



Department of Electrical and Computer Engineering
COMPUTER NETWORKS
ENCS3320
PROJECT #1

By:

Sondos Shahin 1200166
Qamar Sharef 1201930
Mohammad Abu Shama 1200270
Istabraq Mohammad 1170256

Instructor:

Dr. Ibrahim Nemer

Introduction	19
Figure 1/Ping	4
Figure 2/ping www.yale.edu.....	5
Figure 3/ tracert www.yale.edu.....	5
Figure 4/ nslookup www.yale.edu.....	7
Part2-----	
Figure 1/UDP server1.....	8
<i>Figure 2/UDP client1.....</i>	9
<i>Figure 3/TCP client1.....</i>	10
<i>Figure 4/TCP server1.....</i>	11
<i>Figure 5/UDP client2.....</i>	12
<i>Figure 6/UDP server2.....</i>	13
<i>Figure 7/TCP server2.....</i>	14
Figure 8/TCP client.....	15
<i>Figure 9/UDP server3.....</i>	16
<i>Figure 10/UDP client3.....</i>	17
Figure 11/TCP server3.....	18
Part3-----	
Figure 1/en.....	19
Figure 2/ar	20
Figure 3/.html	21
Figure 4/.css	22
Figure 5/.jpg	22
Figure 6/png.....	23
Figure 7phone /ar.....	24
Figure 8phone/en.....	2625
Figure 9phonephone/.css	26
Figure 10phone/.html.....	27
Figure 11phone/notFound.....	28
Figure 12phone/.jpg	29
Task.....	30

Introduction

Our project is built on the super client architecture. Based on the concept of socket programming. In this project, we have built a three-part foolproof network that works as follows:

part one:

We explain what ping, tracert, and nslookup are, and then run the following commands:

- Ping a device in the same network, eg from a laptop to a smartphone
- 2- Bing www.yale.edu
- 3- Tracert www.yale.edu
- 4- nslookup www.yale.edu

and take a screenshot of the playback outputs

part two:

Server and client implementation of both TCP and UDP: The client continuously sends numbers from 0 to 1,000,000 to a server listening on port 5566. The server counts the messages received by socket programming used in Python. For each run, we take screenshots of the run and display the number of packets received. We also measured when packets were sent and when packets were received.

part three:

Using socket programming, implement a simple web server in Python. The program should have an HTML webpage that contains the title, names of group members, identifiers, some information about them, etc., and use CSS to make the page look nice.

We took browser screenshots to show the results and tested the project from the browser on itself computer and from a different computer or phone.

Task:

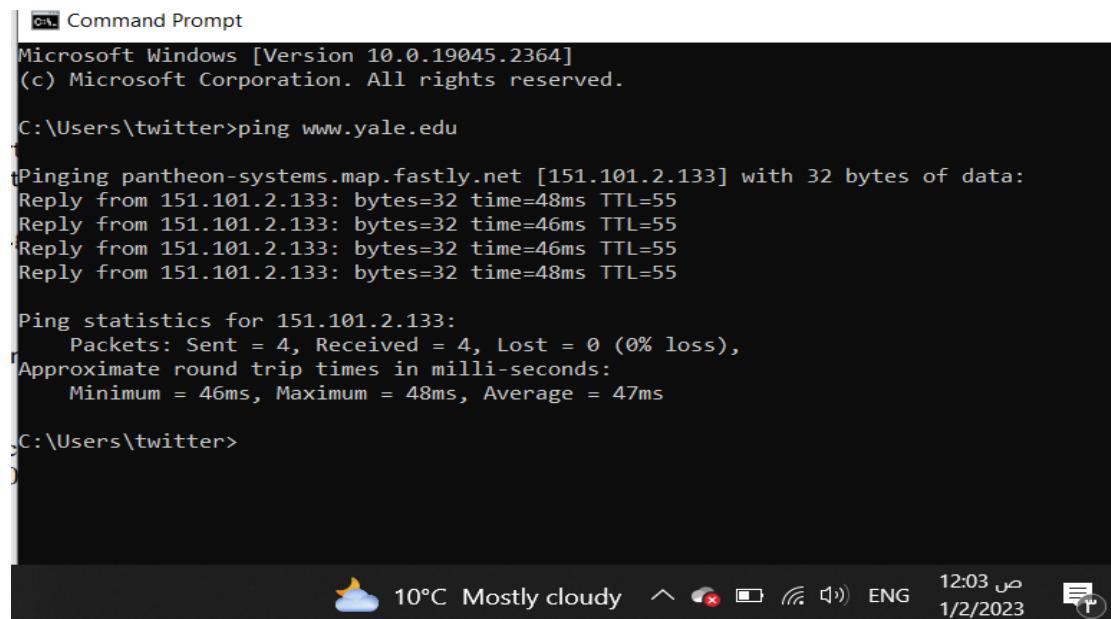
Write program using socket programming a multi-client server TCP program that will do the following:

- 1) Create a connection between one server and two clients
- 2) client1 sends data to SERVER.
- 3) client2 sends data to SERVER.
- 4) SERVER calculates the Cyclic Redundancy Check (CRC) with KEY (=110) for client1 and client1 data.
- 5) SERVER resends the new data along with the CRC bits in binary format to both clients.
- 6) Client checks for error with the same key (=110), and displays the appropriate messages for positive and negative ACKs

Part1:

ping is the primary TCP/IP command used to troubleshoot connectivity, reachability, and name resolution. we used “ping” along with the URL you want to ping, In the image below, we’re pinging ping www.yale.edu and getting a normal response. That response shows the URL you’re pinging, the IP address associated with that URL, and the size of the packets being sent on the first line. The next four lines show the replies from each individual packet, including the time (in milliseconds) it took for the response and the time-to-live (TTL) of the packet, which is the amount of time that must pass before the packet is discarded. At the bottom, you’ll see a summary that shows how many packets were sent and received, as well as the minimum, maximum, and average response time.

And in the next image(pic.2), we’re pinging the router on our local network using its IP address. We’re also getting a normal response from it.



```
Windows [Version 10.0.19045.2364]
(c) Microsoft Corporation. All rights reserved.

C:\Users\twitter>ping www.yale.edu

Pinging pantheon-systems.map.fastly.net [151.101.2.133] with 32 bytes of data:
Reply from 151.101.2.133: bytes=32 time=48ms TTL=55
Reply from 151.101.2.133: bytes=32 time=46ms TTL=55
Reply from 151.101.2.133: bytes=32 time=46ms TTL=55
Reply from 151.101.2.133: bytes=32 time=48ms TTL=55

Ping statistics for 151.101.2.133:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 46ms, Maximum = 48ms, Average = 47ms

C:\Users\twitter>
```

The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window displays the output of the "ping" command for the URL "www.yale.edu". The output includes the IP address of the target host (151.101.2.133), four successful replies with their respective times and TTL values, and a summary of the ping statistics showing 100% packet delivery and an average round-trip time of 47ms. The taskbar at the bottom of the screen shows the date (1/2/2023), time (12:03 PM), weather (10°C, Mostly cloudy), and system status icons.

```
C:\Users\twitter>ping 192.168.68.111

Pinging 192.168.68.111 with 32 bytes of data:
Reply from 192.168.68.111: bytes=32 time=122ms TTL=64
Reply from 192.168.68.111: bytes=32 time=38ms TTL=64
Reply from 192.168.68.111: bytes=32 time=51ms TTL=64
Reply from 192.168.68.111: bytes=32 time=63ms TTL=64

Ping statistics for 192.168.68.111:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 38ms, Maximum = 122ms, Average = 68ms
```

```
C:\Users\twitter>
```



nslookup is a network administration command-line tool available for many computer operating systems for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or for any other specific DNS record.

The information displayed will be the local DNS server and its IP address. You can specify a DNS server (IP address), record type, and domain name, and in the last line show you the www.yale.edu address as aliases.

```
C:\Users\twitter>nslookup www.yale.edu
Server:  one.one.one.one
Address:  1.1.1.1

Non-authoritative answer:
Name:      pantheon-systems.map.fastly.net
Addresses:  2a04:4e42::645
            2a04:4e42:200::645
            2a04:4e42:400::645
            2a04:4e42:600::645
            151.101.2.133
            151.101.66.133
            151.101.130.133
            151.101.194.133
Aliases:   www.yale.edu
```

```
C:\Users\twitter>
```



TRACERT (Trace Route), a command-line utility that you can use to trace the path that an Internet Protocol (IP) packet takes to its destination. We can see that tracert again identified 10 network devices including our router at 192.168.68.1 and all the way through to the target of www.yale.edu, which we can assume uses the public IP address of 151.101.2.133 . As you can see, tracert didn't resolve any hostnames this time, which significantly sped up the process.

```
C:\Users\twitter>tracert www.yale.edu

Tracing route to pantheon-systems.map.fastly.net [151.101.2.133]
over a maximum of 30 hops:

 1   1 ms    1 ms    1 ms  192.168.68.1
 2   3 ms    3 ms    2 ms  192.168.50.1
 3   5 ms    4 ms    3 ms  193.58.151.1
 4   5 ms    6 ms    6 ms  10.1.1.5
 5  12 ms    7 ms    7 ms  bzq-84-110-113-25.cablep.bezeqint.net [84.110.113.25]
 6   *      *      * Request timed out.
 7  10 ms    10 ms   6 ms  bzq-179-124-85.cust.bezeqint.net [212.179.124.85]
 8  51 ms    61 ms   51 ms  10.250.99.4
 9  47 ms    44 ms   45 ms  bzq-25-70-71.cust.bezeqint.net [212.25.70.71]
10  45 ms    44 ms   43 ms  151.101.2.133

Trace complete.

C:\Users\twitter>
```



The screenshot shows a Windows taskbar with various icons. From left to right, they include: a cloud with a sun icon (weather), "10°C Mostly cloudy", a volume icon, a battery icon, a signal strength icon, a "ENG" language setting, the time "12:07", a power button icon, and the date "1/2/2023".

Part2:

UDP server's show the number of packets received is shown as 1 million (a counter representing). When the program is on the same computer (using localhost)

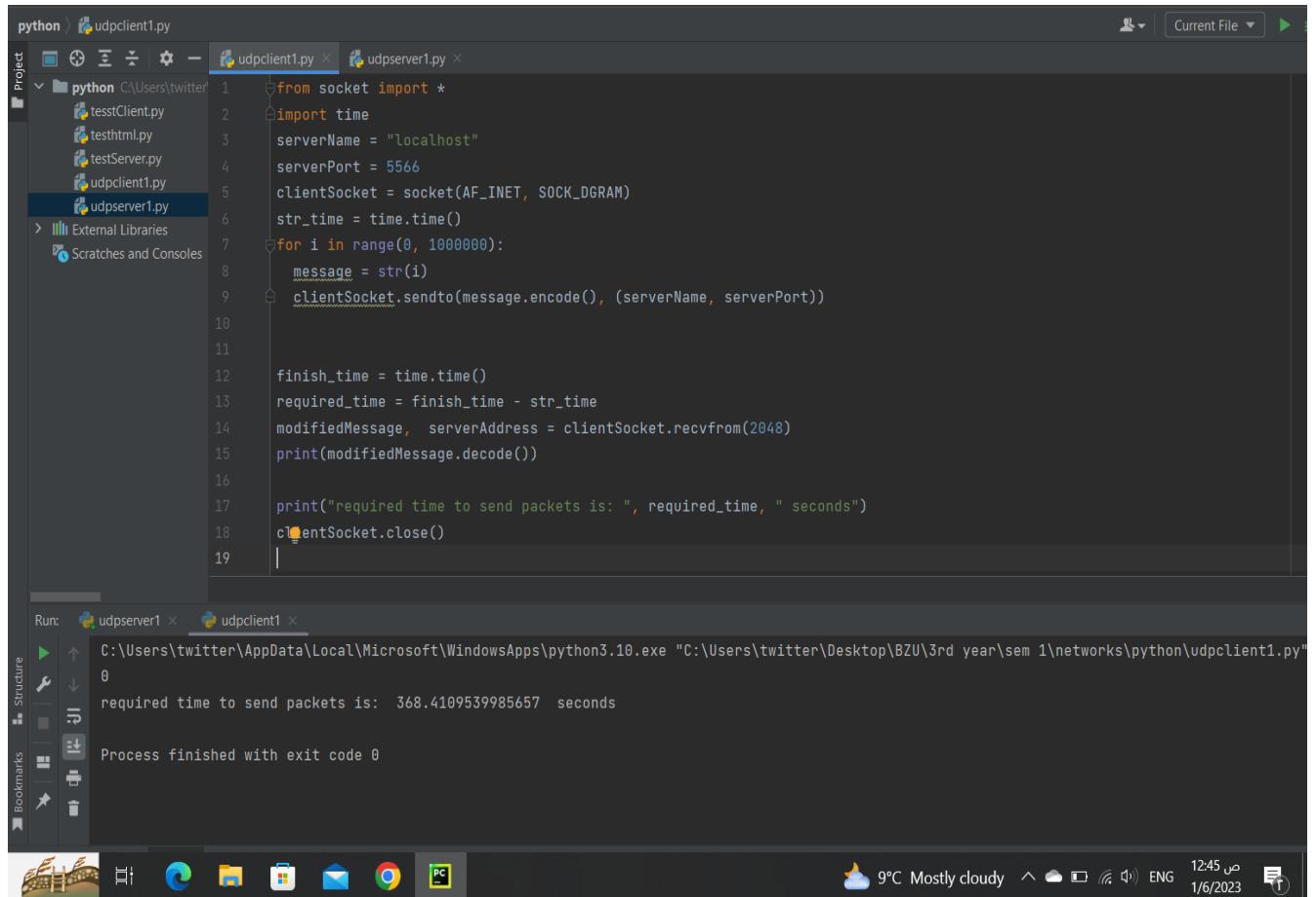
The screenshot shows a Python development environment with the following details:

- Project View:** Shows a folder named "python" containing files: testClient.py, testhtml.py, testServer.py, udpclient1.py, and udpserver1.py.
- Code Editor:** Displays the content of `udpserver1.py`. The code is a UDP server that listens on port 5566, receives messages from clients, decodes them, sends modified messages back, and increments a counter. It prints the counter value and the required time to receive 1 million packets.
- Run Tab:** Shows the output of running `udpserver1.py`. The output window displays the following sequence of messages:

```
counter = 999997
b'999997'
counter = 999998
b'999998'
counter = 999999
b'999999'
counter = 1000000
```
- Bottom Bar:** Includes tabs for Version Control, Run, TODO, Problems, Terminal, Python Packages, Python Console, Services, and a status bar showing the date and time (12:46 1/6/2023).

Figure 1/UDP server1

The client UDP continuously sends numbers from 0 to 1,000,000 to the server. When the program is on the same computer (using localhost), the number of packets sender are 1,000,000 ,and the time to send packets its:368.1095399858657 second



A screenshot of the PyCharm IDE interface. The top bar shows 'python' and the current file 'udpclient1.py'. The project sidebar lists files: testClient.py, testhtml.py, testServer.py, udpclient1.py, and udpserver1.py. The main editor window displays the Python code for the UDP client. The code imports socket and time, sets the server name to 'localhost' and port to 5566, creates a client socket, and loops through a range of 1,000,000 to send messages. It also receives a modified message from the server and prints the required time. The bottom run tab shows the command 'C:\Users\twitter\AppData\Local\Microsoft\WindowsApps\python3.10.exe "C:\Users\twitter\Desktop\BZU\3rd year\sem 1\networks\python\udpclient1.py"' and the output 'required time to send packets is: 368.41095399858657 seconds'. The status bar at the bottom right shows the date and time.

```
from socket import *
import time
serverName = "localhost"
serverPort = 5566
clientSocket = socket(AF_INET, SOCK_DGRAM)
str_time = time.time()
for i in range(0, 1000000):
    message = str(i)
    clientSocket.sendto(message.encode(), (serverName, serverPort))

    finish_time = time.time()
    required_time = finish_time - str_time
    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
    print(modifiedMessage.decode())

print("required time to send packets is: ", required_time, " seconds")
clientSocket.close()
```

Figure 2|UDP client1

The client TCP continuously sends numbers from 0 to 1,000,000 to the server, when the program is on the same computer (using localhost), number of packets sender are 1,000,000 (from 0 to 999999), and the time to send packets and the time to receive packets from first packet to last packet its: 13.095838069915771 second

A screenshot of the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Behavior, Run, Tools, VCS, Window, Help, and project\Network - C:\Users\amora\PycharmProjects\project\Network\venv\TCPclient.py. The main window shows two files: TCPserver.py and TCPclient.py. The TCPclient.py code is as follows:

```
File Edit View Navigate Code Behavior Run Tools VCS Window Help project\Network - C:\Users\amora\PycharmProjects\project\Network\venv\TCPclient.py
TCPserver.py TCPclient.py
1 import socket
2 import time
3
4 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 client_socket.connect(('localhost', 5566))
6
7 start_time = time.time()
8
9 for i in range(1000000):
10     client_socket.send(str(i).encode())
11     print(i, end='\n')
12
13 end_time = time.time()
14 client_socket.close()
15
16 print(f'Time required to send packets: {end_time - start_time} seconds')
```

The bottom panel shows the run output for TCPclient.py:

```
Run: TCPserver x TCPclient x
999992
999993
999994
999995
999996
999997
999998
999999
Time required to send packets: 13.095838069915771 seconds
Process finished with exit code 0
```

The status bar at the bottom right indicates: 1497881 CRLF UTF-8 4 spaces Python 3.10 (project\Network) 49°F Cloudy 12:00 AM 1/7/2023.

Figure 3/TCP client

TCP server when the program is on the same computer (using localhost), number of packets received are 957338 messages

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and project1network - C:\Users\amora\PycharmProjects\project1network\venv\TCPserver.py. The main window displays two files: TCPserver.py and TCPclient.py. The TCPserver.py code is as follows:

```
1 import socket
2 import time
3
4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 server_socket.bind(('localhost', 5566))
6 server_socket.listen()
7 client_socket, client_address = server_socket.accept()
8 start_time = time.time()
9 count = 0
10
11 while True:
12     data = client_socket.recv(1024)
13     if not data:
14         break
15     count += 1
16     end_time = time.time()
17     client_socket.close()
18     server_socket.close()
19
20 print(f'Time required to receive packets: {end_time - start_time} seconds')
21 print(f'Received {count} messages.'
```

The bottom run tab shows the output of running TCPserver.py: "Time required to receive packets: 13.094840288162231 seconds" and "Received 957338 messages.". The status bar at the bottom right indicates "Activate Windows Go to Settings to activate Windows.", the date and time as "2023-06-01 11:57 PM", and the Python version as "Python 3.10 (project1network)".

Figure 4/TCP server

The UDP client continuously sends numbers from 0 to 1,000,000 to the server on two different computers connected through cable, time to send packages and time to receive packages from the First Packet to Last: 18.238933123352 seconds

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with files: main.py, udpclient2.py (selected), and udpclient3.py. The right pane shows the code for udpclient2.py:

```
from socket import *
import time
clientIP = "192.168.68.114"
serverIP = "192.168.68.101"
serverPort = 5566
clientSocket = socket(AF_INET, SOCK_DGRAM)
clientSocket.connect((serverIP, serverPort))
str_time = time.time()
for i in range(0, 1000000):
    message = str(i)
    clientSocket.sendto(message.encode(), (serverIP, serverPort))

finish_time = time.time()
required_time = finish_time - str_time
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())

print("required time to send packets is: ", required_time, " seconds")
clientSocket.close()
```

The 'Run' tab at the bottom shows the command: "C:\Users\DELL i7\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:\Users\DELL i7\PycharmProjects\pythonProject\udpclient2.py". The output window displays:

```
750
required time to send packets is: 18.2389331123352 seconds
Process finished with exit code 0
```

The status bar at the bottom right shows: Activate Windows, Go to Settings to activate Windows. The system tray icons include: Search, File Explorer, Task View, Settings, File Explorer, Mail, Word, Excel, Powerpoint, Edge, Chrome, Firefox, File Explorer, and Task View. The date and time are 1/6/2023 4:00 PM.

Figure 5/UDP client2

UDP server's show the number of packets received are shown as 276889 (a counter representing). When the program is on two different computers connected through cable directly or through a switch

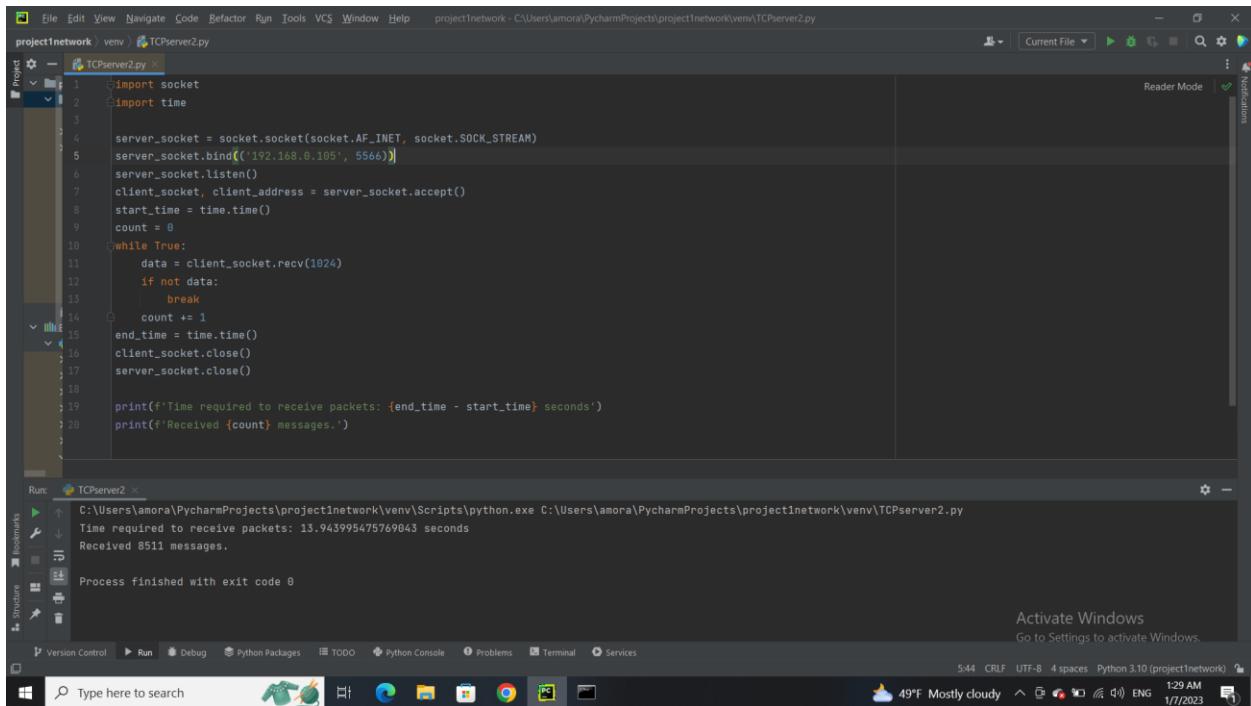
The screenshot shows a Python development environment with the following details:

- Project View:** A tree view of files under a "python" folder. Files listed include: testClient.py, testhtml.py, testServer.py, udpclient1.py, udpclient3.py, udpserver1.py, udpserver2.py, and udpserver3.py.
- Code Editor:** The file "udpserver2.py" is open. The code is a UDP server script. It imports socket and time, sets up a socket to port 5566, binds it to "192.168.68.101", and prints a message indicating it is ready to receive. It then enters a loop where it receives messages from clients, increments a counter, encodes the modified message, and sends it back. It prints the current value of the counter each time a message is received. The code ends with a print statement for the required time to receive the packets.
- Run Output:** Below the editor, the output window shows the following sequence of messages:

```
Run: udpserver2.x
b'999832'
counter = 276887
b'999833'
counter = 276888
b'999863'
counter = 276889
```
- System Tray:** At the bottom right, there is a system tray with icons for weather (10°C Partly sunny), battery, signal strength, language (ENG), and date/time (03:58 1/6/2023).

Figure 6/UDP server2

TCP server *When the program is on two different computers connected through Wi-Fi* , number of packets received are 8511 messages ,and the time required to receive packets:13.943995475643 second



The screenshot shows the PyCharm IDE interface with the following details:

- File:** TCPserver2.py
- Code Content:**

```
project1network venv > TCPserver2.py
1 import socket
2 import time
3
4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 server_socket.bind(('192.168.0.105', 5566))
6 server_socket.listen()
7 client_socket, client_address = server_socket.accept()
8 start_time = time.time()
9 count = 0
10 while True:
11     data = client_socket.recv(1024)
12     if not data:
13         break
14     count += 1
15 end_time = time.time()
16 client_socket.close()
17 server_socket.close()
18
19 print(f'Time required to receive packets: {end_time - start_time} seconds')
20 print(f'Received {count} messages.')
```
- Run Output:**

```
C:\Users\amora\PycharmProjects\project1network\venv\Scripts\python.exe C:\Users\amora\PycharmProjects\project1network\venv\TCPserver2.py
Time required to receive packets: 13.943995475643 seconds
Received 8511 messages.

Process finished with exit code 0
```
- System Status Bar:** Activate Windows Go to Settings to activate Windows.
- System Icons:** 49°F Mostly cloudy, 129 AM, 1/7/2023

Figure 7/TCP server2

TCP client continuously sends numbers from 0 to 1,000,000 to the server, when the program is *on two different computers connected through Wi-Fi*, number of packets sender are 1,000,000 (from 0 to 999999), and the time to send packets and the time to receive packets from first packet to last packet its: 13. 943288803100586 second.

The screenshot shows a Python code editor interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help.
- Project:** pythonProject, TCPclient.py
- Code Editor:** The file TCPclient.py contains the following Python code:

```
pythonProject | TCPclient.py
Project   — TCPclient.py
1 import socket
2 import time
3 serverIP="192.168.0.105"
4 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 client_socket.connect(('192.168.0.105', 5566))
6
7 start_time = time.time()
8
9 for i in range(1000000):
10     client_socket.send(str(i).encode())
11     print(i, end='\n')
12
13 end_time = time.time()
14 client_socket.close()
15
16 print(f'Time required to send packets: {end_time - start_time} seconds')
```
- Run Tab:** Shows the output of the run command:

```
Run: TCPclient x
999996
999997
999998
999999
Time required to send packets: 13.943288803100586 seconds
Process finished with exit code 0
```
- Bottom Status Bar:** Version Control, Run, TODO, Problems, Terminal, Python Packages, Python Console, Services, Indexing completed in 39 sec. Shared indexes were applied to 77% of files (5,908 of 7,641), (56 minutes ago).
- System Tray:** 49°F Mostly cloudy, ENG, 1:28 AM, 1/7/2023.

Figure 8/TCP client2

UDP server's show the number of packets received are shown as 293243 (a counter representing). When the program on 2 different computers connected through wifi.

The screenshot shows the PyCharm IDE interface. The top part displays the code for `udpserver3.py`. The code initializes a UDP socket, binds it to port 5566, and enters a loop to receive messages from clients. It prints each message, increments a counter, encodes the modified message, and sends it back to the client. The bottom part shows the run output window with the title "Run: udpserver3". The output shows the server printing 293243 messages, with the last few lines being:

```
b'996853'  
counter = 293241  
b'996854'  
counter = 293242  
b'996855'  
counter = 293243
```

At the bottom of the interface, there are various toolbars and status indicators, including a weather icon showing 10°C Cloudy, a date and time indicator (01:59 PM, 1/6/2023), and a Python version indicator (Python 3.10).

```
from socket import *
import time

serverPort = 5566
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("192.168.68.101", serverPort))
print("The server is ready to receive")
start_time = time.time()
counter = 0

while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    print(message)
    counter = counter+1
    modifiedMessage = message.decode()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
    print("counter = ", counter)
    end_time = time.time()
    req_time = start_time - end_time
    print("required time to receive packets is: ", req_time, " seconds")
```

Figure 9/UDP server3

UDP client continuously sends numbers from 0 to 1,000,000 to the server on two different computers connected through Wi-Fi , time to send packages and time to receive packages from the First Packet to Last: 41.7681710 72006226 seconds.

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with files: main.py, udpclient3.py, and a venv library root. The main editor window contains the Python code for a UDP client. The code imports socket and time, sets serverIP and serverPort, creates a clientSocket, and enters a loop to send messages from 0 to 1,000,000. It also receives responses from the server and prints the required time to send packets. The run tab at the bottom shows the command "C:\Users\DELL i7\PycharmProjects\pythonProject\venv\Scripts\python.exe" "C:\Users\DELL i7\PycharmProjects\pythonProject\udpclient3.py" and the output "required time to send packets is: 41.768171072006226 seconds". The status bar at the bottom right shows the date and time as 2:01 PM 1/6/2023.

```
from socket import *
import time
serverIP = "192.168.68.101"
serverPort = 5566
clientSocket = socket(AF_INET, SOCK_DGRAM)
str_time = time.time()
for i in range(0, 1000000):
    message = str(i)
    clientSocket.sendto(message.encode(), (serverIP, serverPort))

finish_time = time.time()
required_time = finish_time - str_time
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())

print("required time to send packets is: ", required_time, " seconds")
clientSocket.close()
```

Figure 10/UDP client3

TCP server When the program is on two different computers connected through cable , number of packets received are 8511 messages ,and the time required to receive packets:13.943995475643 second

The screenshot shows the PyCharm IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The current file is TCPserver2.py, located in the project1network\venv directory. The code itself is as follows:

```
project1network\venv> TCPserver2.py
1 import socket
2 import time
3
4 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 server_socket.bind(('192.168.0.105', 5566))
6 server_socket.listen()
7 client_socket, client_address = server_socket.accept()
8 start_time = time.time()
9 count = 0
10 while True:
11     data = client_socket.recv(1024)
12     if not data:
13         break
14     count += 1
15     end_time = time.time()
16     client_socket.close()
17     server_socket.close()
18
19 print(f'Time required to receive packets: {end_time - start_time} seconds')
20 print(f'Received {count} messages.')
```

In the bottom right corner of the code editor, there is a "Reader Mode" button. The bottom panel displays the run output:

```
Run: TCPserver2 >
C:\Users\amora\PycharmProjects\project1network\venv\Scripts\python.exe C:\Users\amora\PycharmProjects\project1network\venv\TCPserver2.py
Time required to receive packets: 13.943995475643 seconds
Received 8511 messages.

Process finished with exit code 0
```

The status bar at the bottom shows "Activate Windows Go to Settings to activate Windows." and system information like "20:37 CRLF UTF-8 4 spaces Python 3.10 (project1network)" and "49°F Mostly cloudy".

Figure 11/TCP server3

TCP client continuously sends numbers from 0 to 1,000,000 to the server, when the program is *on two different computers connected through cable*, number of packets sender are 1,000,000 (from 0 to 99999), and the time to send packets and the time to receive packets from first packet to last packet its: 13.943288803100586 second.

```

PC File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject - TCPclient.py
pythonProject TCPclient.py
Project pythonProject C:\Users\Admin\PycharmProjects\pyth...
  main.py
  TCPclient.py
External Libraries
Scratches and Consoles

TCPclient.py
1 import socket
2 import time
3 clientIP ="192.168.0.111"
4 serverIP="192.168.0.105"
5 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6 client_socket.connect(('192.168.0.105', 5566))

7
8 start_time = time.time()
9
10 for i in range(1000000):
11     client_socket.send(str(i).encode())
12     print(i, end='\n')
13
14 end_time = time.time()
15 client_socket.close()
16
17 print(f'Time required to send packets: {end_time - start_time} seconds')

```

Run: TCPclient x

Structure

Bookmarks

Run Control

Version Control

File

Edit

View

Navigate

Code

Refactor

Run

Tools

VCS

Window

Help

pythonProject - TCPclient.py

Current File

Notification

638 CRLF UTF-8 4 spaces Python 3.9 (pythonProject)

Indexing completed in 39 sec. Shared indexes were applied to 77% of files (5,908 of 7,641). (50 minutes ago)

49°F Mostly cloudy

1:22 AM

1/7/2023

Type here to search

pythonProject - TCPclient.py

Command Prompt

Figure 12/TCP client3

Part3:

Testing website on computer:

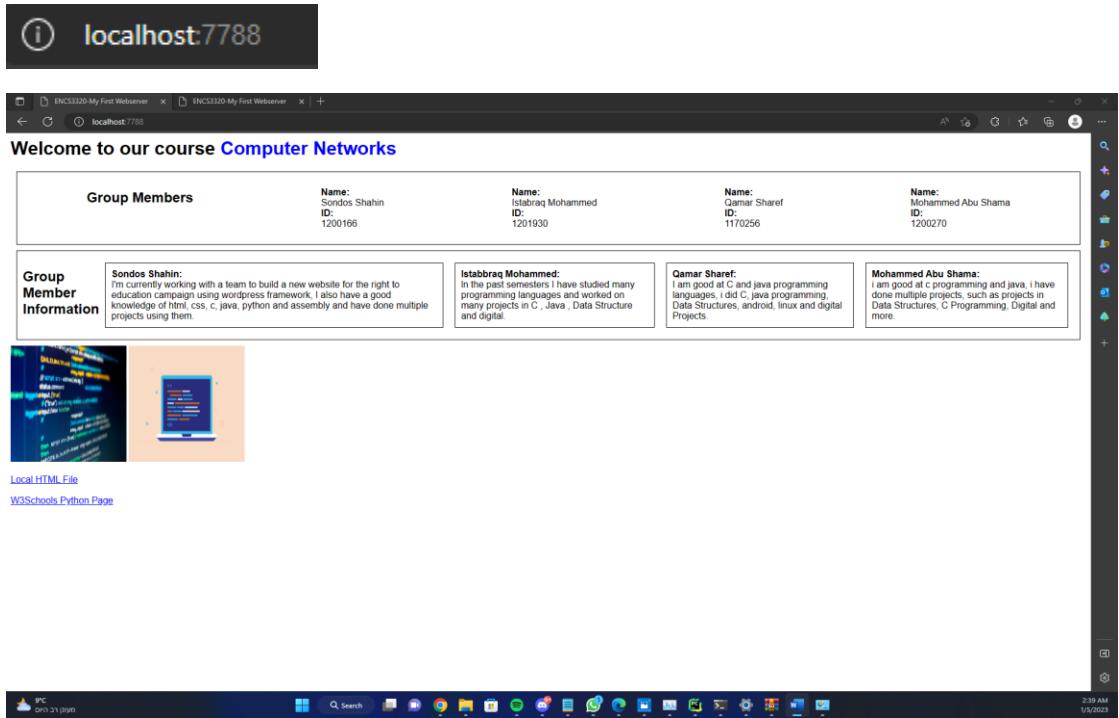


Figure 1/en

For the Arabic page:

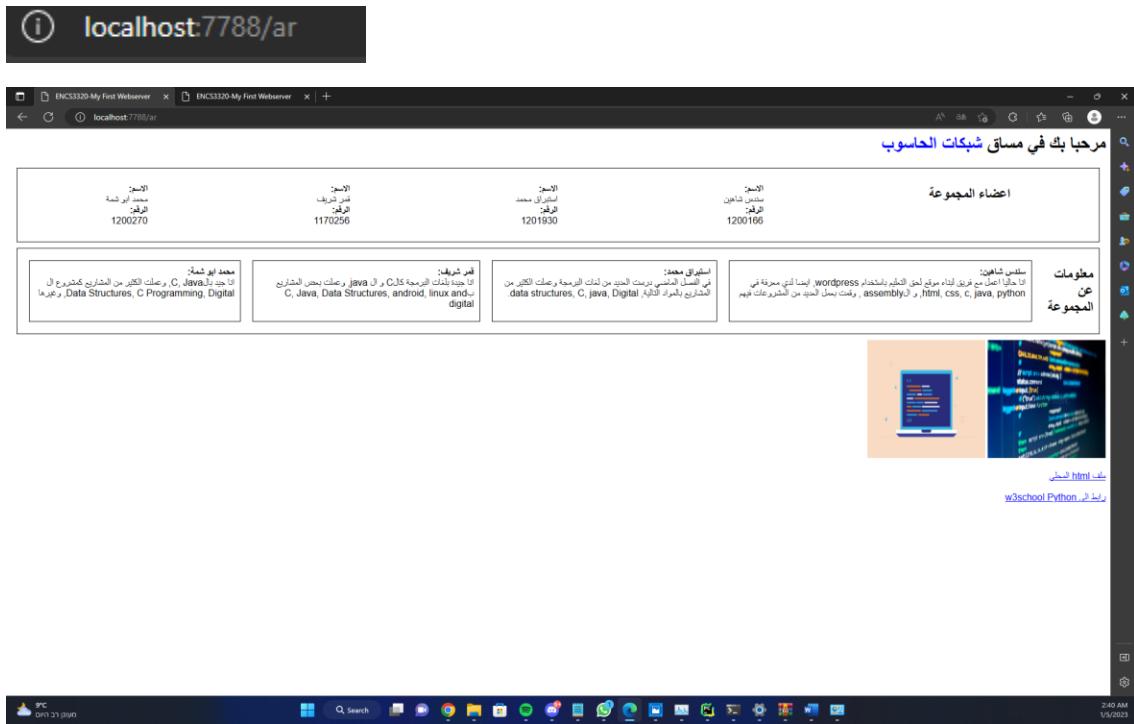
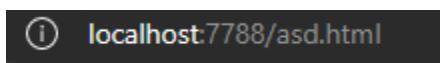


Figure 2/ar

If we request any .html link it display a local_file.html file:



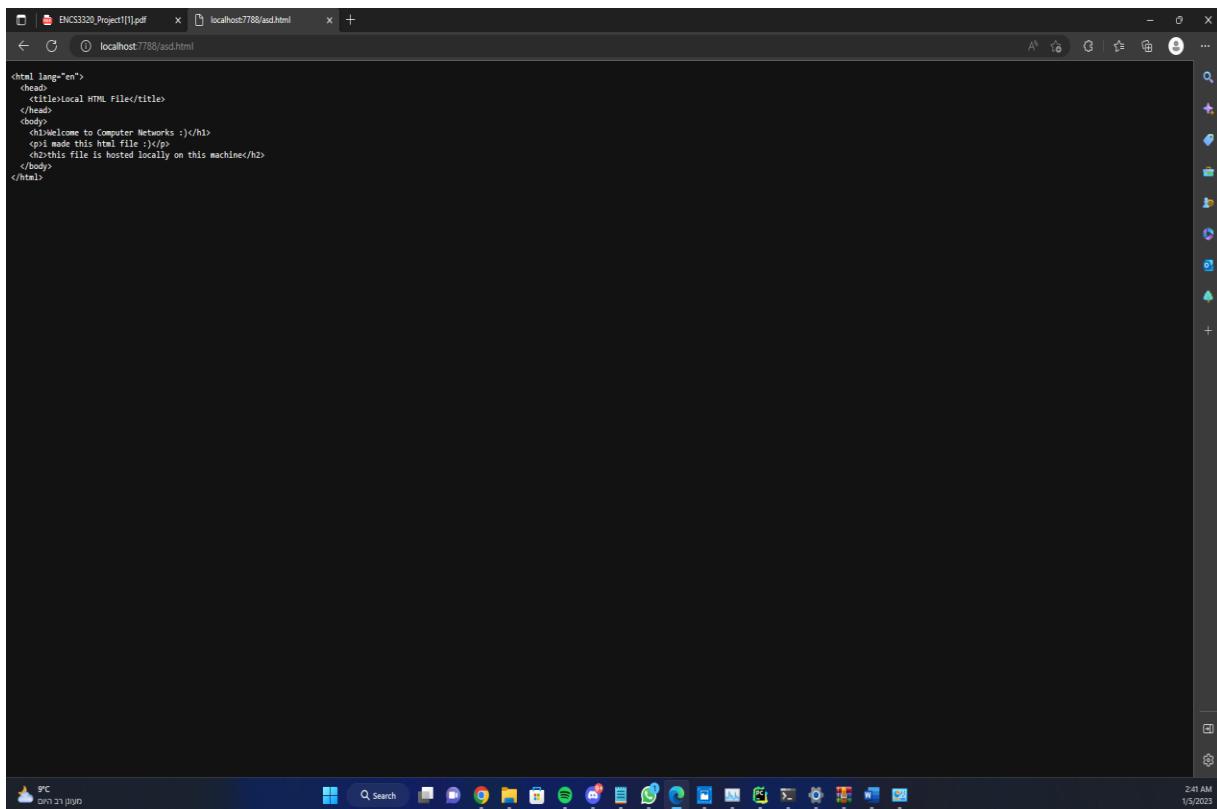
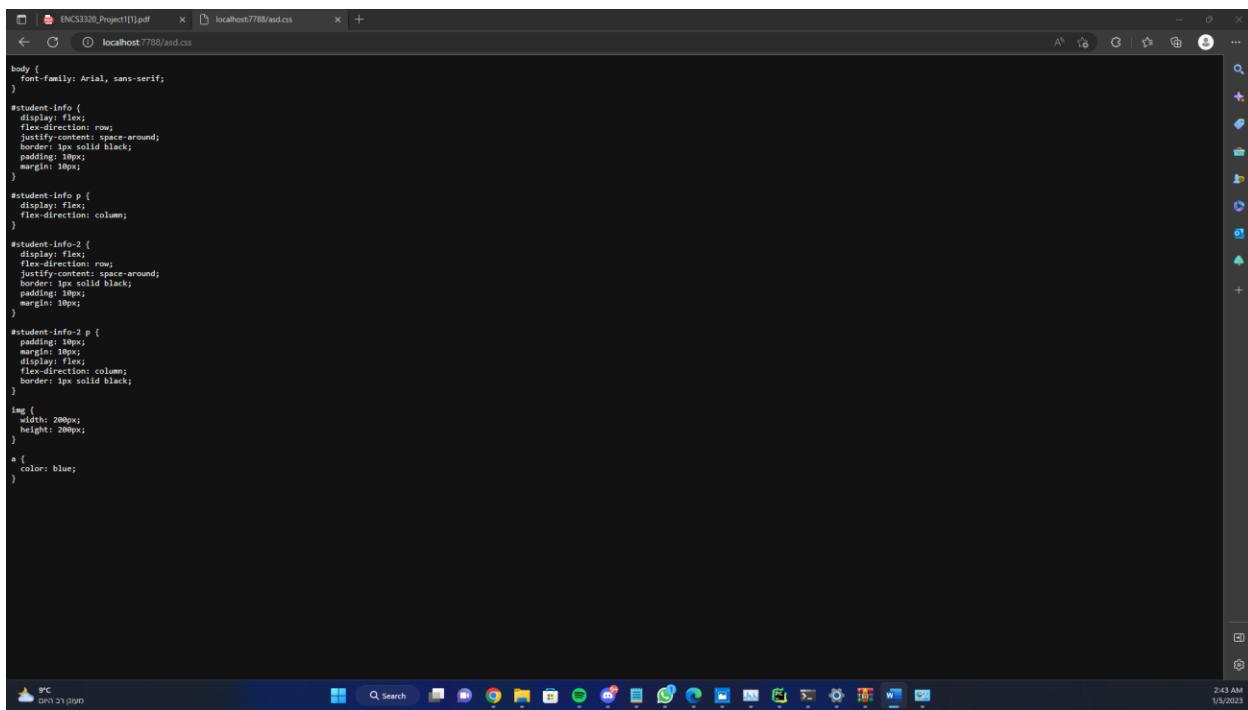


Figure 3/.html

If we request any .css file it display the style.css used to create the main_en and main_ar html files:

(i) localhost:7788/asd.css



A screenshot of a Windows desktop environment. In the center is a Microsoft Edge browser window titled "localhost:7788/asd.css". The page content is a dark-themed CSS code editor interface showing a single file named "asd.css". The code is as follows:

```
body { font-family: Arial, sans-serif; }

#student-info {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
    border: 1px solid black;
    padding: 10px;
    margin: 10px;
}

#student-info p {
    display: flex;
    flex-direction: column;
}

#student-info-2 {
    display: flex;
    flex-direction: row;
    justify-content: space-around;
    border: 1px solid black;
    padding: 10px;
    margin: 10px;
}

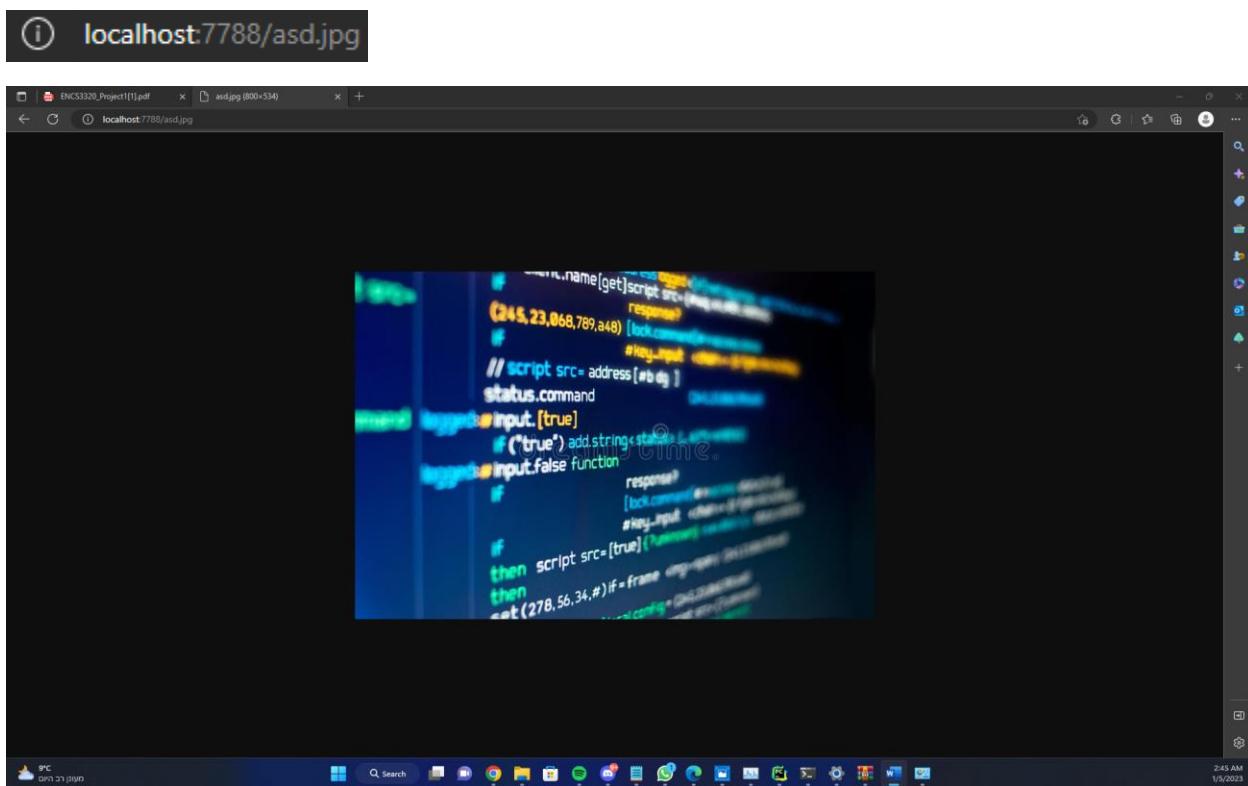
#student-info-2 p {
    padding: 10px;
    margin: 10px;
    display: flex;
    flex-direction: column;
    border: 1px solid black;
}

img {
    width: 200px;
    height: 200px;
}

a {
    color: blue;
}
```

Figure 4/.css

If we request a .jpg file then it displays this image.jpg found online:



If we request a .png file then it displays this image.png found online:

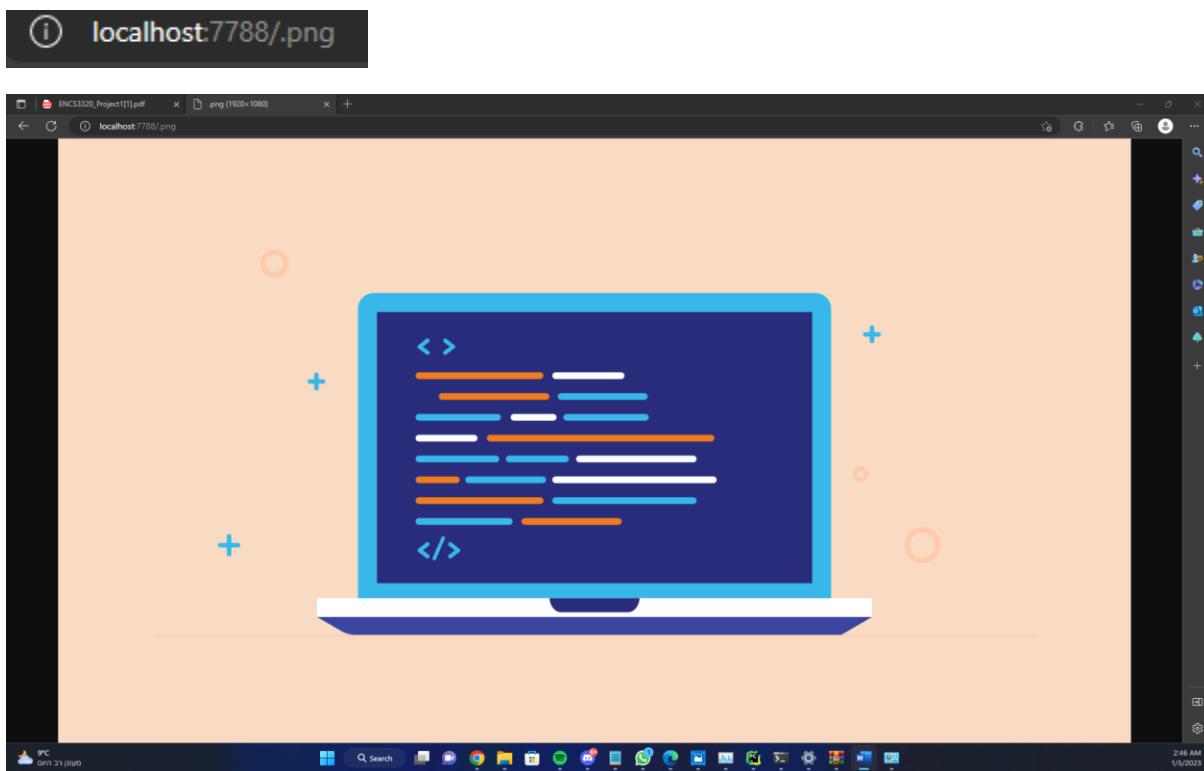


Figure 5.jpg

If we don't request any of the previous, the request doesn't exist so we display the error.html:

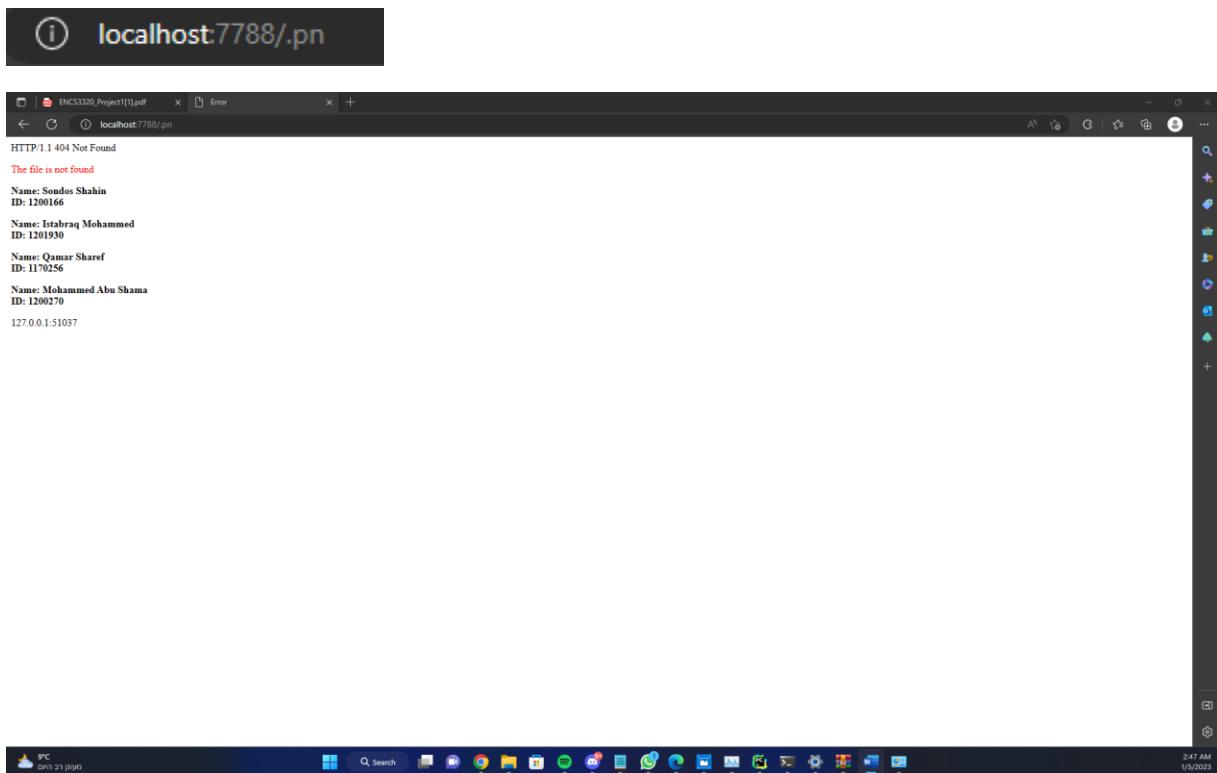


Figure 6/png

Also here are the requests made from phone, to do this we changed the server from local host to our ipv4 local address:



Figure 7 phone /ar

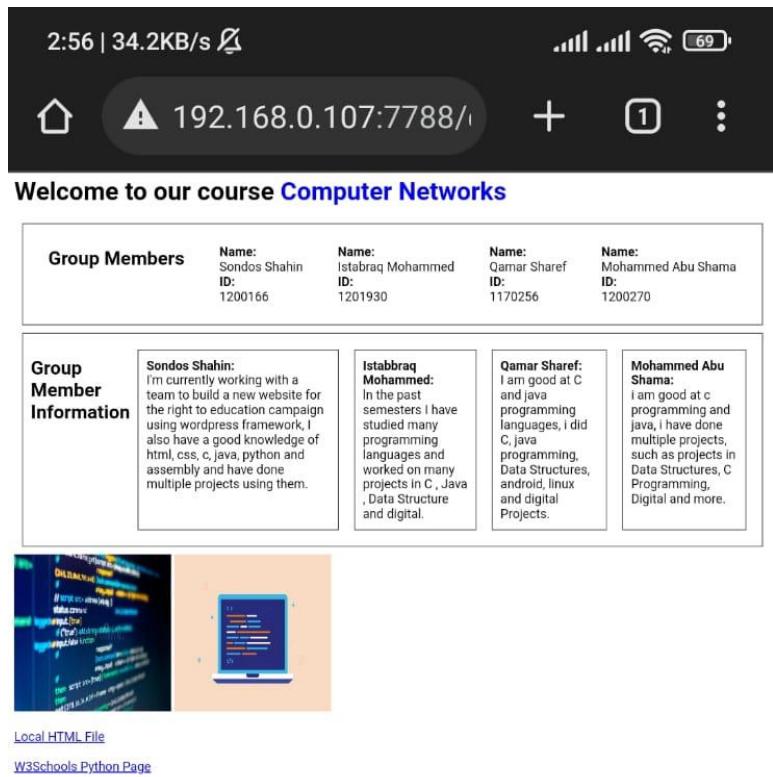
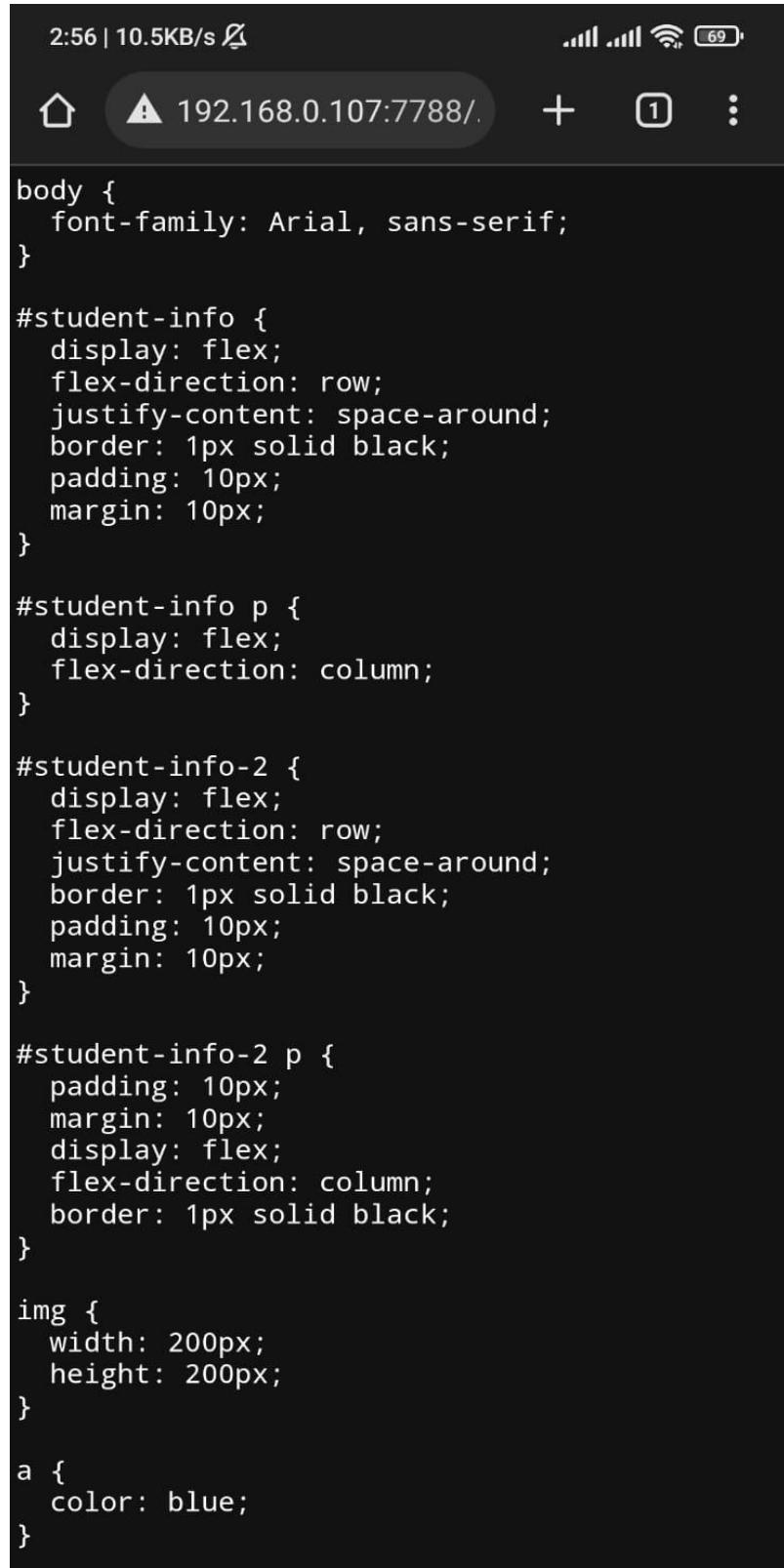


Figure 8 phone/en



A screenshot of a mobile browser displaying a dark-themed page with white text. The browser's header shows the time as 2:56, a speed of 10.5KB/s, signal strength, battery level at 69%, and a URL of 192.168.0.107:7788/. Below the header is a navigation bar with icons for home, back, forward, and tabs. The main content area contains the following CSS code:

```
body {  
    font-family: Arial, sans-serif;  
}  
  
#student-info {  
    display: flex;  
    flex-direction: row;  
    justify-content: space-around;  
    border: 1px solid black;  
    padding: 10px;  
    margin: 10px;  
}  
  
#student-info p {  
    display: flex;  
    flex-direction: column;  
}  
  
#student-info-2 {  
    display: flex;  
    flex-direction: row;  
    justify-content: space-around;  
    border: 1px solid black;  
    padding: 10px;  
    margin: 10px;  
}  
  
#student-info-2 p {  
    padding: 10px;  
    margin: 10px;  
    display: flex;  
    flex-direction: column;  
    border: 1px solid black;  
}  
  
img {  
    width: 200px;  
    height: 200px;  
}  
  
a {  
    color: blue;  
}
```

Figure 9 phonephone/.css

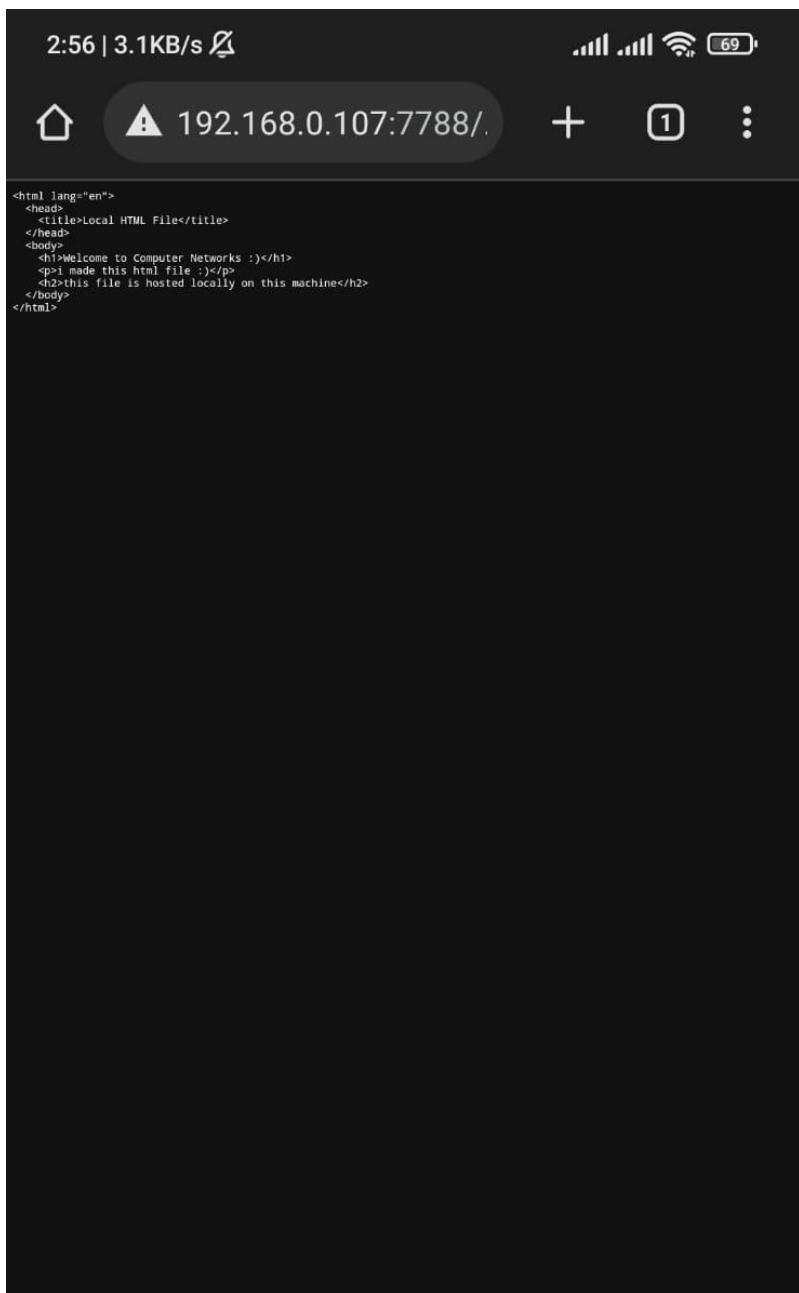


Figure 10 phone/.html

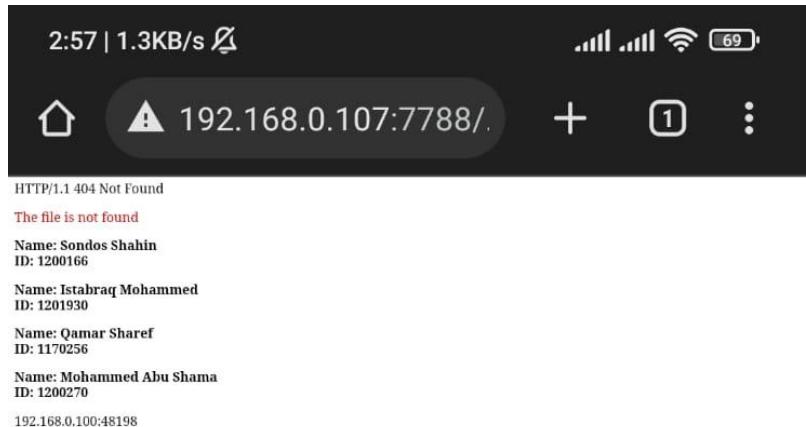


Figure 11 phone/notFound

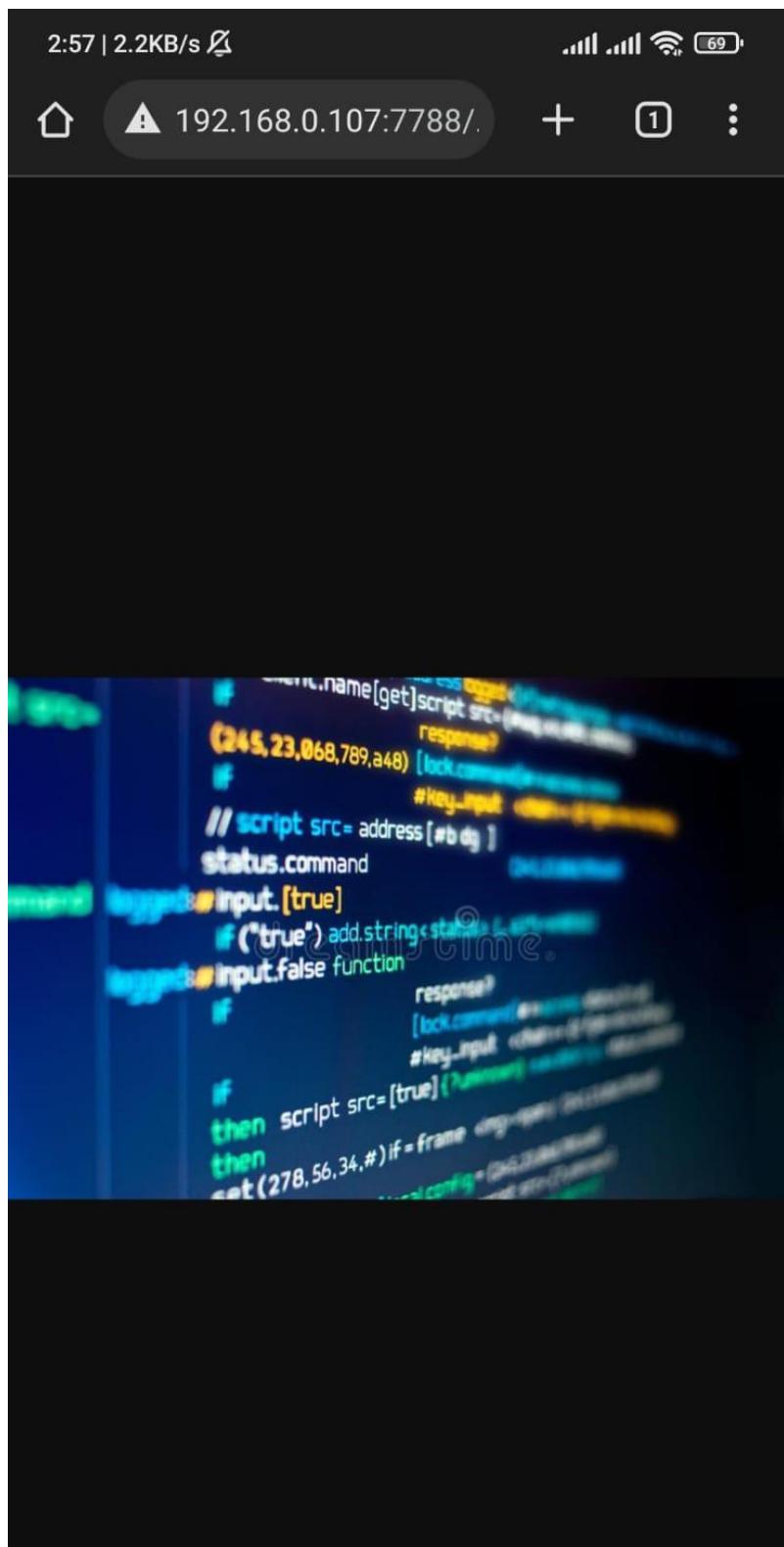


Figure 12phone/.jpg

Task:

First, we run the server:

```
starting up on localhost port 5050
```

Now we run the first client:

```
connecting to localhost port 5050
the message sent: b'hi my name is mohammed' in bytes
```

Now we open the cmd and run the second client:

```
connecting to localhost port 5050
the message sent: b'hi my name is mohammed' in bytes
crc in bytes as received: b'\x00\x00\x00l'
crc in decimal: 108
Negative ACK
```

Because the two clients are now connected the server received two messages from the clients:

```
try:
    # send data to server, the message in bytes format
    message = b'hi my name is mohammed'
    print("the message sent: " + str(message) + " in bytes")
    sock.sendall(message)
```

And now the output for client1 is:

```
connecting to localhost port 5050
the message sent: b'hi my name is mohammed' in bytes
crc in bytes as received: b'\x00\x00\x00l'
crc in decimal: 108
Negative ACK
```

```
Process finished with exit code 0
```

After checking the errors and calculating the CRC for both clients.

The final output of the server:

```
starting up on localhost port 5050
connection from ('127.0.0.1', 64374)
connection from ('127.0.0.1', 64375)
client1:
message: b'hi my name is mohammed'
Cyclic Redundancy Check: 108
client2:
message: b'hi my name is mohammed'
Cyclic Redundancy Check: 108

Process finished with exit code 0
```

Another test run by changing the message to hello there:

Client1:

```
connecting to localhost port 5050
the message sent: b'hello there' in bytes
crc in bytes as received: b'\x00\x00\x00B'
crc in decimal: 66
Negative ACK
```

Client2:

```
connecting to localhost port 5050
the message sent: b'hello there' in bytes
crc in bytes as received: b'\x00\x00\x00B'
crc in decimal: 66
Negative ACK
```

Server:

```
starting up on localhost port 5050
connection from ('127.0.0.1', 64383)
connection from ('127.0.0.1', 64385)
client1:
message: b'hello there'
Cyclic Redundancy Check: 66
client2:
message: b'hello there'
Cyclic Redundancy Check: 66
```