



Department of Electrical and Computer Engineering
ENCS3320 | Computer Networks

Project #1 Report

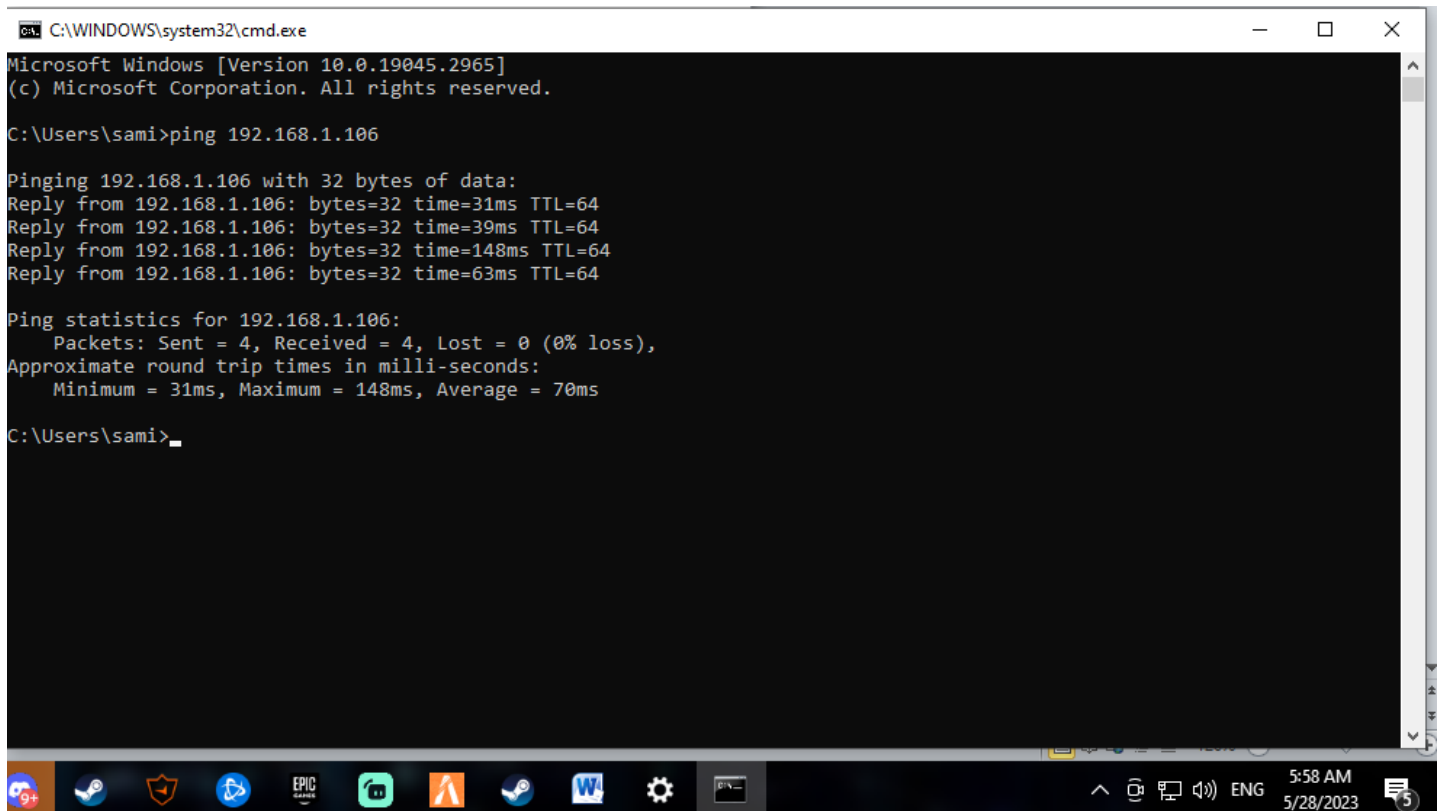
Students:

Abd Khuffash	—	1200970
Sami Moqbel	—	1200751
Saad Alahmad	—	1190326
Raed Al-Adhami	—	1180052

Instructor: Dr. Abdalkarim Awad

Part 1

1. **Ping:** A command that shows how much time it takes to transfer data between the client and the host across network (Total Time).
2. **Tracert (trace route):** A command that shows the path that every packet take and the amount of time it takes whereas the last time is same as Ping.
3. **Nslookup (name server lookup):** A command that shows the DNS of a specific domain or IP.
4. **Telnet:** A command that enables remote access to servers or devices over a network, allowing users to control and manage them through a text-based command-line interface, as if they were physically present.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sami>ping 192.168.1.106

Pinging 192.168.1.106 with 32 bytes of data:
Reply from 192.168.1.106: bytes=32 time=31ms TTL=64
Reply from 192.168.1.106: bytes=32 time=39ms TTL=64
Reply from 192.168.1.106: bytes=32 time=148ms TTL=64
Reply from 192.168.1.106: bytes=32 time=63ms TTL=64

Ping statistics for 192.168.1.106:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 31ms, Maximum = 148ms, Average = 70ms

C:\Users\sami>
```

1) ping 192.168.1.106

The command pings the destination (in this case it is a device connected to the same network “192.168.1.106”) once every second, and prints one line of output for every response received.

It shows a brief summary when complete, and it calculates the round-trip for each packet, and the packet loss percentage.

```
C:\WINDOWS\system32\cmd.exe

C:\Users\sami>ping www.harvard.edu

Pinging pantheon-systems.map.fastly.net [199.232.82.133] with 32 bytes of data:
Reply from 199.232.82.133: bytes=32 time=73ms TTL=53
Reply from 199.232.82.133: bytes=32 time=73ms TTL=53
Reply from 199.232.82.133: bytes=32 time=70ms TTL=53
Reply from 199.232.82.133: bytes=32 time=70ms TTL=53

Ping statistics for 199.232.82.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 70ms, Maximum = 73ms, Average = 71ms

C:\Users\sami>
```

2) ping www.harvard.edu

The destination here is “www.harvard.edu”

Packet Information:

4 Packets Sent, 4 Packets Received, 0% loss

Round-trip Information:

Minimum time = 70ms

Maximum time = 73ms

Average time = 71ms

```
C:\WINDOWS\system32\cmd.exe

C:\Users\sami>tracert www.harvard.edu

Tracing route to pantheon-systems.map.fastly.net [199.232.82.133]
over a maximum of 30 hops:

  1  <1 ms    <1 ms    <1 ms    mada-alarab.ps [192.168.1.254]
  2   4 ms     2 ms     3 ms     ADSL-185.17.235.200.mada.ps [185.17.235.200]
  3   6 ms     4 ms     2 ms     172.16.250.57
  4   1 ms     5 ms     2 ms     172.16.250.1
  5  53 ms    54 ms    56 ms    ae0-165.cr3-fra2.ip4.gtt.net [77.67.93.9]
  6  63 ms    56 ms    61 ms    ae1.cr7-fra2.ip4.gtt.net [89.149.137.14]
  7  55 ms    55 ms    56 ms    ip4.gtt.net [46.33.83.254]
  8  62 ms    56 ms    56 ms    ae-2.r21.frnkge13.de.bb.gin.ntt.net [129.250.6.41]
  9  69 ms    76 ms    63 ms    ae-12.r21.parsfr04.fr.bb.gin.ntt.net [129.250.5.27]
 10  65 ms    64 ms    72 ms    ae-0.r20.parsfr04.fr.bb.gin.ntt.net [129.250.3.47]
 11  71 ms    71 ms    73 ms    ae-17.r00.mrs1fr01.fr.bb.gin.ntt.net [129.250.3.39]
 12  75 ms    72 ms    72 ms    82.112.96.234
 13  73 ms    70 ms    73 ms    199.232.82.133

Trace complete.

C:\Users\sami>
```

3) tracert www.harvard.edu

(Trace Route)

Every line in the output shows the route the request takes until it reaches the destination “www.harvard.edu”, and it displays the round-trip times between the source and the desired destination and also for every point in between. It sends a packet to each router and gives information about the time it takes to reach and come back to the client until it reaches to the destination it will be the same as ping command. (E.g., Harvard ping was 71, and in the above screenshot 73).

```
C:\WINDOWS\system32\cmd.exe

C:\Users\sami>nslookup www.harvard.edu
Server: mada-alarab.ps
Address: 192.168.1.254

Non-authoritative answer:
Name:   pantheon-systems.map.fastly.net
Addresses:  2a04:4e42:54::645
            199.232.82.133
Aliases:  www.harvard.edu

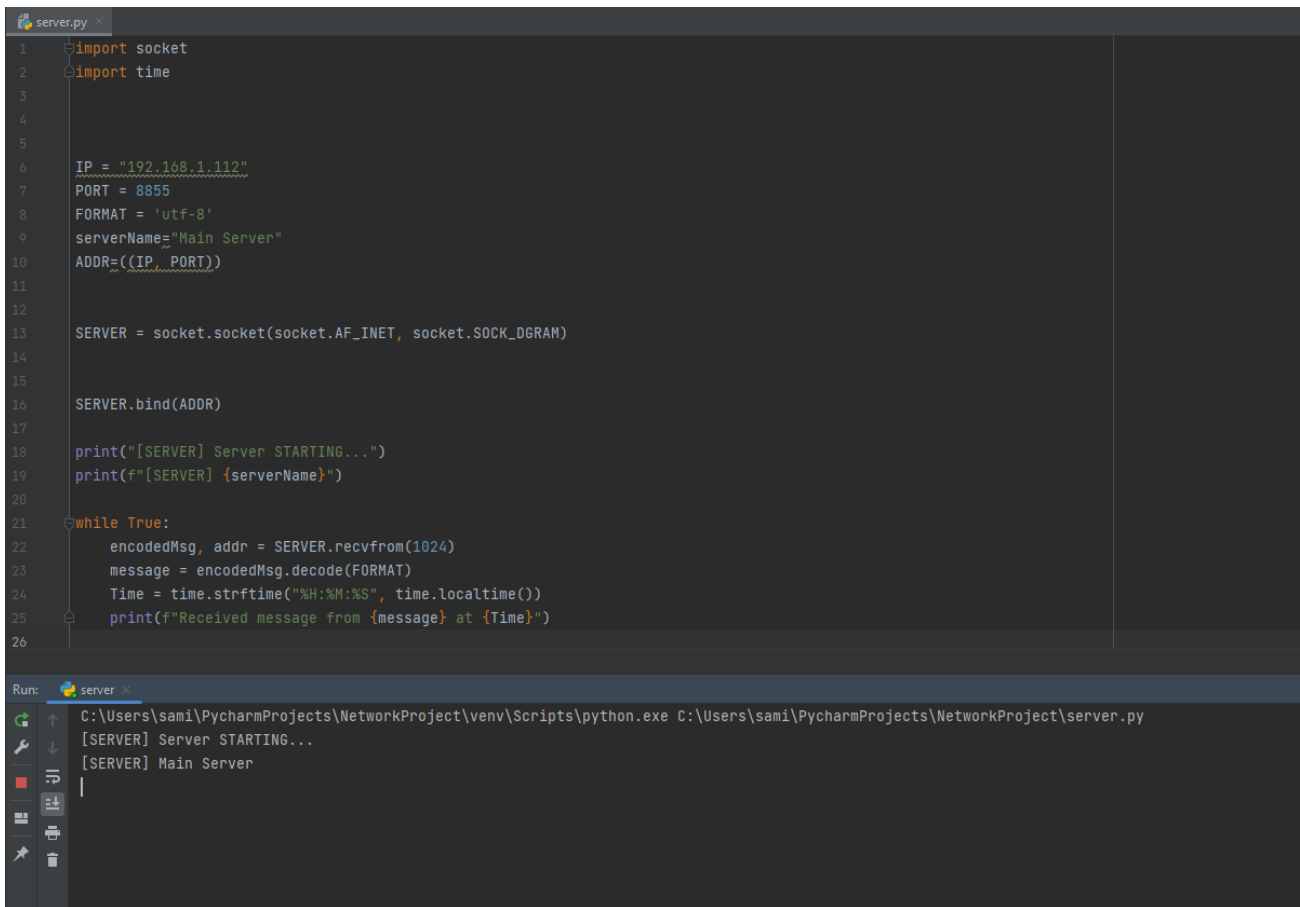
C:\Users\sami>
```

4) nslookup www.harvard.edu

(Name Server Lookup)

The command is used to send queries to a DNS server for DNS records about a specific domain (here we used “www.harvard.edu”) to obtain information about it. The first thing the command sends is the local DNS server in the network and the gateway address of the used device (mada-alarab.ps, 192.168.1.254) and it can be changed by using the command server in the nslookup program (E.g., server 1.1.1.1), then it sends information about the specified URL, including names, and public addresses (IPV4 and IPV6).

Part 2



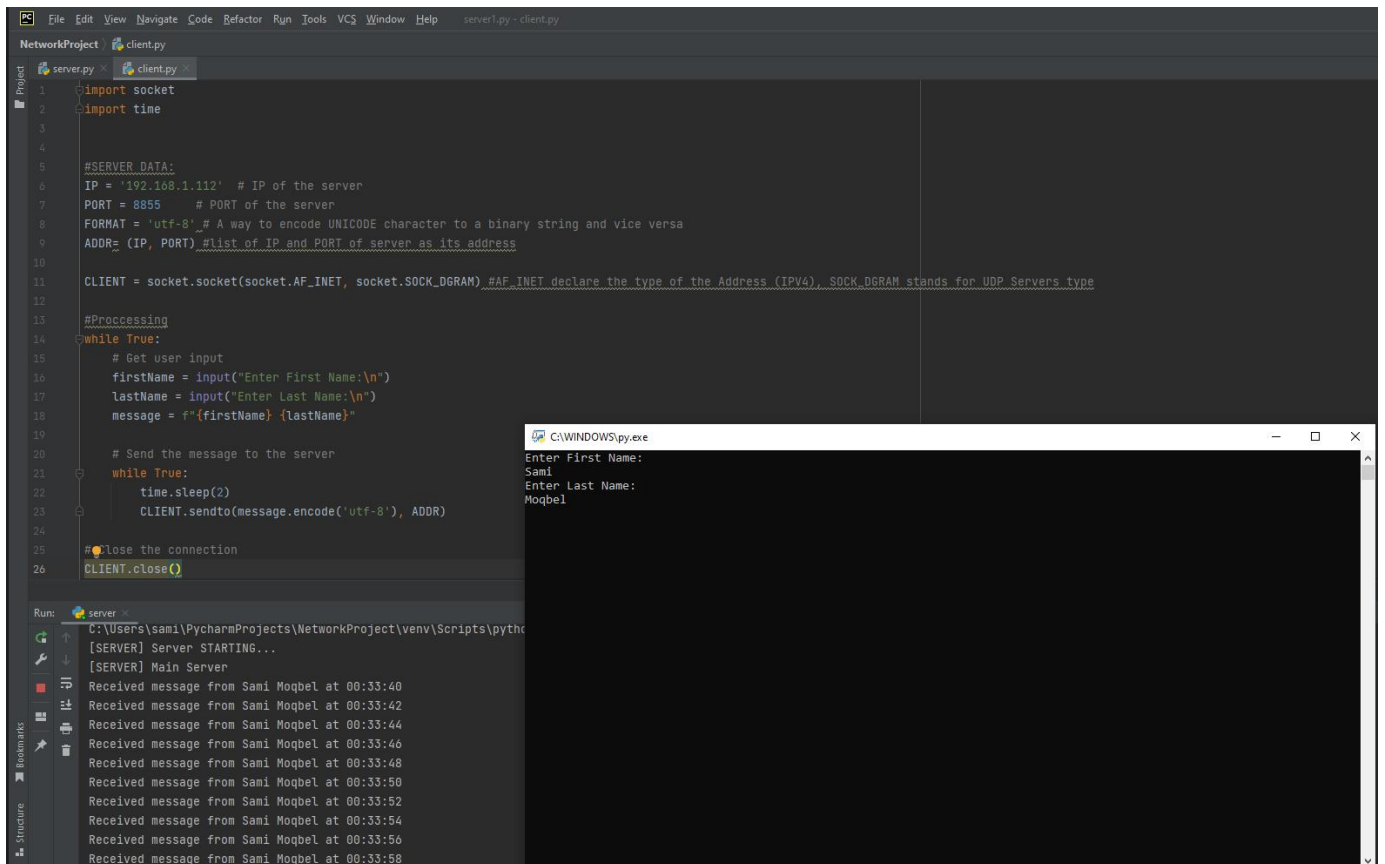
```
server.py
1 import socket
2 import time
3
4
5
6 IP = "192.168.1.112"
7 PORT = 8855
8 FORMAT = 'utf-8'
9 serverName="Main Server"
10 ADDR=((IP, PORT))
11
12
13 SERVER = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
14
15
16 SERVER.bind(ADDR)
17
18 print("[SERVER] Server STARTING...")
19 print(f"[SERVER] {serverName}")
20
21 while True:
22     encodedMsg, addr = SERVER.recvfrom(1024)
23     message = encodedMsg.decode(FORMAT)
24     Time = time.strftime("%H:%M:%S", time.localtime())
25     print(f"Received message from {message} at {Time}")
26
```

Run: server

```
C:\Users\sami\PycharmProjects\NetworkProject\venv\Scripts\python.exe C:\Users\sami\PycharmProjects\NetworkProject\server.py
[SERVER] Server STARTING...
[SERVER] Main Server
|
```

The server runs on python created using socket programming, the port it listens to is “8855”, the format “utf-8” is used for decoding and encoding, and the server’s name is “Main Server”.

The server receives a message from the client every two seconds, the message contains the student's name, and the server sends the last message it receives from any client and the time of that message.



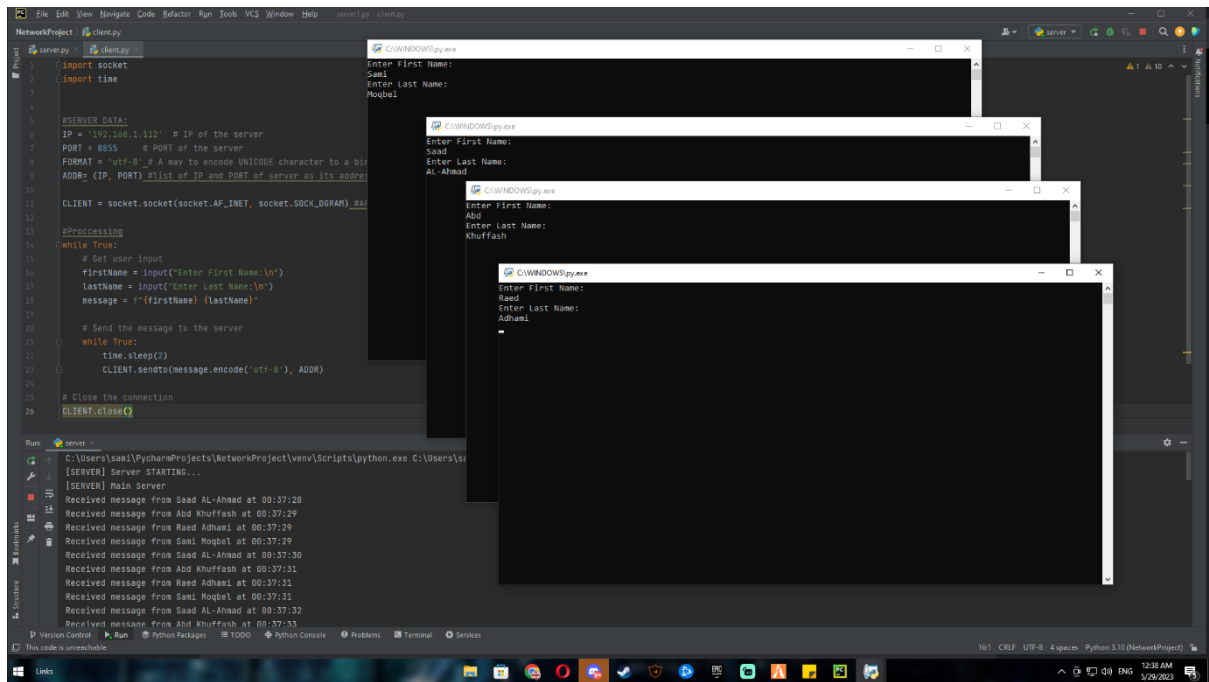
The screenshot displays the PyCharm IDE interface. The main editor window shows the code for `client.py`. The code imports `socket` and `time`, defines server data (IP, PORT, FORMAT, ADDR), creates a `CLIENT` socket, and enters a `while True` loop. Inside the loop, it prompts the user for their first and last names, constructs a message, and sends it to the server every 2 seconds using `CLIENT.sendto(message.encode('utf-8'), ADDR)`. The `CLIENT.close()` method is called at the end of the loop.

Below the editor, the `Run` tab shows the output of the server. It indicates that the server is starting and then receives multiple messages from 'Sami Moqbel' at various timestamps, confirming that the client is sending data to the server.

Overlaid on the bottom right is a terminal window titled `C:\WINDOWS\py.exe`. It shows the user input for the client: 'Enter First Name: Sami' and 'Enter Last Name: Moqbel'.

(The server is running with one client)

The client Asks the user for his first name and last name, and takes them as an input, and sends it to the server every 2 seconds (the server is running in the background in PyCharm IDE, and the client is the python cmd window).



(The server running with 4 clients)

The server shows the last message it receives to all the clients, with the time it received it.

Part 3

Created a server using TCP, and defined its port.

```
1 from socket import *
2 from jinja2 import *
3
4 serverPort = 9977
5 serverSocket = socket(AF_INET, SOCK_STREAM) #TCP Socket
6 #serverSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
7 serverSocket.bind(('localhost', serverPort)) #server socket binded with server port
8 serverSocket.listen(1) #listen for 1 tcp connectio request. Server can handle atleast one queued connection at a time
9
10
11 print("The server is ready to receive")
12 while True:
13     connectionSocket, addr= serverSocket.accept() #clients sends tcp conn request-> connectionsocket made
14     sentence = connectionSocket.recv(2048).decode()
15     print(addr)
16     print(sentence)
17
18     ip = addr[0]
19     port = addr[1]
20
21     #this to write on the html , using jinja2 library; will read the Error_temp.html ,will recognize the ip/port,
22     # and then write on Error.html ip/port
23
24     with open('Error_temp.html', 'r') as file:
25         Error_content = file.read()
26         template = Template(Error_content)
27         variables = {
28             'port': port,
29             'ip': ip
30         }
```

serverSocket.listen(1) defines the length of the backlog queue, it is the number of incoming connections that have been completed by the TCP stack, but not yet accepted by the application, here we have one, and if one more connection makes a request before we call accept, that connection will get dropped.

The server's console prints "The server is ready to receive" when the server is established.

```
    }
    rendered_template = template.render(**variables)
    #print(rendered_template)
    with open("Error.html", 'w') as file:
        file.write(rendered_template)

    #request=sentence.split()[1] #splitting the sentence and accessing second word
    words = sentence.split()

    # Check if the list has at least two elements before accessing the second element
    if len(words) >= 2:
        request = words[1]
        #print("Request:", request)
    else:
        print("Invalid sentence format")
        break;

    print(f"HTTP Request: {request}")
```

```

if (request == '/' or request == '/main-en.html' or request=='/en' or request=='/index.html'): # English ; / /main-en.html
# /en /index.html will result the same html
connectionSocket.send("HTTP/1.1 200 OK \r\n".encode()) #http response status "200 ok"
connectionSocket.send("Content-Type: text/html \r\n".encode()) #send html file text/html
connectionSocket.send("\r\n".encode()) #sends the bytes b'\r\n' through the connectionSocket,
# used to send line ending sequence to indicate the end of a request or a section of data in a network protocol.
f = open("main-en.html", "rb") #loading main-en.html//
#allows you to read the content of the file a binary data^
connectionSocket.send(f.read())

```

The server checks if the request is any of the cases listed above, if the request is “/” or “/main_en.html” or “/index.html” or “/en”, the server sends the same html file which is “main_en.html”.

ENCS3320-My Tiny Webserver

Welcome to our course Computer Networks, This is a tiny webserver

Group Members:

- Abd Khoufash (ID: 1200970)
- Sami Moudel (ID: 1200761)
- Saad AlAhmad (ID: 1160326)
- Raed Al-Ashari (ID: 1160362)

Group Members Information:
 Abd Khoufash: Born in Saffi, 20 years old, Responsible, Administrative, Likes playing cards and Chess, Done Multiple projects, Residential building system.
 Sami Moudel: Born in Ramallah, 20 years old, Multitasking, good listener, Likes playing soccer and tennis, Done Multiple projects, Media Rental System.
 Saad AlAhmad: Born in Ramallah, 21 years old, Fast learner, leadership, Likes playing Video games and football, Done Multiple projects, Library Management System.
 Raed Saad: Born in Ramallah, 22 years old, Problem Solver, create, Likes playing Volleyball and Basketball, Done Multiple projects, Turkey.

Images:



Links:

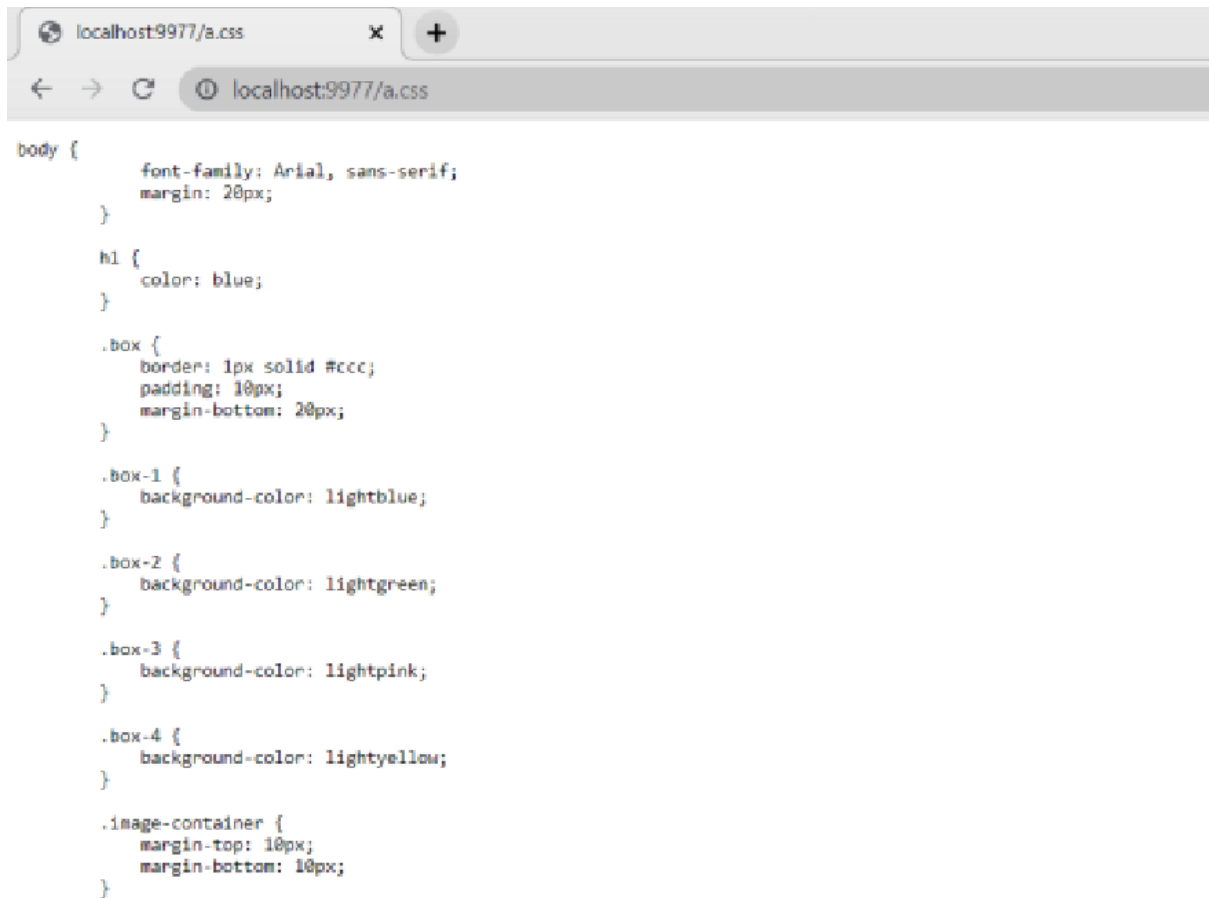
- [Local HTML File](#)
- [Python Multi-Line Strings](#)

‘main_en.html’ contains the title “ENCS332-My Tiny Webserver”, A welcome message, the group members names and IDs, more information about group members like projects and skills, an image with extension ‘.jpg’ and another one with extension ‘.png’, a link to a local html file, and a link to w3schools Python Multiline Strings page.

Links:

- [Local HTML File](#)
- [Python Multi-Line Strings](#)

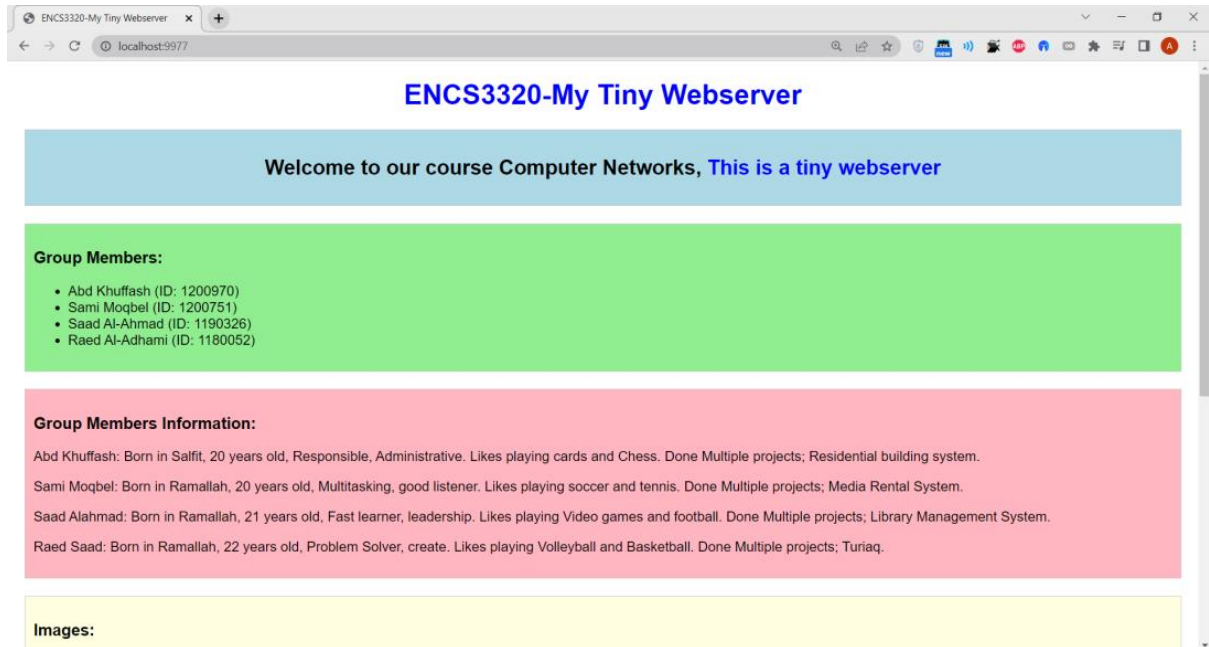
The CSS is in a separate file ‘a.css’:



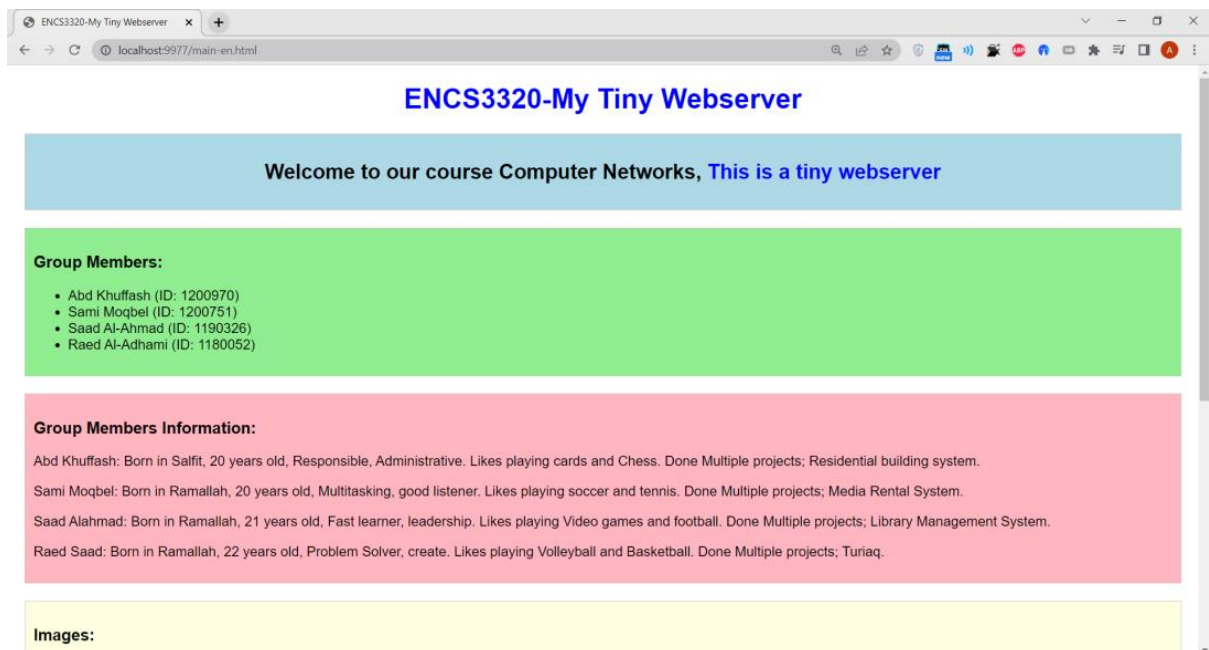
```
body {  
    font-family: Arial, sans-serif;  
    margin: 20px;  
}  
  
h1 {  
    color: blue;  
}  
  
.box {  
    border: 1px solid #ccc;  
    padding: 10px;  
    margin-bottom: 20px;  
}  
  
.box-1 {  
    background-color: lightblue;  
}  
  
.box-2 {  
    background-color: lightgreen;  
}  
  
.box-3 {  
    background-color: lightpink;  
}  
  
.box-4 {  
    background-color: lightyellow;  
}  
  
.image-container {  
    margin-top: 10px;  
    margin-bottom: 10px;  
}
```

Defined some of text in the welcome message on the main html page as ‘hi’ and used it in css to change its color to blue, then added a solid border to each of the boxes in the page, defined a padding and a margin value to them, and gave each of them a color.

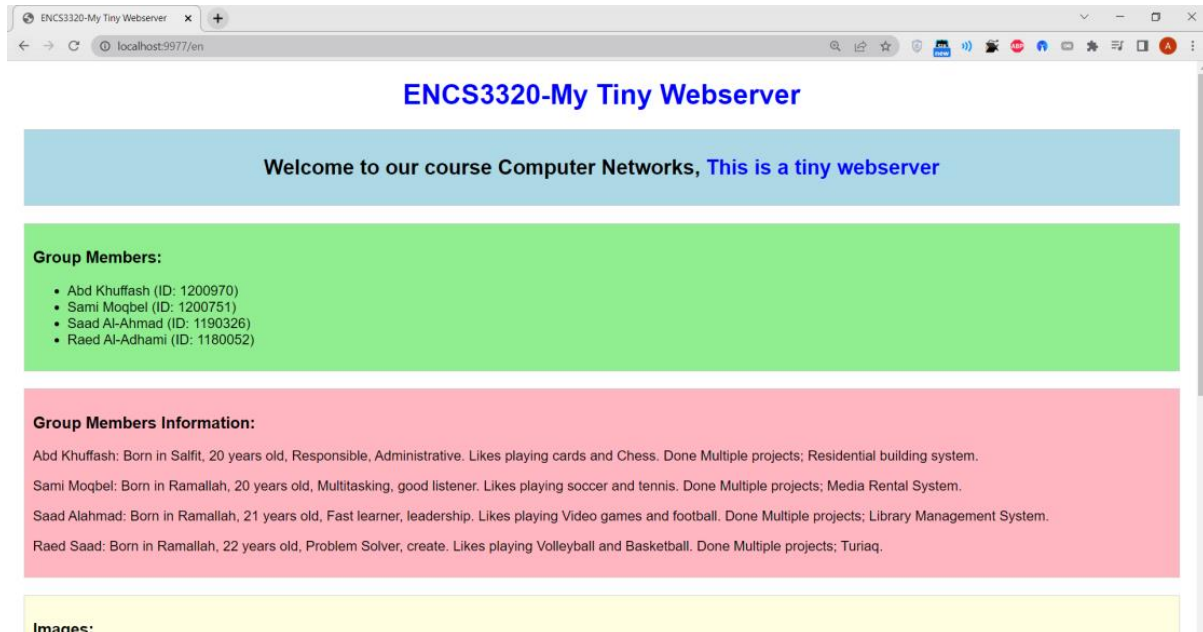
First Case ‘/’:



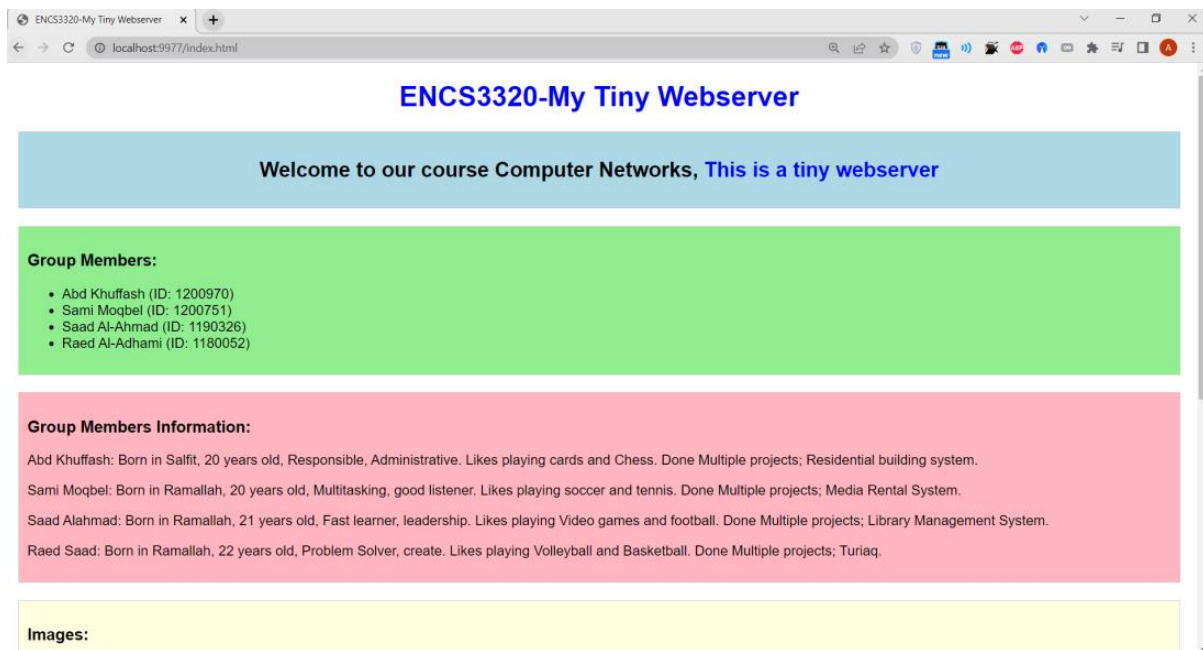
Second Case ‘/main-en.html’:



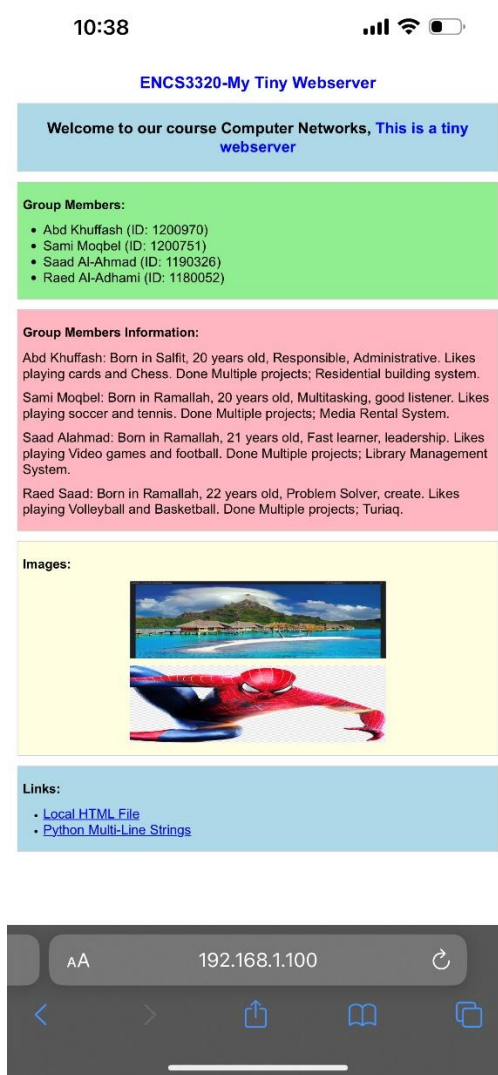
Third Case ‘/en’:



Fourth Case ‘/index.html’:



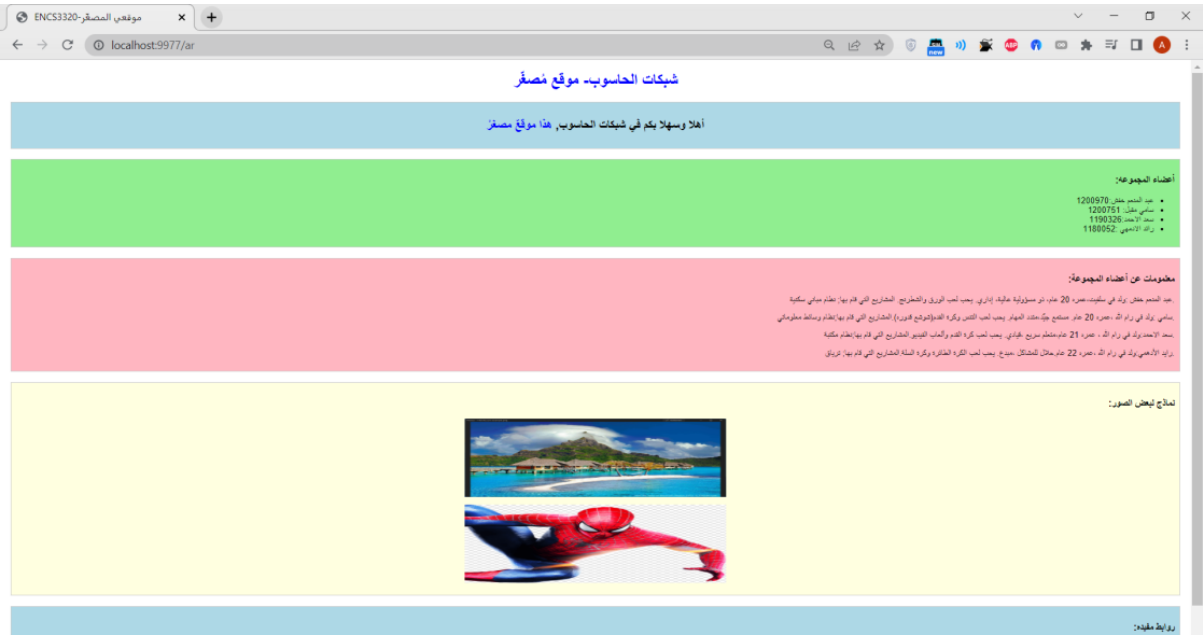
Phone view:



If the request is '/ar':

```
elif (request=='/ar'):#Arabic
    connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
    connectionSocket.send("Content-Type: text/html; charset=utf-8\r\n".encode())
    connectionSocket.send("\r\n".encode())
    f = open("main-ar.html", "rb")
    connectionSocket.send(f.read())
```

The server sends ‘main-ar.html’ file as a response, which contains the same content but in Arabic.



Phone view:



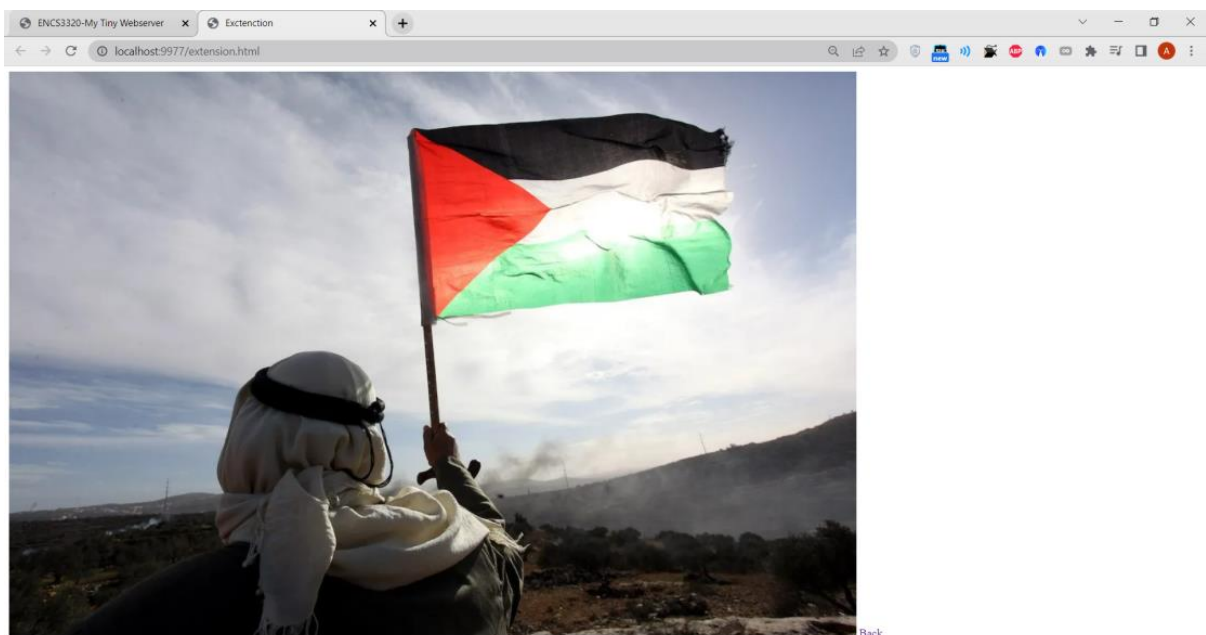
If the request is a local html file:

```
elif(request.endswith('.html')):  
    connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())  
    connectionSocket.send("Content-Type: text/html; charset=utf-8\r\n".encode())  
    connectionSocket.send("\r\n".encode())  
    f = open("extension.html", "rb")  
    connectionSocket.send(f.read())
```

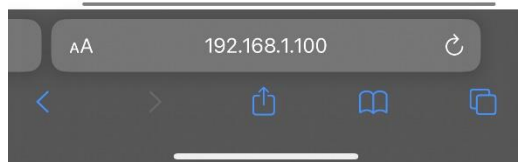
the server sends it with 'Content-Type: text/html':

Links:

- [Local HTML File](#)
- [Python Multi-Line Strings](#)



Phone view:



If the request is a css file:

```
elif (request.endswith('.css')):
    connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
    connectionSocket.send("Content-Type: text/css; charset=utf-8\r\n".encode()) #connection type text/css;
    # to specify that the data being sent is CSS (Cascading Style Sheets) content.
    connectionSocket.send("\r\n".encode())
    f = open("styles.css", "rb")
    connectionSocket.send(f.read())
```

The server sends the css file with ‘Content-Type: text/css’:

A screenshot of a web browser window. The address bar shows 'localhost:9977/a.css'. The page content displays a CSS file with the following rules:

```
body {
  font-family: Arial, sans-serif;
  margin: 20px;
}
h1 {
  color: blue;
}
.box {
  border: 1px solid #ccc;
  padding: 10px;
  margin-bottom: 20px;
}
.box-1 {
  background-color: lightblue;
}
.box-2 {
  background-color: lightgreen;
}
.box-3 {
  background-color: lightpink;
}
.box-4 {
  background-color: lightyellow;
}
.image-container {
  margin-top: 10px;
  margin-bottom: 10px;
}
```

Phone view:

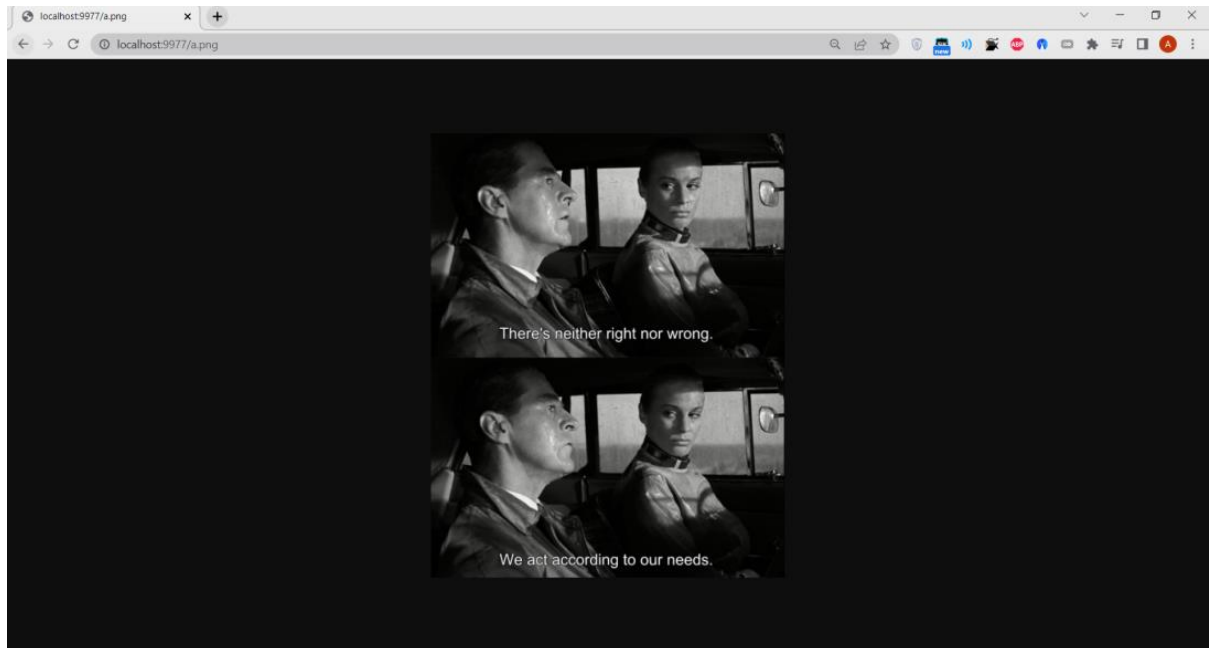
A screenshot of a mobile phone screen. At the top, the status bar shows the time '10:40' and signal/battery icons. Below the status bar, the address bar shows '192.168.1.100'. The page content displays a CSS file with the following rules:

```
body {
  font-family: Arial, sans-serif;
  margin: 20px;
}
h1 {
  color: blue;
}
.box {
  border: 1px solid #ccc;
  padding: 10px;
  margin-bottom: 20px;
}
.box-1 {
  background-color: lightblue;
}
.box-2 {
  background-color: lightgreen;
}
.box-3 {
  background-color: lightpink;
}
.box-4 {
  background-color: lightyellow;
}
.image-container {
  margin-top: 10px;
  margin-bottom: 10px;
}
```

If the request is png file:

```
elif (request.endswith('.png')):
    connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())
    connectionSocket.send("Content-Type: image/png; charset=utf-8\r\n".encode())#to specify that the data being sent is png/image content.
    connectionSocket.send("\r\n".encode())
    f = open("net.PNG", "rb")
    connectionSocket.send(f.read())
```

The server sends the png image with 'Content-Type: image/png':



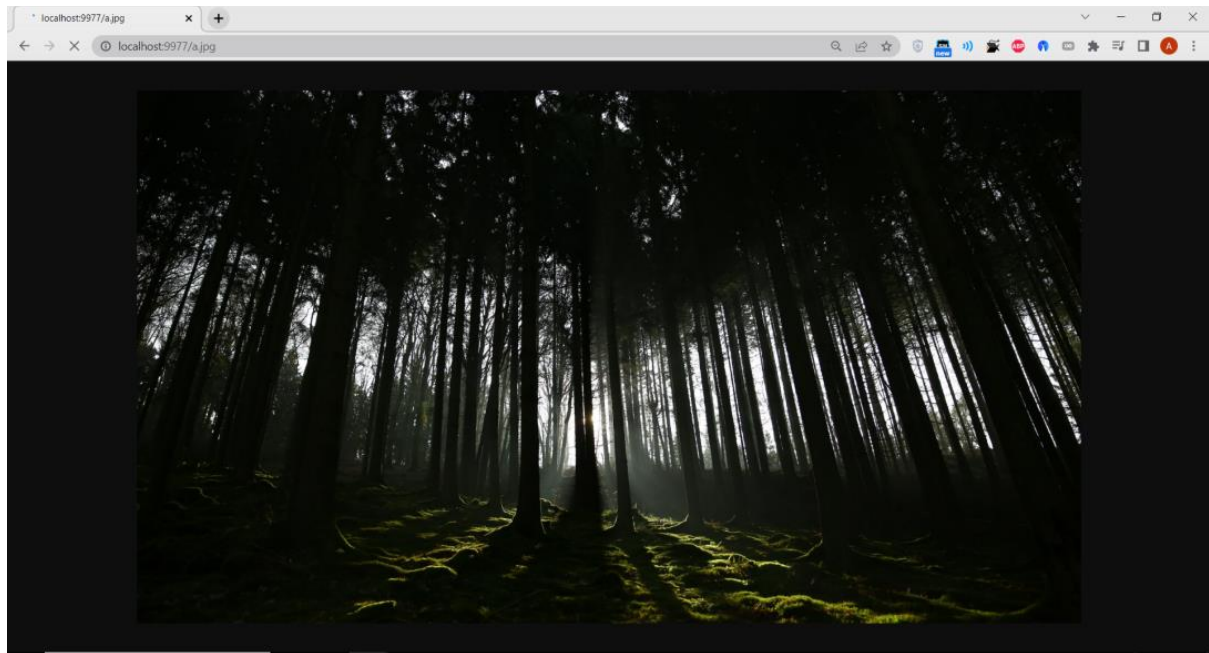
Phone view:



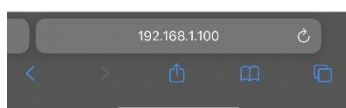
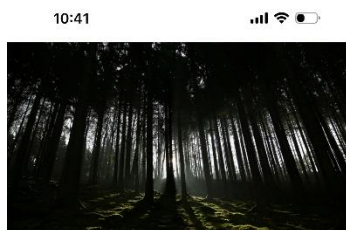
If the request is jpg file:

```
elif (request.endswith('.jpg')):  
    connectionSocket.send("HTTP/1.1 200 OK \r\n".encode())  
    connectionSocket.send("Content-Type: image/jpeg; charset=utf-8\r\n".encode()) #t to specify that the data being sent is jpeg/image content.  
    connectionSocket.send("\r\n".encode())  
    f = open("net.jpg", "rb")  
    connectionSocket.send(f.read())
```

The server sends the jpg file with 'Content-Type: image/jpeg':



Phone view:

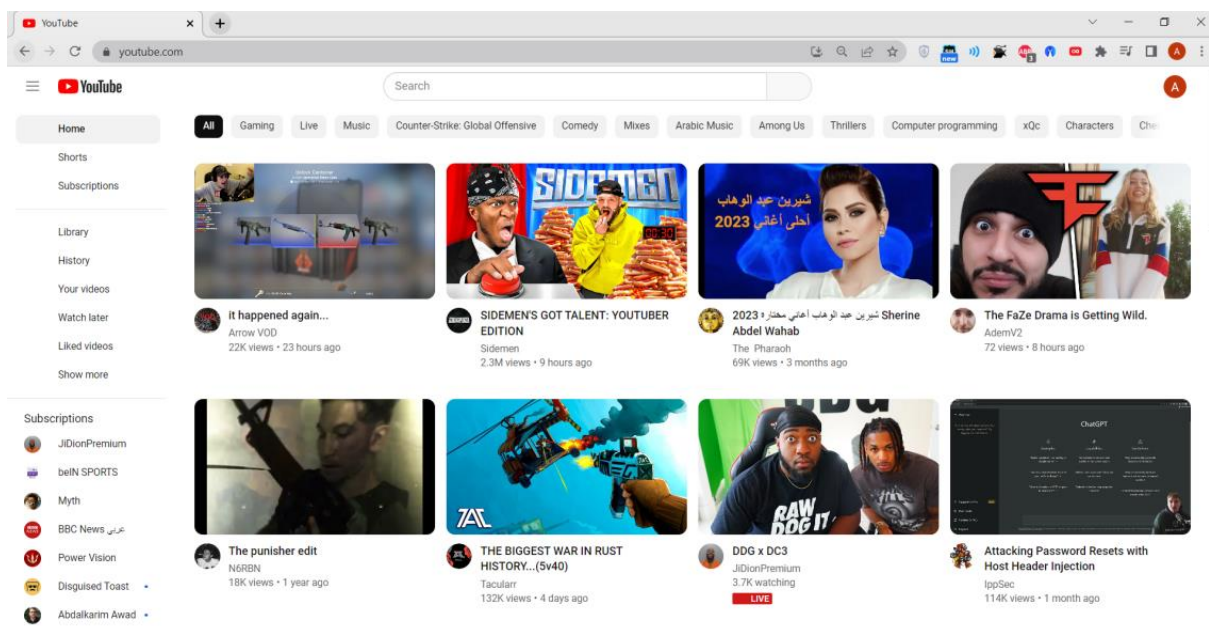


If the request is any of the cases ('/so', or '/yt', or '/rt'):

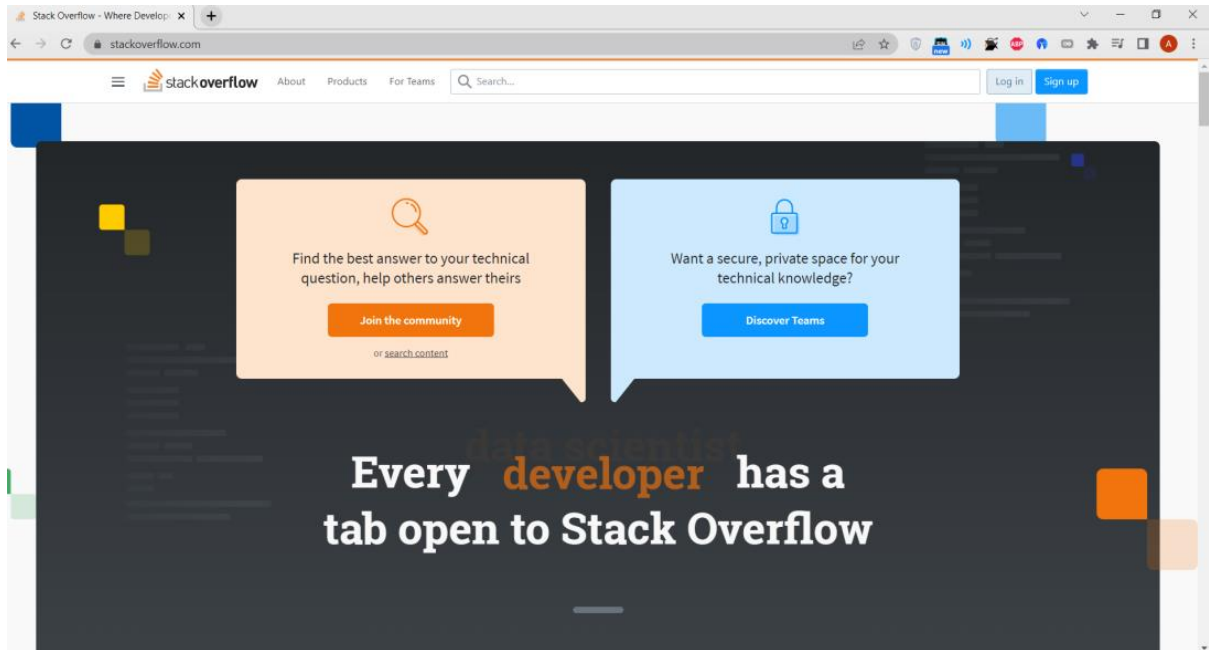
```
elif (request=='/so'):  
    connectionSocket.send("HTTP/1.1 307 Temporary Redirect \r\n".encode())  
    connectionSocket.send("Content-Type: text/html; charset=utf-8 \r\n".encode())  
    connectionSocket.send("Location: https://stackoverflow.com \r\n".encode())  
    connectionSocket.send("\r\n".encode())  
  
elif (request=='/yt'):  
    connectionSocket.send("HTTP/1.1 307 Temporary Redirect \r\n".encode())  
    connectionSocket.send("Content-Type: text/html \r\n".encode())  
    connectionSocket.send("Location: https://youtube.com \r\n".encode())  
    connectionSocket.send("\r\n".encode())  
  
elif (request=='/rt'):  
    connectionSocket.send("HTTP/1.1 307 Temporary Redirect \r\n".encode())  
    connectionSocket.send("Content-Type: text/html \r\n".encode())  
    connectionSocket.send("Location: https://ritaj.birzeit.edu/ \r\n".encode())  
    connectionSocket.send("\r\n".encode())
```

The server sends the status code for redirecting, '307 Temporary Redirect', and redirects the request to another website:

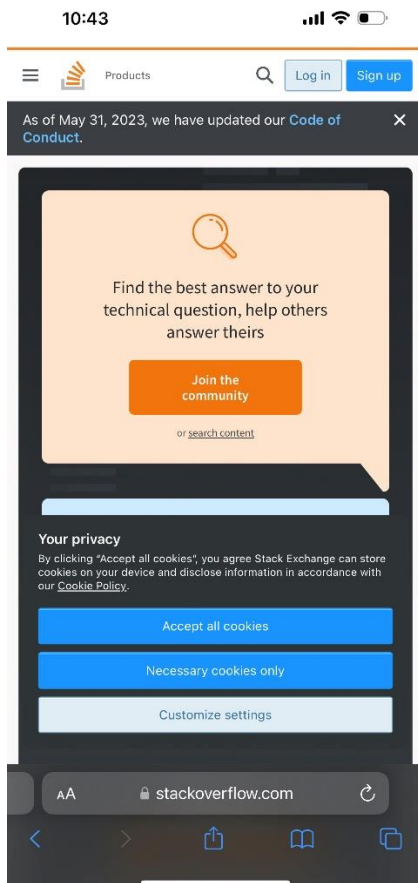
First case '/yt', redirects to YouTube website:



Second case ‘/so’, redirects to stackoverflow.com website:



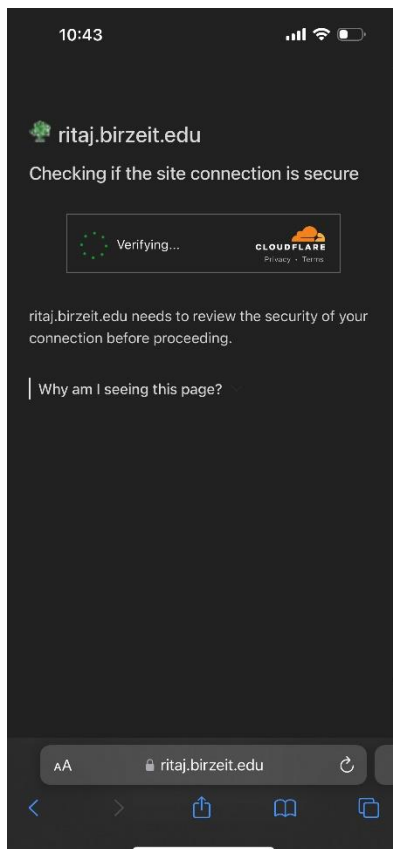
Phone view:



Third case '/rt', redirects to Ritaj website:



Phone view:

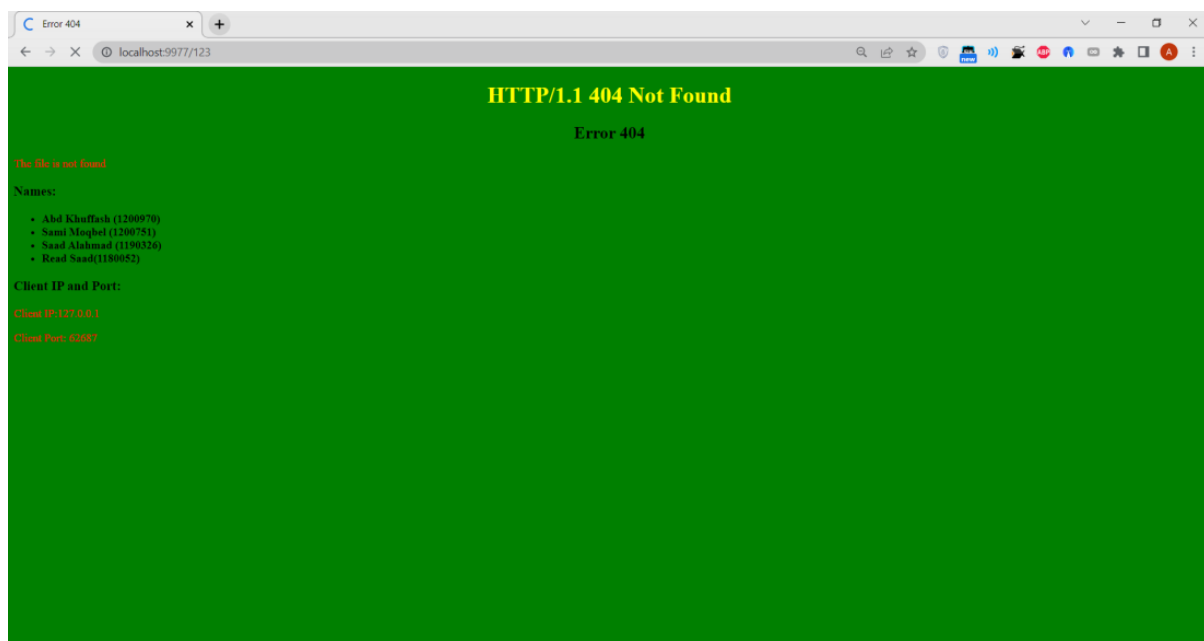


And finally, if the request is wrong or the requested file is not found:

```
else:
    connectionSocket.send("HTTP/1.1 404 Not Found \r\n".encode())
    connectionSocket.send("Content-Type: text/html; charset=utf-8\r\n".encode())
    connectionSocket.send("\r\n".encode())

    f = open("Error.html", "rb")
    connectionSocket.send(f.read())
```

the server will return an error.html file with ‘Content-Type: text/html’ that contains the error code “HTTP/1.1 404 Not Found” in the response status, a title “Error 404”, a red text “The file is not found” in the body, group members names and IDs in Bold, and the IP and port number of the connected client:



Phone view:



Part 4

```
server1.py x client1.py x
1 import socket
2 import time
3
4 IP = '192.168.1.100' # IP address of the server
5 PORT = 8855 # Port number for communication
6 ADDR = (IP, PORT)
7 FORMAT = 'utf-8'
8 CLIENTS={}
9
10 # Create a UDP socket
11 SERVER = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12
13 # Bind the socket to the IP address and port
14 SERVER.bind(ADDR)
15
16 print("[SERVER] Server STARTING...")
17
18 while True:
19     # Receive a message and the client's address
20     message, client_address = SERVER.recvfrom(2048)
21     print(f"[SERVER] Received message: {message.decode(FORMAT)} from {client_address}")
22     receivedMsg=message.decode(FORMAT)
23     clientData=receivedMsg.split('$')
24     print(clientData)
25     Time = time.strftime("%H:%M:%S", time.localtime())
26     clientData.append(Time)
27     CLIENTS[client_address] = clientData
28     print(CLIENTS)
29
30     for key in CLIENTS:
31         messageToSend = f"{key}${CLIENTS[key][0]}${CLIENTS[key][1]}${CLIENTS[key][2]}"
32         print(f"Message : {messageToSend}")
33         response = f"{messageToSend}"
34         for key in CLIENTS:
35             SERVER.sendto(response.encode(FORMAT), key)]
```

```
server1.py x client1.py x
1 import socket
2
3
4 IP = '192.168.1.255' # IP address of the server
5 PORT = 8855 # Port number for communication
6 ADDR = (IP, PORT)
7 FORMAT = 'utf-8'
8 DATA = {}
9
10 # Create a UDP socket
11 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
12
13 # Get user input
14 firstName = input("Enter First Name:\n")
15 lastName = input("Enter Last Name:\n")
16 msg=input("Enter message:\n")
17
18 message = f"{firstName}-{lastName}${msg}"
19
20 # Send the message to the server
21 client_socket.sendto(message.encode(FORMAT), ADDR)
22 i=int(1)
23
24 while True:
25
26     # Receive the response from the server
27     response, server_address = client_socket.recvfrom(2048)
28     msg=response.decode(FORMAT)
29     senderData = msg.split("$")
30     senderADDR=senderData[0]
31     DATA[senderADDR] = senderData
32     senderName=DATA[senderADDR][1]
33     receiveTime=DATA[senderADDR][3]
34     dataList = list(DATA)
```

```

35
36     for address in dataList:
37         if address!=senderADDR:
38             flag=False
39         else:
40             flag=True
41
42     if flag==False:
43         i=1
44
45
46
47     print(f"[CLIENT] {i}- Received a message from {senderName} at {receiveTime}")
48     print("=====\n")
49     i+=1
50
51     choice=input("Enter message number to show followed by d ( S to skip) :\n")
52     choice=choice[0]
53     if choice.lower()=="s":
54         continue
55     else:
56         print(f"Message is: {DATA[dataList[int(choice) - 1]][2]}")
57
58 # Close the socket
59 client_socket.close()

```