Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS 3320

Computer Networks

Project 1.

Simple Web Server.

Prepared By:

Name: Baraa Atta.   ID: 1180445

Name: Ahmad Hamad. ID: 1180060

Instructor: Dr. Abdalkarim Awad

Section: 1

Date: 30/4/2021

Abstract

This project consists of two parts each one of them contains the following

1- Part 1: in this part we are going to try running some network commands which are ping, tracert, and nslookup, also we will explain the output of each one of them in details.
2- Part 2: in this part we will implement a simple but complete web server using python TCP socket programming, that will provide some functions defined by their paths, which are the follows:
    - Main index page (path: / or /index.html)
    - PNG image (path: /imagename.png)
    - JPEG image (path:/ imagename.jpg)
    - Sort a pre-defined file by name (path: /sortName)
    - Sort a pre-defined file by price (path: /sortPrice)
    - Error page for undefined paths (path: / any undefined path)

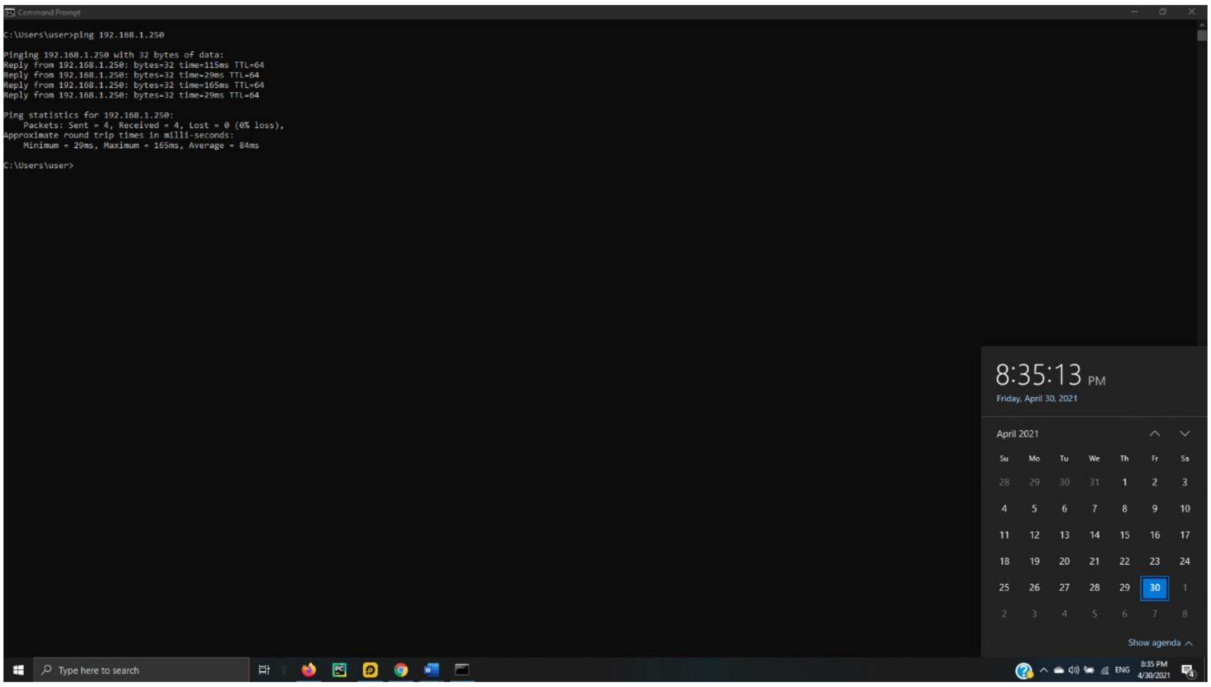
Finally, we will give our own conclusion about the whole experiment.

Contents

Figures

Part 1

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



```
C:\Users\user>ping 192.168.1.250

Pinging 192.168.1.250 with 32 bytes of data:
Reply from 192.168.1.250: bytes=32 time=115ms TTL=64
Reply from 192.168.1.250: bytes=32 time=29ms TTL=64
Reply from 192.168.1.250: bytes=32 time=165ms TTL=64
Reply from 192.168.1.250: bytes=32 time=29ms TTL=64

Ping statistics for 192.168.1.250:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 29ms, Maximum = 165ms, Average = 84ms

C:\Users\user>
```

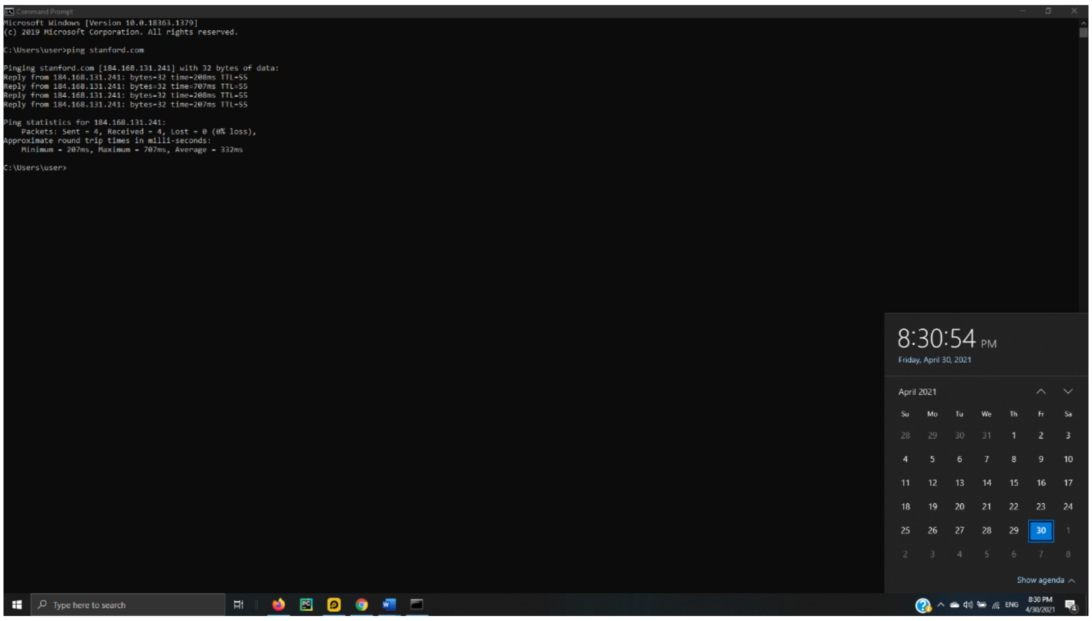
Figure 1: pinging a device in the same network

As we can see from figure 1, we tried to ping my own smart phone by its ip address by sending four 32-bit packets and waiting for them it to return back. In the next lines we can see the response from Stanford's server, which contain 3 main attributes as follows:

1- bytes: which indicates the size of the sent packet.
2- Time: which represents the total time taking to send the packet and receive it back.
3- TTL: which stands for time to live; time to live is the maximum number of hobs the message can pass through before returning back. Note that time to live is usually used to prevent the packet from entering an infinite loop between two or more hops.

Finally, we can see a brief summary of the hall operation, with the total number of sent packets and the total number of acknowledgments received; to find out if there is a loss or not, also we can see the maximum and the minimum response time at all sent packets, and the average response time.

ping www.stanford.edu



```
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>ping stanford.com

Pinging stanford.com [184.168.131.241] with 32 bytes of data:
Reply from 184.168.131.241: bytes=32 time=208ms TTL=55
Reply from 184.168.131.241: bytes=32 time=707ms TTL=55
Reply from 184.168.131.241: bytes=32 time=208ms TTL=55
Reply from 184.168.131.241: bytes=32 time=207ms TTL=55

Ping statistics for 184.168.131.241:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 207ms, Maximum = 707ms, Average = 332ms

C:\Users\user>
```

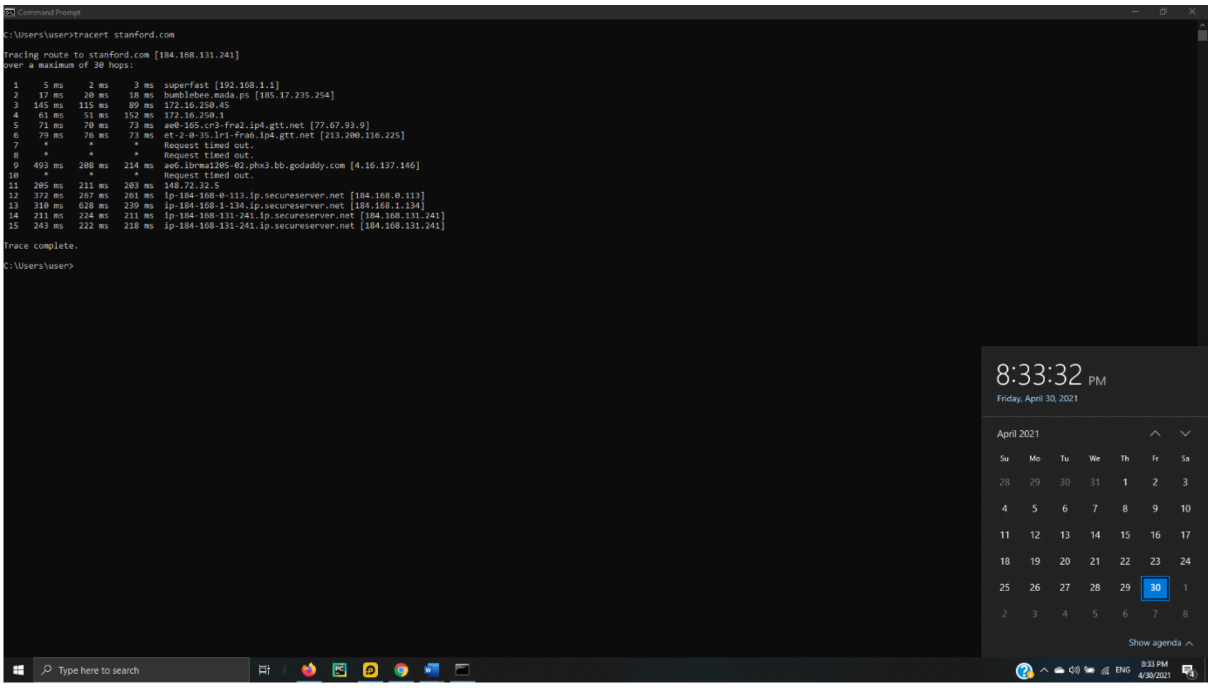
Figure 2: pinging Stanford university server

As we can see from figure 2, we tried to ping Stanford university server by sending four 32-bit packets and waiting for them it to return back. In the next lines we can see the response from Stanford's server, which contain 3 main attributes as follows:

4- bytes: which indicates the size of the sent packet.
5- Time: which represents the total time taking to send the packet and receive it back.
6- TTL: which stands for time to live; time to live is the maximum number of hobs the message can pass through during its trip. Note that time to live is usually used to prevent the packet from entering an infinite loop between two or more hops.

Finally, we can see a brief summary of the hall operation, with the total number of sent packets and the total number of acknowledgments to find out if there is a loss or not, also we can see the maximum and the minimum response time in all 4 sent packets, and also the average response time.

tracert www.stanford.edu

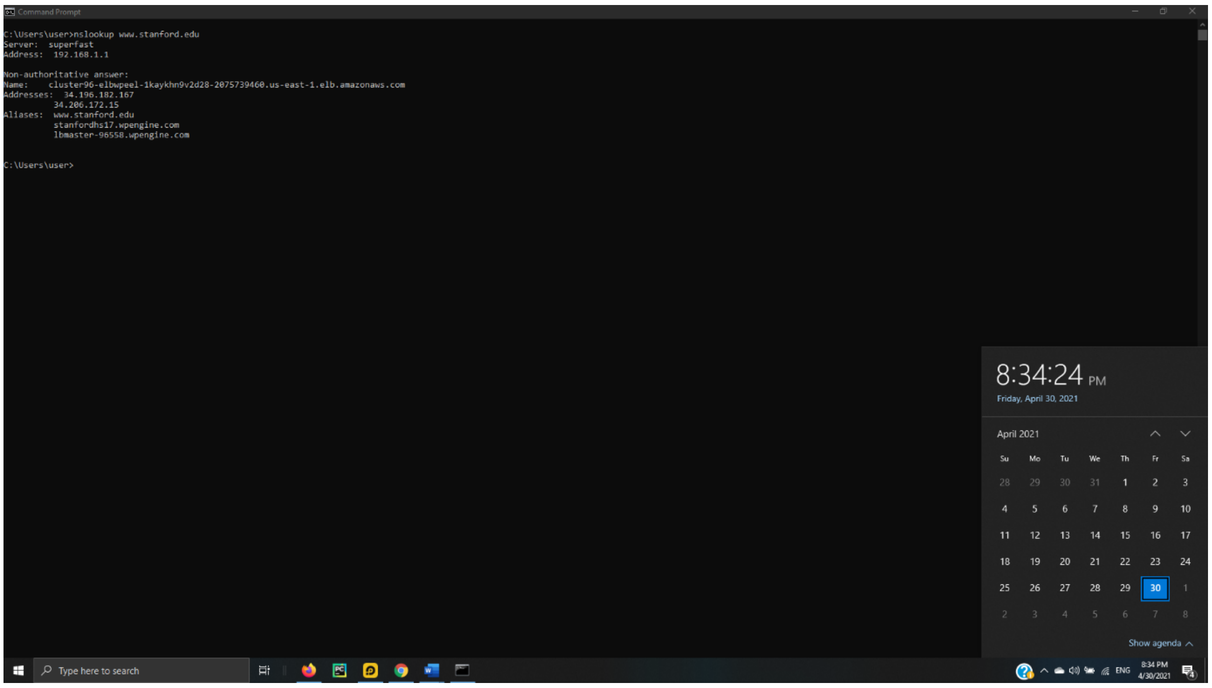


```
C:\Users\user>tracert stanford.com

Tracing route to stanford.com [184.168.131.241]
over a maximum of 30 hops:

  1     5 ms     2 ms     3 ms  superfast [192.168.1.1]
  2    17 ms    20 ms    18 ms  bumblebee.mada.ps [185.17.235.254]
  3   145 ms   115 ms    89 ms  172.16.250.45
  4    61 ms    51 ms   152 ms  172.16.250.1
  5    71 ms    70 ms    73 ms  ae0-165.cr3-fra2.ip4.gtt.net [77.67.93.9]
  6    79 ms    76 ms    73 ms  et-2-0-35.lr1-fra6.ip4.gtt.net [213.200.116.225]
  7     *        *        *     Request timed out.
  8     *        *        *     Request timed out.
  9   493 ms   208 ms   214 ms  ae6.ibrma1205-02.phx3.bb.godaddy.com [4.16.137.146]
 10     *        *        *     Request timed out.
 11   205 ms   211 ms   203 ms  148.72.32.5
 12   372 ms   267 ms   261 ms  ip-184-168-0-113.ip.secureserver.net [184.168.0.113]
 13   310 ms   628 ms   239 ms  ip-184-168-1-134.ip.secureserver.net [184.168.1.134]
 14   211 ms   224 ms   211 ms  ip-184-168-131-241.ip.secureserver.net [184.168.131.241]
 15   243 ms   222 ms   218 ms  ip-184-168-131-241.ip.secureserver.net [184.168.131.241]

Trace complete.

C:\Users\user>
```

Figure 3: tracing the route to Stanford university server

In figure 3, we can see that trace rout command, which tests the network through a specific sever, but in a more details compared to the ping, since it shows as each router the packet passes through before reaching the destination sever, for example if we look at the results we got, we can see that we send three packets to each route and measure the total response time for each one of them. The first line shows the first router the packets pass through which is the home router since the time taken is very low, next they move to the ISP server resulting in an increase on the response time, after that the packets keep travelling through the routers with increasing response times (due to the increasing distance), until they reach the final destination router. Note that there is a router with a time out response, that’s mostly due to some configurations on that router, that allow it to block the trace rout packets, due to security reasons most of the time.

nslookup www.stanford.edu



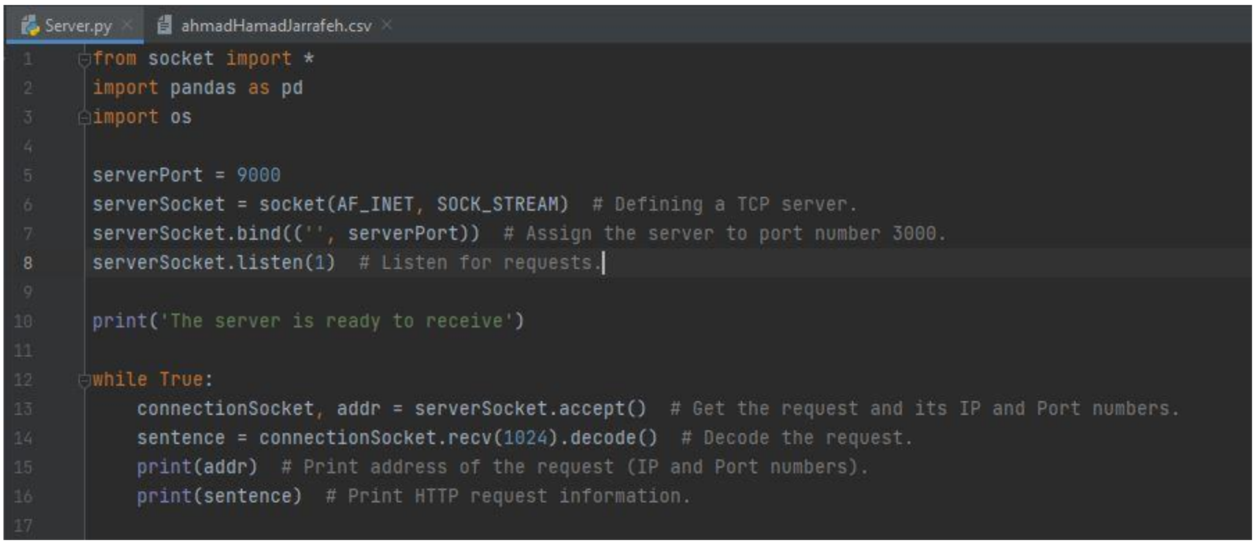
```
C:\Users\user>nslookup www.stanford.edu
Server:  superfast
Address:  192.168.1.1

Non-authoritative answer:
Name:    cluster96-elbwpeel-1kaykhn9v2d28-2075739460.us-east-1.elb.amazonaws.com
Addresses:  34.196.182.167
          34.206.172.15
Aliases:  www.stanford.edu
          stanfordhs17.wpengine.com
          lbmaster-96558.wpengine.com


C:\Users\user>
```

Figure 4: asking the DNS server about the IP address of Stanford university server

In figure 4 we can see the result after applying the nslookup command, which asks the DNS server for the IP address for the corresponding domain name. When we ask the DNS server about the IP address for the domain name standford.edu, it returns the IP addresses 34.196.182.176 and 32.206.172.15, also it returns the aliasing names for the same domain name like stanfordhs17.wpenine.com.

Part 2

Part one of the code is shown in the following figure.



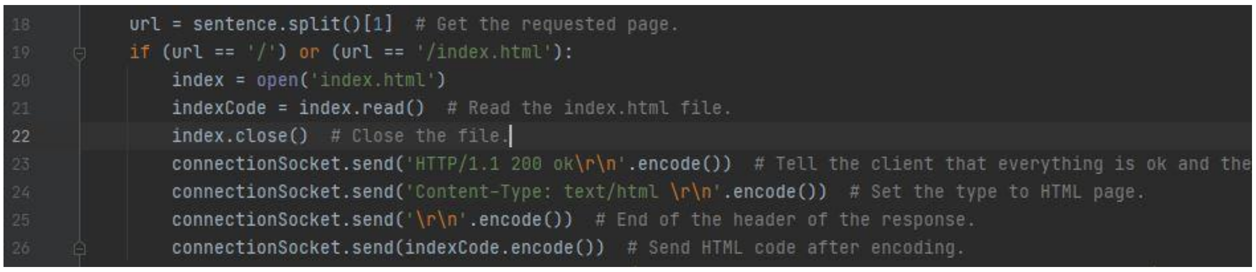
Server.py ahmadHamadJarrafeh.csv

```
1  from socket import *
2  import pandas as pd
3  import os
4
5  serverPort = 9000
6  serverSocket = socket(AF_INET, SOCK_STREAM)  # Defining a TCP server.
7  serverSocket.bind(('', serverPort))  # Assign the server to port number 3000.
8  serverSocket.listen(1)  # Listen for requests.
9
10 print('The server is ready to receive')
11
12 while True:
13     connectionSocket, addr = serverSocket.accept()  # Get the request and its IP and Port numbers.
14     sentence = connectionSocket.recv(1024).decode()  # Decode the request.
15     print(addr)  # Print address of the request (IP and Port numbers).
16     print(sentence)  # Print HTTP request information.
17
```

Figure 5: Part 1 of Server's Code Creating the server)

At first the port of the server is defined to be 9000 then at line 6 an internet server that uses TCP connection is defined and at line 7 the port 9000 is assigned to the created server. Then at line 8, the server starts listening for incoming requests.

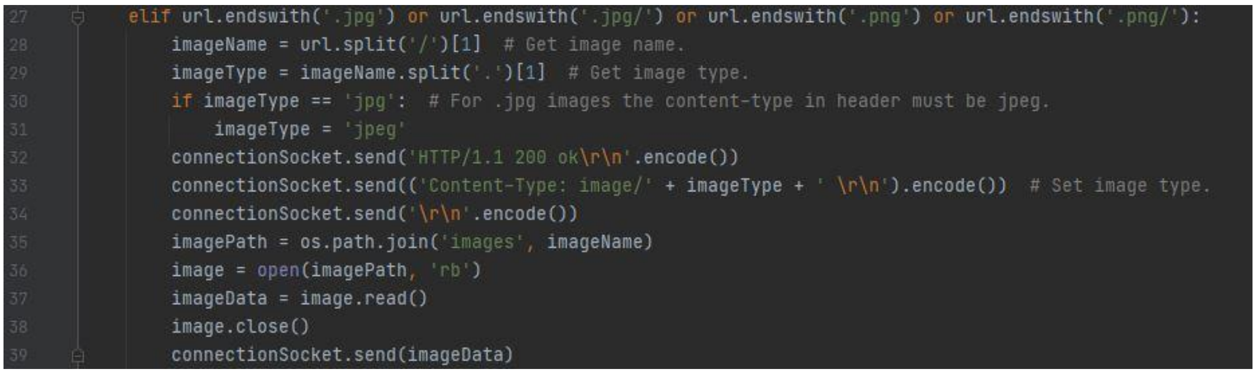
At line 12, the while handles every incoming request, at first the client socket and his address are accepted by the server, then the HTTP request of the client decoded and stored in a variable called **sentence**. Next two lines print to the console the IP and the port numbers of the client as well as the HTPP request. Part 2 of the code is shown in the next figure.



```
18     url = sentence.split()[1]  # Get the requested page.
19     if (url == '/') or (url == '/index.html'):
20         index = open('index.html')
21         indexCode = index.read()  # Read the index.html file.
22         index.close()  # Close the file.
23         connectionSocket.send('HTTP/1.1 200 ok\r\n'.encode())  # Tell the client that everything is ok and the
24         connectionSocket.send('Content-Type: text/html \r\n'.encode())  # Set the type to HTML page.
25         connectionSocket.send('\r\n'.encode())  # End of the header of the response.
26         connectionSocket.send(indexCode.encode())  # Send HTML code after encoding.
```

Figure 6: Part 2 of Server's Code (index.html)

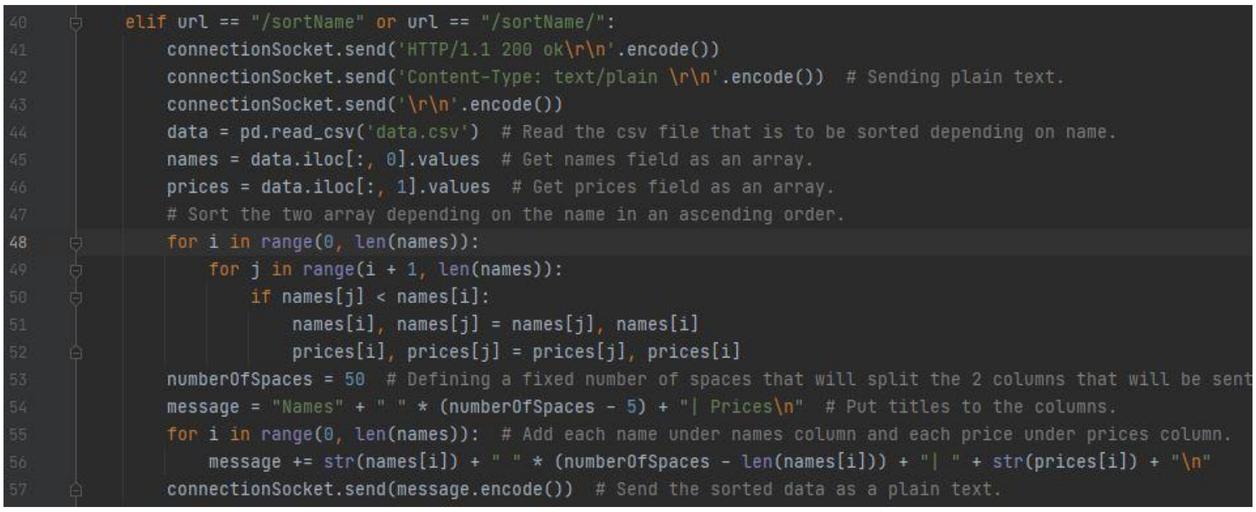
Line 18 extracts the requested URL. Then, the URL is checked. If the URL is / or /index.html then the main page is sent to the client, at first the index.html file is read then the response sends the header to the client which consists of that the HTTP/1.1 request is done successfully (line 23) and the content is an HTML code (line 24) then it sends an indicator for the end of header (line 25). Finally, the data which is the HTML code is sent (line 26). Part 3 of the code is shown in the next figure.



```python
        elif url.endswith('.jpg') or url.endswith('.jpg/') or url.endswith('.png') or url.endswith('.png/'):
            imageName = url.split('/')[1]  # Get image name.
            imageType = imageName.split('.')[1]  # Get image type.
            if imageType == 'jpg':  # For .jpg images the content-type in header must be jpeg.
                imageType = 'jpeg'
            connectionSocket.send('HTTP/1.1 200 ok\r\n'.encode())
            connectionSocket.send(('Content-Type: image/' + imageType + ' \r\n').encode())  # Set image type.
            connectionSocket.send('\r\n'.encode())
            imagePath = os.path.join('images', imageName)
            image = open(imagePath, 'rb')
            imageData = image.read()
            image.close()
            connectionSocket.send(imageData)
```

Figure 7: Part 3 of Server's Code (.jpg & .png Images)

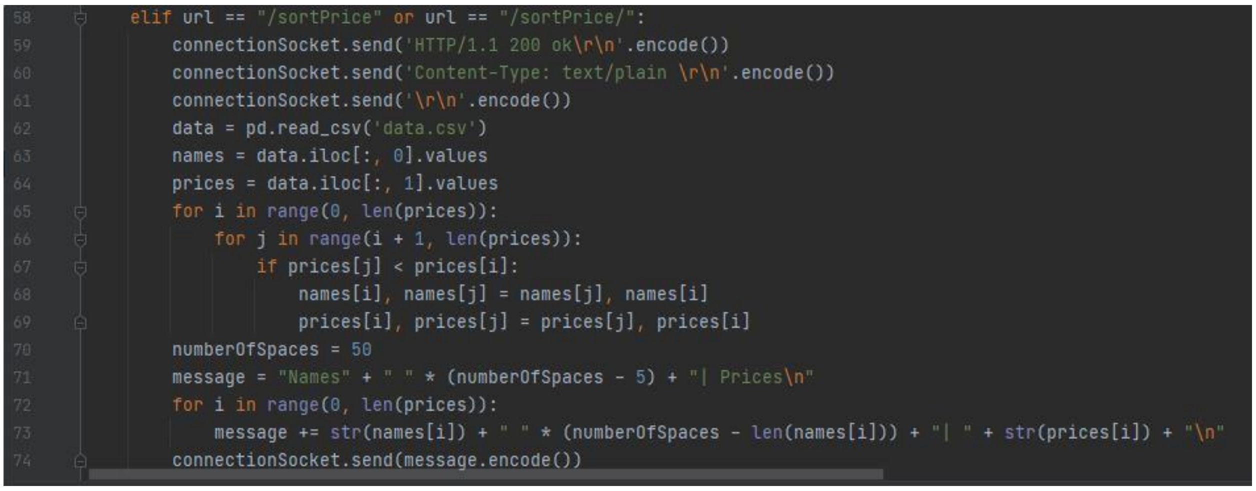
If the requested page is not the index.html file then the server checks if it's an image with either .jpg or .png extensions. At first, the name of the image is extracted from the URL then the type of image is extracted too. Next, the type is checked if it was jpg then the type variable is set to jpeg as the response should contain jpeg as a type for jpg images. Then the response is sent the same way as before, sending the header ➔ ok message with status code 200 and the content type as image/ (imageType variable to work for both extensions). Then the image is read from the images file as all images are stored in a folder called images. Finally, the image is sent to the client. Part 4 of the code is shown in the next figure.



```python
        elif url == "/sortName" or url == "/sortName/":
            connectionSocket.send('HTTP/1.1 200 ok\r\n'.encode())
            connectionSocket.send('Content-Type: text/plain \r\n'.encode())  # Sending plain text.
            connectionSocket.send('\r\n'.encode())
            data = pd.read_csv('data.csv')  # Read the csv file that is to be sorted depending on name.
            names = data.iloc[:, 0].values  # Get names field as an array.
            prices = data.iloc[:, 1].values  # Get prices field as an array.
            # Sort the two array depending on the name in an ascending order.
            for i in range(0, len(names)):
                for j in range(i + 1, len(names)):
                    if names[j] < names[i]:
                        names[i], names[j] = names[j], names[i]
                        prices[i], prices[j] = prices[j], prices[i]
            numberOfSpaces = 50  # Defining a fixed number of spaces that will split the 2 columns that will be sent
            message = "Names" + " " * (numberOfSpaces - 5) + "| Prices\n"  # Put titles to the columns.
            for i in range(0, len(names)):  # Add each name under names column and each price under prices column.
                message += str(names[i]) + " " * (numberOfSpaces - len(names[i])) + "| " + str(prices[i]) + "\n"
            connectionSocket.send(message.encode())  # Send the sorted data as a plain text.
```

Figure 8: Part 4 of Server's Code (/sortPrice)

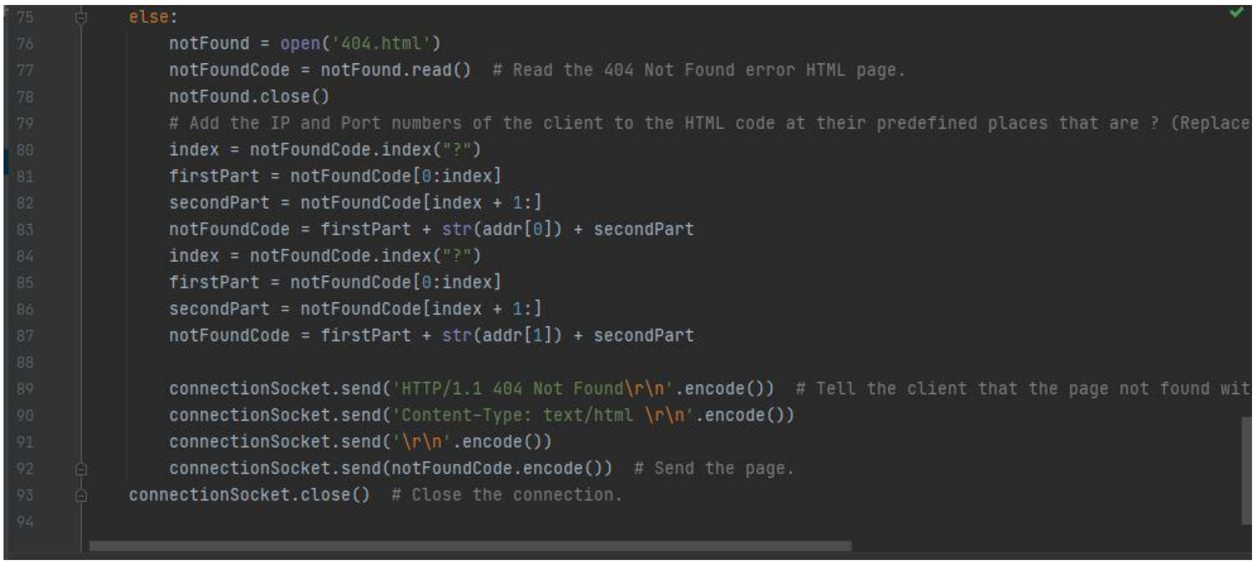
Here the server checks if the requested page is /sortName, in this case a csv file is read and sorted depending on the name field in an ascending order and sent to the client as a plain text. At first, the header is sent to the client with a status code 200 and content type as plain text. Then, the data.csv file is read and converted to two distinct arrays, one array is for the name field and the other for price field, then the two arrays are sorted in an ascending order using bubble sort depending on the name. Then, a string is created to store the sorted arrays and send them to the client. Part 5 is exactly the same as this one except that the sort is done depending on the price, the code is in the following figure.



```
    elif url == "/sortPrice" or url == "/sortPrice/":
        connectionSocket.send('HTTP/1.1 200 ok\r\n'.encode())
        connectionSocket.send('Content-Type: text/plain \r\n'.encode())
        connectionSocket.send('\r\n'.encode())
        data = pd.read_csv('data.csv')
        names = data.iloc[:, 0].values
        prices = data.iloc[:, 1].values
        for i in range(0, len(prices)):
            for j in range(i + 1, len(prices)):
                if prices[j] < prices[i]:
                    names[i], names[j] = names[j], names[i]
                    prices[i], prices[j] = prices[j], prices[i]
        numberOfSpaces = 50
        message = "Names" + " " * (numberOfSpaces - 5) + "| Prices\n"
        for i in range(0, len(prices)):
            message += str(names[i]) + " " * (numberOfSpaces - len(names[i])) + "| " + str(prices[i]) + "\n"
        connectionSocket.send(message.encode())
```

Figure 9: Part 5 of Server's Code (/sortPrice)

Final part of the code which is responsible for sending a 404 page not found is shown in the following figure.



```
    else:
        notFound = open('404.html')
        notFoundCode = notFound.read()  # Read the 404 Not Found error HTML page.
        notFound.close()
        # Add the IP and Port numbers of the client to the HTML code at their predefined places that are ? (Replace
        index = notFoundCode.index("?")
        firstPart = notFoundCode[0:index]
        secondPart = notFoundCode[index + 1:]
        notFoundCode = firstPart + str(addr[0]) + secondPart
        index = notFoundCode.index("?")
        firstPart = notFoundCode[0:index]
        secondPart = notFoundCode[index + 1:]
        notFoundCode = firstPart + str(addr[1]) + secondPart

        connectionSocket.send('HTTP/1.1 404 Not Found\r\n'.encode())  # Tell the client that the page not found wit
        connectionSocket.send('Content-Type: text/html \r\n'.encode())
        connectionSocket.send('\r\n'.encode())
        connectionSocket.send(notFoundCode.encode())  # Send the page.
    connectionSocket.close()  # Close the connection.
```

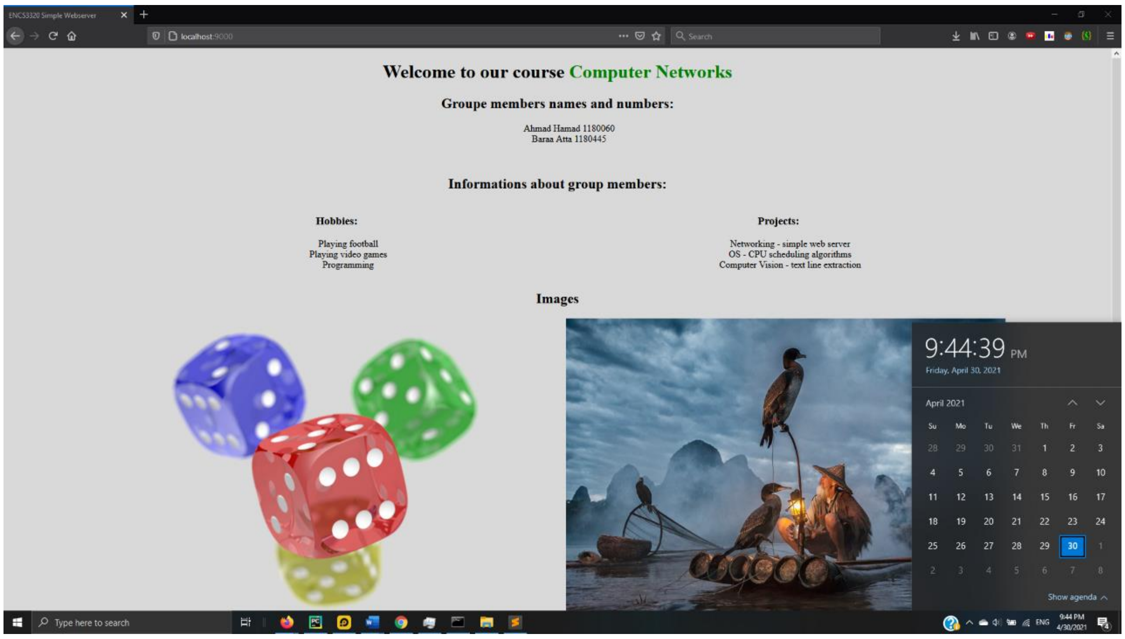
Figure 10: Final Part of Server's Code (404.html)

When the requested URL is unknown a 404 page is sent. At first a 404.html file is read. Next, there are two question marks in the code of the file needs to be replaced by the IP and port numbers of the client, so the index of the first? is found then the IP of the client is inserted by splitting the code to two parts and inserting the IP between them, the port number of the client is inserted using the same way. Next, the header of the response is sent by sending a status code 404 Not Found and setting the content type to html. Finally, the data which the html code is sent to the client and the connection is closed.

Screen shot for each request

Main page (/ or / index.html)

- Client page



Welcome to our course Computer Networks

Groupe members names and numbers:

Ahmad Hamad 1180060
Baraa Atta 1180445

Informations about group members:

Hobbies:

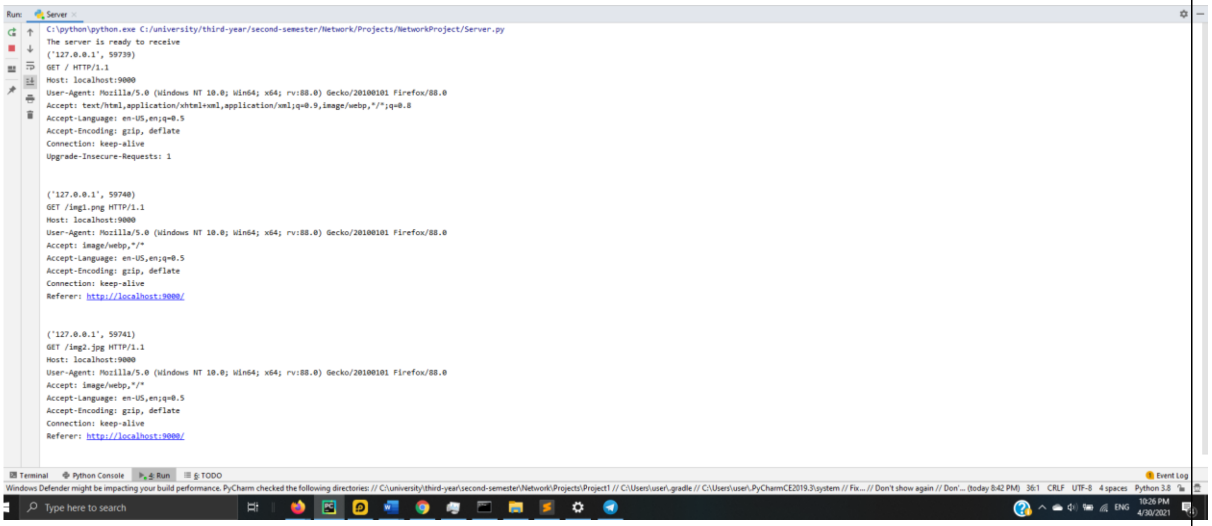
Playing football
Playing video games
Programming

Projects:

Networking - simple web server
OS - CPU scheduling algorithms
Computer Vision - text line extraction

Images

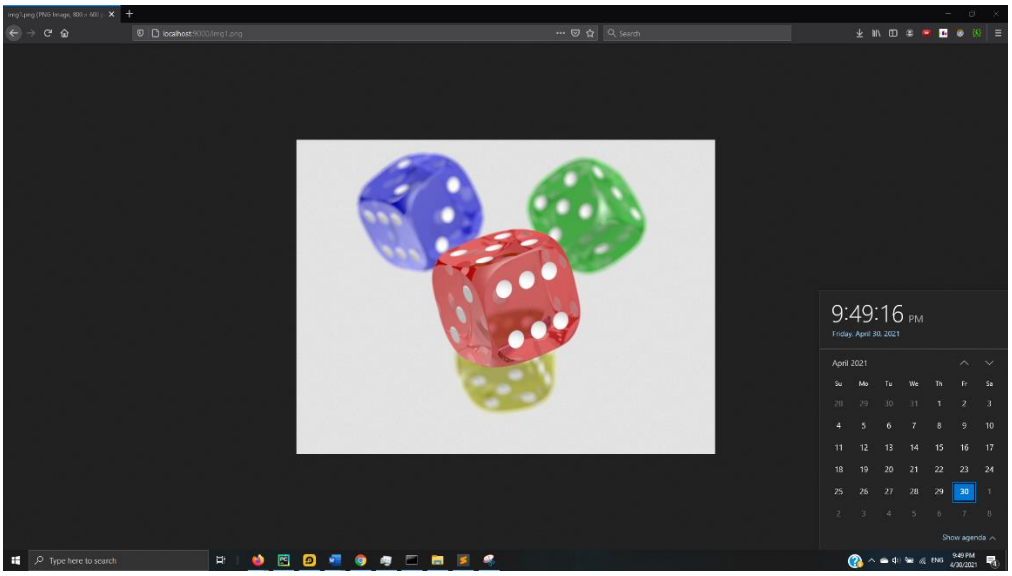
- Requests



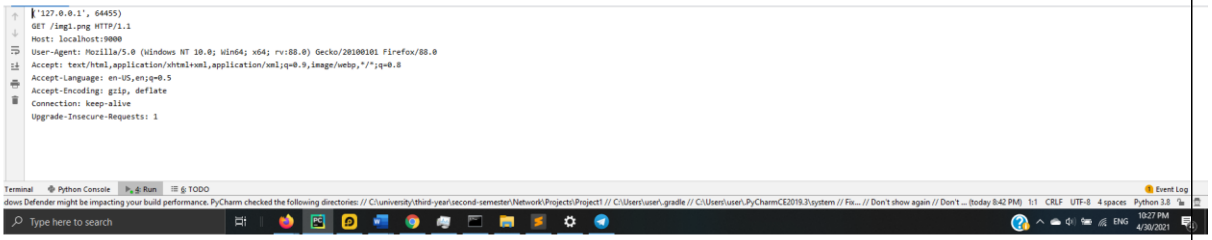
```
C:\python\python.exe C:/university/third-year/second-semester/Network/Projects/NetworkProject/Server.py
The server is ready to receive
('127.0.0.1', 59739)
GET / HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


('127.0.0.1', 59740)
GET /img1.png HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://localhost:9000/


('127.0.0.1', 59741)
GET /img2.jpg HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://localhost:9000/
```

PNG image(/imagename.png)

- Client page



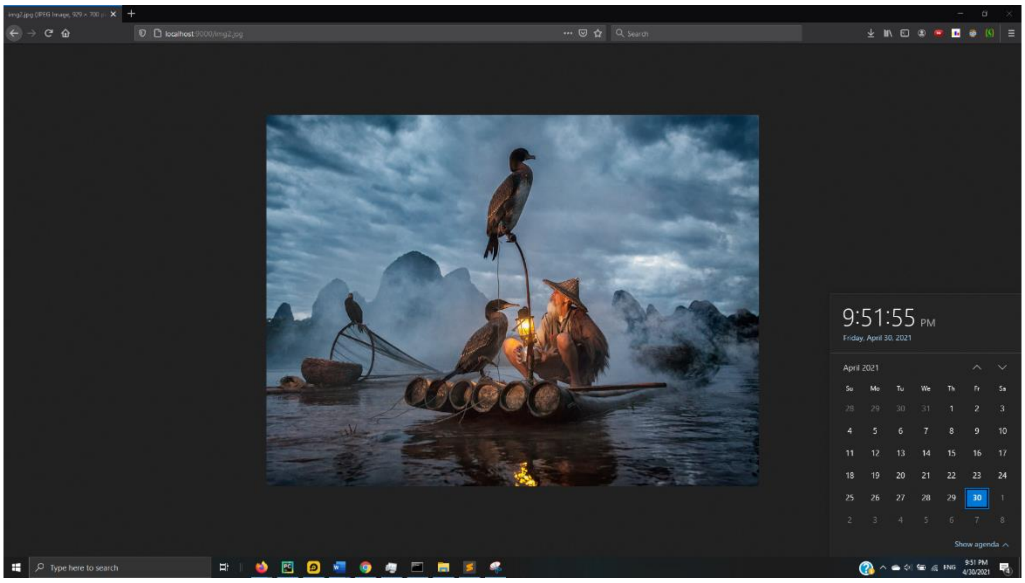
- Request



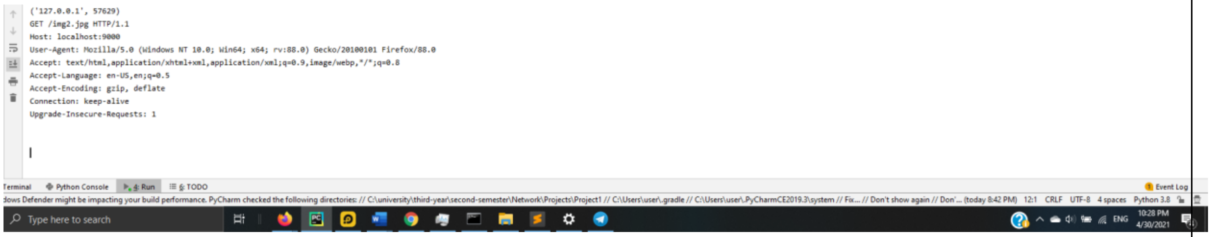
```
('127.0.0.1', 64455)
GET /img1.png HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

JPEG image (/imagename.jpg)

- Client page



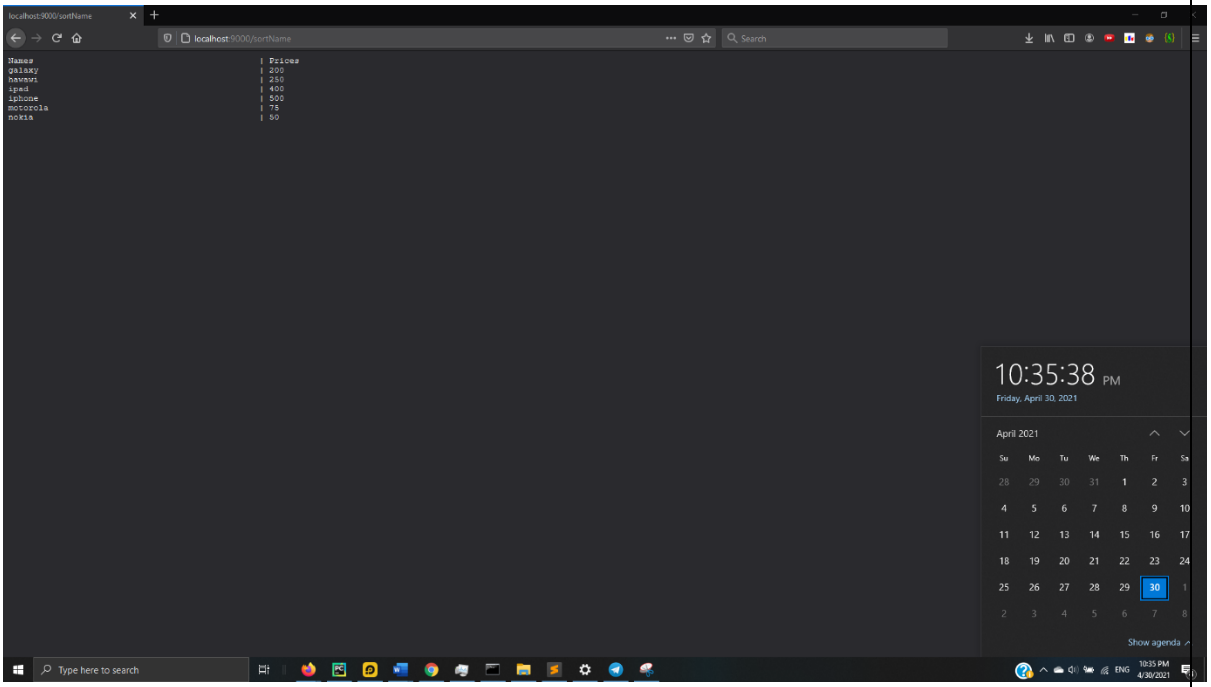
- Request



```
('127.0.0.1', 57629)
GET /img2.jpg HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

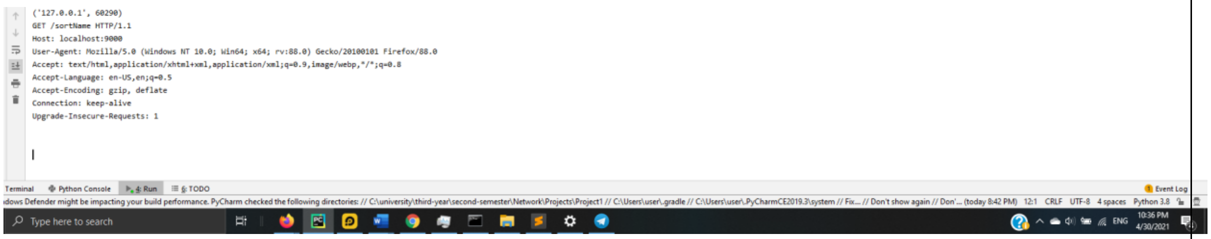
Sorting the file depending on the name (/sortName)

- Client page



```
Names                                          | Prices
galaxy                                         | 200
hawawi                                         | 250
ipad                                           | 400
iphone                                         | 500
motorola                                       | 75
nokia                                          | 50
```

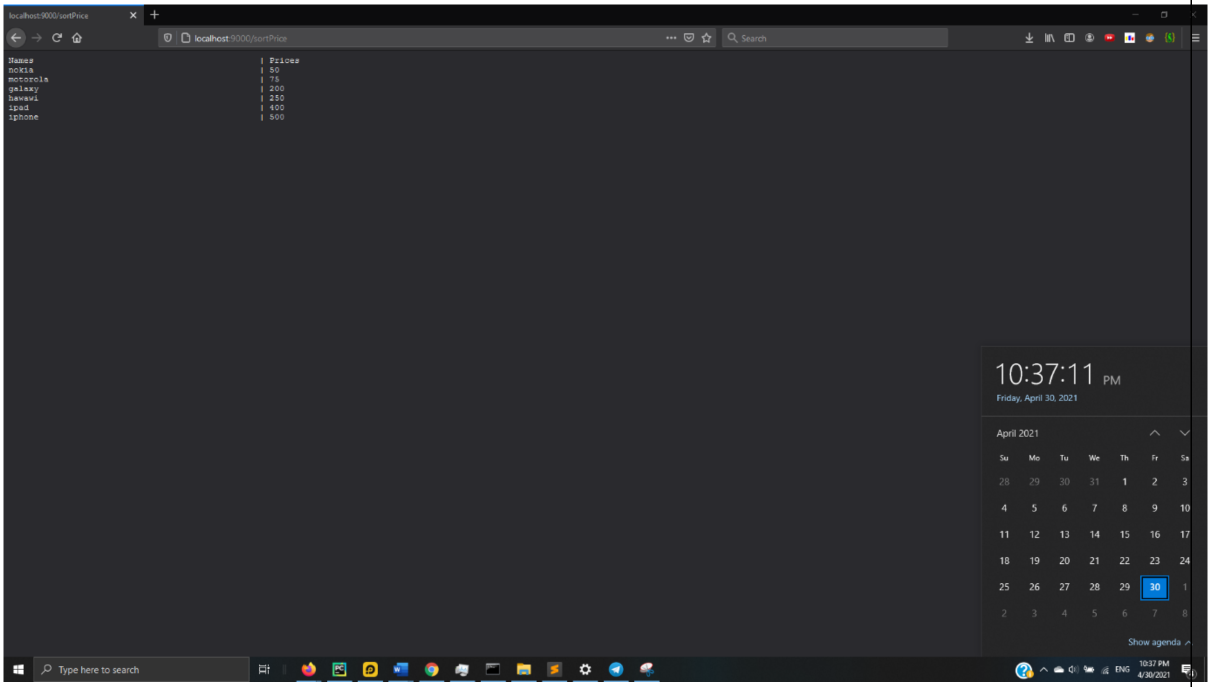
- Request



```
('127.0.0.1', 60290)
GET /sortName HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

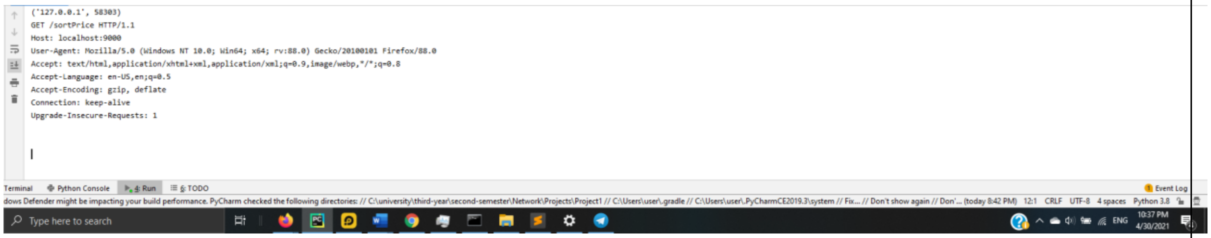
Sort the file depending on the price (/sortPrice)

- Client page



```
Names                                   | Prices
nokia                                   | 50
motorola                                | 75
galaxy                                  | 200
hawawi                                  | 250
ipad                                    | 400
iphone                                  | 500
```

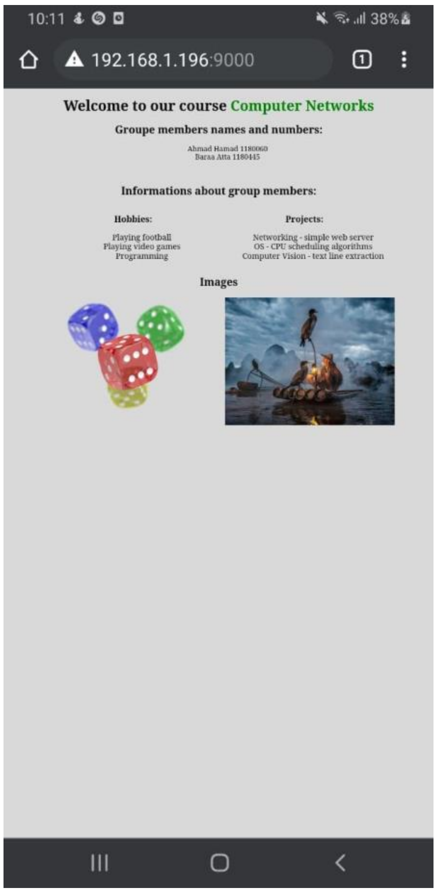
- Request



```
('127.0.0.1', 58303)
GET /sortPrice HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

running the index page on my mobile

- Client page



Welcome to our course Computer Networks

Groupe members names and numbers:

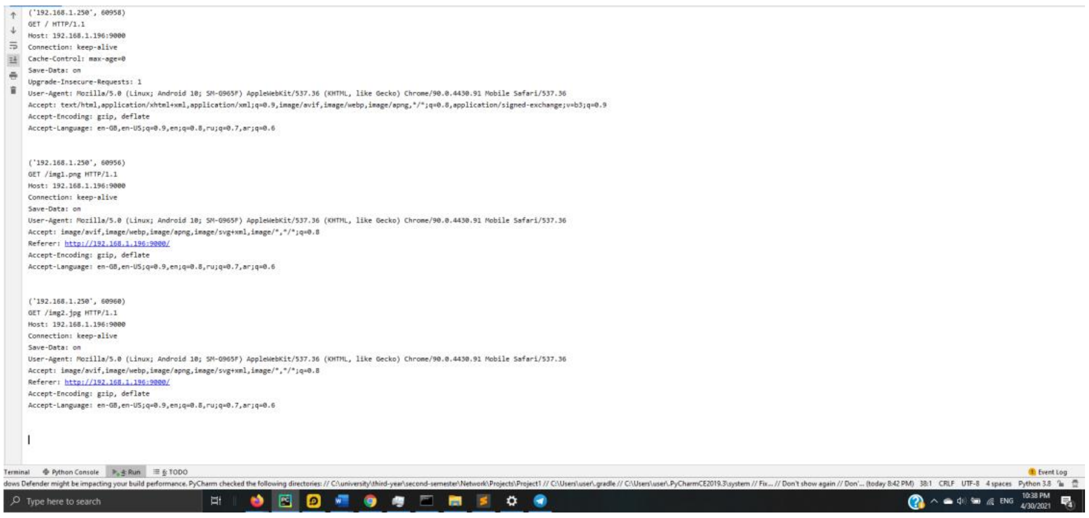
Ahmad Hamad 1180060
Baraa Atta 1180445

Informations about group members:

Hobbies:	Projects:
Playing football	Networking - simple web server
Playing video games	OS - CPU scheduling algorithms
Programming	Computer Vision - text line extraction

Images

- Requests

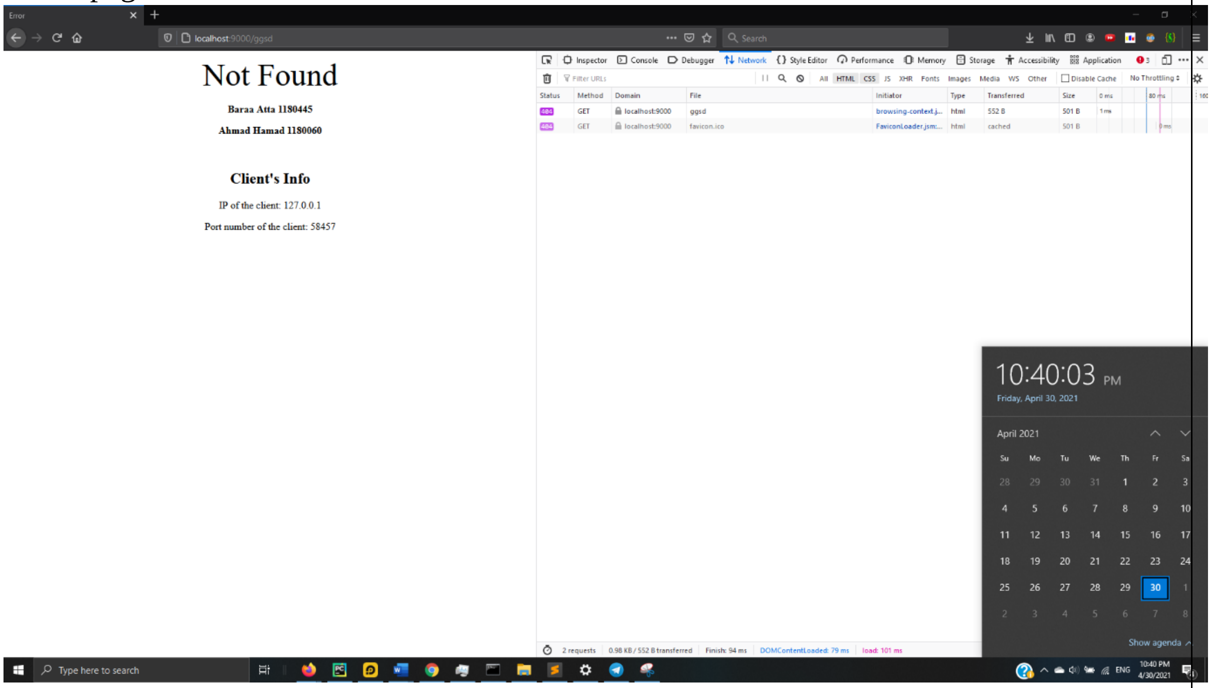


```
('192.168.1.250', 60958)
GET / HTTP/1.1
Host: 192.168.1.196:9000
Connection: keep-alive
Cache-Control: max-age=0
Save-Data: on
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 10; SM-G965F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.91 Mobile Safari/537.36
Accept: 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
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ru;q=0.7,ar;q=0.6


('192.168.1.250', 60956)
GET /img1.png HTTP/1.1
Host: 192.168.1.196:9000
Connection: keep-alive
Save-Data: on
User-Agent: Mozilla/5.0 (Linux; Android 10; SM-G965F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.91 Mobile Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.1.196:9000/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ru;q=0.7,ar;q=0.6


('192.168.1.250', 60960)
GET /img2.jpg HTTP/1.1
Host: 192.168.1.196:9000
Connection: keep-alive
Save-Data: on
User-Agent: Mozilla/5.0 (Linux; Android 10; SM-G965F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.91 Mobile Safari/537.36
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Referer: http://192.168.1.196:9000/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,ru;q=0.7,ar;q=0.6
```

dows Defender might be impacting your build performance. PyCharm checked the following directories: // C:\university\third-year\second-semester\Network\Projects\Project1 // C:\Users\user\.gradle // C:\Users\user\.PyCharmCE2019.3\system // Fix... // Don't show again // Don'... (today 8:42 PM) 38:1 CRLF UTF-8 4 spaces Python 3.8

Showing the error page (/any undefined path)

- Client page



Not Found

Baraa Atta 1180445

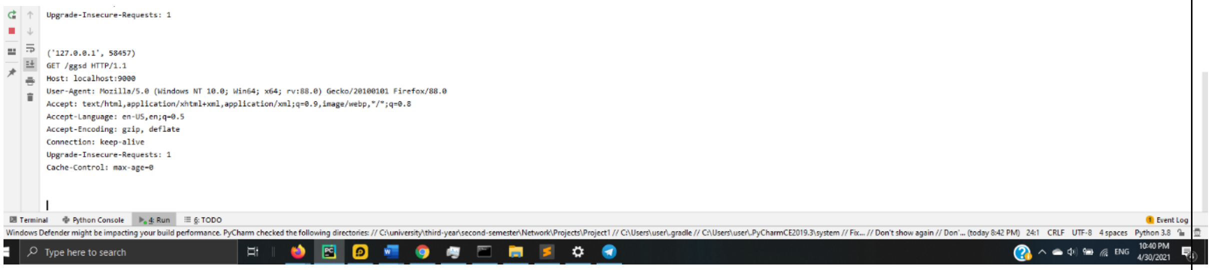
Ahmad Hamad 1180060

Client's Info

IP of the client: 127.0.0.1

Port number of the client: 58457

- Request



```
Upgrade-Insecure-Requests: 1


('127.0.0.1', 58457)
GET /ggsd HTTP/1.1
Host: localhost:9000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Conclusion

In this project we have learned about some main commands, that are used to test the network, like the ping, traceroute, and nslookup, moreover we saw the output after executing them and what each line means, and how it can be used to determine where the problem occurs.

At the second part we implemented a basic web server, using TCP socket programming, this part was very useful, since we learned from it how we can handle different types of requests and deal with each one of them separately.

Finally, we can say that we were interested in working at this project, because it takes from the theoretical analysis of networks to the actual field in which those protocols are used.