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

Table of figures

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:

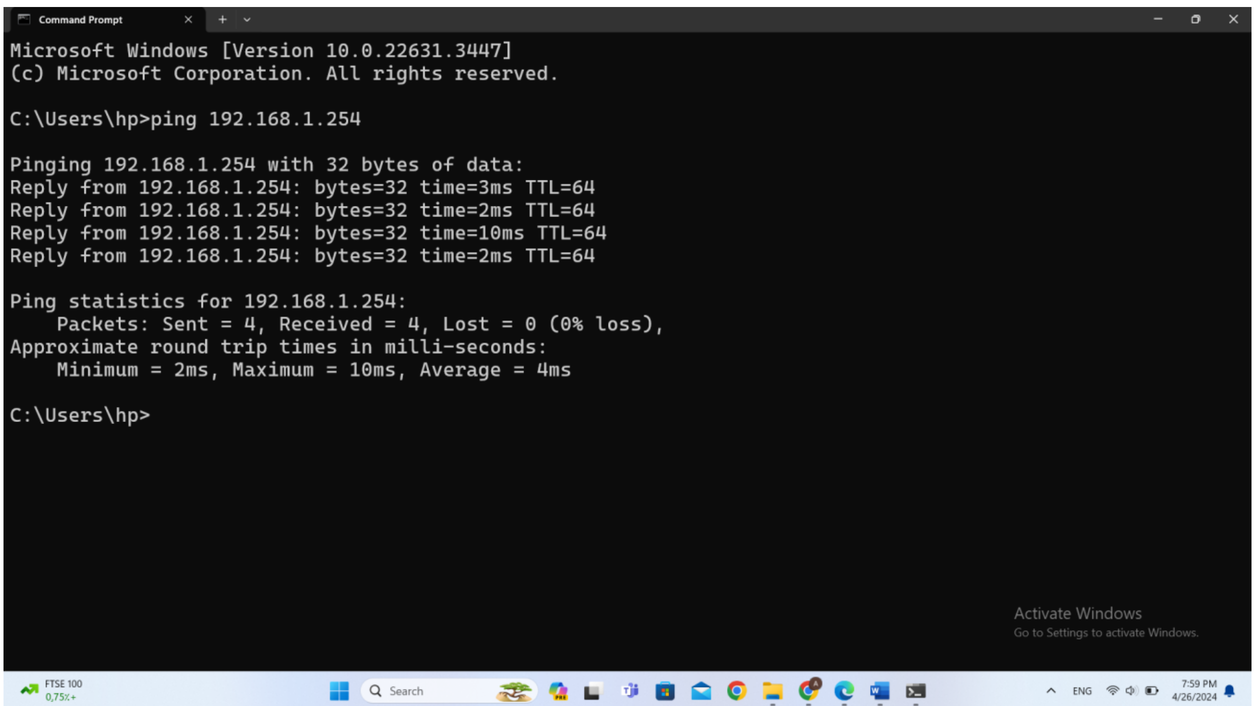

Figure 1: pinging a device from the same network

- 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:

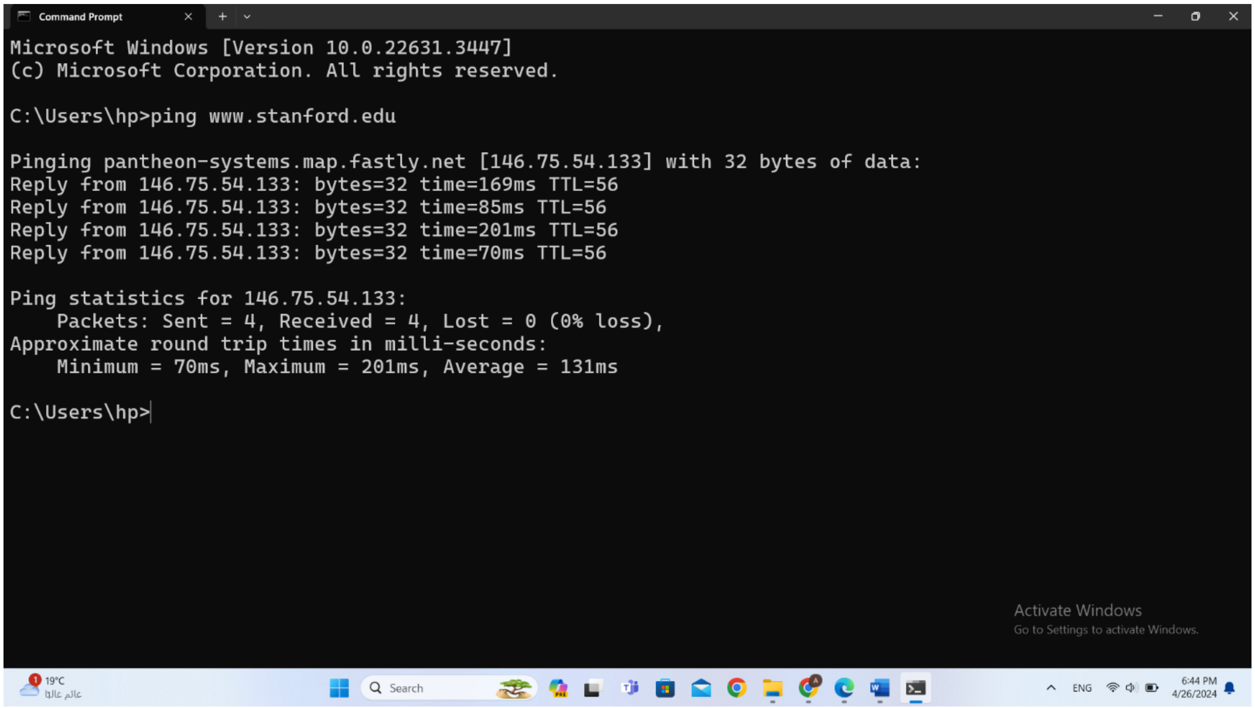

Figure 2: pinging stanford.edu

- 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 msecs with an average of 131 msecs.

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:

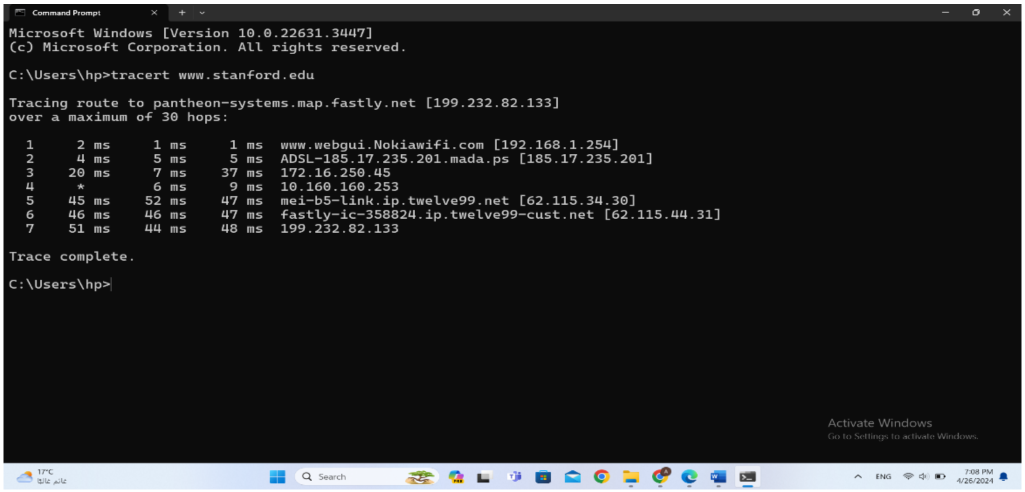

Figure 3: tracert stanford.edu

- 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:

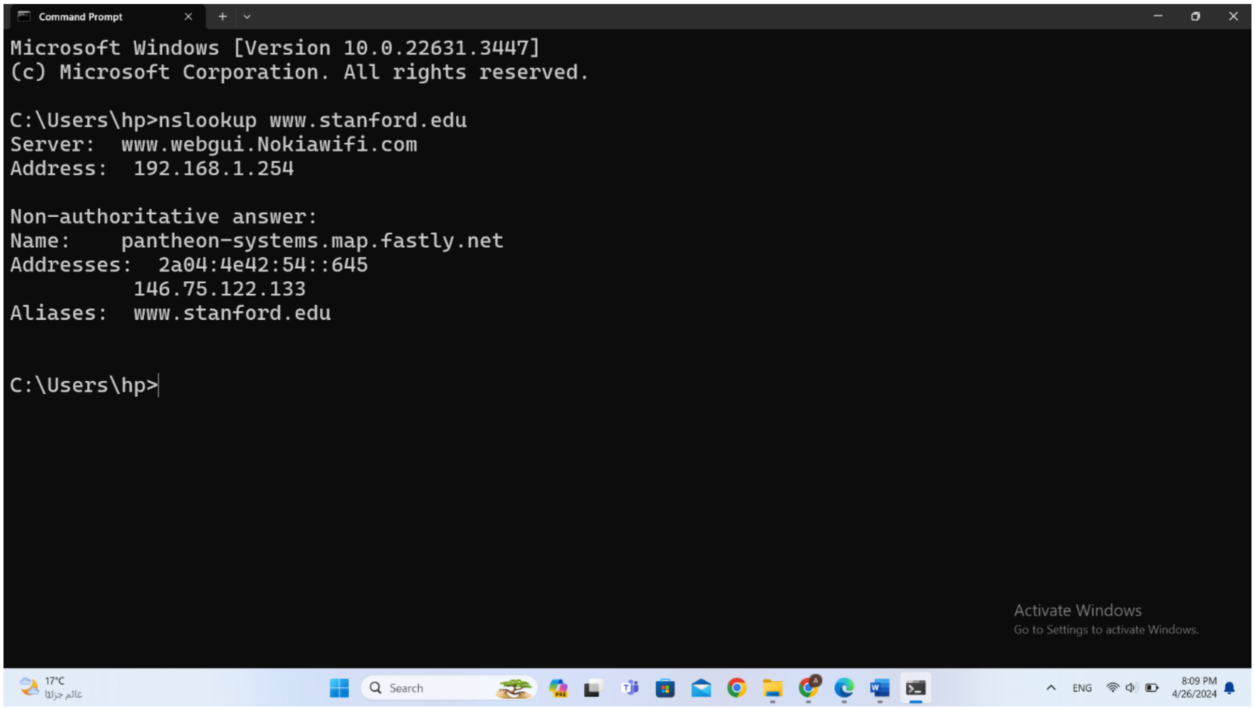

Figure 4: nslookup stanford.edu

- 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:

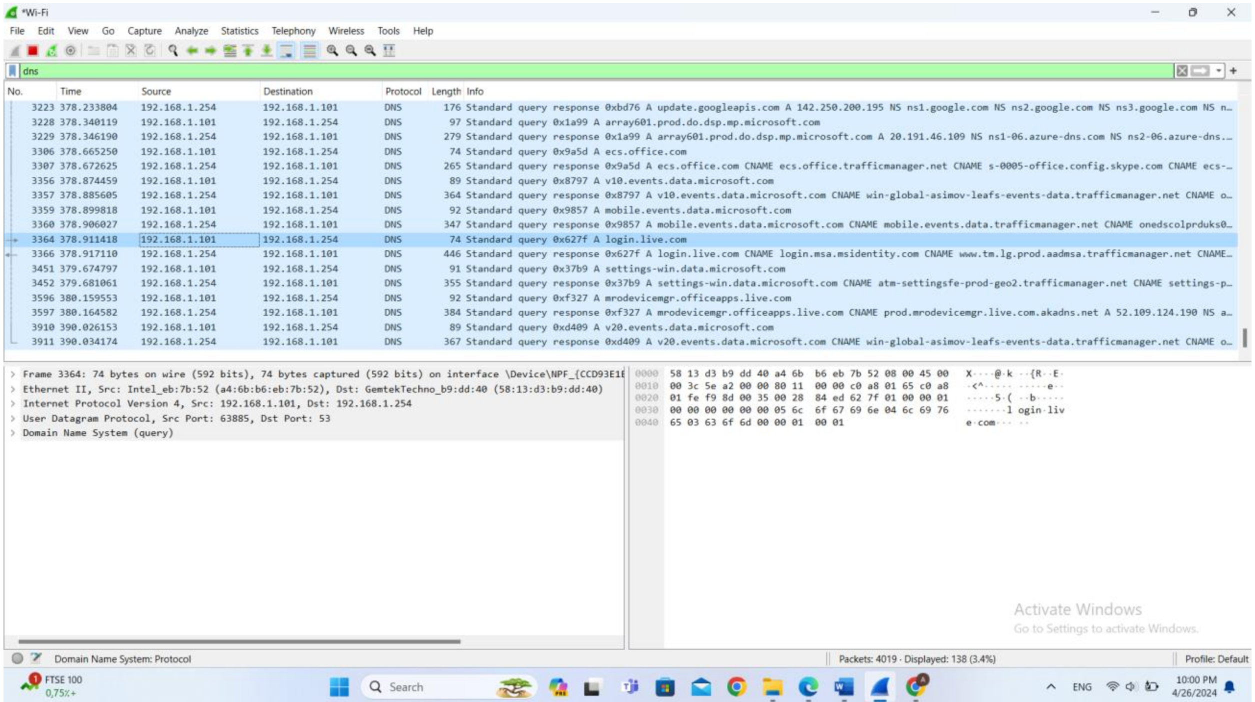

Figure 5: Capturing DNS messages

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.

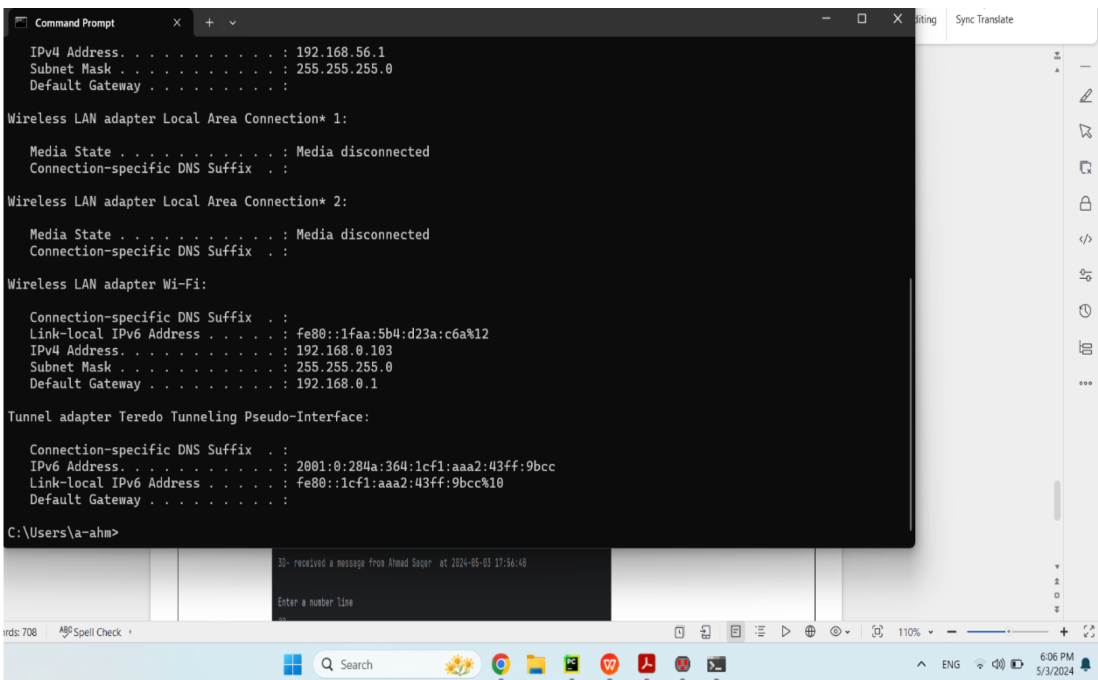

Figure 6 : Calculating boradcast address

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 = "1D"
    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

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

```

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.

```
# handling requests
while True:
    connection_socket, addr = server_socket.accept()
    client_ip = addr[0]
    client_port = addr[1]
    print("Got connection from IP = " + client_ip + "Port = " + str(client_port))
    sentence = connection_socket.recv(1024).decode()
    print("request is: " + sentence )

    # obtain the requset part from the whole received message
    request = sentence.split()[1]
```

Figure 12: obtainning 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 …..)

```python
    if  (request == '/' or request == '/index.html' or request == '/main_en.html' or request == '/en'):
        connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
        connection_socket.send('Content-Type: text/html \r\n'.encode())
        connection_socket.send('\r\n'.encode())
        with open("main_en.html", "rb") as f:
            content = f.read()
            connection_socket.send(content)

    elif  (request == '/ar'):
        connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
        connection_socket.send('Content-Type: text/html \r\n'.encode())
        connection_socket.send('\r\n'.encode())
        with open("main_ar.html", "rb") as f:
            content = f.read()
            connection_socket.send(content)

    elif  ( request.endswith('.html')):
        file_name = request[1:] # to remove the '/' cahracter
        try:
            with open(file_name, "rb") as f:
                connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
                connection_socket.send('Content-Type: text/html \r\n'.encode())
                connection_socket.send('\r\n'.encode())
                content = f.read()
                connection_socket.send(content)
        except FileNotFoundError:
            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).

```python
    elif  ( request.endswith('.css')):
        file_name = request[1:] # to remove the '/' cahracter
        try:
            with open(file_name, "rb") as f:
                connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
                connection_socket.send('Content-Type: text/css \r\n'.encode())
                connection_socket.send('\r\n'.encode())
                content = f.read()
                connection_socket.send(content)
        except FileNotFoundError:
            send_error_file(connection_socket, client_ip, client_port)

    elif  ( request.endswith('.png') and not request.startswith('/myform.html?name=')):
        file_name = "images/"+ request[1:] # to remove the '/' cahracter
        try:
            with open(file_name, "rb") as f:
                connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
                connection_socket.send('Content-Type: image/png \r\n'.encode())
                connection_socket.send('\r\n'.encode())
                content = f.read()
                connection_socket.send(content)
        except FileNotFoundError:
            send_error_file(connection_socket, client_ip, client_port)

    elif  ( request.endswith('.jpg') and not request.startswith('/myform.html?name=')):
        file_name ="images/"+ request[1:] # to remove the '/' cahracter
        try:
            with open(file_name, "rb") as f:
                connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
                connection_socket.send('Content-Type: image/jpg \r\n'.encode())
                connection_socket.send('\r\n'.encode())
                content = f.read()
                connection_socket.send(content)
        except FileNotFoundError:
            send_error_file(connection_socket, client_ip, client_port)
```

Figure 14: code part2

```python
    elif (request.endswith('.png')):
        # Split the string using "=" as a delimiter
        parts = request.split("=")
        # Get the second part
        request = parts[1]
        try:
            with open('images/' + request, "rb") as f:
                connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
                connection_socket.send('Content-Type: image/png; charset=utf-8\r\n'.encode())
                connection_socket.send('\r\n'.encode())
                content = f.read()
                connection_socket.send(content)
        except FileNotFoundError:
            send_error_file(connection_socket, client_ip, client_port)

    elif (request.endswith('.jpg')):
        # Split the string using "=" as a delimiter
        parts = request.split("=")
        # Get the second part
        request = parts[1]
        try:
            with open('images/' + request, "rb") as f:
                connection_socket.send('HTTP/1.1 200 OK \r\n'.encode())
                connection_socket.send('Content-Type: image/jpg; charset=utf-8\r\n'.encode())
                connection_socket.send('\r\n'.encode())
                content = f.read()
                connection_socket.send(content)
        except FileNotFoundError:
            send_error_file(connection_socket, client_ip, client_port)

```

Figure 16:code part3

```python

    elif  (request == '/so'):
        connection_socket.send('HTTP/1.1 307 Temporary Redirect \r\n'.encode())
        connection_socket.send('Content-Type: text/html \r\n'.encode())
        connection_socket.send('Location: https://stackoverflow.com/ \r\n'.encode())

    elif  (request == '/itc'):
        connection_socket.send('HTTP/1.1 307 Temporary Redirect \r\n'.encode())
        connection_socket.send('Content-Type: text/html \r\n'.encode())
        connection_socket.send('Location: https://itc.birzeit.edu/ \r\n'.encode())

    else:
        send_error_file(connection_socket, client_ip, client_port)
    # close the connection
    connection_socket.close()
```

Figure 15:code part4

3.5. Testing the code

- Http request in the command line:

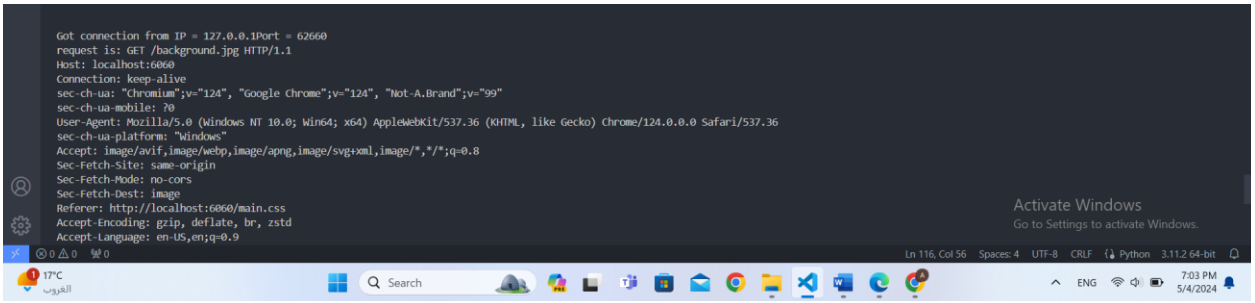

Figure 17: http request at command line

- English page request (/en or /index.html or /main_en.html or /):

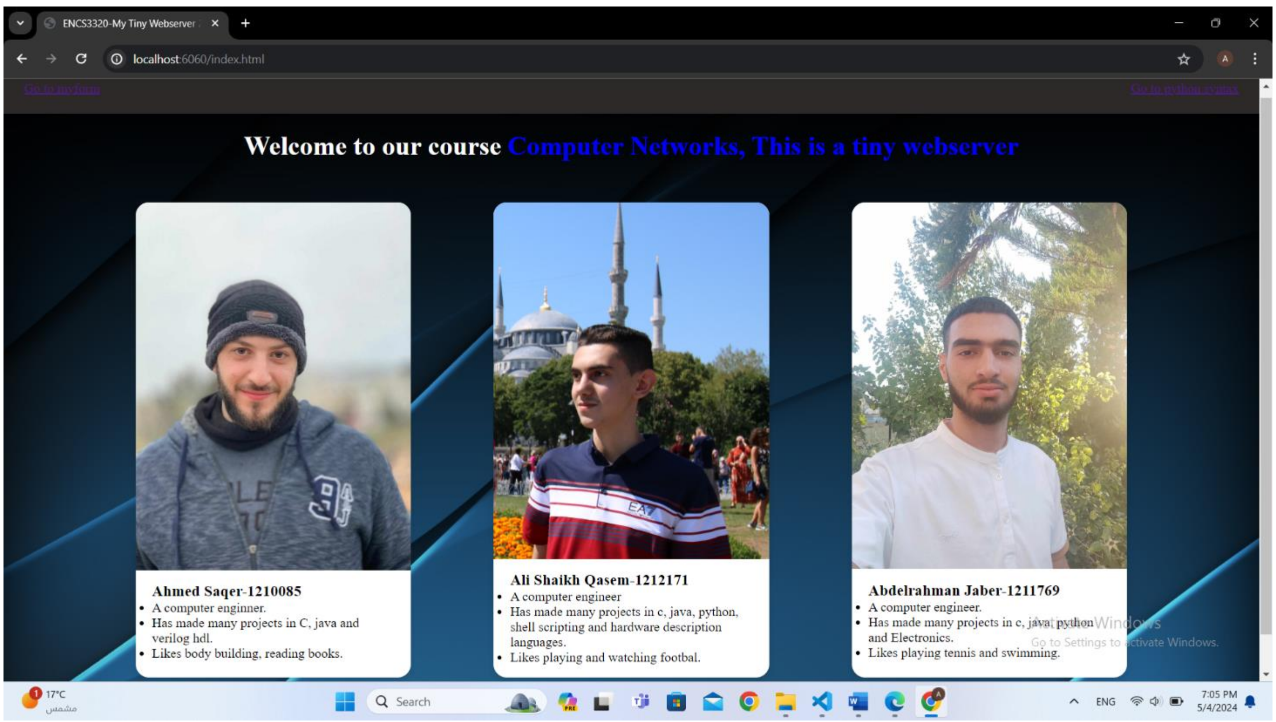

Figure 18: english page

- Arabic page request(/ar):

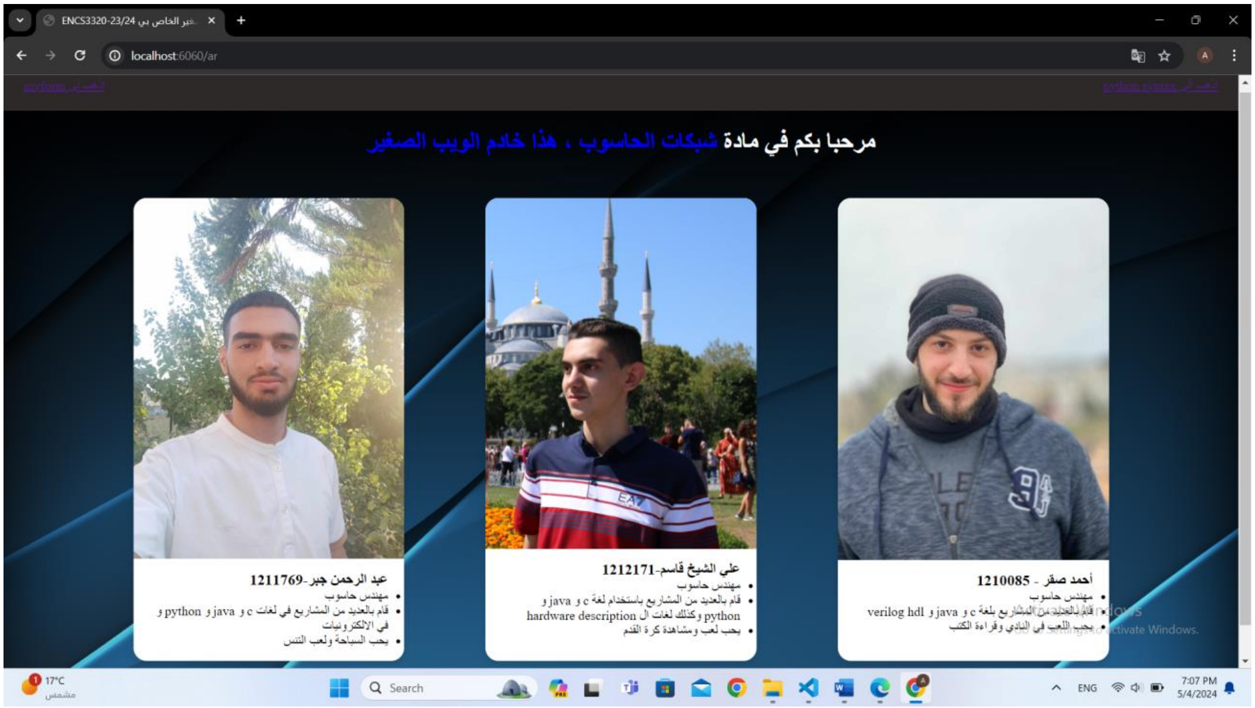

Figure 19: Arabic page

- css file request:

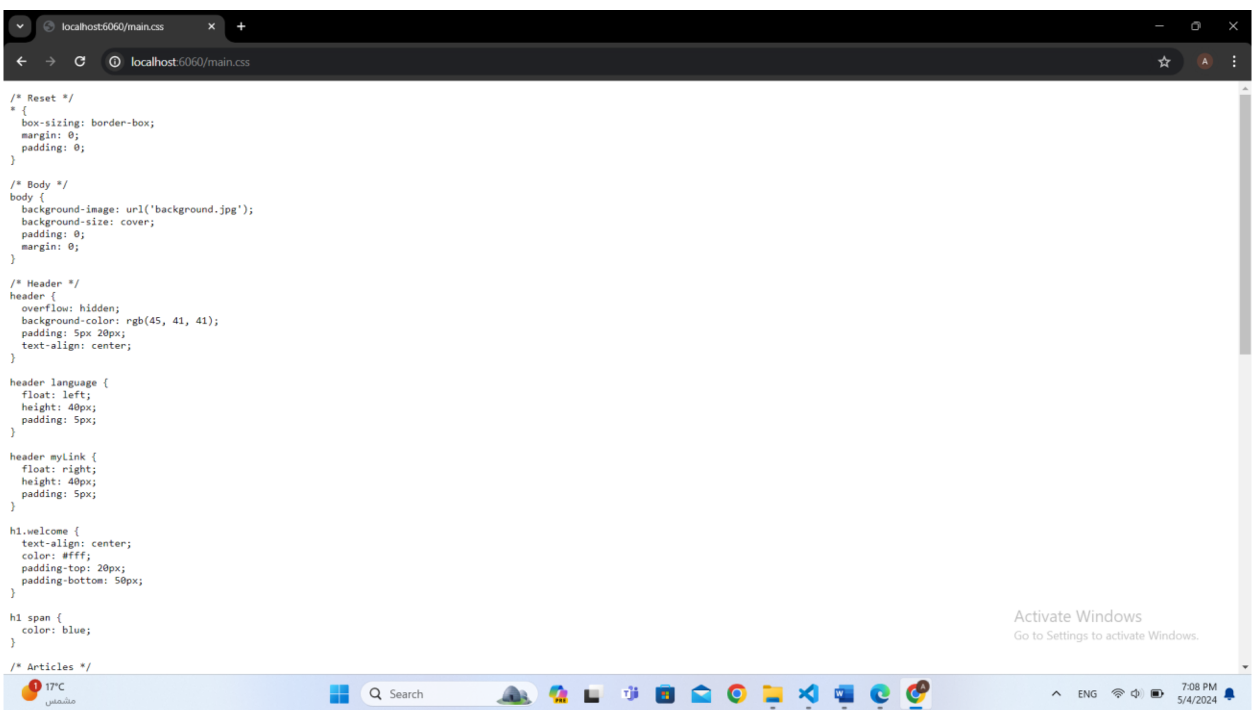

Figure 20: Css file request

- Png image request (from basic browser):

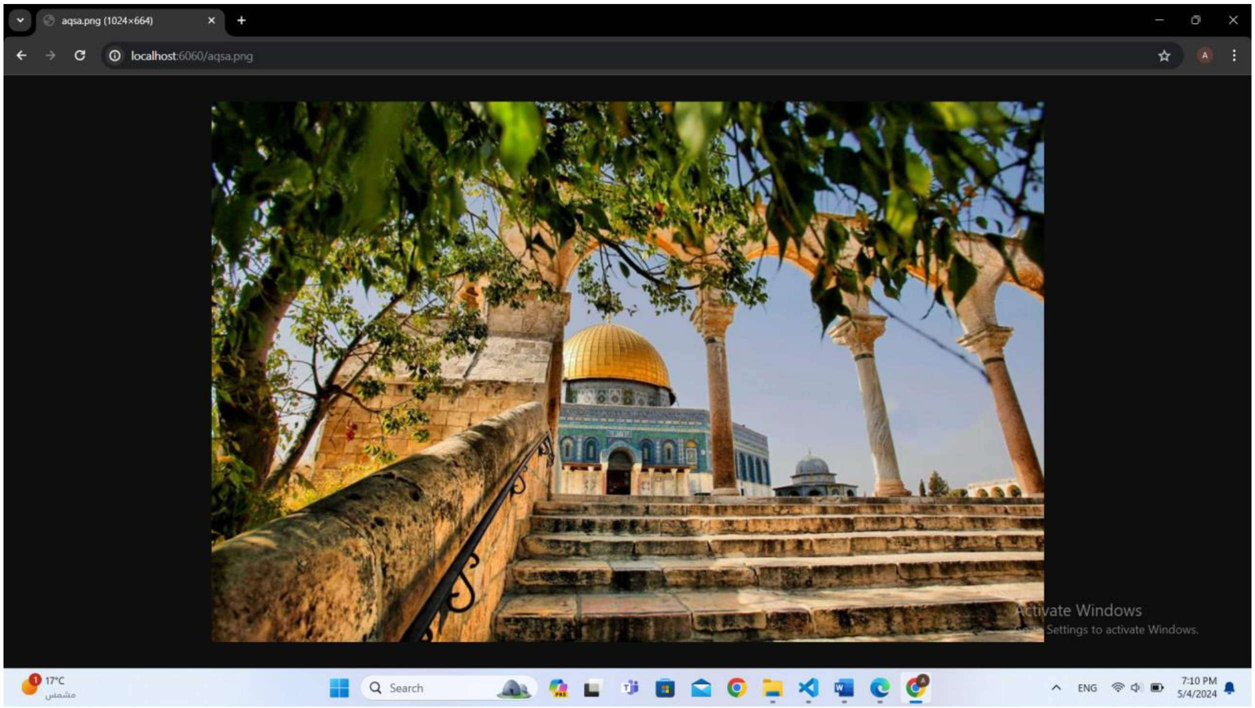

Figure 21: png image request

- Jpg image request (from myform.html):

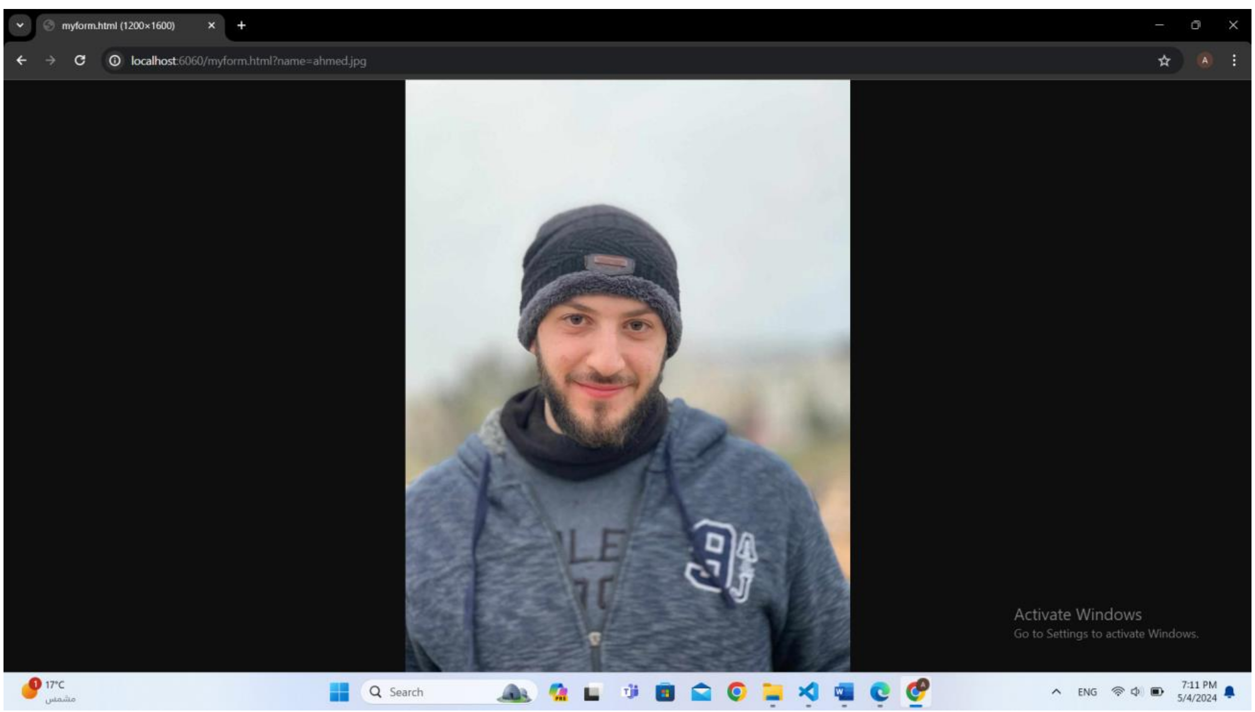

Figure 22: jpg image request

- Itc request:

Figure 23: itc request

- Stack overflow request:

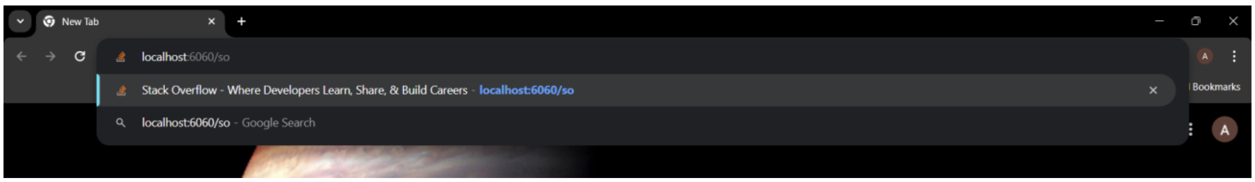

Figure 24: stack overflow request

- Wrong request test:

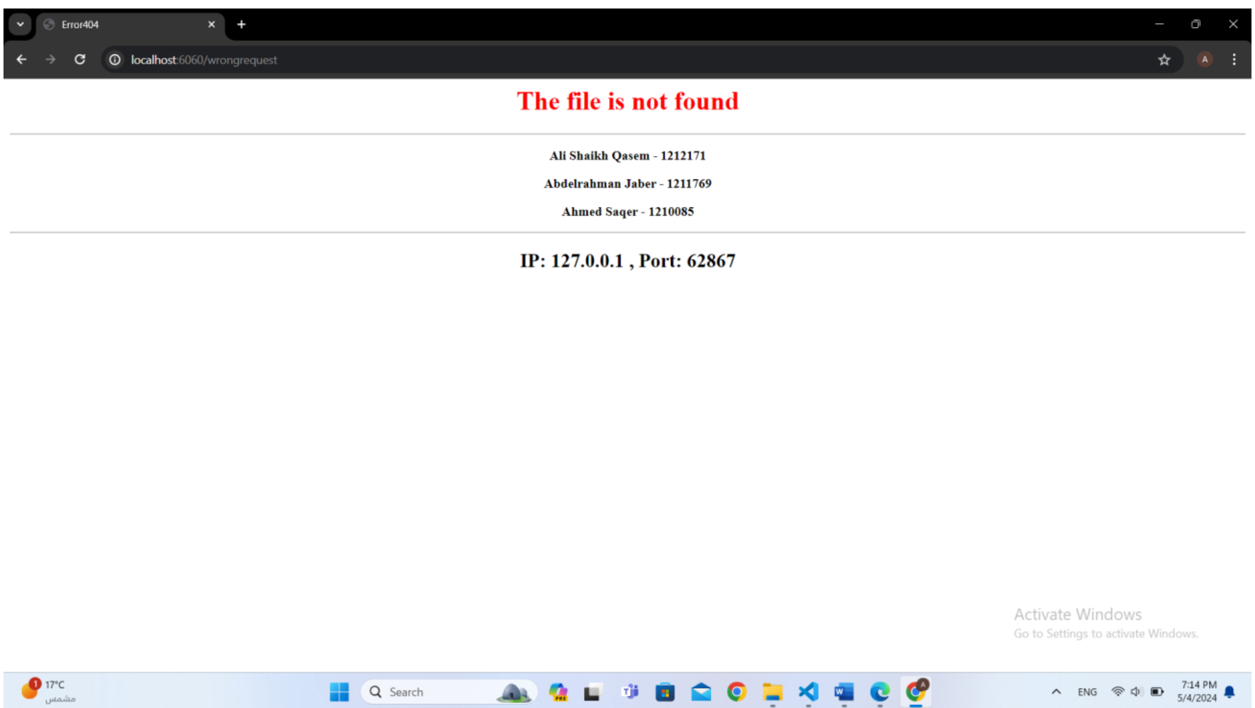

Figure 25: wrong request test

- Opening the page from a phone:

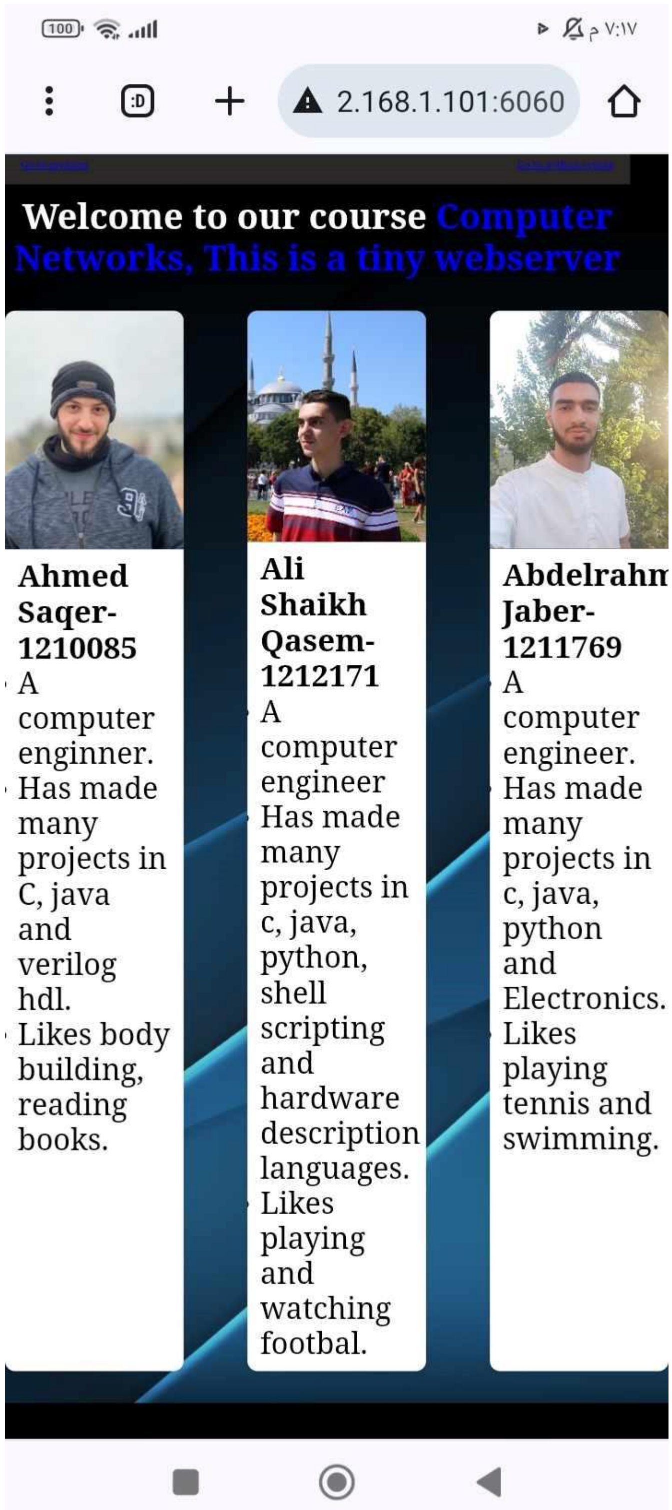

Figure 26: opening the page from a phone

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.