

بسم الله الرحمن الرحيم



Computer Systems Engineering Department

Computer Networks

ENEE3320

Networks Project#1

Instruction: Dr. Abdelkarim Awwad.

Students Names & ID:

Nafe Abubaker, 1200047

Mohammad Abushamma, 1200270

Mohammad Suliman, 1201261

Table of Contents

| | |
|----------|-------------------------|
| 3 | First Part |
| 3 | 1- |
| 3 | 2- |
| 6 | Second Part |
| 21 | Codes: |
| 21 | 1-Python Code |
| 27 | 2- Main_en.html Code: |
| 33 | 3- Main_ar.html Code: |
| 38 | 5-Local_file.html |
| 41 | 6- Labtops.txt content: |

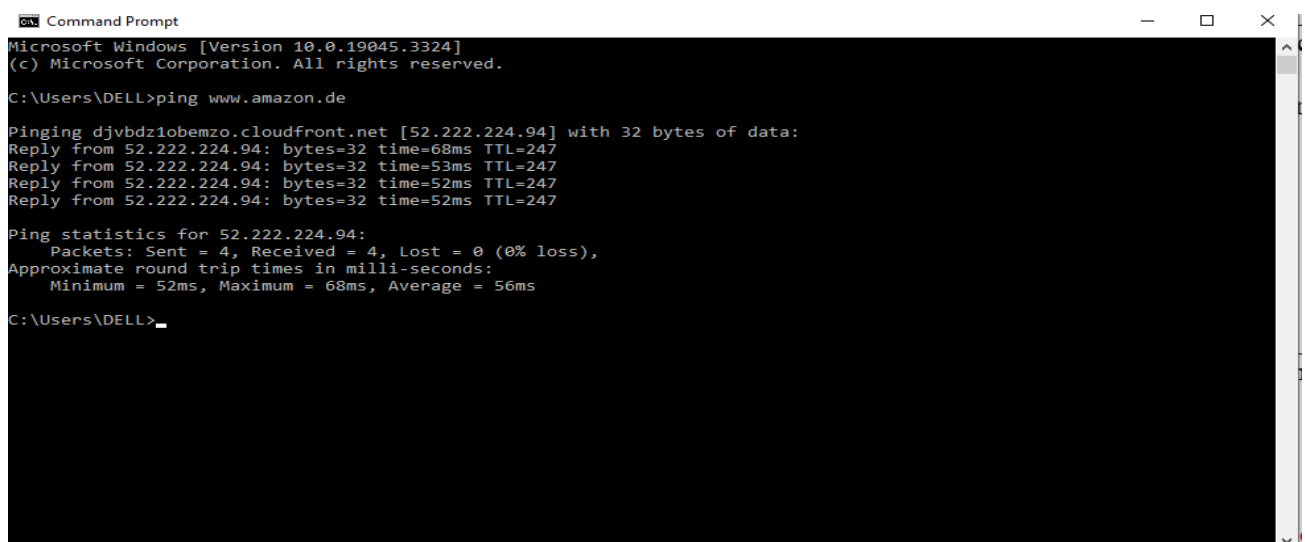
First Part

1-

- 1- **Ping**: network utility that tests the reachability of a host or server by sending small packets of data and measuring the time it takes for the packets to travel to the host and back.
- 2- **Tracert (tracert)**: traces the route taken by data packets as they travel across a network, showing the intermediate hops and their response times.
- 3- **Nslookup**: tool used to query the Domain Name System (DNS) to retrieve information about domain names, such as IP addresses and other DNS records.
- 4- **Telnet**: tool that allows a user to remotely access and manage a computer or server over a network, often used for troubleshooting and administration tasks.

2-

Ping www.amazon.de



```
Command Prompt
Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>ping www.amazon.de

Pinging djvbdz1obemzo.cloudfront.net [52.222.224.94] with 32 bytes of data:
Reply from 52.222.224.94: bytes=32 time=68ms TTL=247
Reply from 52.222.224.94: bytes=32 time=53ms TTL=247
Reply from 52.222.224.94: bytes=32 time=52ms TTL=247
Reply from 52.222.224.94: bytes=32 time=52ms TTL=247

Ping statistics for 52.222.224.94:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 52ms, Maximum = 68ms, Average = 56ms

C:\Users\DELL>
```

A packet of 32 bytes of data was sent, response time took 68 milliseconds, TTL represents number of hops the packet can take before it dies(discarded). There was no packet loss, and the average round-trip times for the packets sent and received was 56 milliseconds.

Tracert www.amazon.de

```
Command Prompt
Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>tracert www.amazon.de

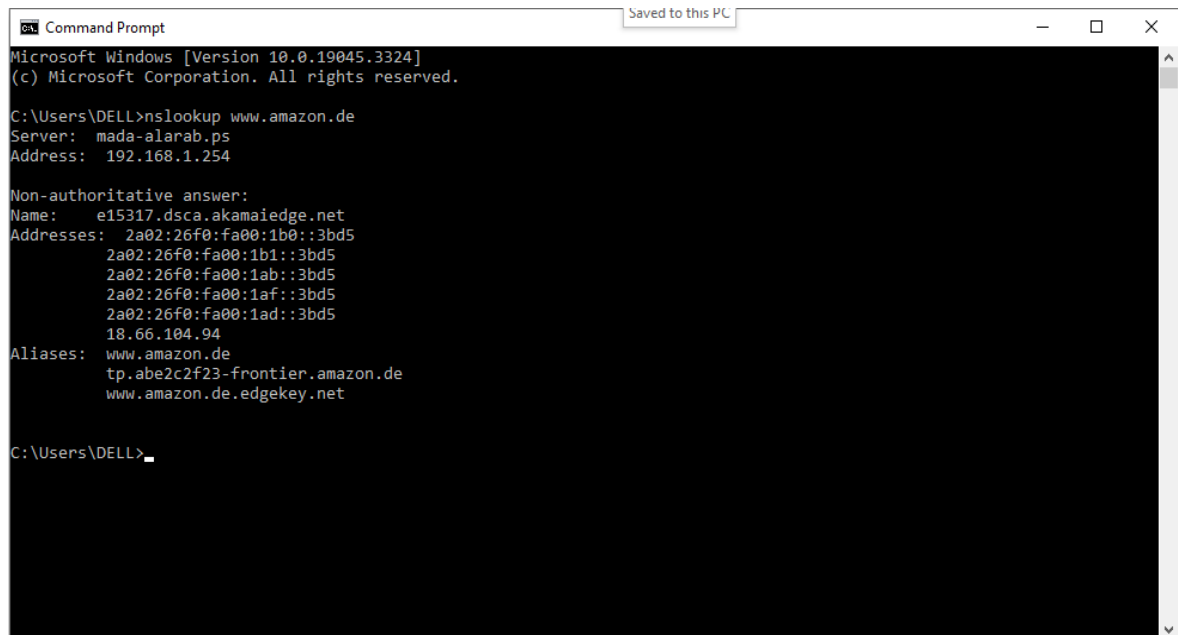
Tracing route to djbvbdzlobemzo.cloudfront.net [52.222.224.94]
over a maximum of 30 hops:
  0  4 ms  1 ms  2 ms  mada-alarab.ps [192.168.1.254]
  1  4 ms  5 ms  6 ms  ADSL-185.17.235.204.mada.ps [185.17.235.204]
  2  6 ms  5 ms  6 ms  172.16.250.93
  3  *      *      *      Request timed out.
  4  59 ms  57 ms  59 ms  ae0-165.cr3-fra2.ip4.gtt.net [77.67.93.9]
  5  60 ms  67 ms  60 ms  ae0.cr7-fra2.ip4.gtt.net [89.149.137.6]
  6  59 ms  62 ms  57 ms  ip4.gtt.net [154.14.139.245]
  7  *      *      *      Request timed out.
  8  *      *      *      Request timed out.
  9  *      *      *      Request timed out.
 10  *      *      *      Request timed out.
 11  53 ms  54 ms  54 ms  15.230.131.163
 12  *      *      *      Request timed out.
 13  51 ms  54 ms  54 ms  server-52-222-224-94.fra56.r.cloudfront.net [52.222.224.94]

Trace complete.

C:\Users\DELL>
```

- The command traces the route that packets take to get from your computer to a specific destination(www.amazon.de).
- The first column shows the hop number.
- The second column shows the round-trip time in milliseconds (ms) for the packet to reach that hop and return.
 - The third column shows the round-trip time for the second packet.
 - The fourth column shows the round-trip time for the third packet.
- The remainder of the columns typically provide information about the IP address or domain name of the hop," Request timed out" means that no response was given at an expected time.
- According the final destination, where the server is being served, the 'fra56.r' shows that this server is located in Frankfurt.

Nslookup www.amazon.de



```
Command Prompt
Saved to this PC

Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>nslookup www.amazon.de
Server: mada-alarab.ps
Address: 192.168.1.254

Non-authoritative answer:
Name: e15317.dsca.akamaiedge.net
Addresses: 2a02:26f0:fa00:1b0::3bd5
           2a02:26f0:fa00:1b1::3bd5
           2a02:26f0:fa00:1ab::3bd5
           2a02:26f0:fa00:1af::3bd5
           2a02:26f0:fa00:1ad::3bd5
           18.66.104.94
Aliases: www.amazon.de
         tp.abe2c2f23-frontier.amazon.de
         www.amazon.de.edgekey.net

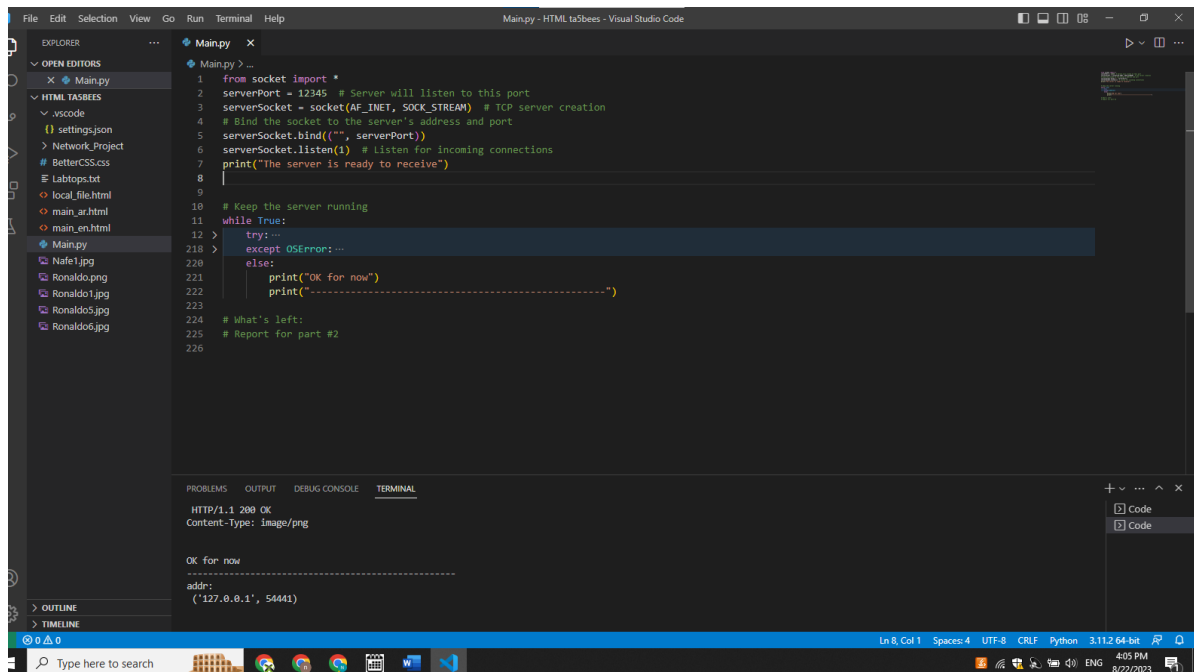
C:\Users\DELL>
```

by using the **nslookup** command, we've gained insight into how the domain **www.amazon.de** is resolved to actual IP addresses and the underlying infrastructure, such as the use of the Akamai CDN, to serve content to users.

Second Part

In this part I will be explaining codes, runs, etc.

And since we are required to insert the date and time of our code, here is a screenshot of the whole code with the date and time in the above screen. Because I will be explaining every code alone.



The screenshot shows a Visual Studio Code editor window with a Python file named `Main.py` open. The file contains a simple HTTP server implementation using the `socket` module. The code is as follows:

```
1 from socket import *
2 serverPort = 12345 # Server will listen to this port
3 serverSocket = socket(AF_INET, SOCK_STREAM) # TCP server creation
4 # Bind the socket to the server's address and port
5 serverSocket.bind(('', serverPort))
6 serverSocket.listen(1) # Listen for incoming connections
7 print("The server is ready to receive")
8
9
10 # Keep the server running
11 while True:
12     try:
13         # Handle incoming connections
14         clientSocket, addr = serverSocket.accept()
15         # Send a response
16         clientSocket.send("HTTP/1.1 200 OK\nContent-Type: image/png\n\n")
17         # Print the address of the client
18         print("OK for now")
19         print("-----")
20         print(addr)
21     except OSError:
22         # Report for part #2
23         print("-----")
24
25 # What's left:
26 # Report for part #2
```

The terminal window at the bottom shows the output of the program, which is an HTTP response and the client's address:

```
HTTP/1.1 200 OK
Content-Type: image/png

OK for now
-----
addr:
('127.0.0.1', 54441)
```

```
1  from socket import *
2  serverPort = 12345 # Server will listen to this port
3  serverSocket = socket(AF_INET, SOCK_STREAM) # TCP server creation
4  # Bind the socket to the server's address and port
5  serverSocket.bind(("", serverPort))
6  serverSocket.listen(1) # Listen for incoming connections
7  print("The server is ready to receive")
8
```

- First line imports all the functions, constants, and classes available in the **socket** module. The **socket** module is a built-in Python library that provides a way to communicate over networks using the socket programming paradigm.
- Second line This line defines a variable named **serverPort** and assigns it the value **12345**. This value represents the port number on which the server will listen for incoming client connections. In the context of network communications, a port number is used to identify a specific process to which network messages are sent.
- The **socket()** function is used to create a new socket object. This object will be used to manage the server's network communications.
- After all this is done the code prints out that the server is ready to receive, waiting for incoming client connections.

```

while True:
    try:
        # Accept any incoming connection and retrieve the client's socket and address
        connectionSocket, addr = serverSocket.accept()
        # Receive data from the client and decode it
        sentence = connectionSocket.recv(2048).decode()
        print("addr:\n", addr)
        print(sentence)

        request_parts = sentence.split()
        if len(request_parts) < 2:
            # Construct an HTTP response with a "Malformed Request" message
            response = "HTTP/1.1 400 Bad Request\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
            response += "<html><head><title>Error 400</title></head><body>"
            response += "<h1>HTTP/1.1 400 Bad Request</h1>"
            response += "<p>Your request is malformed and could not be understood by the server.</p>"
            response += "</body></html>"

            # Send the response to the client (browser)
            connectionSocket.send(response.encode())
            connectionSocket.close()
            continue

        requested_path = request_parts[1]

```

The **accept()** method of the **serverSocket** object waits for a client to establish a connection. Once a connection is established, this method returns two values:

- **connectionSocket**: A new socket object used for sending and receiving data on the connection. This socket is specific to the connected client.
- **addr**: A tuple containing the IP address and port number of the client.
- The **recv()** method of the **connectionSocket** object waits for data sent by the client. The argument **2048** specifies the maximum amount of data (in bytes) that can be received at once.
- Once data is received, it is decoded from bytes to a string using the **decode()** method. The resulting string (which should be an HTTP request) is stored in the variable **sentence**.
- Print functions are used to print the client's address and the received HTTP request to the console. It provides a log of the incoming requests and their sources.
- “request_parts = sentence.split()” This line splits the received HTTP request (stored in **sentence**) into its individual components

using spaces as delimiters. The resulting list of components is stored in the variable **request_parts**. This operation helps in parsing the HTTP request and extracting necessary information from it.

- “requested_path = request_parts[1]” If the request was valid (i.e., not malformed), this line extracts the path requested by the client. This is typically the second part of an HTTP request (after the HTTP method). For instance, in the request **GET /index.html HTTP/1.1**, the requested path is **/index.html**. The path helps the server determine which resource or file the client is asking for.
- If the request is malformed, a **400 Bad Request** response is sent; otherwise, the server extracts the requested path for further processing.

```

# Check the requested path and send appropriate responses back to the client
if requested_path == '/' or requested_path == '/index.html' or requested_path == '/main_en.html' or requested_path == '/en':
    response_header = "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
    with open("main_en.html", "rb") as f:
        data = f.read()
    connectionSocket.send(response_header.encode())
    connectionSocket.send(data)
    print("Response Header: \n", response_header)
    # print("data: \n", data)

elif requested_path == '/ar':
    response_header = "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
    with open("main_ar.html", "rb") as f:
        data = f.read()
    connectionSocket.send(response_header.encode())
    connectionSocket.send(data)
    print("Response Header: \n", response_header)

```

- If the requested path from the client was any of the shown in code, the response will be **http/1.1 ok**, and the sent content will be in **HTML** form.

Next image shows the content of the request + response as shown in terminal:

```

addr:
('127.0.0.1', 54496)
GET /ar HTTP/1.1
Host: localhost:12345
Connection: keep-alive
sec-ch-ua: "Not/A)Brand";v="99", "Google Chrome";v="115", "Chromium";v="115"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

Response Header:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

```

```

elif requested_path == '/SortByPrice':
    # If the client wants laptops sorted by price
    try:
        laptops = []
        with open("Labtops.txt", "r") as f:
            lines = f.readlines()
            for line in lines:
                parts = line.strip().split(":")
                if len(parts) == 2:
                    laptop_name = parts[0]
                    laptop_price = int(parts[1])
                    laptops.append((laptop_name, laptop_price))

        sorted_laptops = sorted(laptops, key=lambda x: x[1])

        # Construct then send the HTML response
        response_header = "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
        response = response_header + "<html><head><title>Laptops</title></head><body>"
        response += "<h1>Laptops Sorted by Price</h1>"
        response += "<ul>"
        for laptop_name, laptop_price in sorted_laptops:
            response += f"<li>{laptop_name.upper()}: ${laptop_price}</li>"
        response += "</ul>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

    except FileNotFoundError:
        response = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
        response += "<html><head><title>Error 404</title></head><body>"
        response += "<h1>HTTP/1.1 404 Not Found</h1>"
        response += "<p>The file 'Labtops.txt' is not found</p>"
        response += "<p>Nafe Abubaker</p>"
        response += "<p>IP: {} Port: {}".format(
            addr[0], addr[1]) + "</p>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

```

- If the requested path was SortByPrice, then the code will read labtops and their prices from a textfile called Labtops.txt that I will provide the content of at the end. After reading the file the code will store the names and prices in a list, then sort them by the price, after this the code will constructs an HTML response that presents the laptops sorted by price. And then send the constructed HTML response to the client.
- The code also handles the case where the **Labtops.txt** file is not found, sends a **404 Not Found** response to the client indicating the file is missing.

```
Laptops.txt
1 Apple MacBook Air (M2):949
2 MacBook Pro (14-inch and 16-inch):1999
3 14-inch MacBook Pro (2021):2699
4 Asus Zenbook Pro Duo 14:2012
5 Asus ROG Zephyrus G14:1900
6 Asus Chromebook Flip CX5:649
7 MacBook Pro 16 (M2 Pro and M2 Max):2299
8 MacBook Air 15:1099
9 Framework Laptop:849
10 Dell XPS 13:999
11 Asus Zenbook S 13 OLED:1400
12 Microsoft Surface Laptop 5 (13.5-inch):900
13 Lenovo 2022 Newest Ideapad 3 Laptop:327
14 MSI Thin GF63 15.6" 144Hz Gaming Laptop:989
15 MSI Wide GF73 16.5" 288Hz Gaming Laptop:999
```

Figure 1 Laptops.txt

Laptops Sorted by Price

LENOVO 2022 NEWEST IDEAPAD 3 LAPTOP: \$327

ASUS CHROMEBOOK FLIP CX5: \$649

FRAMEWORK LAPTOP: \$849

MICROSOFT SURFACE LAPTOP 5 (13.5-INCH): \$900

APPLE MACBOOK AIR (M2): \$949

MSI THIN GF63 15.6" 144HZ GAMING LAPTOP: \$989

DELL XPS 13: \$999

MSI WIDE GF73 16.5" 288HZ GAMING LAPTOP: \$999

MACBOOK AIR 15: \$1099

ASUS ZENBOOK S 13 OLED: \$1400

ASUS ROG ZEPHYRUS G14: \$1900

MACBOOK PRO (14-INCH AND 16-INCH): \$1999

ASUS ZENBOOK PRO DUO 14: \$2012

MACBOOK PRO 16 (M2 PRO AND M2 MAX): \$2299

14-INCH MACBOOK PRO (2021): \$2699

Figure 2 Laptops Sorted Output

```

elif requested_path == '/SortByName':
    # If the client wants laptops sorted by name
    try:
        laptops = []
        with open("Labtops.txt", "r") as f:
            lines = f.readlines()
            for line in lines:
                parts = line.strip().split(":")
                if len(parts) == 2:
                    laptop_name = parts[0]
                    laptop_price = int(parts[1])
                    laptops.append((laptop_name, laptop_price))

        sorted_laptops = sorted(laptops, key=lambda x: x[0].upper())
        response_header = "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
        response = response_header + "<html><head><title>Laptops</title></head><body>"
        response += "<h1>Laptops Sorted by Name</h1>"
        response += "<ul>"
        for laptop_name, laptop_price in sorted_laptops:
            response += f"<li>{laptop_name.upper()}: ${laptop_price}</li>"
        response += "</ul>"
        response += "</body></html>"

        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

    except FileNotFoundError:
        response = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
        response += "<html><head><title>Error 404</title></head><body>"
        response += "<h1>HTTP/1.1 404 Not Found</h1>"
        response += "<p>The file 'Labtops.txt' is not found</p>"
        response += "<p>Nafe Abubaker</p>"
        response += "<p>IP: {} Port: {}".format(
            addr[0], addr[1]) + "</p>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

```

- If the requested path was SortByName, then the code will read labtops and their prices from a textfile called Labtops.txt. After reading the file the code will store the names and prices in a list, then sort them by the name, after this the code will constructs an HTML response that presents the laptops sorted by price. And then send the constructed HTML response to the client.
- The code also handles the case where the **Labtops.txt** file is not found, sends a **404 Not Found** response to the client indicating the file is missing.

Laptops Sorted by Name

14-INCH MACBOOK PRO (2021): \$2699

APPLE MACBOOK AIR (M2): \$949

ASUS CHROMEBOOK FLIP CX5: \$649

ASUS ROG ZEPHYRUS G14: \$1900

ASUS ZENBOOK PRO DUO 14: \$2012

ASUS ZENBOOK S 13 OLED: \$1400

DELL XPS 13: \$999

FRAMEWORK LAPTOP: \$849

LENOVO 2022 NEWEST IDEAPAD 3 LAPTOP: \$327

MACBOOK AIR 15: \$1099

MACBOOK PRO (14-INCH AND 16-INCH): \$1999

MACBOOK PRO 16 (M2 PRO AND M2 MAX): \$2299

MICROSOFT SURFACE LAPTOP 5 (13.5-INCH): \$900

MSI THIN GF63 15.6" 144HZ GAMING LAPTOP: \$989

MSI WIDE GF73 16.5" 288HZ GAMING LAPTOP: \$999

```

# Redirect the client to external websites based on the path
elif requested_path == '/azn':
    response_header = "HTTP/1.1 307 Temporary Redirect\r\nLocation: https://www.amazon.com\r\n\r\n"
    connectionSocket.send(response_header.encode())
    print("Response Header: \n", response_header)
elif requested_path == '/so':
    response_header = "HTTP/1.1 307 Temporary Redirect\r\nLocation: https://stackoverflow.com\r\n\r\n"
    connectionSocket.send(response_header.encode())
    print("Response Header: \n", response_header)
elif requested_path == '/bzu':
    response_header = "HTTP/1.1 307 Temporary Redirect\r\nLocation: https://www.birzeit.edu\r\n\r\n"
    connectionSocket.send(response_header.encode())
    print("Response Header: \n", response_header)

```

- In this part of the code, we used **HTTP/1.1 307**, to redirect the client into a specific link that depends on what the client requested.
- Here is the requested and response part as shown in terminal, where we used the **HTTP/1.1 307**, the client was redirected to Location: www.amazon.com

```

addr:
('127.0.0.1', 58043)

addr:
('127.0.0.1', 58056)
GET /azn HTTP/1.1
Host: localhost:12345
Connection: keep-alive
sec-ch-ua: "Not(A)Brand";v="99", "Google Chrome";v="115", "Chromium";v="115"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

Response Header:
HTTP/1.1 307 Temporary Redirect
Location: https://www.amazon.com

OK for now
-----

```

```

# If the client requests a CSS file
elif requested_path.endswith('.css'):
    try:
        with open(requested_path[1:], 'rb') as f:
            data = f.read()
        response_header = "HTTP/1.1 200 OK\r\nContent-Type: text/css; charset=utf-8\r\n\r\n"
        connectionSocket.send(response_header.encode())
        connectionSocket.send(data)
        print("Response Header: \n", response_header)
    except FileNotFoundError:
        # Handle 404 for CSS file
        response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
        connectionSocket.send(response_header.encode())
        print("Response Header: \n", response_header)
        print("\nCSS file not found")

```

- This part of code is for any requested **.css** file that is needed, for example: the client(browser) will ask for the **css** file when receiving **Main_en.html** file.
- Request and response part as shown in terminal when asked for **css** file:


```

addr:
 ('127.0.0.1', 58153)
GET /BetterCSS.css HTTP/1.1
Host: localhost:12345
Connection: keep-alive
sec-ch-ua: "Not/A)Brand";v="99", "Google Chrome";v="115", "Chromium";v="115"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: text/css,*/*;q=0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://localhost:12345/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

Response Header:
 HTTP/1.1 200 OK
Content-Type: text/css; charset=utf-8

```

```

# If the client requests a PNG file
elif requested_path.endswith('.png'):
    try:
        with open(requested_path[1:], 'rb') as f:
            data = f.read()
            response_header = "HTTP/1.1 200 OK\r\nContent-Type: image/png\r\n\r\n"
            connectionSocket.send(response_header.encode())
            connectionSocket.send(data)
            print("Response Header: \n", response_header)
    except FileNotFoundError:
        # Handle 404 for PNG file
        response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
        connectionSocket.send(response_header.encode())
        print("Response Header: \n", response_header)
        print("\nPNG file not found")

```

- This part of code is for any requested **.png** file that is needed, for example: the client(browser) will ask for the **png** file when receiving **Main_ar.html** file.
- Request and Reponse content as shown in terminal when asked for **png** file:

```

addr:
  ('127.0.0.1', 58156)
GET /Ronaldo.png HTTP/1.1
Host: localhost:12345
Connection: keep-alive
sec-ch-ua: "Not(A)Brand";v="99", "Google Chrome";v="115", "Chromium";v="115"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 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:12345/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

Response Header:
HTTP/1.1 200 OK
Content-Type: image/png

OK for now
-----

```

```

# If the client requests a JPG or JPEG file
elif requested_path.endswith('.jpg') or requested_path.endswith('.jpeg'):
    try:
        with open(requested_path[1:], 'rb') as f:
            data = f.read()
            response_header = "HTTP/1.1 200 OK\r\nContent-Type: image/jpeg\r\n\r\n"
            connectionSocket.send(response_header.encode())
            connectionSocket.send(data)
            print("Response Header: \n", response_header)
    except FileNotFoundError:
        # Handle 404 for JPG file
        response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
        connectionSocket.send(response_header.encode())
        print("Response Header: \n", response_header)
        print("JPG file not found")

```

- This part of code is for any requested **.jpg** or **.jpeg** file that is needed, for example: the client(browser) will ask for the **jpg** file when receiving **Main_ar.html** file.
- Request and response content as shown in terminal when asked for **jpg** file.

```
addr:
('127.0.0.1', 58155)
GET /Nafel.jpg HTTP/1.1
Host: localhost:12345
Connection: keep-alive
sec-ch-ua: "Not(A)Brand";v="99", "Google Chrome";v="115", "Chromium";v="115"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 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:12345/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

Response Header:
HTTP/1.1 200 OK
Content-Type: image/jpeg

OK for now
-----
```

```
elif requested_path == '/local_file.html':
    try:
        with open("local_file.html", "rb") as f:
            data = f.read()
            response_header = "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
            connectionSocket.send(response_header.encode())
            connectionSocket.send(data)
            print("Response Header: \n", response_header)
    except FileNotFoundError:
        response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
        connectionSocket.send(response_header.encode())
        print("Local_file page not found\n")
        print("Response Header: \n", response_header)
```

Because we have inserted a link in both Main_en.html and Main_ar.html, that referred to **local_file.html**. We implemented an html file of our own.

```

else:
    # Handle 404 Not Found
    response = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html; charset=utf-8\r\n\r\n"
    response += "<html><head><title>Error 404</title></head><body>"
    response += "<h1>HTTP/1.1 404 Not Found</h1>"
    response += "<p style='color: red;*>The file is not found</p>"
    response += "<p style='font-weight: bold;*>Name : Nafe Abubaker | ID : 1200047</p>"
    response += "<p>IP: {} Port: {}".format(addr[0], addr[1]) + "</p>"
    response += "</body></html>"
    connectionSocket.send(response.encode())
    print("Response Header: \n", response)

```

- If the requested path from the user were none of the implemented ones in the project, for example /Abdelkarim, since there is no implementation for /Abdelkarim. Then the response will be done using **HTTP/1.1 404 Not Found**.
- This is the content of response + request when the requested path was /Abdelkarim, and since this path did not meet any condition in the code, **404 Not Found error** was shown to the user.

```

addr:
  ('127.0.0.1', 52939)
GET /Abdelkarim HTTP/1.1
Host: localhost:12345
Connection: keep-alive
sec-ch-ua: "Not(A)Brand";v="99", "Google Chrome";v="115", "Chromium";v="115"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9

Response Header:
  HTTP/1.1 404 Not Found
  Content-Type: text/html; charset=utf-8

<html><head><title>Error 404</title></head><body><h1>HTTP/1.1 404 Not Found</h1><p style='color: red;*>The file is not found</p><p style='font-weight: bold;*>Name : Nafe Abubaker | ID : 1200047</p><p>IP: 127.0.0.1 Port: 52939</p></body></html>
OK for now

```

Codes:

1-Python Code

```
from socket import *
serverPort = 12345 # Server will listen to this port
serverSocket = socket(AF_INET, SOCK_STREAM) # TCP server creation
# Bind the socket to the server's address and port
serverSocket.bind(("", serverPort))
serverSocket.listen(1) # Listen for incoming connections
print("The server is ready to receive")

# Keep the server running
while True:
    try:
        # Accept any incoming connection and retrieve the client's
        # socket and address
        connectionSocket, addr = serverSocket.accept()
        # Receive data from the client and decode it
        sentence = connectionSocket.recv(2048).decode()
        print("addr:\n",)
        print("Sentence:", sentence) # sentence == request

        request_parts = sentence.split()
        # bo5d kol el request message o befselhom 3an ba3ad o b5znhom f
        # tuple
        print(request_parts)
        if len(request_parts) < 2:
            # Construct an HTTP response with a "Malformed Request"
            # message
            response = "HTTP/1.1 400 Bad Request\r\nContent-Type:
            text/html; charset=utf-8\r\n\r\n"
            response += "<html><head><title>Error
            400</title></head><body>"
            response += "<h1>HTTP/1.1 400 Bad Request</h1>"
            response += "<p>Your request is malformed and could not be
            understood by the server.</p>"
            response += "</body></html>"
            # Send the response to the client (browser)
            connectionSocket.send(response.encode())
            connectionSocket.close()
            continue
```

```

        # requested_path : bo5d el matloob mn el request o bkon el
tarteeb ta3o el tane fel tuple
        requested_path = request_parts[1]

        # Check the requested path and send appropriate responses back
to the client
        if requested_path == '/' or requested_path == '/index.html' or
requested_path == '/main_en.html' or requested_path == '/en':
            response_header = "HTTP/1.1 200 OK\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"
            with open("main_en.html", "rb") as f:
                data = f.read()
            connectionSocket.send(response_header.encode())
            connectionSocket.send(data)
            print("Response Header: \n", response_header)
            # print("data: \n", data)

        elif requested_path == '/ar':
            response_header = "HTTP/1.1 200 OK\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"
            with open("main_ar.html", "rb") as f:
                data = f.read()
            connectionSocket.send(response_header.encode())
            connectionSocket.send(data)
            print("Response Header: \n", response_header)

        elif requested_path == '/SortByPrice':
            # If the client wants laptops sorted by price
            try:
                laptops = []
                with open("Labtops.txt", "r") as f:
                    lines = f.readlines()
                    for line in lines:
                        parts = line.strip().split(":")
                        if len(parts) == 2:
                            laptop_name = parts[0]
                            laptop_price = int(parts[1])
                            laptops.append((laptop_name, laptop_price))

                sorted_laptops = sorted(laptops, key=lambda x: x[1])

                # Construct the HTML response
                response_header = "HTTP/1.1 200 OK\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"
                response = response_header +
"<html><head><title>Laptops</title>"
                response += "<link rel='stylesheet'
href='BetterCSS.css'>"

```

```

        response += "</head><body>"
        response += "<div class='container'>"
        response += "<h1 class='heading'>Laptops Sorted by
Price</h1>"
        response += "<ul class='laptop-list'>"
        for laptop_name, laptop_price in sorted_laptops:
            response += f"<li class='laptop-
item'>{laptop_name.upper()}: ${laptop_price}</li>"
        response += "</ul>"
        response += "</div>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

    except FileNotFoundError:
        response = "HTTP/1.1 404 Not Found\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"
        response += "<html><head><title>Error
404</title></head><body>"
        response += "<h1>HTTP/1.1 404 Not Found</h1>"
        response += "<p>The file 'Labtops.txt' is not
found</p>"
        response += "<p>Nafe Abubaker</p>"
        response += "<p>IP: {} Port: {}".format(
            addr[0], addr[1]) + "</p>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

    elif requested_path == '/SortByName':
        # If the client wants laptops sorted by name
        try:
            laptops = []
            with open("Labtops.txt", "r") as f:
                lines = f.readlines()
                for line in lines:
                    parts = line.strip().split(":")
                    if len(parts) == 2:
                        laptop_name = parts[0]
                        laptop_price = int(parts[1])
                        laptops.append((laptop_name, laptop_price))

            sorted_laptops = sorted(laptops, key=lambda x:
x[0].upper())

            # Construct the HTML response
            response_header = "HTTP/1.1 200 OK\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"

```

```

        response = response_header +
"<html><head><title>Laptops</title>"
        response += "<link rel='stylesheet'
href='BetterCSS.css'>"
        response += "</head><body>"
        response += "<div class='container'>"
        response += "<h1 class='heading'>Laptops Sorted by
Name</h1>"
        response += "<ul class='laptop-list'>"
        for laptop_name, laptop_price in sorted_laptops:
            response += f"<li class='laptop-
item'>{laptop_name.upper()}: ${laptop_price}</li>"
        response += "</ul>"
        response += "</div>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

    except FileNotFoundError:
        response = "HTTP/1.1 404 Not Found\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"
        response += "<html><head><title>Error
404</title></head><body>"
        response += "<h1>HTTP/1.1 404 Not Found</h1>"
        response += "<p>The file 'Labtops.txt' is not
found</p>"
        response += "<p>Nafe Abubaker</p>"
        response += "<p>IP: {} Port: {}".format(
            addr[0], addr[1]) + "</p>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

    # Redirect the client to external websites based on the path
    elif requested_path == '/azn':
        response_header = "HTTP/1.1 307 Temporary
Redirect\r\nLocation: https://www.amazon.com\r\n\r\n"
        connectionSocket.send(response_header.encode())
        print("Response Header: \n", response_header)
    elif requested_path == '/so':
        response_header = "HTTP/1.1 307 Temporary
Redirect\r\nLocation: https://stackoverflow.com\r\n\r\n"
        connectionSocket.send(response_header.encode())
        print("Response Header: \n", response_header)
    elif requested_path == '/bzu':
        response_header = "HTTP/1.1 307 Temporary
Redirect\r\nLocation: https://www.birzeit.edu\r\n\r\n"
        connectionSocket.send(response_header.encode())

```



```

        print("Response Header: \n", response_header)

    # If the client requests a CSS file
    elif requested_path.endswith('.css'):
        try:
            with open(requested_path[1:], 'rb') as f:
                data = f.read()
                response_header = "HTTP/1.1 200 OK\r\nContent-Type:
text/css; charset=utf-8\r\n\r\n"
                connectionSocket.send(response_header.encode())
                connectionSocket.send(data)
                print("Response Header: \n", response_header)
        except FileNotFoundError:
            # Handle 404 for CSS file
            response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
            connectionSocket.send(response_header.encode())
            print("Response Header: \n", response_header)
            print("\nCSS file not found")

    # If the client requests a PNG file
    elif requested_path.endswith('.png'):
        try:
            with open(requested_path[1:], 'rb') as f:
                data = f.read()
                response_header = "HTTP/1.1 200 OK\r\nContent-Type:
image/png\r\n\r\n"
                connectionSocket.send(response_header.encode())
                connectionSocket.send(data)
                print("Response Header: \n", response_header)
        except FileNotFoundError:
            # Handle 404 for PNG file
            response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
            connectionSocket.send(response_header.encode())
            print("Response Header: \n", response_header)
            print("\nPNG file not found")

    # If the client requests a JPG or JPEG file
    elif requested_path.endswith('.jpg') or
requested_path.endswith('.jpeg'):
        try:
            with open(requested_path[1:], 'rb') as f:
                data = f.read()
                response_header = "HTTP/1.1 200 OK\r\nContent-Type:
image/jpeg\r\n\r\n"
                connectionSocket.send(response_header.encode())
                connectionSocket.send(data)
                print("Response Header: \n", response_header)
        except FileNotFoundError:

```

```

        # Handle 404 for JPG file
        response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
        connectionSocket.send(response_header.encode())
        print("Response Header: \n", response_header)
        print("JPG file not found")

    elif requested_path == '/local_file.html':
        try:
            with open("local_file.html", "rb") as f:
                data = f.read()
                response_header = "HTTP/1.1 200 OK\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"
                connectionSocket.send(response_header.encode())
                connectionSocket.send(data)
                print("Response Header: \n", response_header)
            except FileNotFoundError:
                response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
                connectionSocket.send(response_header.encode())
                print("Local_file page not found\n")
                print("Response Header: \n", response_header)

        else:
            # Handle 404 Not Found
            response = "HTTP/1.1 404 Not Found\r\nContent-Type:
text/html; charset=utf-8\r\n\r\n"
            response += "<html><head><title>Error 404</title>"
            response += "<style>"
            response += "body { font-family: Arial, sans-serif;
background-color: #f4f4f4; margin: 0; padding: 0; display: flex;
justify-content: center; align-items: center; height: 100vh; }"
            response += ".container { text-align: center; background-
color: white; padding: 20px; border-radius: 10px; box-shadow: 0px 0px
10px rgba(0, 0, 0, 0.2); }"
            response += ".header { color: #ff5252; font-size: 28px;
margin-bottom: 10px; }"
            response += ".message { font-size: 18px; margin-bottom:
20px; }"
            response += ".link { color: #1976d2; text-decoration: none;
font-weight: bold; }"
            response += ".link:hover { text-decoration: underline; }"
            response += "</style>"
            response += "</head><body>"
            response += "<div class='container'>"
            response += "<div class='header'>Oops, something went
wrong!</div>"
            response += "<div class='message'>We couldn't find the page
you're looking for.</div>"

```

```

        response += "<p style='font-weight: bold;'>Name: Nafe
Abubaker | ID: 1200047</p>"
        response += "<p>Your IP: {} | Port: {}".format(
            addr[0], addr[1]) + "</p>"
        response += "<p>You can always go back to our <a
class='link' href='/main_en.html'>Main Page</a>.</p>"
        response += "</div>"
        response += "</body></html>"
        connectionSocket.send(response.encode())
        print("Response Header: \n", response)

    # Close the connection with the client because we are using TCP
I guess
    connectionSocket.close()
except OSError:
    print("IO error")
else:
    print("OK for now")
    print("-----")

```

2- Main_en.html Code:

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>ENCS3320-My Tiny Webserver</title>
  <link rel="stylesheet" href="BetterCSS.css">
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600
&display=swap" rel="stylesheet">
  <style>
    body {
      font-family: 'Poppins', sans-serif;
      background-image: url(https://images.unsplash.com/photo-
1483232539664-d89822fb5d3e?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVuFDB8fHx8fA%3D%3D&auto=fo
rmat&fit=crop&w=1664&q=80);
      background-size: cover;
      background-position: center;
      margin: 0;
      padding: 0;
      color: #fff;
    }

    header {
      text-align: center;
      background-color: rgba(0, 0, 0, 0.7);
      padding: 20px;
    }

    h1 {
      margin: 0;
      font-size: 36px;
    }

    .blue-box {
      background-color: rgba(0, 0, 0, 0.6);
      padding: 20px;
      text-align: center;
    }

    .blue-text {
      color: #00aaff;
    }

    .group-members {

```

```
        background-color: rgba(0, 0, 0, 0.7);
        padding: 20px;
    }

    h2 {
        font-size: 24px;
    }

    ul {
        list-style: none;
        padding: 0;
    }

    li {
        margin-bottom: 10px;
    }

    .student-info {
        display: flex;
        justify-content: center;
        flex-wrap: wrap;
    }

    .info-box {
        background-color: rgba(0, 0, 0, 0.7);
        padding: 20px;
        margin: 20px;
        border-radius: 10px;
        flex: 1;
        max-width: 300px;
    }

    .info-box h3 {
        margin: 0;
        font-size: 24px;
    }

    .info-box p {
        line-height: 1.6;
        margin-top: 10px;
    }

    .red-text {
        color: #ff5252;
    }

    .image-container {
        display: flex;
```

```

        justify-content: center;
        margin-top: 30px;
    }

    .responsive {
        max-width: 100%;
        height: auto;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
        transition: transform 0.3s;
    }

    .responsive:hover {
        transform: scale(1.05);
    }

    .links {
        text-align: center;
        margin-top: 30px;
    }

    .links a {
        color: #00aaff;
        text-decoration: none;
        font-weight: 600;
        margin: 0 10px;
    }

    .links a:hover {
        text-decoration: underline;
    }

    .student-info {
        display: flex;
        justify-content: center;
        flex-wrap: wrap;
        background-color: rgba(0, 0, 0, 0.7);
        /* Add background color */
        padding: 40px 0;
        /* Adjust padding for spacing */
    }
</style>
</head>

<body>
    <header>
        <h1>ENCS3320-My Tiny Webserver</h1>
    </header>

```

```

<div class="blue-box">
  <p>Welcome to our course <span class="blue-text">Computer
Networks</span>. This is a tiny webserver.</p>
</div>
<div class="group-members">
  <h2>Group Members:</h2>
  <ul>
    <li>Nafe Abubaker - #: 1200047</li>
    <li>Mohammad Abushamma - #: 1200270</li>
    <li>Mohammad Suliman - #: 7</li>
  </ul>
</div>
<div class="student-info">
  <div class="info-box">
    <h3>Nafé Abubaker</h3>
    <p>I'm a computer engineer, passionate about networks.
      <br><span class="red-text">Previous Projects:</span>
      <br>1. Connect Five game with AI.
      <br>2. Online Shopping Application with a database.
      <br>3. Verilog prime numbers and Fibonacci counter.
      <br><span class="red-text">Hobbies:</span>
      <br>1. Programming.
      <br>2. Football.
    </p>
  </div>
  <div class="info-box">
    <h3>Mohammad Abushamma</h3>
    <p>I'm a computer engineer who loves soccer.
      <br><span class="red-text">Previous Projects:</span>
      <br>1. Tic Tac Toe game.
      <br>2. Car selling app with a database.
      <br><span class="red-text">Hobbies:</span>
      <br>1. Programming.
      <br>2. Football.
    </p>
  </div>
  <div class="info-box">
    <h3>Mohammad Suliman</h3>
    <p>I'm a computer engineer who enjoys soccer.
      <br><span class="red-text">Previous Projects:</span>
      <br>1. Snake game.
      <br>2. Verilog sequence detection.
      <br><span class="red-text">Hobbies:</span>
      <br>1. Programming.
      <br>2. Football.
    </p>
  </div>
</div>
</div>

```

```
<div class="image-container">
  <div class="image-background">
    
  </div>
  <div class="image-background">
    
  </div>
</div>

<div class="links">
  <a href="local_file.html">Link to Local HTML File</a>
  <a
href="https://www.w3schools.com/python/python_tuples.asp">Link to
w3schools</a>
  </div>
</body>
</html>
```


3- Main_ar.html Code:

```
<!DOCTYPE html>
<html lang="ar">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>ENC3320- خادم الويب الصغير </title>
  <link rel="stylesheet" href="BetterCSS.css">
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600
&display=swap" rel="stylesheet">
  <style>
    body {
      font-family: 'Poppins', sans-serif;
      background-image: url(https://images.unsplash.com/photo-
1483232539664-d89822fb5d3e?ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVuFDB8fHx8fA%3D%3D&auto=fo
rmat&fit=crop&w=1664&q=80);
      background-size: cover;
      background-position: center;
      margin: 0;
      padding: 0;
      color: #fff;
    }

    header {
      text-align: center;
      background-color: rgba(0, 0, 0, 0.7);
      padding: 20px;
    }

    h1 {
      margin: 0;
      font-size: 36px;
    }

    .blue-box {
      background-color: rgba(0, 0, 0, 0.6);
      padding: 20px;
      text-align: center;
    }

    .blue-text {
      color: #00aaff;
    }
  </style>
</head>

<body>
  <div class="blue-box">
    <h1 class="blue-text">ENC3320- خادم الويب الصغير
  </div>
</body>
</html>
```

```
.group-members {
  background-color: rgba(0, 0, 0, 0.7);
  padding: 20px;
}

h2 {
  font-size: 24px;
}

ul {
  list-style: none;
  padding: 0;
}

li {
  margin-bottom: 10px;
}

.student-info {
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
}

.info-box {
  background-color: rgba(0, 0, 0, 0.7);
  padding: 20px;
  margin: 20px;
  border-radius: 10px;
  flex: 1;
  max-width: 300px;
}

.info-box h3 {
  margin: 0;
  font-size: 24px;
}

.info-box p {
  line-height: 1.6;
  margin-top: 10px;
}

.red-text {
  color: #ff5252;
}

.image-container {
```

```

        display: flex;
        justify-content: center;
        margin-top: 30px;
    }

    .responsive {
        max-width: 100%;
        height: auto;
        border-radius: 10px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
        transition: transform 0.3s;
    }

    .responsive:hover {
        transform: scale(1.05);
    }

    .links {
        text-align: center;
        margin-top: 30px;
    }

    .links a {
        color: #00aaff;
        text-decoration: none;
        font-weight: 600;
        margin: 0 10px;
    }

    .links a:hover {
        text-decoration: underline;
    }

    .student-info {
        display: flex;
        justify-content: center;
        flex-wrap: wrap;
        background-color: rgba(0, 0, 0, 0.7);
        /* Add background color */
        padding: 40px 0;
        /* Adjust padding for spacing */
    }
</style>
</head>

<body>
    <header>
        <h1>ENCS3320 - خادم الويب الصغير </h1>

```

```
</header>
<div class="blue-box">
  <p>هذا خادم .<span class="blue-text">شبكات الحاسوب</span> مرحبًا بك في دورتنا </p>
</div>
<div class="group-members">
  <h2>أعضاء المجموعة</h2>
  <ul>
    <li>1200047 :# نافع أبوبكر </li>
    <li>1200270 :# محمد أبوشمة </li>
    <li>7 :# محمد سليمان </li>
  </ul>
</div>
<div class="student-info">
  <div class="info-box">
    <h3>نافع أبوبكر</h3>
    <p>أنا مهندس كمبيوتر ، متحمس لشبكات الاتصال</p>
    <br><span class="red-text">مشاريع سابقة</span>
    <br>1. لعبة "Connect Five" مع الذكاء الاصطناعي
    <br>2. تطبيق التسوق عبر الإنترنت
    <br>3. أرقام أولية وعدد فيبوناتشي باستخدام فيريلو ج
    <br><span class="red-text">هوايات</span>
    <br>1. البرمجة
    <br>2. كرة القدم
  </p>
</div>
<div class="info-box">
  <h3>محمد أبوشمة</h3>
  <p>أنا مهندس كمبيوتر وأحب كرة القدم</p>
  <br><span class="red-text">مشاريع سابقة</span>
  <br>1. لعبة "Tic Tac Toe".
  <br>2. تطبيق بيع السيارات مع قاعدة بيانات
  <br><span class="red-text">هوايات</span>
  <br>1. البرمجة
  <br>2. كرة القدم
</p>
</div>
<div class="info-box">
  <h3>محمد سليمان</h3>
  <p>أنا مهندس كمبيوتر وأستمتع بمشاهدة كرة القدم</p>
  <br><span class="red-text">مشاريع سابقة</span>
  <br>1. لعبة "Snake".
  <br>2. الكشف عن تسلسل باستخدام فيريلو ج
  <br><span class="red-text">هوايات</span>
  <br>1. البرمجة
  <br>2. كرة القدم
</p>
</div>
```

```
</div>

<div class="image-container">
  <div class="image-background">
    
  </div>
  <div class="image-background">
    
  </div>
</div>

<div class="links">
  <a href="local_file.html">رابط لملف HTML محلي</a>
  <a
href="https://www.w3schools.com/python/python_tuples.asp">رابط إلى
w3schools</a>
  </div>
</body>

</html>
```

5-Local_file.html

```
5- <!DOCTYPE html>
6- <html lang="en">
7-
8- <head>
9-     <meta charset="UTF-8">
10-    <meta name="viewport" content="width=device-width, initial-
        scale=1.0">
11-    <title>Cristiano Ronaldo's Career</title>
12-    <link rel="stylesheet" href="BetterCSS.css">
13-    <link
        href="https://cdn.pixabay.com/photo/2017/01/19/16/15/background-
        1992713_1280.jpg" rel="stylesheet">
14-    <style>
15-        body {
16-            font-family: 'Poppins', sans-serif;
17-            background-image:
18-            url(https://cdn.pixabay.com/photo/2017/01/19/16/15/background-
19-            1992713_1280.jpg);
20-            background-size: cover;
21-            background-position: center;
22-            margin: 0;
23-            padding: 0;
24-            color: #fff;
25-        }
26-
27-        /* Additional styles for the Ronaldo page */
28-        .hero-section {
29-            background-color: #333;
30-            color: #fff;
31-            padding: 50px 0;
32-            text-align: center;
33-        }
34-
35-        .ronaldo-images {
36-            display: flex;
37-            justify-content: space-around;
38-            flex-wrap: wrap;
39-            margin-top: 20px;
40-        }
41-
42-        .ronaldo-images img {
43-            flex: 0 0 calc(30% - 20px);
44-            margin: 10px;
45-            border-radius: 10px;
46-            box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
47-            transition: transform 0.3s;
```

```

46-     }
47-
48-     .ronaldo-images img:hover {
49-         transform: scale(1.1);
50-     }
51- </style>
52-</head>
53-
54-<body>
55-
56-     <header class="hero-section">
57-         <h1>Cristiano Ronaldo</h1>
58-         <p>The greatest footballers of all time</p>
59-     </header>
60-
61-     <section class="content-section">
62-         <h2>Introduction</h2>
63-         <p>
64-             Cristiano Ronaldo, often referred to as CR7, is a
65-             Portuguese professional footballer widely considered one
66-             of the greatest football players in history. With
67-             numerous awards, titles, and records under his belt,
68-             Ronaldo's prowess on the field is unmatched.
69-         </p>
70-         <h2>Career Highlights</h2>
71-         <ul>
72-             <li>Began his professional career with Sporting CP in
73-             Portugal.</li>
74-             <li>Joined Manchester United in 2003, winning three
75-             Premier League titles and a UEFA Champions League.</li>
76-             <li>Moved to Real Madrid in 2009, becoming the club's
77-             all-time top scorer.</li>
78-             <li>Won four Champions League titles with Real
79-             Madrid.</li>
80-             <li>Joined Juventus in 2018, continuing to break
81-             records in Serie A.</li>
82-             <li>Returned to Manchester United in 2021.</li>
83-             <li>Has won five Ballon d'Or awards, given to the
84-             best player in the world.</li>
85-         </ul>
86-
87-         <div class="ronaldo-images">
88-             <!-- Placeholder images; ensure you have the rights
89-             to use any images you include -->
90-             
91-             
92-             

```

```
85-         </div>
86-     </section>
87-
88- </body>
89-
90- </html>
```


6- Labtops.txt content:

Apple MacBook Air (M2):949
MacBook Pro (14-inch and 16-inch):1999
14-inch MacBook Pro (2021):2699
Asus Zenbook Pro Duo 14:2012
Asus ROG Zephyrus G14:1900
Asus Chromebook Flip CX5:649
MacBook Pro 16 (M2 Pro and M2 Max):2299
MacBook Air 15:1099
Framework Laptop:849
Dell XPS 13:999
Asus Zenbook S 13 OLED:1400
Microsoft Surface Laptop 5 (13.5-inch):900
Lenovo 2022 Newest Ideapad 3 Laptop:327
MSI Thin GF63 15.6" 144Hz Gaming Laptop:989
MSI Wide GF73 16.5" 288Hz Gaming Laptop:999

يعطيك العافية مس كاتبة