



Faculty of Engineering and Technology  
Department of Electrical and Computer Engineering

### **Computer Networks**

**ENEE3320**

Project #1

### **Socket Programming**

---

**Prepared by:**

- Ali Shaikh Qasem      ID: 1212171
- Abdalrahman Juber      ID: 1211769
- Ahmed Saqer      ID: 1210085

**Instructor:** Dr. Ibrahim Nemer.

**Sections:** 1,4.

**Date:** 26/4/2024

Birzeit University

2023-2024

## Abstract

This project aims to study and practice networks principles. Specifically, to apply some networking commands and observe their output. In addition, to be able to use socket programming to make a simple client-server UDP application and a complete server to respond to user requests.

## Table of Contents

<b>1. Part one</b>	<b>1</b>
<b>1.1. Defining the commands</b>	<b>1</b>
<b>1.2. Running some commands</b>	<b>1</b>
<b>1.3. Capturing some DNS messages</b>	<b>4</b>
<b>2. Part two</b>	<b>6</b>
<b>2.1. General Description and explanation</b>	<b>6</b>
<b>2.2. Testing the Code:</b>	<b>9</b>
<b>3. Part Three</b>	<b>10</b>
<b>3.1. Question:</b>	<b>10</b>
<b>3.2. Initializing the server</b>	<b>10</b>
<b>3.3. Waiting for connection and obtaining the request</b>	<b>10</b>
<b>3.4. Providing the required file based on the request</b>	<b>11</b>
<b>3.5. Testing the code</b>	<b>14</b>
<b>4. Conclusion</b>	<b>19</b>

## Table of figures

Figure 1: pinging a device from the same network .....	1
Figure 2: pinging stanford.edu .....	2
Figure 3: tracert stanford.edu .....	3
Figure 4: nslookup stanford.edu .....	4
Figure 5: Capturing DNS messages .....	5
Figure 6 : Calculating boradcast address .....	6
Figure 7 :Sender function.....	7
Figure 8 : Receiver function .....	8
Figure 9 :Display function .....	8
Figure 10 : Testing the code.....	9
Figure 11: initialize the server.....	10
Figure 12: obtainning user request.....	10
Figure 13: code part1 .....	11
Figure 14: code part2 .....	12
Figure 16:code part3.....	13
Figure 15:code part4.....	13
Figure 17: http request at command line .....	14
Figure 18: english page .....	14
Figure 19: Arabic page .....	15
Figure 20: Css file request.....	15
Figure 21: png image request .....	16
Figure 22: jpg image request .....	16
Figure 23: itc request .....	17
Figure 24: stack overflow request.....	17
Figure 25: wrong request test.....	17
Figure 26: opening the page from a phone .....	18

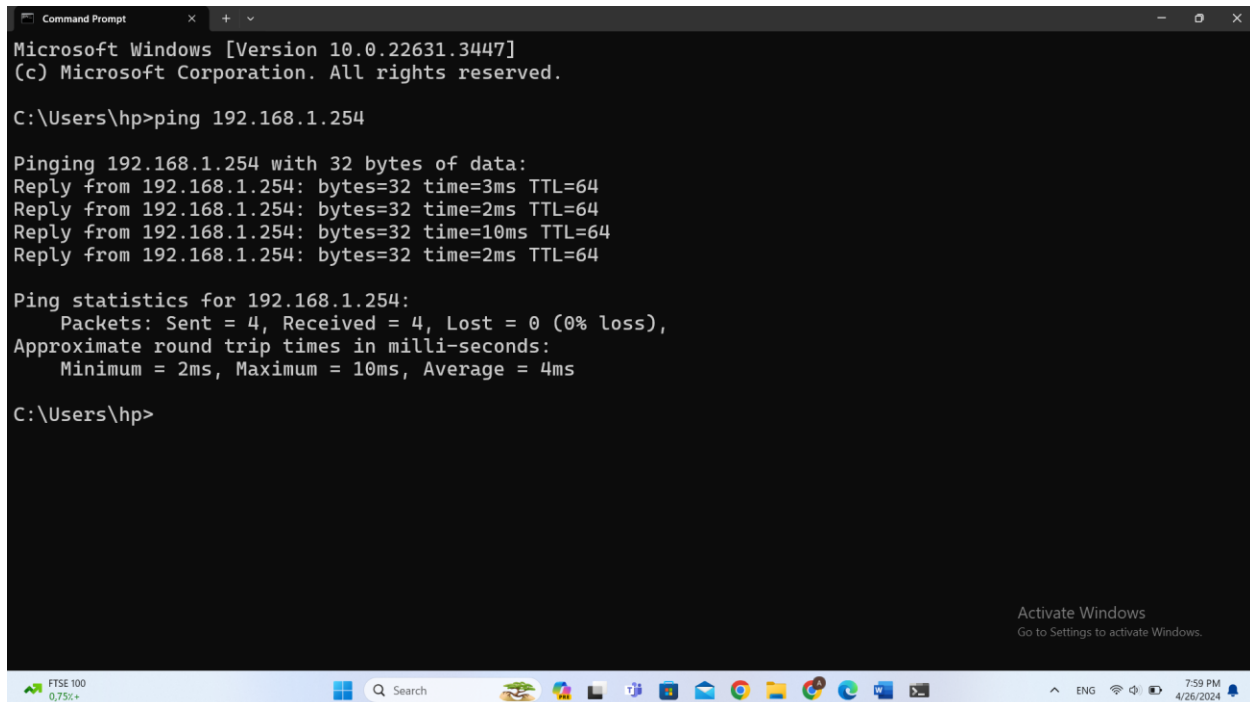
## 1. Part one

### 1.1. Defining the commands

- **Ping:** a network utility used to send packets to a target host and measure the response time considering both the TTL (time to live) and RTT (round trip time).
- **tracert:** a command that provides delay measurement from source to destination along the end-end path.
- **nslookup:** a command that queries DNS to get information about a domain, such as its IP address, IPV4 address, and retrieve various records associated with the DNS.
- **telnet:** a network protocol allows us to connect to a server or host and interact with it by a command-line interface.

### 1.2. Running some commands

a) Ping a device from the same network:



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>ping 192.168.1.254

Pinging 192.168.1.254 with 32 bytes of data:
Reply from 192.168.1.254: bytes=32 time=3ms TTL=64
Reply from 192.168.1.254: bytes=32 time=2ms TTL=64
Reply from 192.168.1.254: bytes=32 time=10ms TTL=64
Reply from 192.168.1.254: bytes=32 time=2ms TTL=64

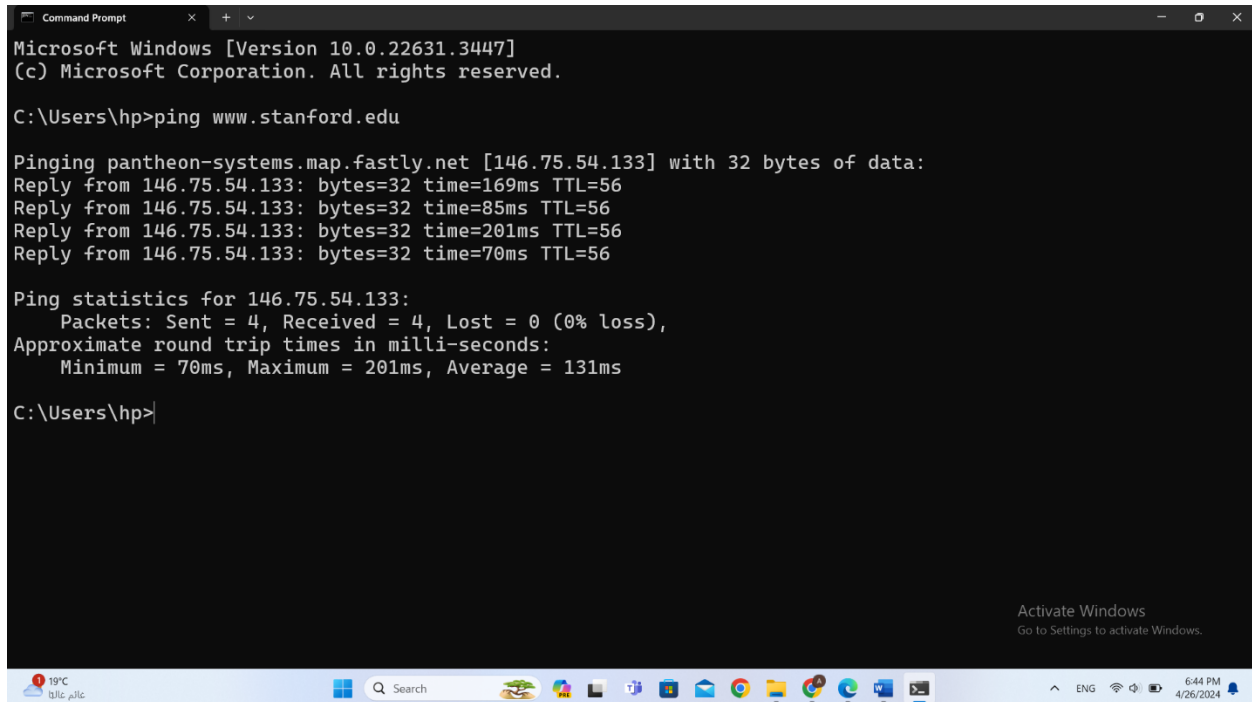
Ping statistics for 192.168.1.254:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2ms, Maximum = 10ms, Average = 4ms

C:\Users\hp>
```

Figure 1: pinging a device from the same network

- The figure above shows a ping from laptop to the home router (whose IP = 192.168.1.254) in same network. It's shown that four packets were sent, each of them is 32 bytes, all of them were successfully received. The round-trip time differ for each packet and changes from 2 to 10 msecs with an average of 4 msecs.

b) ping [www.stanford.edu](http://www.stanford.edu):



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>ping www.stanford.edu

Pinging pantheon-systems.map.fastly.net [146.75.54.133] with 32 bytes of data:
Reply from 146.75.54.133: bytes=32 time=169ms TTL=56
Reply from 146.75.54.133: bytes=32 time=85ms TTL=56
Reply from 146.75.54.133: bytes=32 time=201ms TTL=56
Reply from 146.75.54.133: bytes=32 time=70ms TTL=56

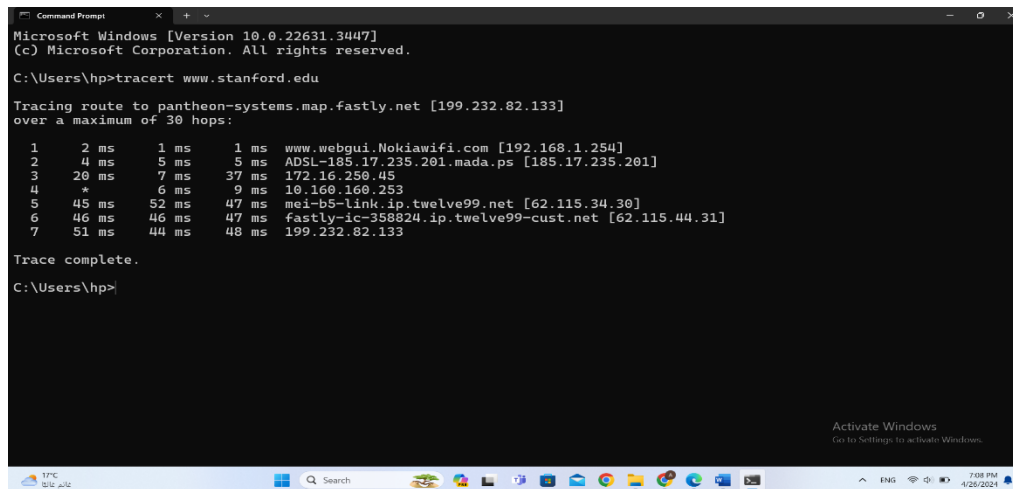
Ping statistics for 146.75.54.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 70ms, Maximum = 201ms, Average = 131ms

C:\Users\hp>
```

Figure 2: pinging stanford.edu

- The figure above shows a ping from laptop to the domain Stanford.edu (whose IP = 146.75.54.133) in same network. It's shown that four packets were sent, each of them is 32 bytes, all of them were successfully received. The round-trip time differ for each packet and changes from 70 to 201 msec with an average of 131 msec.
- c) From the ping results seen in this example, we notice that average RTT time is quite high compared with those in part (a), but this doesn't necessarily mean that the response is from USA, the Stanford university site might be distributed in some servers close to us.

d) `tracert` [www.stanford.edu](http://www.stanford.edu):



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>tracert www.stanford.edu

Tracing route to pantheon-systems.map.fastly.net [199.232.82.133]
over a maximum of 30 hops:
  0  0 ms  0 ms  0 ms  www.webgui.Nokiawifi.com [192.168.1.254]
  1  4 ms  5 ms  5 ms  ADSL-185.17.235.201.mada.ps [185.17.235.201]
  2 20 ms  7 ms 37 ms 172.16.250.45
  3  *      6 ms  9 ms 10.160.160.253
  4 45 ms 52 ms 47 ms mei-b5-link.ip.twelve99.net [62.115.34.30]
  5 46 ms 46 ms 47 ms fastly-ic-358824.ip.twelve99-cust.net [62.115.44.31]
  6 51 ms 44 ms 48 ms 199.232.82.133

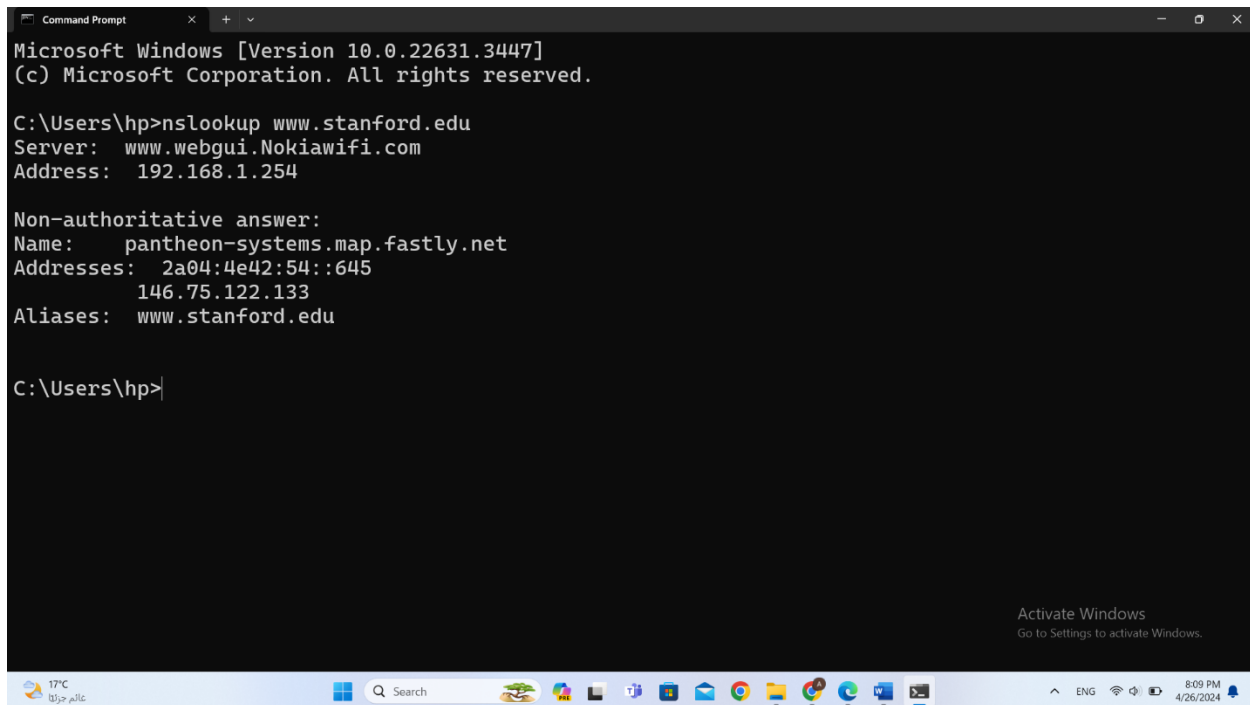
Trace complete.

C:\Users\hp>
```

Figure 3: `tracert` stanford.edu

- From the figure above, we notice that `tracert` command shows the end-end path from source to destination and provide three delays measurements for each node. We can see that the path from our local router to the target domain (Stanford.edu) took only 7 nodes, starts from the local routers and ISPs such as mada.ps and going towards global ISPs like fastly.net. We also notice that the measured delays is increasing when we go down, which is normal since we are going to further nodes with longer distance and congestion as going down. It's also noticed that some of delays at node number 4 is represented by asterisk (\*) this means the router in node number 4 did not respond the message during measuring this delay.

e) nslookup [www.stanford.edu](http://www.stanford.edu):



```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>nslookup www.stanford.edu
Server: www.webgui.Nokiawifi.com
Address: 192.168.1.254

Non-authoritative answer:
Name:    pantheon-systems.map.fastly.net
Addresses: 2a04:4e42:54::645
          146.75.122.133
Aliases:  www.stanford.edu

C:\Users\hp>
```

Figure 4: nslookup stanford.edu

- From the figure above, we notice that nslookup command provide some information about our host server such as server name, IP address. As well as some information about the domain we searched for such as name, addresses and aliases names.

### 1.3. Capturing some DNS messages

In this part, we used wireshark software to capture some DNS messages:



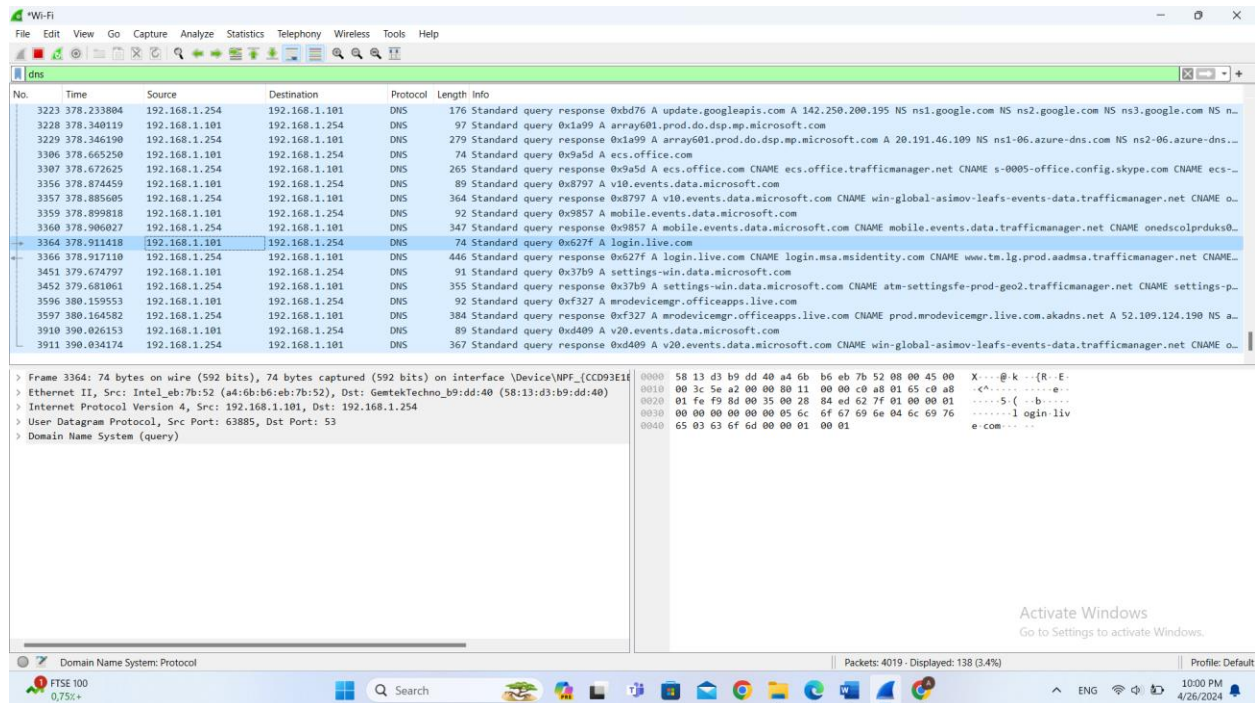


Figure 5: Capturing DNS messages

After we start the capturing in Wireshark, we filter the packets such that only DNS messages are displayed, then we opened a website (e.g. ritaj website) to capture some DNS queries. The figure above shows multiple DNS queries, if we take the line number 3364 for example: **3364 378.911418 192.168.1.101 192.168.1.254 DNS 74 Standard query 0x627f A login.live.com**. This line provides many information about the query such as:

- **Packet Information:** It includes details like packet number (3364), time (378.911418), source IP address (192.168.1.101), destination IP address (192.168.1.254), protocol (DNS), and packet length (74).
- **Query details:** standard query with ID = 0x627f and type A.
- **Domain name:** the domain name being queried is login.live.com.

## 2. Part two

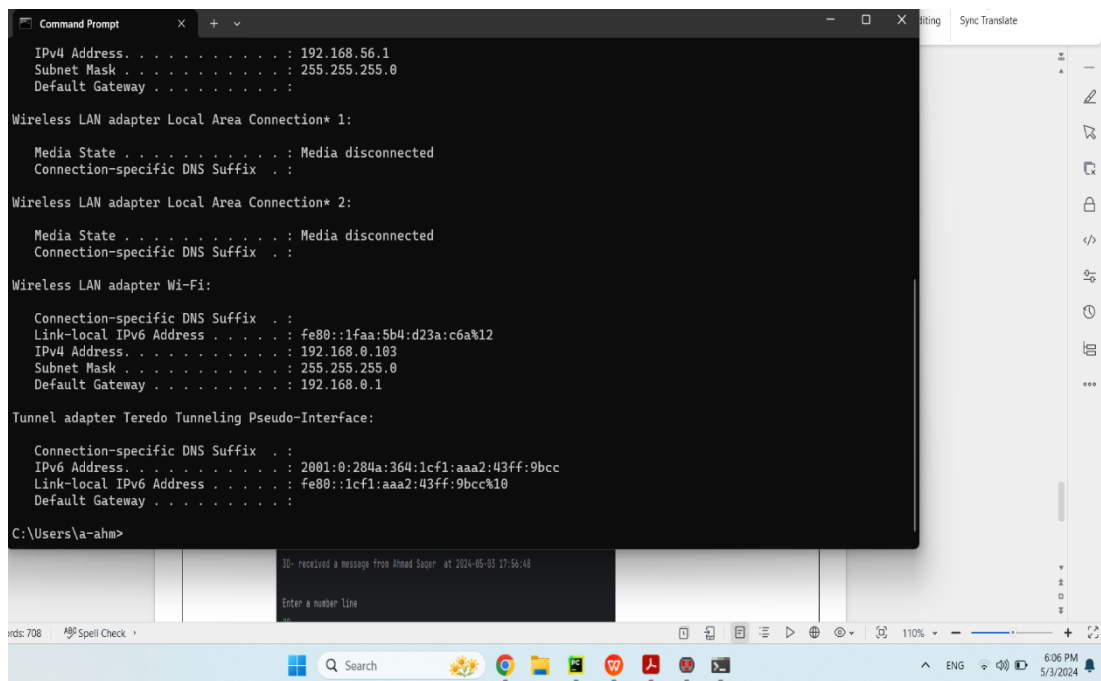
### 2.1. General Description and explanation

The idea for this part is to able the peers to send and receive messages to the broadcast address therefore,all peers must be connected on the same network and having same subnet mask.

We can get the broadcast address by taking the subnet mask and determine the last octet for instance the below figure shows the IP address and subnet mask by typing ipconfig command in cmd.

As seen in figure below we are interesting in the sharing network which is wireless LAN adapter Wi-Fi.

Which having subnet mask 255.255.255.0 and the peer IP address which is 192.168.0.103 then the broadcast address equals 192.168.0.255 because the subnet mask has only one zero octet and it is last part.



```
Command Prompt
IPv4 Address. . . . . : 192.168.56.1
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :

Wireless LAN adapter Local Area Connection* 1:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix  . :
Link-local IPv6 Address . . . . . : fe80::1faa:5b4:d23a:c6a%12
IPv4 Address. . . . . : 192.168.0.103
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.0.1

Tunnel adapter Teredo Tunneling Pseudo-Interface:

Connection-specific DNS Suffix  . :
IPv6 Address. . . . . : 2001:0:284a:364:1cf1:aaa2:43ff:9bcc
Link-local IPv6 Address . . . . . : fe80::1cf1:aaa2:43ff:9bcc%10
Default Gateway . . . . . :

C:\Users\A-ahm>
```

Figure 6 : Calculating boradcast address

For writing code for this part python language was used and those libraries was import threading ,socket and time.

- Threading was import for letting the peers sending and receiving messages at the same time .
- Socket to implement UDP connection for every peers to the broadcast address at port 5051.
- Time to calculating the time was a messages sent until it reach at receiver part.

Each peer should send a message contain the flowing format :

FirstnameLastName : message

The first and last name of a sender and the message he wants to send .  
Colon will cut off the name and message from each other.

**Three functions was built :**

```
def Sender():
    message = input('For Sending : Enter your first and last name: And yor message. For peers sending\n')
    line_num = "10"
    PeerSend = socket(AF_INET, SOCK_DGRAM) #Creating UDP connection
    PeerSend.setsockopt(SOL_SOCKET, SO_BROADCAST, _value: 1) #Enabling broadcasting
    PeerSend.sendto(line_num.encode(), (BroadcastAddress, broadcastPort))
    PeerSend.sendto(message.encode(), (BroadcastAddress, broadcastPort))
    PeerSend.close()
```

Figure 7 :Sender function

Sender function will ask the peer to enter a message and should be same as format as specified above .

The sender function will send two messages the first one is the number of line which the display function will be able to know how many messages was sent.

The real message containing the format .

As seen in figure 5 we enable broadcast address by setting the socket to 1 for SO\_BROADCAST.

### The Receiver function:

```
def Receiver():
    PeerReceive = socket(AF_INET, SOCK_DGRAM) #Creating UDP connection
    PeerReceive.bind('', broadcastPort)
    counter = ""
    string_list = []
    print(f"For listening on broadcast messages on {BroadcastAddress}:{broadcastPort}...")
    while True:
        message, PeerAddress = PeerReceive.recvfrom(2048)
        if len(message.decode()) == 2:
            string_list.append(message.decode())
            counter = message.decode()

        else:
            length = len(string_list)
            display(message.decode(), counter: f"{length}D")
```

Figure 8 : Receiver function

The receiver function will create a UDB connection with port 5051 number .

This function will keep listening to the sender messages and will call display function .

### The display function:

```
def display(message, counter):
    current_time = time.strftime( format: "%Y-%m-%d %H:%M:%S", time.localtime())
    # Find the index of the first occurrence of ":"
    colon_index = message.find(":")
    # Extract the First and Last name
    name = message[:colon_index]
    saved_messages[counter] = message[colon_index + 1:]
    if counter == "1D":
        print(f"Peer {name}\n")
    print(f"{counter}- received a message from {name} at {current_time}\n")
    if counter == "3D":
        num = input("Enter a number line\n")
        print(f"{saved_messages[num]}\n")
```

Figure 9 :Display function

The display function will print the messages in lines with there names and times.

This function required 3 messages sent to it for display the message since the receiver part will keep in the loop listening to the messages were sent.

## 2.2. Testing the Code:

For testing this code two computer were used one of them send messages and the other is listening to those messages. the figure below shows 3 messages the first line contains the first and last name of the listening peer, Then the second line contains the message from that peer.

The other messages contain the messages from the other peers.

```
For Sending : Enter your first and last name: And yor message. For peers sending
For Listening on broadcast messages on 192.168.0.255:5051...
Abdalrhman Juber: Hello from Abdalrhman
Peer Abdalrhman Juber

1D- received a message from Abdalrhman Juber at 2024-05-03 17:42:33

2D- received a message from Ali Shaikh Qasem at 2024-05-03 17:56:27

3D- received a message from Ahmad Saqer at 2024-05-03 17:56:48

Enter a number line
2D
Hello from Ali
```

Figure 10 : Testing the code

Figure 8 shows that Abdalrhman Juber is listening to other peers. Second line contains the message that Abdalrhman sent. And other lines are from Ali Shaikh Qasem and Ahmad Saqer.

If we want to see the message that Ali sent we must write 2D as showing above then we will have that message was sent printed on the screen.

At last we used 2 thread calling both sender and receiver function at the main.

### 3. Part Three

In this part, we used socket programming to construct a simple and complete web server that accepts user's request and provide him the appropriate file based on the request

#### 3.1. Question:

From rfce2616, what is Entity Tag Cache Validators in the HTTP protocol and why do we need it?

- Entity Tag (ETag) Cache Validators in HTTP are unique identifiers assigned by servers to resources. They help clients to determine if their cached copies are still valid or need updating by comparing ETags. This reduces unnecessary network traffic and improves performance by allowing clients to use cached resources when appropriate.

#### 3.2. Initializing the server

We initialized the server by defining a TCP connection socket and define a port = 6060 to listen for user request

```
24  # initialize the server
25  server_port = 6060
26  server_socket = socket(AF_INET, SOCK_STREAM)
27  server_socket.bind(('', server_port))
28  server_socket.listen(1)
29  print("The server is ready to receive requests")
30
```

Figure 11: initialize the server

#### 3.3. Waiting for connection and obtaining the request

The sever keep reading requests from user in a loop, after reading it, we split the user sentence to get the exact request.

```
32  # handling requests
33  while True:
34      connection_socket, addr = server_socket.accept()
35      client_ip = addr[0]
36      client_port = addr[1]
37      print("Got connection from IP = " + client_ip + "Port = " + str(client_port))
38      sentence = connection_socket.recv(1024).decode()
39      print("request is: " + sentence )
40
41      # obtain the request part from the whole received message
42      request = sentence.split()[1]
```

Figure 12: obtaining user request

### 3.4. Providing the required file based on the request

After getting the request, the server determine which file to send based on the request structure, (e.g. .css, .html, .png, index file .....)

```
44     if (request == '/' or request == '/index.html' or request == '/main_en.html' or request == '/en'):
45         connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
46         connection_socket.send('Content-Type: text/html \r\n'.encode())
47         connection_socket.send('\r\n'.encode())
48         with open("main_en.html", "rb") as f:
49             content = f.read()
50             connection_socket.send(content)
51
52     elif (request == '/ar'):
53         connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
54         connection_socket.send('Content-Type: text/html \r\n'.encode())
55         connection_socket.send('\r\n'.encode())
56         with open("main_ar.html", "rb") as f:
57             content = f.read()
58             connection_socket.send(content)
59
60     elif (request.endswith('.html')):
61         file_name = request[1:] # to remove the '/' cahracter
62         try:
63             with open(file_name, "rb") as f:
64                 connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
65                 connection_socket.send('Content-Type: text/html \r\n'.encode())
66                 connection_socket.send('\r\n'.encode())
67                 content = f.read()
68                 connection_socket.send(content)
69         except FileNotFoundError:
70             send_error_file(connection_socket, client_ip, client_port)
```

Figure 13: code part1

The code segment below shows two different kinds of png or jpg images requests, part 2 of the code shows the requests that doesn't start with (/myform.html?name) , this part handles the image requests from the main browser, while part 3 of the code handles the requests that start with (/myform.html?name), which are requests from the form (myform.html).

```
71
72 ✓ elif ( request.endswith('.css')):
73     file_name = request[1:] # to remove the '/' cahracter
74 ✓     try:
75 ✓         with open(file_name, "rb") as f:
76             connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
77             connection_socket.send('Content-Type: text/css \r\n'.encode())
78             connection_socket.send('\r\n'.encode())
79             content = f.read()
80             connection_socket.send(content)
81 ✓     except FileNotFoundError:
82         send_error_file(connection_socket, client_ip, client_port)
83
84 ✓ elif ( request.endswith('.png') and not request.startswith('/myform.html?name=')):
85     file_name = "images/" + request[1:] # to remove the '/' cahracter
86 ✓     try:
87 ✓         with open(file_name, "rb") as f:
88             connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
89             connection_socket.send('Content-Type: image/png \r\n'.encode())
90             connection_socket.send('\r\n'.encode())
91             content = f.read()
92             connection_socket.send(content)
93 ✓     except FileNotFoundError:
94         send_error_file(connection_socket, client_ip, client_port)
95
96 ✓ elif ( request.endswith('.jpg') and not request.startswith('/myform.html?name=')):
97     file_name = "images/" + request[1:] # to remove the '/' cahracter
98 ✓     try:
99 ✓         with open(file_name, "rb") as f:
100             connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
101             connection_socket.send('Content-Type: image/jpg \r\n'.encode())
102             connection_socket.send('\r\n'.encode())
103             content = f.read()
104             connection_socket.send(content)
105 ✓     except FileNotFoundError:
106         send error file(connection socket, client ip, client port)
```

Figure 14: code part2



```

107 elif (request.endswith('.png')):
108     # Split the string using "=" as a delimiter
109     parts = request.split("=")
110     # Get the second part
111     request = parts[1]
112     try:
113         with open('images/' + request, "rb") as f:
114             connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
115             connection_socket.send('Content-Type: image/png; charset=utf-8\r\n'.encode())
116             connection_socket.send('\r\n'.encode())
117             content = f.read()
118             connection_socket.send(content)
119     except FileNotFoundError:
120         send_error_file(connection_socket, client_ip, client_port)
121
122 elif (request.endswith('.jpg')):
123     # Split the string using "=" as a delimiter
124     parts = request.split("=")
125     # Get the second part
126     request = parts[1]
127     try:
128         with open('images/' + request, "rb") as f:
129             connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
130             connection_socket.send('Content-Type: image/jpg; charset=utf-8\r\n'.encode())
131             connection_socket.send('\r\n'.encode())
132             content = f.read()
133             connection_socket.send(content)
134     except FileNotFoundError:
135         send_error_file(connection_socket, client_ip, client_port)
136

```

Figure 16:code part3

```

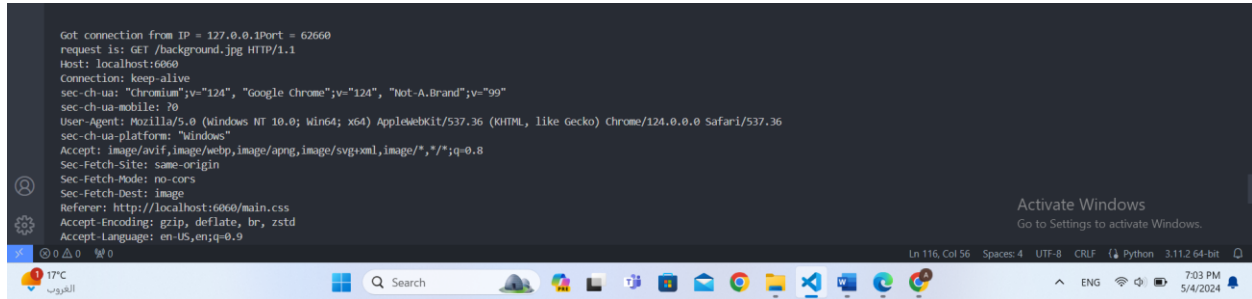
137
138 elif (request == '/so'):
139     connection_socket.send('HTTP/1.1 307 Temporary Redirect \r\n'.encode())
140     connection_socket.send('Content-Type: text/html \r\n'.encode())
141     connection_socket.send('Location: https://stackoverflow.com/ \r\n'.encode())
142
143 elif (request == '/itc'):
144     connection_socket.send('HTTP/1.1 307 Temporary Redirect \r\n'.encode())
145     connection_socket.send('Content-Type: text/html \r\n'.encode())
146     connection_socket.send('Location: https://itc.birzeit.edu/ \r\n'.encode())
147
148 else:
149     send_error_file(connection_socket, client_ip, client_port)
150     # close the connection
151     connection_socket.close()
152
153
154

```

Figure 15:code part4

### 3.5. Testing the code

- Http request in the command line:



```
Got connection from IP = 127.0.0.1 Port = 62660
request is: GET /background.jpg HTTP/1.1
Host: localhost:6060
Connection: keep-alive
sec-ch-ua: "Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:6060/main.css
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
```

Figure 17: http request at command line

- English page request ( /en or /index.html or /main\_en.html or /):

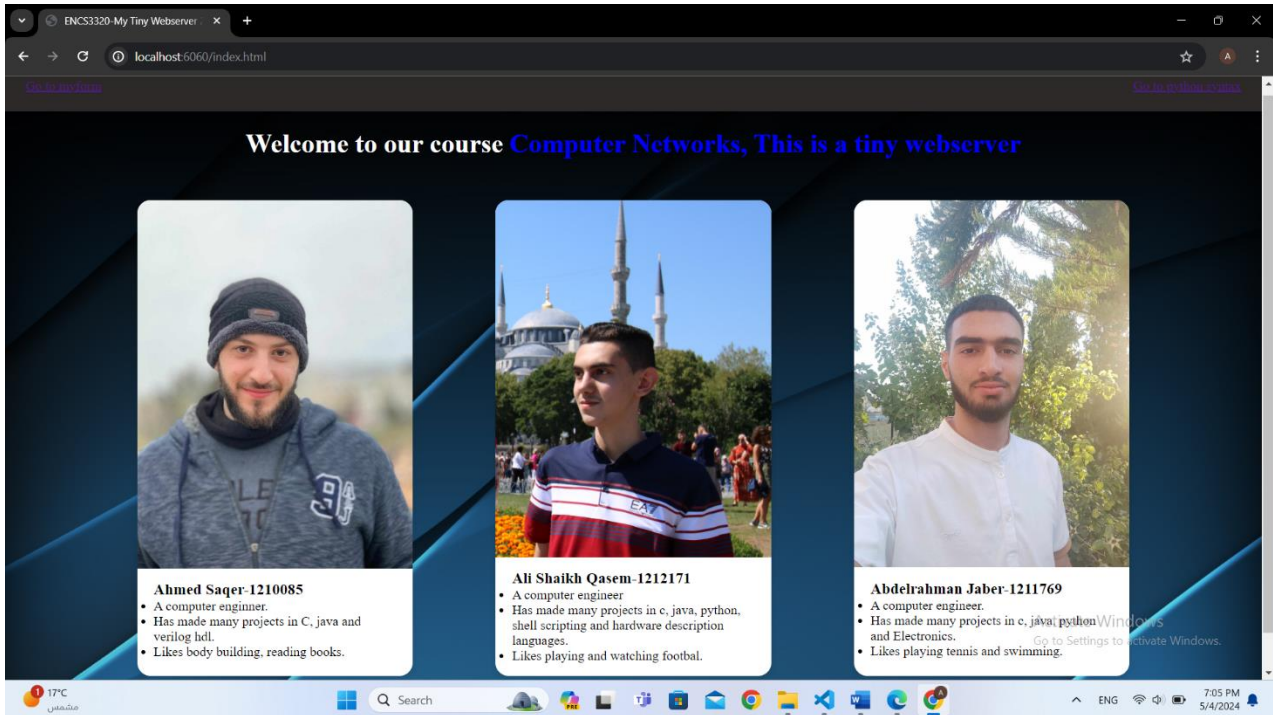


Figure 18: english page

- Arabic page request(/ar):

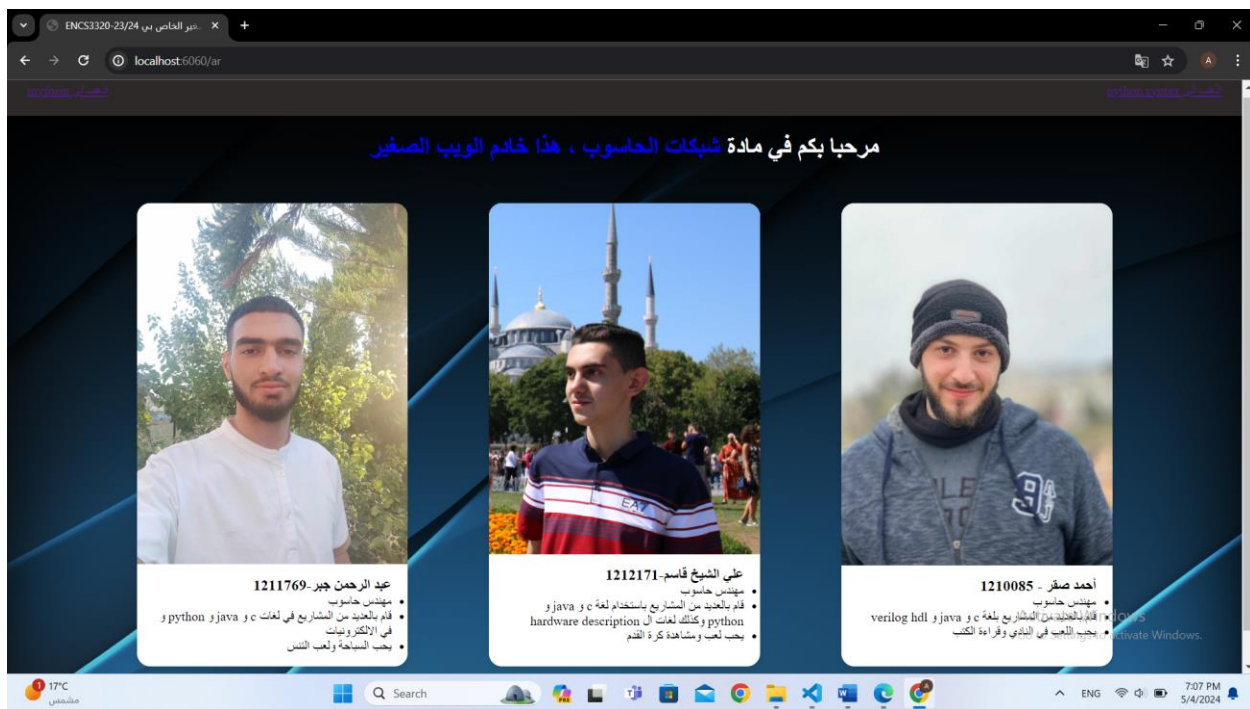


Figure 19: Arabic page

- css file request:

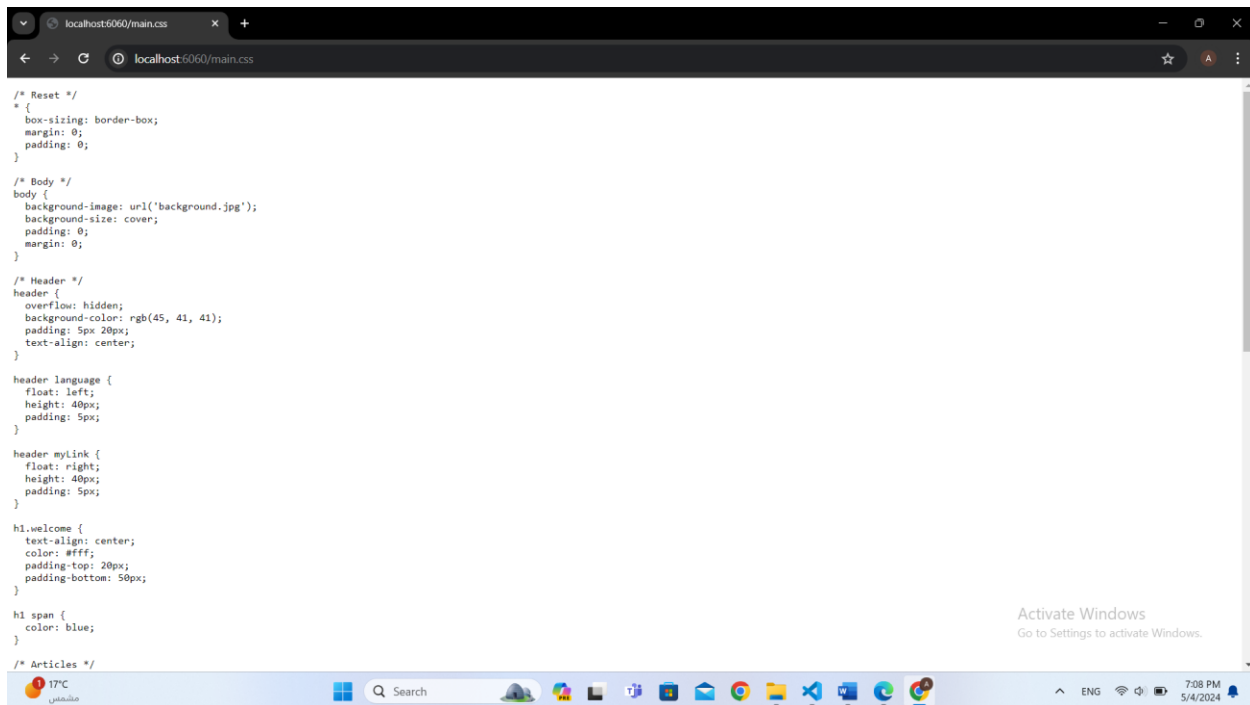


Figure 20: Css file request

- Png image request (from basic browser):

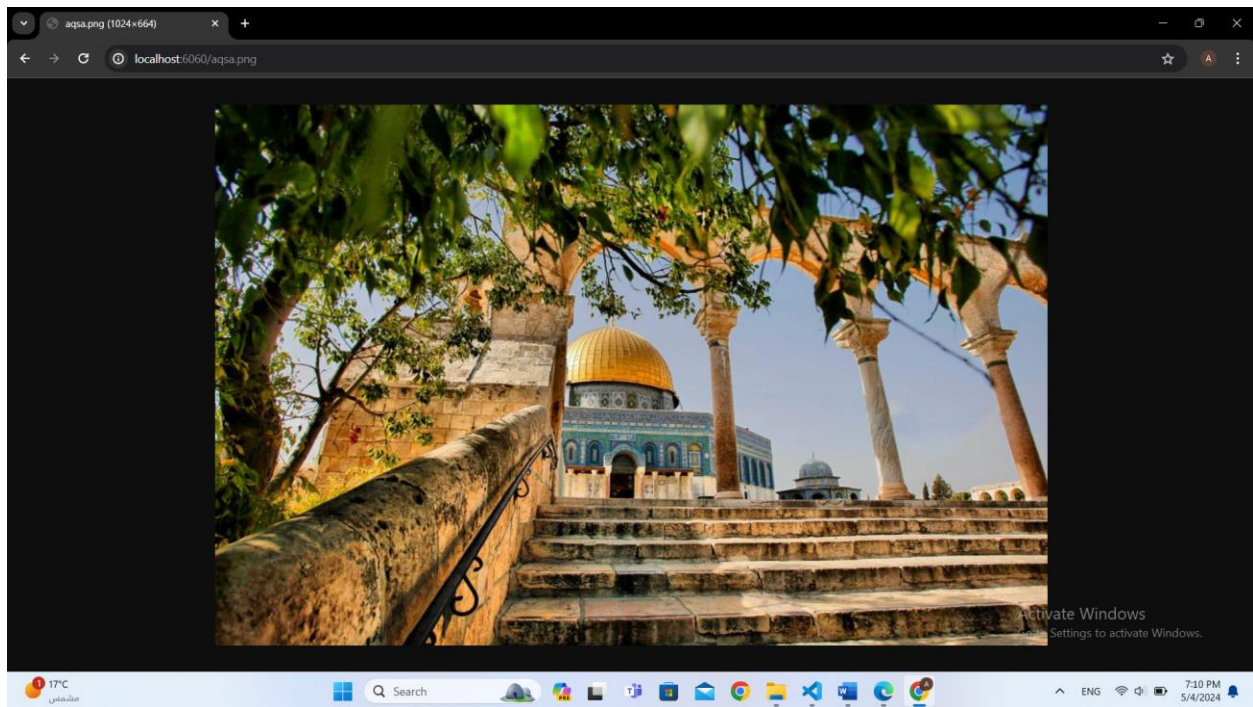


Figure 21: png image request

- Jpg image request (from myform.html):

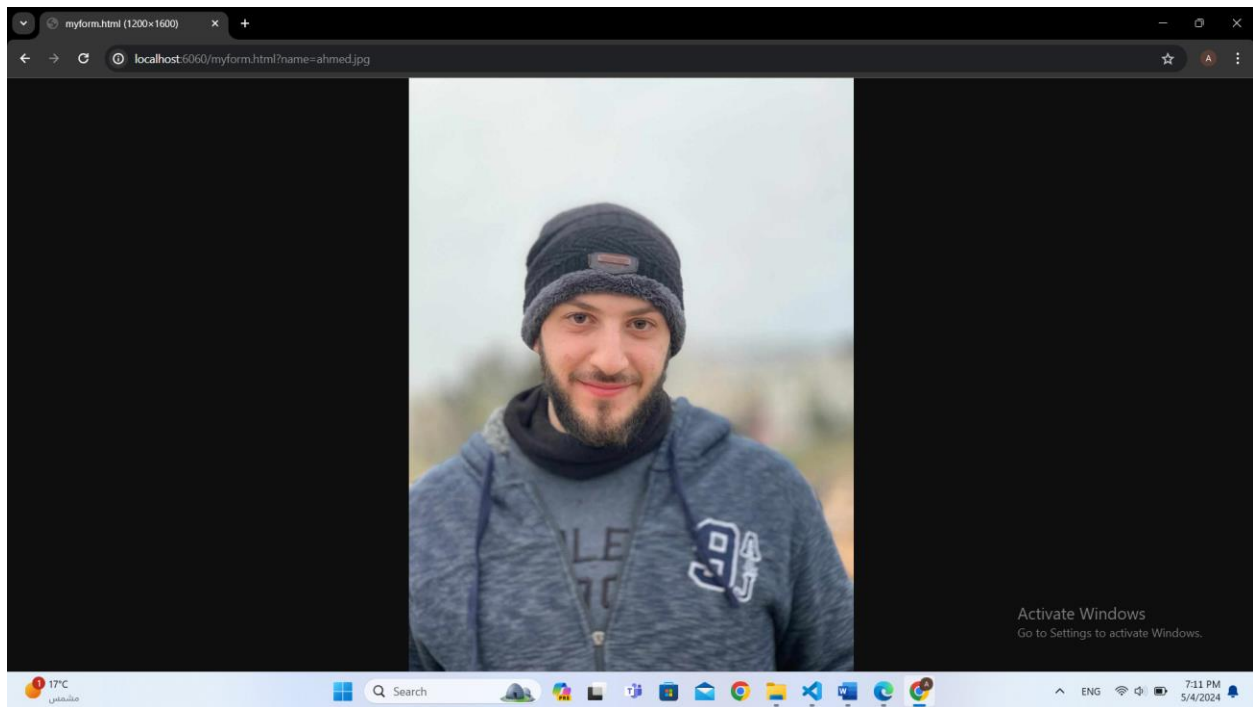


Figure 22: jpg image request

- Itc request:

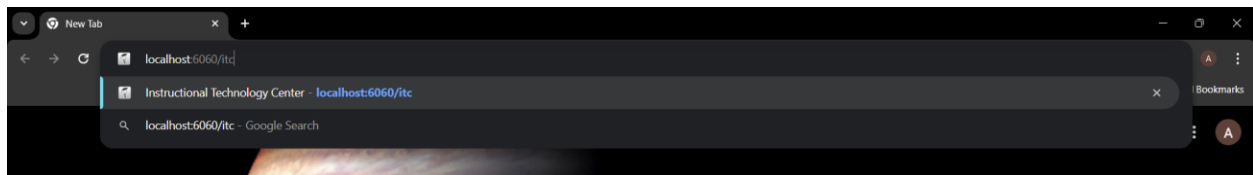


Figure 23: itc request

- Stack overflow request:

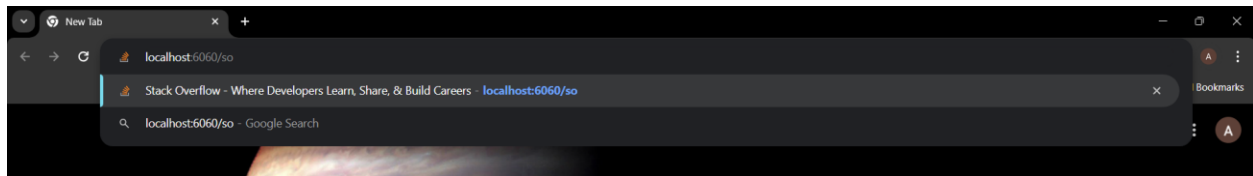


Figure 24: stack overflow request

- Wrong request test:

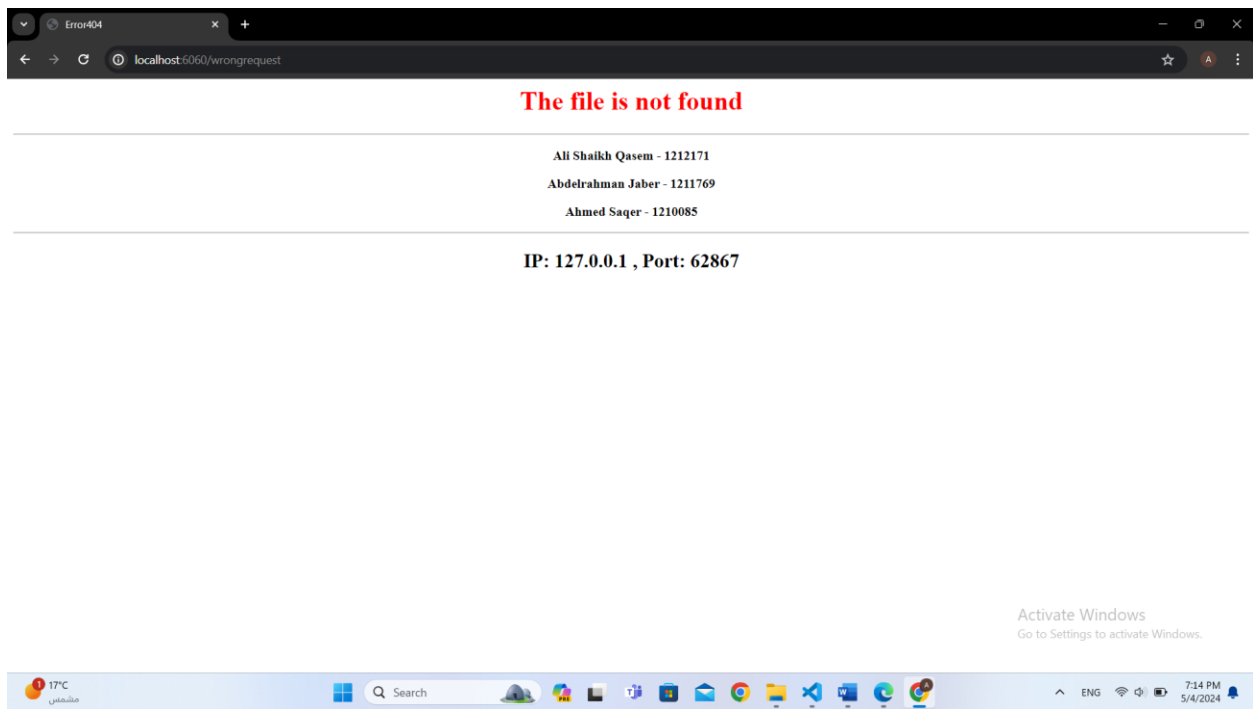


Figure 25: wrong request test



- Opening the page from a phone:

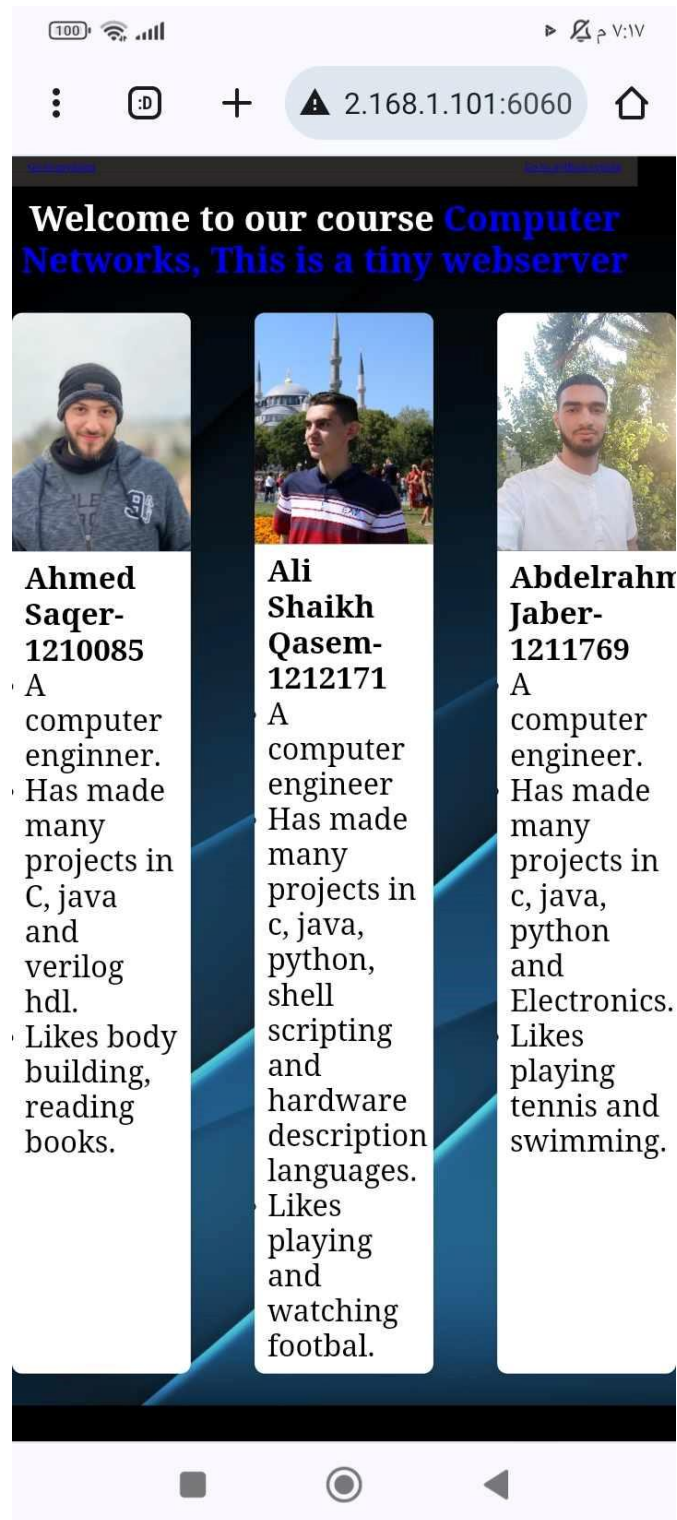


Figure 26: opening the page from a phone

## 4. Conclusion

In this project, we have gained many skills in computer networks principles. First, we have tried some network utilities using command line window such as ping, tracert and nslookup, we also used wire shark software to trace some http and DNS requests and explore their contents. Additionally, in the second section, we developed Python code that enables linked peers to share a broadcast address, enabling simultaneous message sending and receiving for all peers. After that we used socket programming to develop a simple server that keeps listening for user requests and provide them different types of files such as html, css, png and jpg files, the main idea of the server code was splitting the whole http request from user to obtain the exact request and then provide users the appropriate files. The combination of theoretical knowledge and practical experience in this project provided us a valuable learning opportunity and a strong understanding of network principles.