



BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Networks

ENCS3320

Course Project

WEB SERVER IMPLEMENTATION

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

Student's Name: Mariam Taweeel **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
1.1.	What are Root Servers?.....	1
2.	Procedure.....	2
2.1.	Part1: Running Commands:	2
2.1.1:	Ping a Device in The Same Network:	2
2.1.2:	Ping b.root-servers.net:.....	4
2.1.3:	Tracert Command:	5
2.1.4:	Nslookup Command:.....	6
2.2.	Part2: HTTP Response and Elapsed Time:.....	7
2.3.	Part3: Implanting Web Server:	9
2.3.1:	Request1: localhost:6500/ localhost: 6500/index.html.....	9
2.3.2:	Request2: localhost:6500/file.css	13
2.3.3:	Request3: localhost:6500/Network1.png.....	15
2.3.4:	Request3: localhost:6500/Network1.jpg.....	18
2.3.5:	Request4: localhost:6500/other.html	19
2.3.6:	Request3: localhost:6500/ File does not find.....	21
2.3.7:	Request3: localhost:6500/ SortByName.....	23
2.3.8:	Request3: localhost:6500/ SortByPrice	25
3.	Conclusion.....	27
4.	References.....	28
5.	Appendixes.....	29
5.1:	part3: Html Code (Main.html) :	29
5.2:	part3: CSS Code (style.css) :.....	31
5.3:	part3: Python Code (Server.py) :.....	33
5.4:	part2: Python Code (Question2.py) :	35

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.

IP Address	192.168.1.5
Subnet Mask	255.255.255.0
Router	192.168.1.1

Figure (2.1.1): IP address of the smartphone

```
C:\ Command Prompt - time
C:\Users\Ho182>ping 192.168.1.5
Pinging 192.168.1.5 with 32 bytes of data:
Reply from 192.168.1.5: bytes=32 time=759ms TTL=64
Reply from 192.168.1.5: bytes=32 time=844ms TTL=64
Reply from 192.168.1.5: bytes=32 time=349ms TTL=64
Reply from 192.168.1.5: bytes=32 time=1232ms TTL=64

Ping statistics for 192.168.1.5:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 349ms, Maximum = 1232ms, Average = 796ms

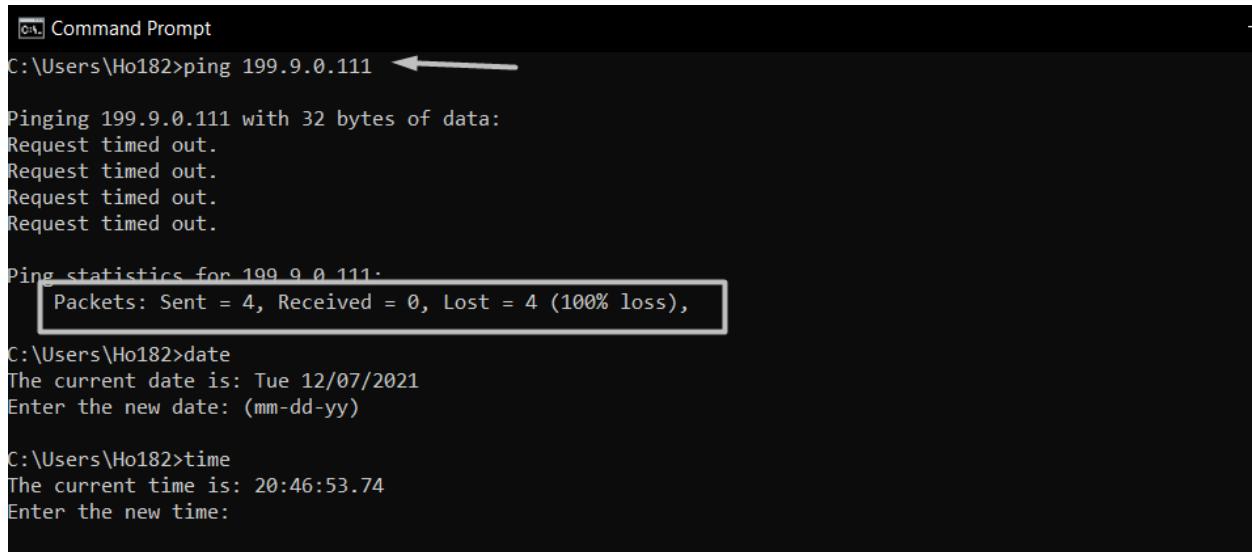
C:\Users\Ho182>date
The current date is: Tue 12/07/2021
Enter the new date: (mm-dd-yy)

C:\Users\Ho182>time
The current time is: 20:00:54.07
Enter the new time:
```

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).



The screenshot shows a Windows Command Prompt window with the following text:

```
C:\Users\Ho182>ping 199.9.0.111 ←
Pinging 199.9.0.111 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 199.9.0.111:
  Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Below the ping command, there is a date and time command:

```
C:\Users\Ho182>date
The current date is: Tue 12/07/2021
Enter the new date: (mm-dd-yy)
```

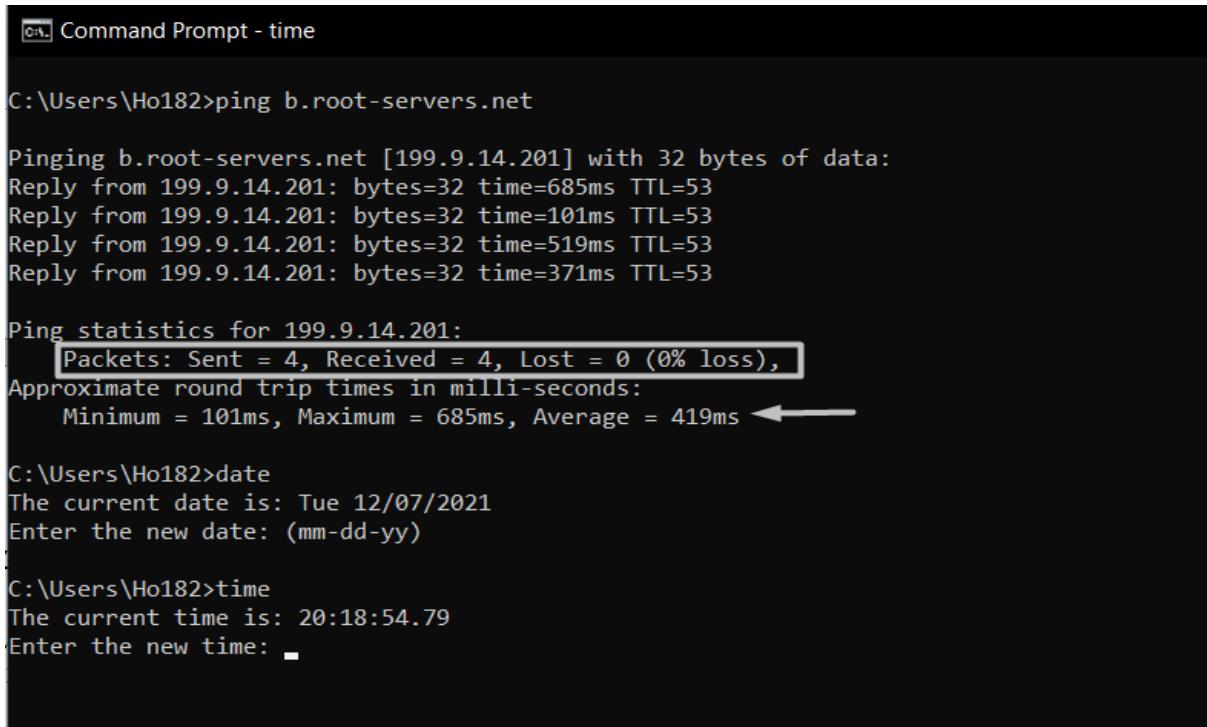
Finally, there is a time command:

```
C:\Users\Ho182>time
The current time is: 20:46:53.74
Enter the new time:
```

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].



Command Prompt - time

```
C:\Users\Ho182>ping b.root-servers.net

Pinging b.root-servers.net [199.9.14.201] with 32 bytes of data:
Reply from 199.9.14.201: bytes=32 time=685ms TTL=53
Reply from 199.9.14.201: bytes=32 time=101ms TTL=53
Reply from 199.9.14.201: bytes=32 time=519ms TTL=53
Reply from 199.9.14.201: bytes=32 time=371ms TTL=53

Ping statistics for 199.9.14.201:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 101ms, Maximum = 685ms, Average = 419ms ←

C:\Users\Ho182>date
The current date is: Tue 12/07/2021
Enter the new date: (mm-dd-yy)

C:\Users\Ho182>time
The current time is: 20:18:54.79
Enter the new time: -
```

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 to 199.9.14.201. 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**.

```
Command Prompt - time
C:\Users\Ho182>tracert b.root-servers.net

Tracing route to b.root-servers.net [199.9.14.201]
over a maximum of 30 hops:

 1  43 ms    92 ms    50 ms  Broadcom.Home [192.168.1.1]
 2  1548 ms   680 ms   368 ms  10.74.32.234
 3  1349 ms    *    2797 ms  10.74.42.22
 4    81 ms    104 ms   125 ms  10.18.28.1
 5  1019 ms    391 ms   502 ms  win-b2-link.ip.twelve99.net [213.248.86.98]
 6    398 ms   1077 ms   277 ms  win-bb4-link.ip.twelve99.net [62.115.114.182]
 7    712 ms    393 ms   164 ms  ffm-bb2-link.ip.twelve99.net [62.115.138.22]
 8    116 ms     99 ms    98 ms  adm-bb4-link.ip.twelve99.net [62.115.122.200]
 9    958 ms   1560 ms   134 ms  adm-b2-link_ip_twelve99.net [62.115.141.37]
10   168 ms    158 ms  1095 ms  b.root-servers.net [199.9.14.201]

Trace complete.

C:\Users\Ho182>date
The current date is: Tue 12/07/2021
Enter the new date: (mm-dd-yy)

C:\Users\Ho182>time
The current time is: 20:34:08.96
Enter the new time:
```

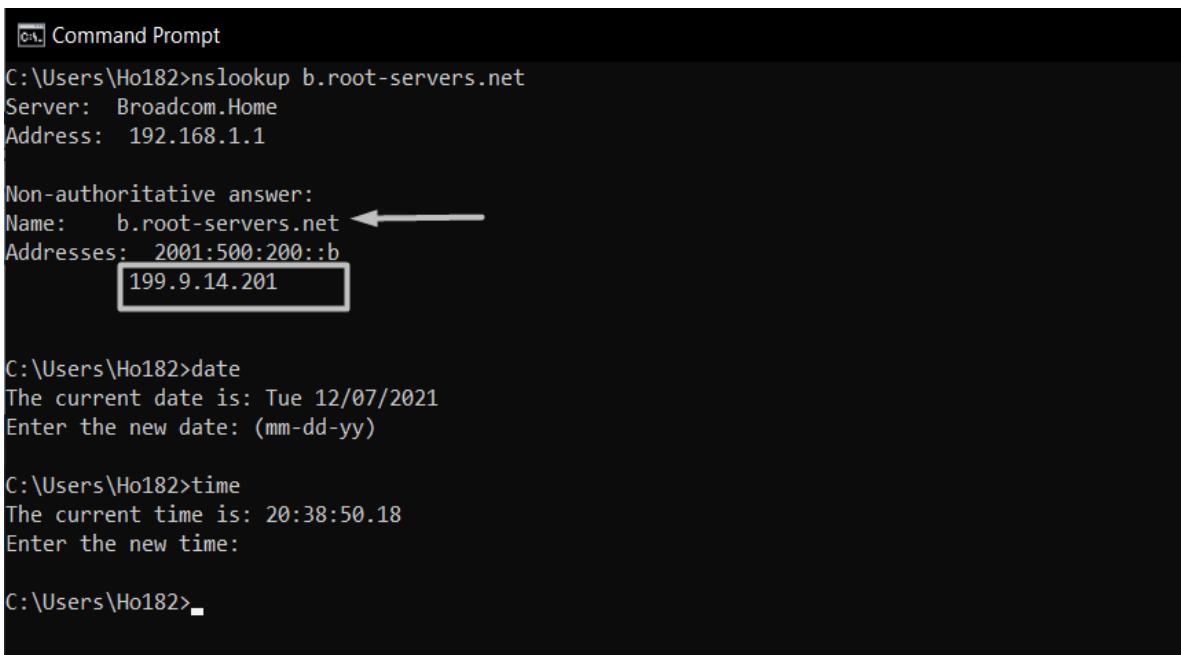
Figure (2.1.5): the result Results of pinging b.root-servers.net

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]**.

```
10   168 ms    158 ms  1095 ms  b.root-servers.net [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, asks 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**.



```
Command Prompt
C:\Users\Ho182>nslookup b.root-servers.net
Server:  Broadcom.Home
Address: 192.168.1.1

Non-authoritative answer:
Name:    b.root-servers.net ←
Addresses: 2001:500:200::b
          199.9.14.201

C:\Users\Ho182>date
The current date is: Tue 12/07/2021
Enter the new date: (mm-dd-yy)

C:\Users\Ho182>time
The current time is: 20:38:50.18
Enter the new time:

C:\Users\Ho182>
```

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]:

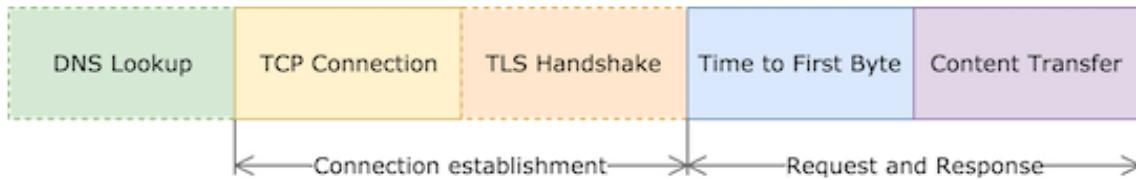


Figure (2.2.1): timeline of a usual HTTP Request.

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.

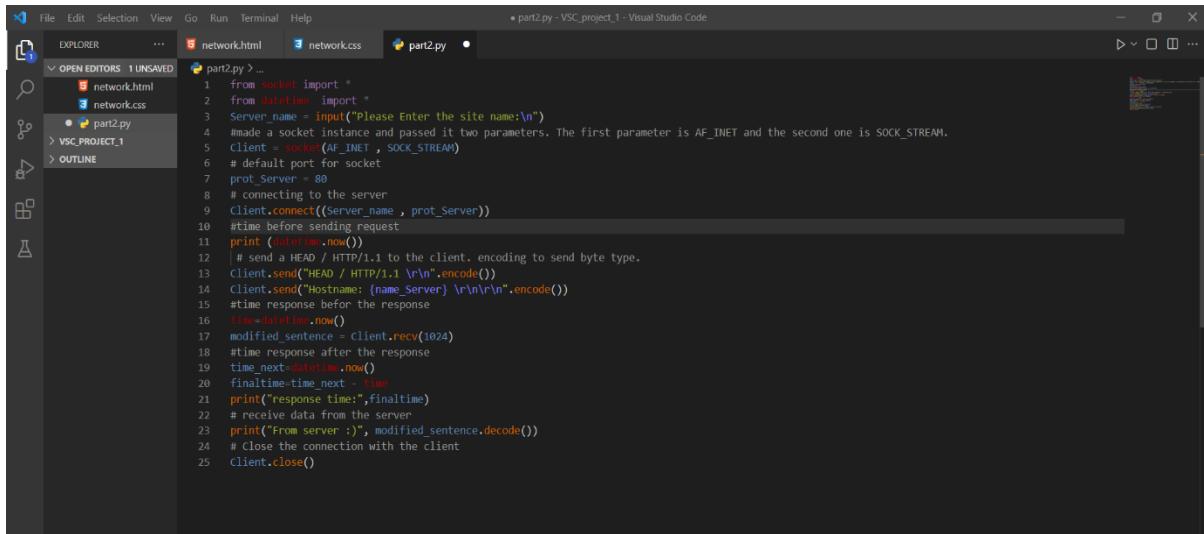
TCP Connection: Time it took to establish TCP connection between a source host and destination host.

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.



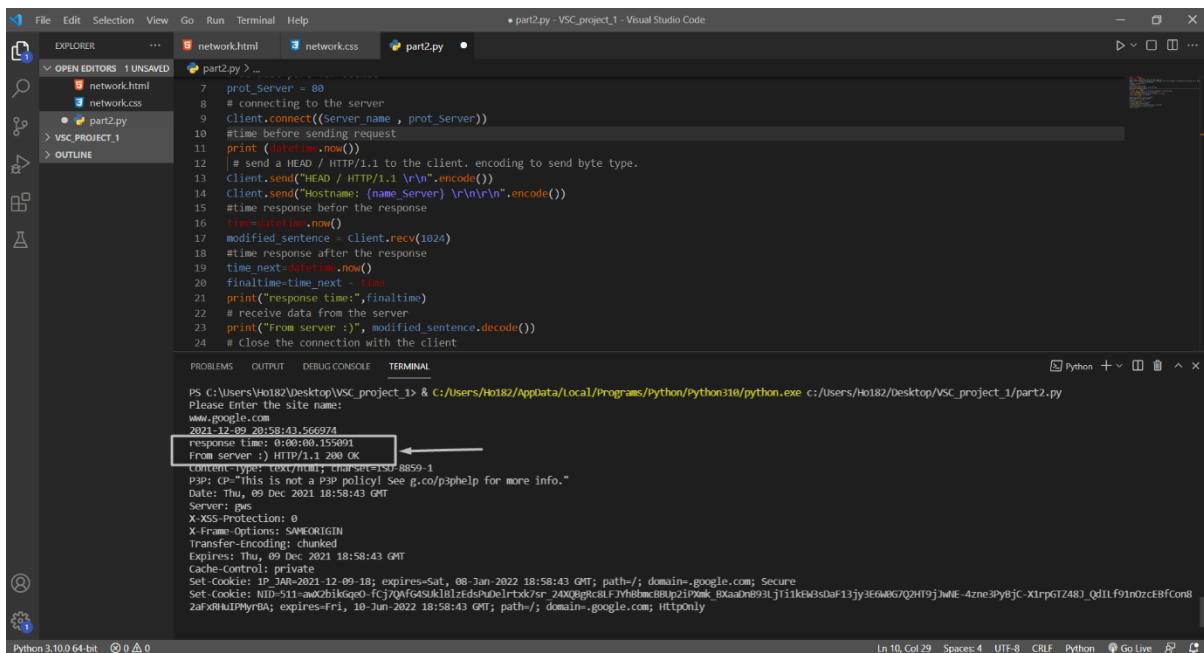
```

File Edit Selection View Go Run Terminal Help
part2.py - VSC_project_1 - Visual Studio Code
EXPLORER OPEN EDITORS 1 UNSAVED network.html network.css part2.py ...
VSC_PROJECT_1 OUTLINE

part2.py > ...
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: {}_Server\r\n\r\n".format(Server_name).encode())
#time response before 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



```

File Edit Selection View Go Run Terminal Help
part2.py - VSC_project_1 - Visual Studio Code
EXPLORER OPEN EDITORS 1 UNSAVED network.html network.css part2.py ...
VSC_PROJECT_1 OUTLINE

part2.py > ...
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: {}_Server\r\n\r\n".format(Server_name).encode())
#time response before 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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\H0182\Desktop\VSC_project_1> & C:/Users/H0182/AppData/Local/Programs/Python/Python310/python.exe c:/Users/H0182/Desktop/VSC_project_1/part2.py
Please enter the site name:
www.google.com
2021-12-09 20:58:43.566974
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: ID=511-a02b1kgq0-fc7jQAfGSKl1zfd5u0elrtkx/sr_24X0gRcaLFYhBmcBBUp2jPwMk; XaaDnB93Lj1i1EW3dAf13jy3E6W0G7Q2HT9jJwHE-4zne3PybjC_XTrpgTz48J_Qd1Lf91n0zctBFcon82afxRhuIMMyrBa; expires=Fri, 10-Jun-2022 18:58:43 GMT; path=/; domain=.google.com; Secure
Set-Cookie: NID=511-a02b1kgq0-fc7jQAfGSKl1zfd5u0elrtkx/sr_24X0gRcaLFYhBmcBBUp2jPwMk; XaaDnB93Lj1i1EW3dAf13jy3E6W0G7Q2HT9jJwHE-4zne3PybjC_XTrpgTz48J_Qd1Lf91n0zctBFcon82afxRhuIMMyrBa; expires=Fri, 10-Jun-2022 18:58:43 GMT; path=/; domain=.google.com; Secure
Python 3.10.0 64-bit
Ln 10, Col 29 Spaces: 4 UTF-8 CR/LF Python Go Live

```

Figure (2.2.3): the response time for requesting www.google.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 handles eight different requests specified and shows the HTTP requests on the terminal window.

➤ 2.3.1: Request1: localhost:6500/ localhost: 6500/index.html

The screenshot shows the Visual Studio Code interface with the Server.py file open in the editor. The terminal window displays the following HTTP request message:

```

GET / HTTP/1.1
Host: localhost:6500
Connection: keep-alive
sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
sec-ch-ua-mobile: "Windows NT 10.0; Win64; x64 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36"
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/96.0.4664.45 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6

[{"GET / HTTP/1.1", "Host": "localhost:6500", "connection": "keep-alive", "sec-ch-ua": "Not A;Brand";v="99", "chromium";v="96", "Google chrome";v="96"}, {"sec-ch-ua-mobile": "Windows NT 10.0; Win64; x64 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36", "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/96.0.4664.45 Safari/537.36", "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9", "sec-fetch-site": "none", "sec-fetch-mode": "navigate", "sec-fetch-dest": "document", "accept-encoding": "gzip, deflate, br", "accept-language": "en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6"}]
Content-type: text/html

```

Figure (2.3.1): HTTP request for the html file

The screenshot shows the Visual Studio Code interface with the Server.py file open in the editor. The terminal window displays the following HTTP request message:

```

HTTP/1.1 200 OK
Content-Type: text/html

Tue Dec 7 17:27:09 2021
[{"GET /file.css HTTP/1.1", "Host": "localhost:6500", "connection": "keep-alive", "sec-ch-ua": "Not A;Brand";v="99", "chromium";v="96", "Google chrome";v="96"}, {"sec-ch-ua-mobile": "Windows NT 10.0; Win64; x64 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36", "sec-ch-ua-platform": "Windows", "accept": "text/css,*/*;q=0.1", "sec-fetch-site": "same-origin", "sec-fetch-mode": "no-store", "sec-fetch-dest": "style", "Referer": "http://127.0.0.1:6500/", "accept-encoding": "gzip, deflate, br", "accept-language": "en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6"}]
Content-Type: text/css
Date: Tue Dec 7 17:27:09 2021

```

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 .

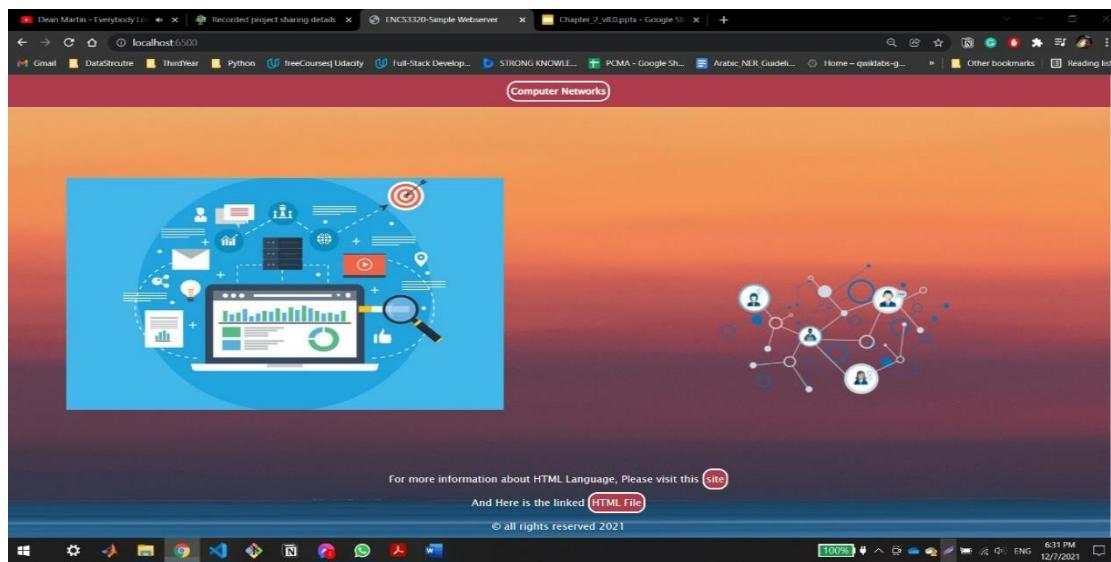
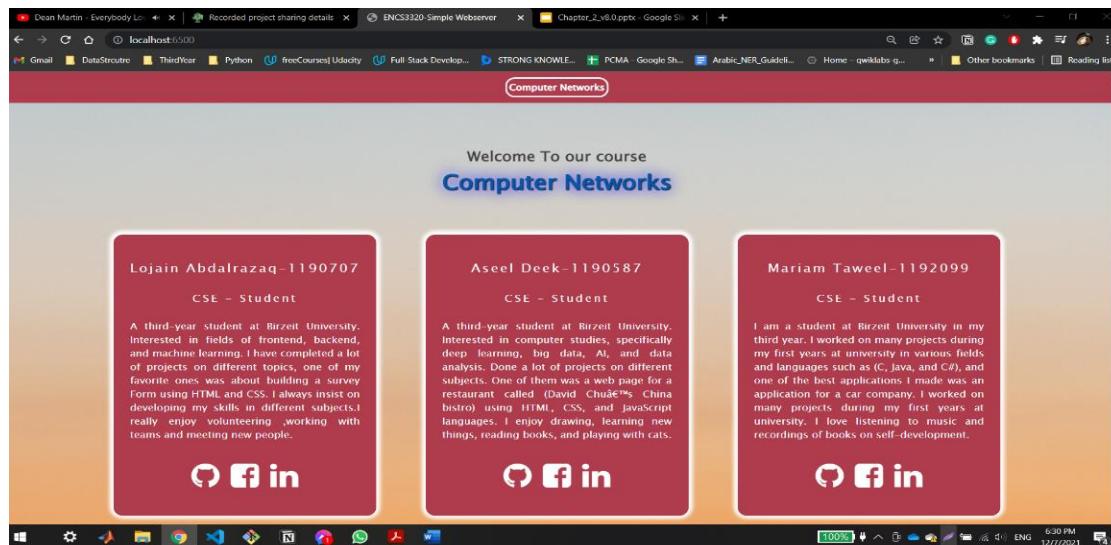


Figure (2.3.3): the styled HTML file

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

The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal window displays the following text:

```

HTTP/1.1 200 OK [P]
connectionSocket.send(bytes('Content-type: image/png\r\n\r\n', "UTF-8"))
print("Content-Type: image/png\r\n\r\n")

[...]
odt/Desktop/ENCS3320_Project/Server.py"
the server is ready to receive
('127.0.0.1', 6275)
GET /file.css HTTP/1.1
Host: localhost:6500
Connection: keep-alive
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
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/96.0.4664.45 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=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6

[{"GET /file.css HTTP/1.1\r", "Host: localhost:6500\r", "Connection: keep-alive\r", "sec-ch-ua: \" Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\", \"Google Chrome\";v=\"96\"\r", "sec-ch-ua-mobile: ?0\r", "sec-ch-ua-platform: \"Windows\"\r", "Upgrade-Insecure-Requests: 1\r", "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36\r", "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=0.9\r", "Sec-Fetch-Site: none\r", "Sec-Fetch-Mode: navigate\r", "Sec-Fetch-User: ?1\r", "Sec-Fetch-Dest: document\r", "Accept-Encoding: gzip, deflate, br\r", "Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6\r"]
HTTP/1.1 200 OK

Content-type: text/css

Date: Tue Dec 7 19:15:31 2021
[...]

```

The terminal also shows the file structure of the ENCS3320_PROJECT folder, which includes files like Server.py, main.html, style.css, linked.html, file.css, and Cars.csv.

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.

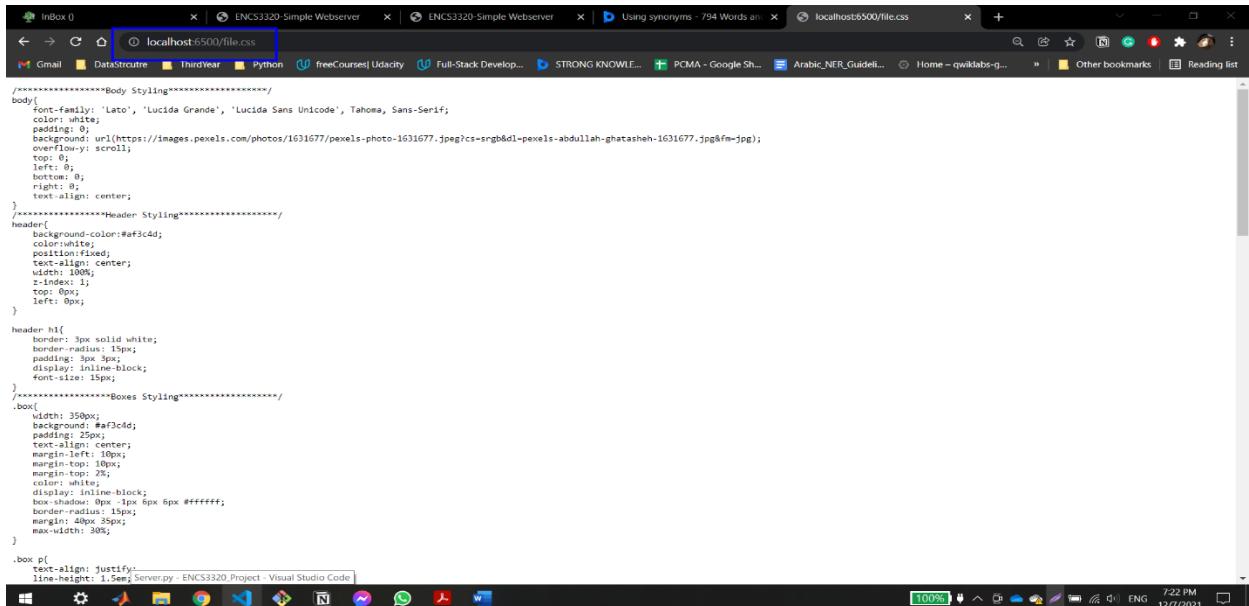


Figure (2.3.7): the result of requesting localhost:6500/file



A screenshot of a smartphone displaying a CSS file in a browser. The browser's status bar shows the time as 8:57 and battery level at 85%. The address bar shows the URL `192.168.1.145:6500/file.css`. The main content area contains the following CSS code:

```
*****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*****
```

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

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the project: Server.py, main.html, style.css, linked.html, file.html, and Cars.csv.
- Terminal:** Displays the output of the Python script Server.py running on port 5000. The server is listening at 127.0.0.1:5000 and responding to a GET request from a Chromium browser on Windows 10. The response includes headers like Content-Type: image/png and a Content-Length of 108.
- Status Bar:** Shows Python 3.10.0 64-bit, a progress bar at 100%, and system status icons.

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.

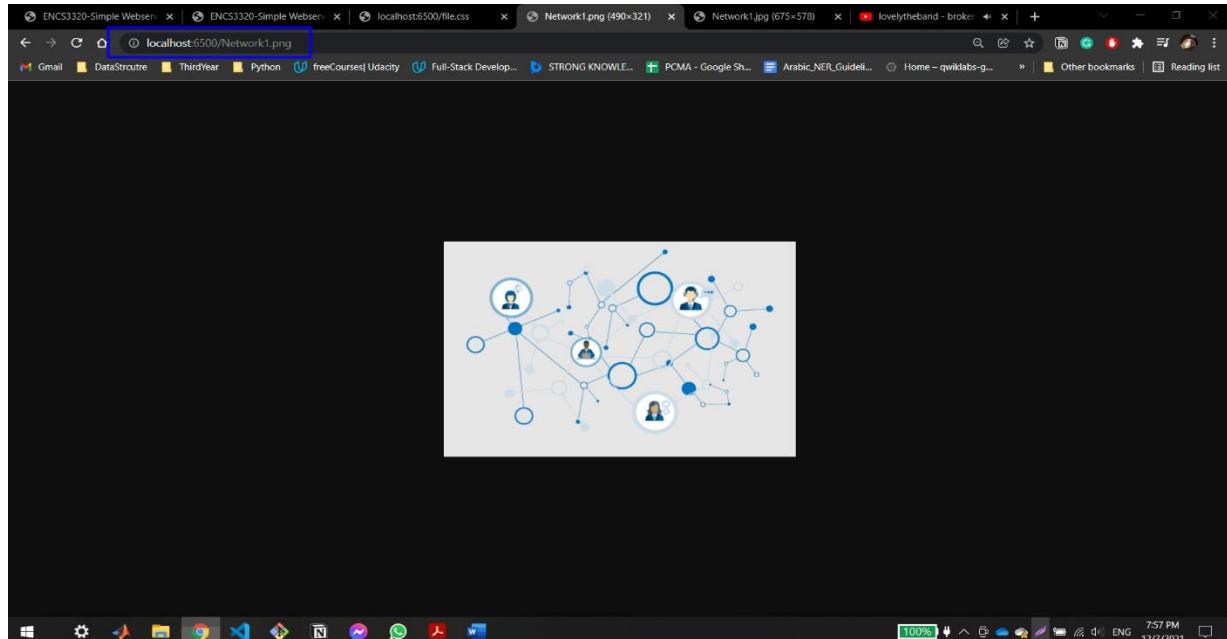


Figure (2.3.10): the result of localhost:6500/Network1.png request



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

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there are files like Server.py, main.html, style.css, linked.html, file.html, and Cars.csv. The Server.py file contains Python code for a web server. The terminal window shows the command `python Server.py` being run, followed by the HTTP request message:

```

GET /Network1.jpg HTTP/1.1
Host: localhost:6500
Connection: keep-alive
sec-ch-ua: "Not A Brand";v="0", "Chromium";v="96", "Google Chrome";v="96"
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/96.0.4664.45 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=0.9
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6
Content-type: image/png
b''
```

The status line at the bottom indicates the request was successful: `[200 OK]`.

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.

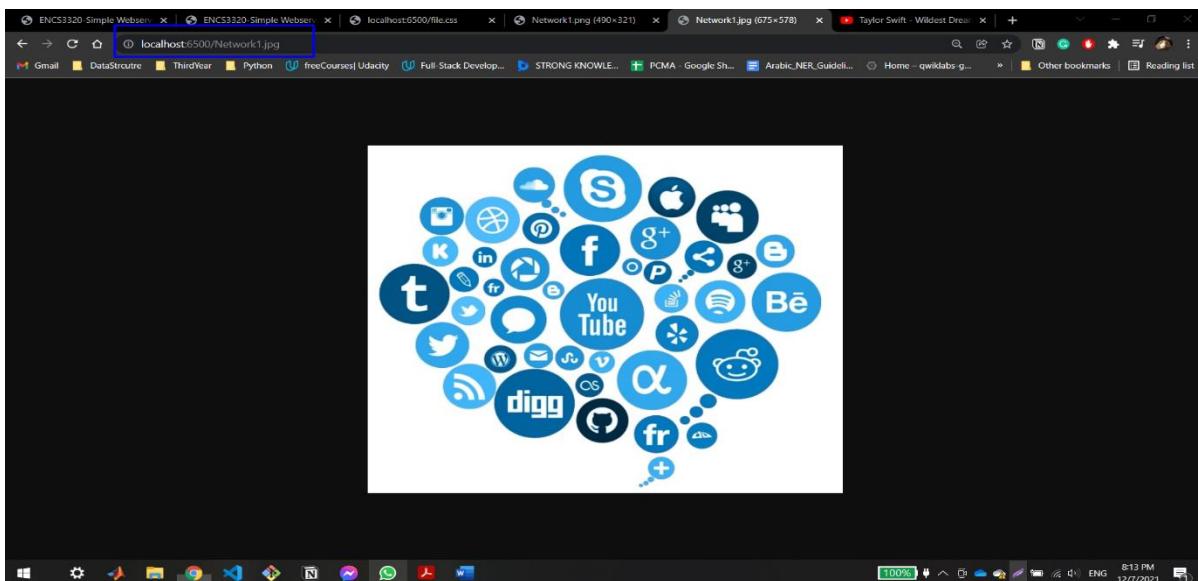


Figure (2.3.13): the result of localhost:6500/Network1.jpg request



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

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files like Server.py, main.html, style.css, linked.html, other.html, and Cars.csv.
- Code Editor:** Displays Python code for a server, specifically handling a GET request for 'other.html'.
- Terminal:** Shows the command-line output of the server's response to a GET request for 'other.html' at port 6500. The response includes headers and the status line "HTTP/1.1 200 OK".
- Bottom Status Bar:** Shows system information including battery level (71%), date (12/8/2021), and time (9:25 AM).

```

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' : #done
    fin = open("images/FirstPic.jpg", "rb")
    connectionSocket.sendall(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
    print("HTTP/1.1 200 OK \r\n")
    connectionSocket.sendall(bytes("Content-type: image/jpg\r\n\r\n", "UTF-8"))

skttop/ENCS3320_Project/Server.py"
the server is ready to receive
("127.0.0.1", 59857)
GET /other.html HTTP/1.1
Host: localhost:6500
Connection: keep-alive
sec-ch-user: "Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
sec-ch-u-mobile: 20
sec-ch-u-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 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=0.9
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: Document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6

[ "GET /other.html HTTP/1.1", "Host: localhost:6500", "Connection: keep-alive", "sec-ch-user: \"Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\"\\r\\n", "sec-ch-u-mobile: 20\\r\\n", "sec-ch-u-platform: \"Windows\"\\r\\n", "Upgrade-Insecure-Requests: 1\\r\\n", "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36\\r\\n", "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=0.9\\r\\n", "sec-fetch-site: none\\r\\n", "sec-fetch-mode: navigate\\r\\n", "sec-fetch-user: ?1\\r\\n", "sec-fetch-dest: document\\r\\n", "Accept-Encoding: gzip, deflate, br\\r\\n", "Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6\\r\\n", "\\r\\n" ]
Content-type: text/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.

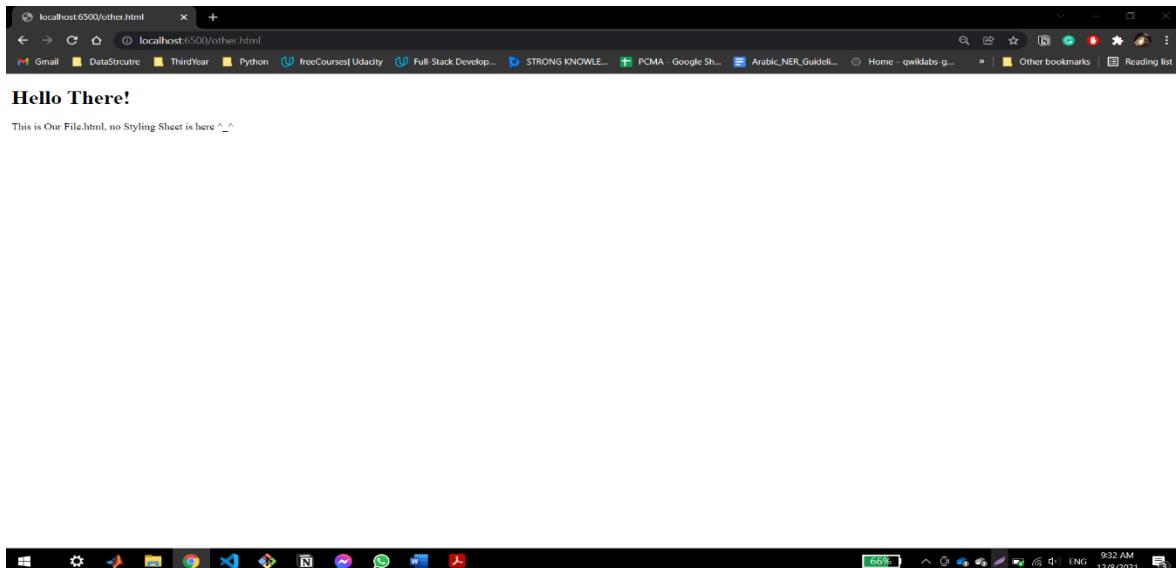


Figure (2.3.16): the result of localhost:6500/other.html



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

```

File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS ...
Server.py main.html style.css linked.html other.html Cars.csv
Server.py ...
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' : #done
    fin = open("images/firstPic.jpg", "rb")
    connectionSocket.sendall(b"HTTP/1.1 200 OK\r\n\r\n", "UTF-8"))

('127.0.0.1', 52354)
GET /SortName HTTP/1.1
Host: localhost:6500
Connection: keep-alive
sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
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/96.0.4664.45 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6

[GET /SortName HTTP/1.1\r\n, 'Host: localhost:6500\r\n', 'Connection: keep-alive\r\n', 'sec-ch-ua: "Not A;Brand";v="99", "chromium";v="96", "Google Chrome";v="96"\r\n', 'sec-ch-ua-mobile: 20\r\n', 'sec-ch-ua-platform: "Windows"\r\n', 'Upgrade-Insecure-Requests: 1\r\n', 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36\r\n', 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n', 'Sec-Fetch-Site: none\r\n', 'Sec-Fetch-Mode: navigate\r\n', 'Sec-Fetch-User: ?1\r\n', 'Sec-Fetch-Dest: document\r\n', 'Accept-Encoding: gzip, deflate, br\r\n', 'Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6\r\n', '\r\n', '\r\n']

this is the not found function
HTTP/1.1 200 OK
Content-Type: text/html

```

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.

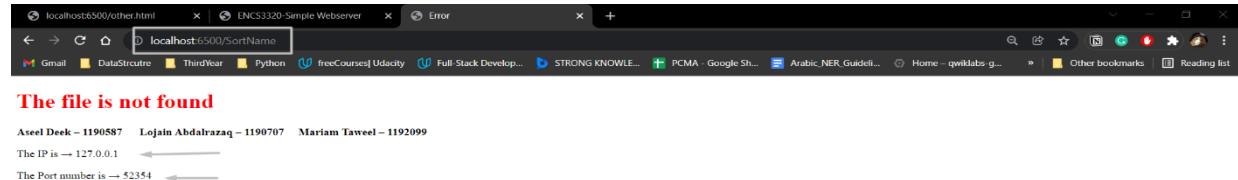


Figure (2.3.19): the result of localhost:6500/ SortName

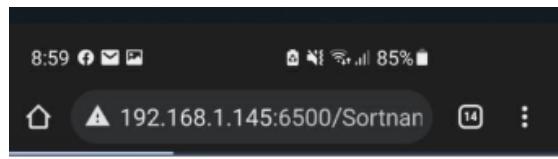
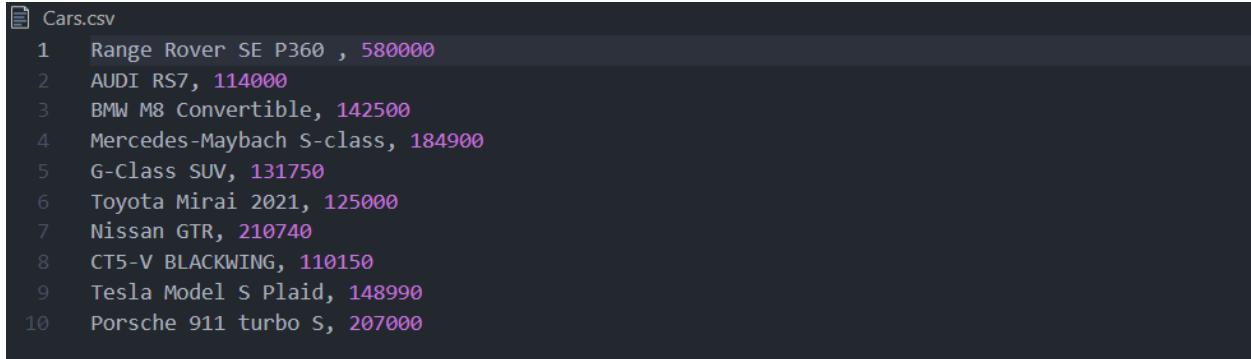


Figure (2.3.20): the required SortName from a smartphone

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.

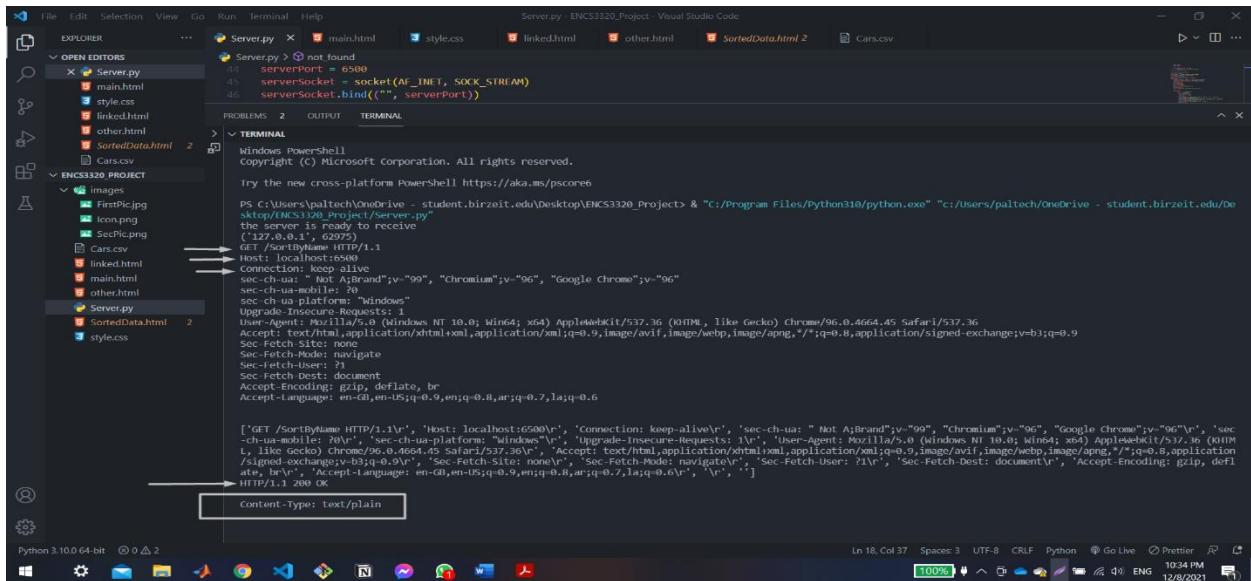


```

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



The screenshot shows a Visual Studio Code interface with the following details:

- Project Structure:** The project is named "ENC3320_PROJECT". It contains several files: Server.py, main.html, style.css, linked.html, other.html, and SortedData.html. There is also a "Cars.csv" file located in the root directory.
- Terminal Output:**

```

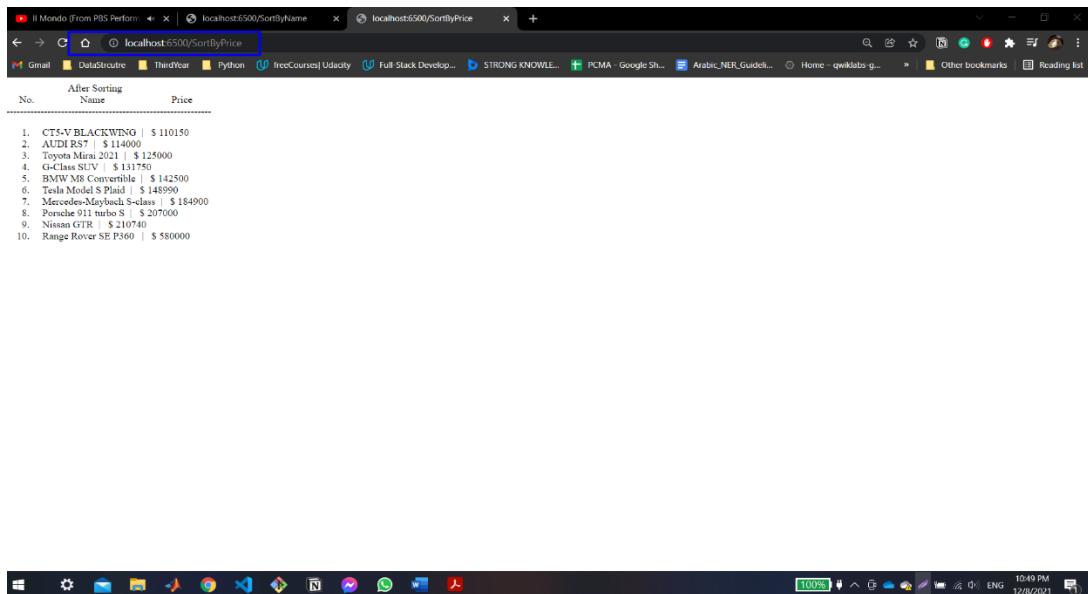
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\paltech\OneDrive - student.birzeit.edu\Desktop\ENC3320_Project> python Server.py
the server is ready to receive
('127.0.0.1', 62975)
[04/Aug/2021 11:11:11] "GET /SortByName HTTP/1.1"
Host: localhost:6500
Connection: keep-alive
sec-ch-ua: "Not A;Brand";v="99", "Chromium";v="96", "Google Chrome";v="96"
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/96.0.4664.45 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,ta;q=0.6,fr;q=0.5,de;q=0.4,es;q=0.3,pt;q=0.2,ru;q=0.1,he;q=0.05
Content-type: text/plain

```
- Bottom Status Bar:** Shows Python 3.10.0 64-bit, 0:00:22, and system status including battery level (100%), signal strength, and network connection.

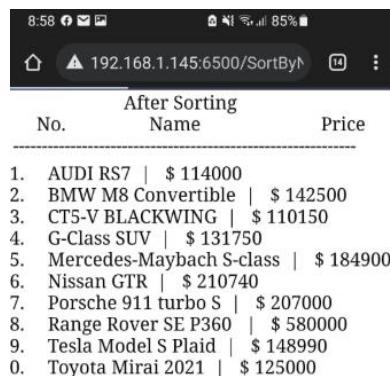
Figure (2.3.22): the request message for localhost:6500/SortByName

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.



No.	Name	Price
1.	CT5-V BLACKWING	\$ 110150
2.	AUDI RS7	\$ 114000
3.	Toyota Mirai 2021	\$ 125000
4.	G-Class SUV	\$ 131750
5.	BMW M8 Convertible	\$ 142500
6.	Tesla Model S Plaid	\$ 148990
7.	Mercedes-Maybach S-class	\$ 184900
8.	Porsche 911 turbo S	\$ 207000
9.	Nissan GTR	\$ 210740
10.	Range Rover SE P360	\$ 580000

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



No.	Name	Price
1.	AUDI RS7	\$ 114000
2.	BMW M8 Convertible	\$ 142500
3.	CT5-V BLACKWING	\$ 110150
4.	G-Class SUV	\$ 131750
5.	Mercedes-Maybach S-class	\$ 184900
6.	Nissan GTR	\$ 210740
7.	Porsche 911 turbo S	\$ 207000
8.	Range Rover SE P360	\$ 580000
9.	Tesla Model S Plaid	\$ 148990
10.	Toyota Mirai 2021	\$ 125000

Figure (2.3.24): the required SortByName from a smartphone

➤ 2.3.8: Request3: localhost:6500/ SortByPrice

The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal window displays the following log entries:

```

Date: Wed Dec 8 22:34:18 2021
("177.0.0.1", 62976)
GET /SortByPrice HTTP/1.1
Host: localhost:6500
Connection: keep-alive
sec-ch-ua: "Not A;Brand";v="99", "chromium";v="96", "Google Chrome";v="96"
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/96.0.4664.45 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=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6

["GET /SortByPrice HTTP/1.1", "Host: localhost:6500\r", "Connection: keep-alive\r", "sec-ch-ua: \"Not A;Brand\";v=\"99\", \"chromium\";v=\"96\", \"Google Chrome\";v=\"96\"\r", "sec-ch-ua-mobile: ?0\r", "sec-ch-ua-platform: \"Windows\"\r", "Upgrade-Insecure-Requests: 1\r", "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36\r", "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=0.9\r", "Sec-Fetch-Site: none\r", "Sec-Fetch-Mode: navigate\r", "Sec-Fetch-User: ?1\r", "Sec-Fetch-Dest: document\r", "Accept-Encoding: gzip, deflate, br\r", "Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ar;q=0.7,la;q=0.6\r"]
HTTP/1.1 200 OK
content-type: text/plain

Date: Wed Dec 8 22:34:28 2021

```

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.

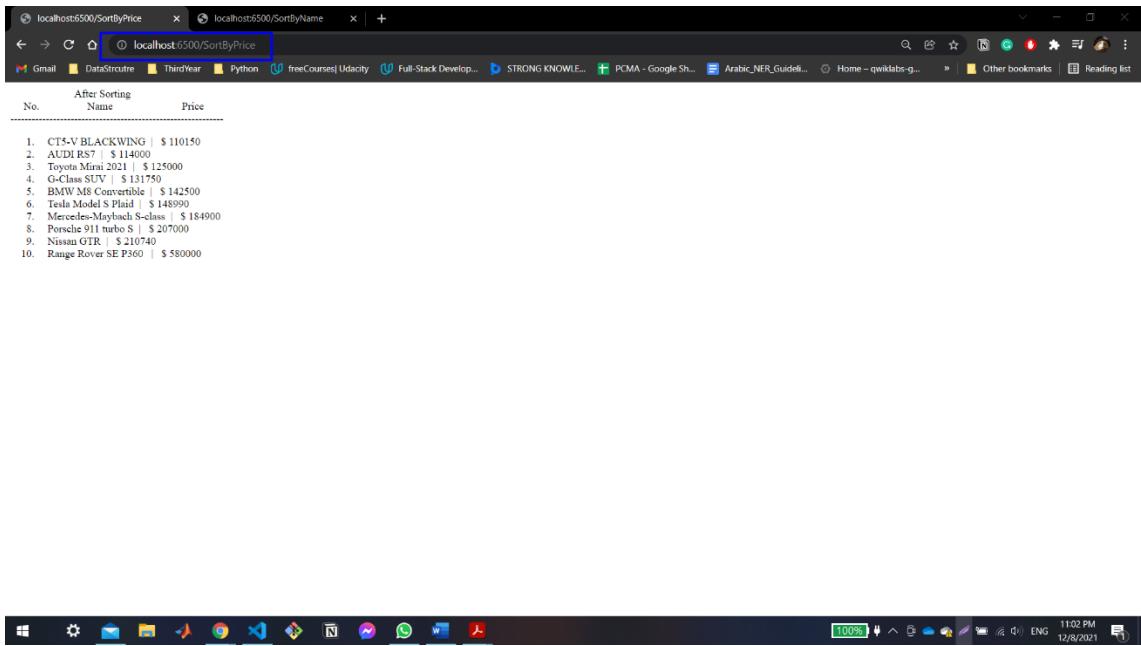


Figure (2.3.26): the result of the localhost:6500/SortByPrice

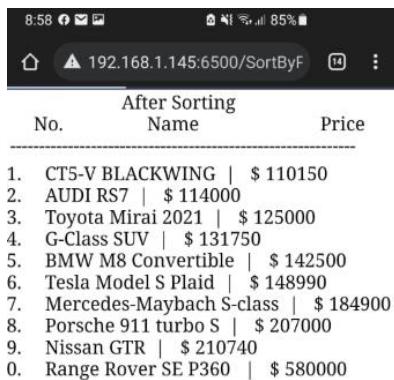


Figure (2.3.27): the required SortByPrice from a smartphone

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) :

```
1  <!DOCTYPE html>
2  <html>
3
4      <!-- ****-->
5      <!--The head part of the HTML Webpage...-->
6      <head>
7
8          <meta charset="utf-8">
9          <script src="https://use.fontawesome.com/d1341f9b7a.js"></script>
10         <link rel="stylesheet" href=" http://localhost:6500/file.css" #style.css http://localhost:6500/file.css
11         <!--Putting the icon of the page using the <link> tag-->
12         <link rel = "icon" href ="https://img.icons8.com/external-wanicon-flat-wanicon/2x/external-network-computer-hardware-wanicon-flat-wanico
13         <!--The title of the web page-->
14         <title>ENCS3320-Simple Webserver</title>
15     </head>
16     <!-- ****-->
17     <!--The body part of the HTML Webpage...-->
18     <body>
19         <header>
20             <!-- The main title of the Webpage-->
21             <h1>Computer Networks</h1>
22         </header>
23         <!-- the section tag is for defining the sections of the page such as chapters or footers etc.....-->
24
25         <section class="banner">
26             <!-- we gave it a class in order to style it later -->
27             <div class="WelcomingSent">
28                 <br><br><br>
29
30                 <!-- ****-->
31                 <!-------Welcoming Sentence----->
32                 <!-- ****-->
33                 <!-- Here is the welcoming message as required in the project-->
34                 <h2>Welcome To our course <br><span>Computer Networks</span></h2>
35             </div>
36         </section>
37
38     <!-- ****-->
39     <main>
40         <!--main content of the page which is the group members name and the IDs-->
41         <!--Some information about each students-->
42         <article>
43             <!-- ****-->
44             <!-------Students Information----->
45             <!-- ****-->
46
47             <!--The class box was given in order to style them like boxes using css language-->
48             <div class="box">
49
50                 <!-------Lojain Abdalrazaq's part----->
51                 <!--The student name-->
52                 <h3>Lojain Abdalrazaq-1190707</h3>
53                 <h4>CSE - Student</h4>
54                 <!--the information in the <p> tag-->
55                 <p>
56                     A third-year student at Birzeit University. Interested in fields of frontend, backend, and machine learning.
57                     I have completed a lot of projects on different topics, one of my favorite ones was about building a survey Form
58                     using HTML and CSS. I always insist on developing my skills in different subjects. I really enjoy volunteering
59                     , working with teams and meeting new people.
60                 </p>
61                 <ul>
62                     <!--List of Lojain's accounts(Github, Facebook and Linkedin) using <ul> tag-->
63                     <li><a href="https://github.com/Lojain-Abdalrazaq"><i class="fa fa-github" aria-hidden="true"></i></a></li>
64                     <li><a href="https://www.facebook.com/lojain.jalal"><i class="fa fa-facebook-square" aria-hidden="true"></i></a></li>
65                     <li><a href="http://linkedin.com/in/lojain-jalal-60a074206"><i class="fa fa-linkedin" aria-hidden="true"></i></a></li>
66                 </ul>
67             </div>
68
69             <!--The class box was given in order to style them like boxes using css language-->
70             <div class="box">
71                 <!-------Aseel Deek's part----->
72                 <!--The student name-->
73                 <h3>Aseel Deek-1190587</h3>
74                 <h4>CSE - Student</h4>
75                 <!--the information in the <p> tag-->
76             </div>
```

```

75    <!--the information in the <p> tag-->
76    <p>
77      A third-year student at Birzeit University. Interested in computer studies, specifically deep learning, big data, AI,
78      and data analysis. Done a lot of projects on different subjects. One of them was a web page
79      for a restaurant called (David Chu's China bistro) using HTML, CSS, and Javascript languages.
80      I enjoy drawing, learning new things, reading books, and playing with cats.
81    </p>
82    <ul>
83      <!--List of Aseel's accounts(Github, Facebook and Linkedin) using <ul> tag-->
84      <li><a href="https://github.com/aseeldeek"><i class="fa fa-github" aria-hidden="true"></i></a></li>
85      <li><a href="https://www.facebook.com/aseel.aldeek.1806"><i class="fa fa-facebook-square" aria-hidden="true"></i></a></li>
86      <li><a href="http://linkedin.com/in/aseel-deek-a106201b4"><i class="fa fa-linkedin" aria-hidden="true"></i></a></li>
87    </ul>
88  </div>
89
90  <!--The class box was given in order to style them like boxes using css language-->
91  <div class="box">
92    <!-------Mariam Taweel's part----->
93    <!--The student name-->
94    <h3>Mariam Taweel-1192099</h3>
95    <h4>CSE - Student</h4>
96    <!--The information in the <p> tag-->
97    <p>
98      I am a student at Birzeit University in my third year. I worked on many projects during my first years at university in various
99      and languages such as (C, Java, and C#), and one of the best applications I made was an application for a car company.
100     I worked on many projects during my first years at university.
101     I love listening to music and recordings of books on self-development.
102   </p>
103   <ul>
104     <!--List of Mariam's accounts(Github, Facebook and Linkedin) using <ul> tag-->
105     <li><a href="https://github.com/maryamaltawil4"><i class="fa fa-github" aria-hidden="true"></i></a></li>
106     <li><a href="https://www.facebook.com/profile.php?id=100003974872539"><i class="fa fa-facebook-square" aria-hidden="true"></i></a></li>
107     <li><a href="http://linkedin.com/in/maryam-altawil-467900225"><i class="fa fa-linkedin" aria-hidden="true"></i></a></li>
108   </ul>
109 </div>
110 </article>
111 <!--*****-->
112 <!-------The JPG and PNG images----->
113 <!--*****-->
114 <!--*****-->
115   <ul class="imgs">
116     <!--The List of two images using the <ul> tag-->
117     <!--JPG Extension-->
118     <li></li>
119     <!--PNG Extension-->
120     <li></li>
121   </ul>
122 </main>
123
124 <!--*****-->
125 <!-------The Links (Webpage & HTML File)----->
126 <!--*****-->
127
128 <section>
129   <!--Link to https://www.w3schools.com/tags/att_img_src.asp -->
130   <p class="linked"> For more information about HTML Language, Please visit this <a href="https://www.w3schools.com/tags/att_img_src.asp">
131   <!--Link to a local HTML File-->
132   <p class="linked"> And Here is the linked <a href="http://localhost:6500/linked.html"> HTML File </a> </p>
133 </section>
134 <!--The footer part....-->
135 <footer> &copy; all rights reserved 2021 </footer>
136
137 </body>
138
139 </html>
140 <!--The HTML part of the Webpage ends here....-->

```

➤ 5.2: part3: CSS Code (style.css) :

```
1  ****Body Styling*****
2 body{
3     font-family: 'Lato', 'Lucida Grande', 'Lucida Sans Unicode', Tahoma, Sans-Serif;
4     color: white;
5     padding: 0;
6     background: url(https://images.pexels.com/photos/1631677/pexels-photo-1631677.jpeg?cs=srgb&dl=pexels-abdullah-ghatasheh-1631677.jpg&fm=jpg);
7     overflow-y: scroll;
8     top: 0;
9     left: 0;
10    bottom: 0;
11    right: 0;
12    text-align: center;
13 }
14 ****Header Styling*****
15 header{
16     background-color: #af3c4d;
17     color: white;
18     position:fixed;
19     text-align: center;
20     width: 100%;
21     z-index: 1;
22     top: 0px;
23     left: 0px;
24 }
25
26 header h1{
27     border: 3px solid white;
28     border-radius: 15px;
29     padding: 3px 3px;
30     display: inline-block;
31     font-size: 15px;
32 }
33 ****Boxes Styling*****
34 .box{
35     width: 350px;
36     background: #af3c4d;
37     padding: 25px;
38     text-align: center;
39     margin-left: 10px;
40     margin-top: 10px;
41     margin-bottom: 2px;
42     color: white;
43     display: inline-block;
44     box-shadow: 0px -1px 6px 6px white;
45     border-radius: 15px;
46     margin: 40px 35px;
47     max-width: 30%;
48 }
49
50 .box p{
51     text-align: justify;
52     line-height: 1.5em;
53 }
54
55 .box li{
56     display: inline-block;
57     margin: 6px;
58     list-style: none;
59 }
60
61 .box li a{
62     color: white;
63     text-decoration: none;
64     font-size: 50px;
65     transition: all ease-in-out 250ms;
66 }
67
68 .box h3{
69     font-size: 20px;
70     letter-spacing: 3px;
71     font-weight: 100;
72 }
73
74 ul{
```

```

74     ul{
75         margin: 0;
76         padding: 0;
77     }
78
79     .box h4{
80         font-size: 18px;
81         letter-spacing: 3px;
82         font-weight: 100;
83     }
84
85     .box li a:hover{
86         color:#e09163;
87     }
88     /*****The welcoming Sentence Styling*****/
89     .WelcomingSent{
90
91         padding: 4px;
92         position: sticky;
93         top: 20px;
94         color:rgba(31, 30, 30, 0.795);
95     }
96
97
98     .banner h2 span{
99
100        /* 1.4 times of the previous size */
101        font-size: 1.5em;
102        color:#0757ad;
103        text-shadow: 1px 1px 2px black, 0 0 25px blue, 0 0 5px white;
104    }
105
106    /*****Images Styling*****/
107    .imgs{
108        text-align: center;
109        white-space: nowrap;
110        margin: 80px 0;
111    }
112
113    .imgs li{
114        display: inline-block;
115        width: 40%;
116        margin: 20px 5%;
117    }
118
119    .imgs li img{
120        max-width: 100%;
121    }
122    /*****Site & HTML File Styling*****/
123    .linked a {
124        border:3px solid white;
125        border-radius: 15px;
126        background-color: #af3c4d;
127        color:white;
128        text-decoration: none;
129        padding: 3px 3px 3px 3px;
130    }
131
132    /*****Media Styling*****/
133    @media screen and (max-width:1000px){
134
135        .WelcomingSent h2{
136            font-size:15px;
137        }
138        .box p{
139            font-size:15px;
140        }
141        .box h3{
142            font-size:15px;
143        }
144        .box h4{
145            font-size:15px;
146        }
147        .linked p{
148            font-size: 10px;
149        }
150    }
151
152    @media screen and (max-width:700px) {
153
154        .WelcomingSent h2{
155            font-size:13px;
156        }
157        .box p{
158            font-size:12px;
159        }
160        .box h3{
161            font-size:13px;
162        }
163        .box h4{
164            font-size:12px;
165        }
166        .linked p{
167            font-size: 5px;
168        }

```

➤ 5.3: part3: Python Code (Server.py):

```
1  from socket import *
2  import csv ,operator # for reading csv file
3  import datetime # for displaying the time and date
4  # ! NOTE : This Project Was Build Using The Last Version Of Python 3.10.0
5  # function to deal with wrong requests and
6  def not_found(file):
7      print("this is the not_found function")
8
9      content = '<!DOCTYPE html><html><head><title>Error</title> </head><body><h1><span style="color:red;">The file is not found </span></h1><p id="'
10     connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
11     print("HTTP/1.1 200 OK \r\n")
12     connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
13     print("Content-Type: text/html\r\n")
14     connectionSocket.send(bytes("\r\n", "UTF-8"))
15     print("\r\n")
16     connectionSocket.sendall(bytes(content, "UTF-8"))
17     x = datetime.datetime.now()
18     print("Date:", x.strftime("%c")) # showing the date
19
20 # end of the function
21
22 # function for reading the csv file
23 def printTohtml(Alist, htmlfile):
24     html = "<html>\n<head> &emsp;&emsp;&emsp;&emsp;&emsp;&emsp;After Sorting</head>\n<style>p { margin: 0 !important; }</style>\n<body>\n"
25
26     title = "&nbsp; No. &nbsp;&nbsp;&nbsp;&nbsp; Name &nbsp;&nbsp;&nbsp;&nbsp; Price "
27     deco= "-----\n"
28
29     html += '\n<p>' + title + '</p>\n' + deco
30
31     strat = '<ol>\n'
32     html +=strat
33     for line in Alist:
34         para = '<li>&nbsp;' + '&nbsp; $'.join(line) + '</li>\n'
35         html += para
36     end ='</ol>'
37     html +=end
38
39     with open(htmlfile, 'w') as f:
40         f.write(html + "\n</body>\n</html>")
41     # end of reading csv file
42
43 # the start of the project
44 serverPort = 6500
45 serverSocket = socket(AF_INET, SOCK_STREAM)
46 serverSocket.bind(("", serverPort))
47 serverSocket.listen(1)
48 print("the server is ready to receive")
49 while True:
50     connectionSocket, address = serverSocket.accept()
51     sentence = connectionSocket.recv(2048).decode()
52     print(address)
53     print(sentence)
54     ip = address[0]
55     port = address[1]
56     headers = sentence.split('\n')
57     print(headers)
58     file = headers[0].split()[1] # index.html
59
60     match file:
61         case '/' | '/index.html' : # when requesting / or /index.html
62             file = 'main.html'
63             fin = open(file)
64             content = fin.read()
65             fin.close()
66             connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
67             print("HTTP/1.1 200 OK \r\n")
68             connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
69             print("Content-Type: text/html\r\n")
70             connectionSocket.send(bytes("\r\n", "UTF-8"))
71             print("\r\n")
72             connectionSocket.sendall(bytes(content, "UTF-8"))
73             x = datetime.datetime.now() # to print the date and time
74             print(x.strftime("%c")) # print the date and time in the terminal
75
```

```

76
77     case '/Network1.jpg' : # when requesting /Network1.jpg
78         fin = open("images/FirstPic.jpg", "rb")
79         connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
80         print("HTTP/1.1 200 OK \r\n")
81         connectionSocket.send(bytes("Content-Type: image/jpg\r\n\r\n\r\n", "UTF-8"))
82         print("Content-Type: image/jpg\r\n\r\n\r\n")
83         connectionSocket.send(fin.read())
84         print(str(fin.read()))
85         x = datetime.datetime.now() # to print the date and time
86         print("Date:", x.strftime("%c")) # print the date and time in the terminal
87
88     case '/other.html' : # when requesting /other.html
89         file = 'other.html'
90         fin = open(file)
91         content = fin.read()
92         fin.close()
93         connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
94         print("HTTP/1.1 200 OK \r\n")
95         connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
96         print("Content-Type: text/html\r\n")
97         connectionSocket.send(bytes("\r\n", "UTF-8"))
98         print("\r\n")
99         connectionSocket.sendall(bytes(content, "UTF-8"))
100        x = datetime.datetime.now() # to print the date and time
101        print("Date:", x.strftime("%c"))# print the date and time in the terminal
102
103    case '/Network1.png' : # when requesting /Network1.png
104        fin = open("images/SecPic.png", "rb")
105        connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
106        print("HTTP/1.1 200 OK \r\n")
107        connectionSocket.send(bytes("Content-Type: image/png\r\n\r\n\r\n", "UTF-8"))
108        print("Content-Type: image/png\r\n\r\n\r\n")
109        connectionSocket.send(fin.read()) # send the contents of the picture
110        print(str(fin.read()))
111        x = datetime.datetime.now() # to print the date and time
112        print("Date:", x.strftime("%c")) # print the date and time in the terminal
113
114    case '/file.css' : # when requesting /file.css
115        file = 'style.css'
116        fin = open(file)
117        content = fin.read()
118        fin.close()
119        connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
120        print("HTTP/1.1 200 OK \r\n")
121        connectionSocket.send(bytes("Content-Type: text/css\r\n", "UTF-8"))
122        print("Content-Type: text/css\r\n")
123        connectionSocket.send(bytes("\r\n", "UTF-8"))
124        print("\r\n")
125        connectionSocket.sendall(bytes(content, "UTF-8"))
126        x = datetime.datetime.now() # to print the date and time
127        print("Date:", x.strftime("%c")) # print the date and time in the terminal
128
129    case '/SortByName': # when requesting /SortByName
130        sample = open('Cars.csv','r')
131        csv1 = csv.reader(sample, delimiter=',')
132        sorted_prices = sorted(csv1,key=operator.itemgetter(0))
133        printTohtml(sorted_prices, 'SortedData.html')
134        file = 'SortedData.html'
135        fin = open(file)
136        content = fin.read()
137        fin.close()
138        connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
139        print("HTTP/1.1 200 OK \r\n")
140        connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8")) # should be always html
141        print("Content-Type: text/plain\r\n")
142        connectionSocket.send(bytes("\r\n", "UTF-8"))
143        print("\r\n")
144        connectionSocket.sendall(bytes(content, "UTF-8"))
145        x = datetime.datetime.now() # to print the date and time
146        print("Date:", x.strftime("%c")) # print the date and time in the terminal
147
148    case '/SortByPrice': # when requesting /SortByPrice
149        sample = open('Cars.csv','r')
150        csv1 = csv.reader(sample, delimiter=',')

```

```

149     csv1 = csv.reader(sample, delimiter=',')
150     sorted_prices = sorted(csv1, key=operator.itemgetter(1))
151     printToHtml(sorted_prices, 'SortedData.html')
152     file = 'Sorteddata.html'
153     fin = open(file)
154     content = fin.read()
155     fin.close()
156     connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
157     print("HTTP/1.1 200 OK \r\n")
158     connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8")) # should be always html
159     print("Content-Type: text/plain\r\n")
160     connectionSocket.send(bytes("\r\n", "UTF-8"))
161     print("\r\n")
162     connectionSocket.sendall(bytes(content, "UTF-8"))
163     x = datetime.datetime.now() # to print the date and time
164     print("Date:", x.strftime("%c")) # print the date and time in the terminal
165
166 case'linked.html':
167     file = 'linked.html'
168     fin = open(file)
169     content = fin.read()
170     fin.close()
171     connectionSocket.send(bytes("HTTP/1.1 200 OK \r\n", "UTF-8"))
172     print("HTTP/1.1 200 OK \r\n")
173     connectionSocket.send(bytes("Content-Type: text/html\r\n", "UTF-8"))
174     print("Content-Type: text/html\r\n")
175     connectionSocket.send(bytes("\r\n", "UTF-8"))
176     print("\r\n")
177     connectionSocket.sendall(bytes(content, "UTF-8"))
178
179
180 case unknown_command: # when requesting Wrong file
181     connectionSocket.send(bytes("HTTP/1.1 404 Not Found \r\n", "UTF-8"))
182     print("HTTP/1.1 404 Not Found \r\n")
183     not_found(file)
184
185
186

```

➤ 5.4: part2: Python Code (Question2.py) :

⌚ Question2.py > ...

```

1  from socket import *
2  from datetime import *
3  Server_name = input("Please Enter the site name:\n")
4  #made a socket instance and passed it two parameters. The first parameter is AF_INET and the second one is SOCK_STREAM.
5  Client = socket(AF_INET , SOCK_STREAM)
6  # default port for socket
7  prot_Server = 80
8  # connecting to the server
9  Client.connect((Server_name , prot_Server))
10 #time before sending request
11 print (datetime.now())
12 # send a HEAD / HTTP/1.1 to the client. encoding to send byte type.
13 Client.send("HEAD / HTTP/1.1 \r\n".encode())
14 Client.send("Hostname: {name_Server} \r\n\r\n".encode())
15 #time response befor the response
16 time=datetime.now()
17 modified_sentence = Client.recv(1024)
18 #time response after the response
19 time_next=datetime.now()
20 finaltime=time_next - time
21 print("response time:",finaltime)
22 # receive data from the server
23 print("From server :)", modified_sentence.decode())
24 # close the connection with the client
25 Client.close()

```