the information in the <p> tag-->
                    <p>
```

```
                <!--the information in the <p> tag-->
                <p>
                   A third-year student at Birzeit University. Interested in computer studies, specifically deep learning, big data, AI,
                   and data analysis. Done a lot of projects on different subjects. One of them was a web page
                   for a restaurant called (David Chu's China bistro) using HTML, CSS, and JavaScript languages.
                   I enjoy drawing, learning new things, reading books, and playing with cats.
                </p>
                <ul>
                    <!--List of Aseel's accounts(Gihub, Facebook and Linkedin) using <ul> tag-->
                    <li><a href="https://github.com/aseeldeek"><i class="fa fa-github" aria-hidden="true"></i></a></li>
                    <li><a href="https://www.facebook.com/aseel.aldeek.1806"><i class="fa fa-facebook-square" aria-hidden="true"></i></a></l
                    <li><a href="http://linkedin.com/in/aseel-deek-a106201b4"><i class="fa fa-linkedin" aria-hidden="true"></i></a></li>
                </ul>
            </div>

            <!--The class box was given in order to style them like boxes using css language-->
            <div class="box">
                <!-----------------------------------Mariam Taweel's part---------------------------------------------->
                <!--The student name-->
                <h3>Mariam Taweel-1192099</h3>
                <h4>CSE - Student</h4>
                <!--The information in the <p> tag-->
                <p>
                  I am a student at Birzeit University in my third year. I worked on many projects during my first years at university in v
                  and languages such as (C, Java, and C#), and one of the best applications I made was an application for a car company.
                  I worked on many projects during my first years at university.
                  I love listening to music and recordings of books on self-development.
                </p>
                <ul>
                    <!--List of Mariam's accounts(Gihub, Facebook and Linkedin) using <ul> tag-->
                    <li><a href="https://github.com/maryamaltawil4"><i class="fa fa-github" aria-hidden="true"></i></a></li>
                    <li><a href="https://www.facebook.com/profile.php?id=100003974872539"><i class="fa fa-facebook-square" aria-hidden="true
                    <li><a href="http://linkedin.com/in/maryam-altawil-467900225"><i class="fa fa-linkedin" aria-hidden="true"></i></a></li>
                </ul>
            </div>
        </article>
        <!--**********************************-->
        <!-----------------------------------The JPG and PNG images---------------------------------------------->
        <!--**********************************-->
        <!-----------------------------------The JPG and PNG images---------------------------------------------->
        <!--**********************************-->

        <ul class="imgs">
            <!--The List of two images using the <ul> tag-->
            <!--JPG Extinsion-->
            <li><img src="https://cdn.datafloq.com/cache/blog_pictures/878x531/data-driven-marketing-2017-marketers-data-critical.jpg" alt=
            <!--PNG Extinsion-->
            <li><img src="https://www.transparentpng.com/thumb/networking/networking-picture-png-2.png" alt="PNG image"></li>
        </ul>
    </main>

    <!--**********************************-->
    <!------------------------------The Links (Webpage & HTML File)-------------------------------------->
    <!--**********************************-->

    <section>
    <!--Link to https://www.w3schools.com/tags/att_img_src.asp -->
    <p class="linked"> For more information about HTML Language, Please visit this <a href="https://www.w3schools.com/tags/att_img_src.asp'
    <!--Link to a local HTML File-->
    <p class="linked"> And Here is the linked <a href="http://localhost:6500/linked.html"> HTML File </a> </p>
    </section>
    <!--The footer part...-->
    <footer> &copy; all rights reserved 2021 </footer>

</body>

</html>
<!--The HTML part of the Webpage ends here...-->
```

## ➢ 5.2: part3: **CSS Code (style.css) :**

```css
/****************Body Styling*****************/
body{
    font-family: 'Lato', 'Lucida Grande', 'Lucida Sans Unicode', Tahoma, Sans-Serif;
    color: white;
    padding: 0;
    background: url(https://images.pexels.com/photos/1631677/pexels-photo-1631677.jpeg?cs=srgb&dl=pexels-abdullah-ghatasheh-1631677.jpg&fm=jpg);
    overflow-y: scroll;
    top: 0;
    left: 0;
    bottom: 0;
    right: 0;
    text-align: center;
}
/****************Header Styling*****************/
header{
    background-color:#af3c4d;
    color:white;
    position:fixed;
    text-align: center;
    width: 100%;
    z-index: 1;
    top: 0px;
    left: 0px;
}

header h1{
    border: 3px solid white;
    border-radius: 15px;
    padding: 3px 3px;
    display: inline-block;
    font-size: 15px;
}
/*****************Boxes Styling*****************/
.box{
    width: 350px;
    background: #af3c4d;
    padding: 25px;
    text-align: center;
    margin-left: 10px;
    margin-top: 10px;
    margin-top: 2%;
    color: white;
    display: inline-block;
    box-shadow: 0px -1px 6px 6px #ffffff;
    border-radius: 15px;
    margin: 40px 35px;
    max-width: 30%;
}

.box p{
    text-align: justify;
    line-height: 1.5em;
}

.box li{
    display: inline-block;
    margin: 6px;
    list-style: none;
}

.box li a{
    color: white;
    text-decoration: none;
    font-size: 50px;
    transition: all ease-in-out 250ms;
}

.box h3{
    font-size: 20px;
    letter-spacing: 3px;
    font-weight: 100;
}

ul{
```

```css
ul{
    margin: 0;
    padding: 0;
}

.box h4{
    font-size: 18px;
    letter-spacing: 3px;
    font-weight: 100;
}

.box li a:hover{
    color:#e09163;
}
/**************The welcoming Sentence Styling********************/
.WelcomingSent{

    padding: 4px;
    position: sticky;
    top: 20px;
    color:rgba(31, 30, 30, 0.795);

}

.banner h2 span{

    /* 1.4 times of the previus size */
    font-size: 1.5em;
    color:#0757ad;
    text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px white;

}
/*****************Images Styling*********************/
.imgs{
    text-align: center;
    white-space: nowrap;
    margin: 80px 0;
}

.imgs li{
    display: inline-block;
    width: 40%;
    margin: 20px 5%;
}

.imgs li img{
    max-width: 100%;
}
/*****************Site & HTML File Styling******************/
.linked a {
    border:3px solid white;
    border-radius: 15px;
    background-color: #af3c4d;
    color:white;
    text-decoration: none;
    padding: 3px 3px 3px 3px;
}
/*********************************************/

@media screen and (max-width:1000px){

    .WelcomingSent h2{
        font-size:15px;
    }
    .box p{
        font-size:15px;
    }
    .box h3{
        font-size:15px;
    }
    .box h4{
        font-size:15px;
    }
    .linked p{
    .linked p{
        font-size: 10px;
    }
}

@media screen and ( max-width:700px ) {

    .WelcomingSent h2{
        font-size:13px;
    }
    .box p{
        font-size:12px;
    }
    .box h3{
        font-size:13px;
    }
    .box h4{
        font-size:12px;
    }
    .linked p{
        font-size: 5px;
    }
```

## ➢ 5.3: part3: **Python Code (Server.py) :**

```python
from socket import *
import csv ,operator # for reading csv file
import datetime # for displaying the time and date
# ! NOTE : This Ptoject Was Bulid Using The Last Version Of Python 3.10.0
# function to deal with wrong requests and
def not_found(file):
    print("this is the not_found function")

    content = '<!DOCTYPE html><html><head><title>Error</title> </head><body><h1><span style="color:red;">The file is not found </span></h1><p id=
    connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
    print("HTTP/1.1 200 OK \r\n")
    connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
    print("Content-Type: text/html\r\n")
    connectionSocket.send(bytes("\r\n", "UTF-8"))
    print("\r\n")
    connectionSocket.sendall(bytes(content, "UTF-8"))
    x = datetime.datetime.now()
    print("Date:", x.strftime("%c")) # showing the date

# end of the function

# function for reading the csv file
def printTohtml(Alist, htmlfile):
    html =  "<html>\n<head>       After Sorting</head>\n<style>p { margin: 0 !important; }</style>\n<body>\n"

    title = "  No.      Name        Price "
    deco= "-----------------------------------------------------------\n"

    html += '\n<p>' + title + '</p>\n'+ deco

    strat = '<ol>\n'
    html +=strat
    for line in Alist:
        para = '<li> ' + '  |   $'.join(line) + '</li>\n'
        html += para
    end ='</ol>'
    html +=end

    with open(htmlfile, 'w') as f:
        f.write(html + "\n</body>\n</html>")
# end of reading csv file

# the strart of the project
serverPort = 6500
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(("", serverPort))
serverSocket.listen(1)
print("the server is ready to receive")
while True:
    connectionSocket, address = serverSocket.accept()
    sentence = connectionSocket.recv(2048).decode()
    print(address)
    print(sentence)
    ip = address[0]
    port = address[1]
    headers = sentence.split('\n')
    print(headers)
    file = headers[0].split()[1] # indesx.html

    match file:
        case '/' | '/index.html' : # when requesting / or /index.html
           file = 'main.html'
           fin = open(file)
           content = fin.read()
           fin.close()
           connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
           print("HTTP/1.1 200 OK \r\n")
           connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
           print("Content-Type: text/html\r\n")
           connectionSocket.send(bytes("\r\n", "UTF-8"))
           print("\r\n")
           connectionSocket.sendall(bytes(content, "UTF-8"))
           x = datetime.datetime.now() # to print the date and time
           print(x.strftime("%c")) # print the date and time in the terminal

```

```
            case '/Network1.jpg' :  # when requesting /Network1.jpg
                fin = open("images/FirstPic.jpg", "rb")
                connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
                print("HTTP/1.1 200 OK \r\n")
                connectionSocket.send(bytes("Content-Type: image/jpg\r\n\r\n", "UTF-8"))
                print("Content-Type: image/jpg\r\n\r\n")
                connectionSocket.send(fin.read())
                print(str(fin.read()))
                x = datetime.datetime.now() # to print the date and time
                print("Date:", x.strftime("%c")) # print the date and time in the terminal

            case '/other.html' : # # when requesting /other.html
                file = 'other.html'
                fin = open(file)
                content = fin.read()
                fin.close()
                connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
                print("HTTP/1.1 200 OK \r\n")
                connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
                print("Content-Type: text/html\r\n")
                connectionSocket.send(bytes("\r\n", "UTF-8"))
                print("\r\n")
                connectionSocket.sendall(bytes(content, "UTF-8"))
                x = datetime.datetime.now() # to print the date and time
                print("Date:", x.strftime("%c"))# print the date and time in the terminal

            case '/Network1.png' : # when requesting /Network1.png
                fin = open("images/SecPic.png", "rb")
                connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
                print("HTTP/1.1 200 OK \r\n")
                connectionSocket.send(bytes("Content-Type: image/png\r\n\r\n", "UTF-8"))
                print("Content-Type: image/png\r\n\r\n")
                connectionSocket.send(fin.read())  # send the contents of the picture
                print(str(fin.read()))
                x = datetime.datetime.now() # to print the date and time
                print("Date:", x.strftime("%c")) # print the date and time in the terminal

            case '/file.css' :  # when requesting /file.css
                file = 'style.css'
                fin = open(file)
                content = fin.read()
                fin.close()
                connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
                print("HTTP/1.1 200 OK \r\n")
                connectionSocket.send(bytes("Content-Type: text/css\r\n", "UTF-8"))
                print("Content-Type: text/css\r\n")
                connectionSocket.send(bytes("\r\n", "UTF-8"))
                print("\r\n")
                connectionSocket.sendall(bytes(content, "UTF-8"))
                x = datetime.datetime.now() # to print the date and time
                print("Date:", x.strftime("%c")) # print the date and time in the terminal

            case '/SortByName': # when requesting /SortByName
                sample = open('Cars.csv','r')
                csv1 = csv.reader(sample, delimiter=',')
                sorted_prices = sorted(csv1,key=operator.itemgetter(0))
                printTohtml(sorted_prices, 'SortedData.html')
                file = 'SortedData.html'
                fin = open(file)
                content = fin.read()
                fin.close()
                connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
                print("HTTP/1.1 200 OK \r\n")
                connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8")) # should be always html
                print("Content-Type: text/plain\r\n")
                connectionSocket.send(bytes("\r\n", "UTF-8"))
                print("\r\n")
                connectionSocket.sendall(bytes(content, "UTF-8"))
                x = datetime.datetime.now() # to print the date and time
                print("Date:", x.strftime("%c")) # print the date and time in the terminal

            case '/SortByPrice': # when requesting /SortByPrice
                sample = open('Cars.csv','r')
                csv1 = csv.reader(sample, delimiter=',')
```

```
            csv1 = csv.reader(sample, delimiter=',')
            sorted_prices = sorted(csv1,key=operator.itemgetter(1))
            printTohtml(sorted_prices, 'SortedData.html')
            file = 'SortedData.html'
            fin = open(file)
            content = fin.read()
            fin.close()
            connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
            print("HTTP/1.1 200 OK \r\n")
            connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8")) # should be always html
            print("Content-Type: text/plain\r\n")
            connectionSocket.send(bytes("\r\n", "UTF-8"))
            print("\r\n")
            connectionSocket.sendall(bytes(content, "UTF-8"))
            x = datetime.datetime.now() # to print the date and time
            print("Date:", x.strftime("%c")) # print the date and time in the terminal

        case'/linked.html':
            file = 'linked.html'
            fin = open(file)
            content = fin.read()
            fin.close()
            connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
            print("HTTP/1.1 200 OK \r\n")
            connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
            print("Content-Type: text/html\r\n")
            connectionSocket.send(bytes("\r\n", "UTF-8"))
            print("\r\n")
            connectionSocket.sendall(bytes(content, "UTF-8"))


        case unknown_command: # when requesting Wrong file
            connectionSocket.send(bytes("HTTP/1.1 404 Not Found \r\n", "UTF-8"))
            print("HTTP/1.1 404 Not Found \r\n")
            not_found(file)
```

## ➢ 5.4: part2: **Python Code (Question2.py) :**

Question2.py > ...

```python
from socket import *
from datetime  import *
Server_name = input("Please Enter the site name:\n")
#made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM.
Client = socket(AF_INET , SOCK_STREAM)
# default port for socket
prot_Server = 80
# connecting to the server
Client.connect((Server_name , prot_Server))
#time before sending request
print (datetime.now())
 # send a HEAD / HTTP/1.1 to the client. encoding to send byte type.
Client.send("HEAD / HTTP/1.1 \r\n".encode())
Client.send("Hostname: {name_Server} \r\n\r\n".encode())
#time response befor the response
time=datetime.now()
modified_sentence = Client.recv(1024)
#time response after the response
time_next=datetime.now()
finaltime=time_next - time
print("response time:",finaltime)
# receive data from the server
print("From server :)", modified_sentence.decode())
# Close the connection with the client
Client.close()
```