Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

ENCS3320 – Computer Networks

Project #1 (Socket Programming)

Team Members:

Shahd Khalaf – 1210545 – Section #1

Sama Wahidee – 1211503 – Section #1

Majd Hamarsheh – 1212306 – Section #5

Submission Date : Dec 1, 2024

Table of Contents:

Table of figures:

Abstract:

The project, which is a part of the ENCS3320 Computer Networks course, uses tasks involving the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) to investigate the fundamentals of socket programming and networking. Developing an interactive multiplayer trivia game, building a custom web server, and getting practical expertise with key network commands are among the goals. The project, which covers important ideas including the HTTP protocol, socket connectivity, and server-client interaction, places an emphasis on both theoretical comprehension and real-world application. Working together, we used our programming expertise to develop a scalable, modular solution using Wireshark and other network analysis tools. This report describes the procedures, results, and difficulties that arose throughout the project's execution.

Task 1- Network Commands and Wireshark:

Section A:

1. **Ipconfig - Internet Protocol configuration**: [7]
   - Displays details about your computer's current network setup, including IP address and TCP/IP settings. It also allows refreshing DHCP and DNS configurations to fix network issues.

2. **ping**: [8]
   - Checks if a device or website is reachable by sending ICMP Echo Request packets over a network and waiting for a reply (ICMP Echo Reply). It measures response time to help troubleshoot connectivity issues.

3. **tracert**: [9]
   - Traces the path data takes to reach a destination by showing each router or hop and the time taken for the journey. This helps diagnose network delays or routing issues.

4. **telnet - teletype network**: [10]
   - Allows remote access to another computer via a simple text-based interface for diagnostics, testing open ports, and monitoring network services. It typically uses port 23 but can be configured to use other ports.

5. **nslookup - DNS lookup**: [11]
   - Retrieves the IP address of a domain or confirms domain functionality to verify and troubleshoot DNS records and configurations. It is useful for checking DNS propagation and resolving domain issues.

Section B:

1. **Run the ipconfig /all command:**

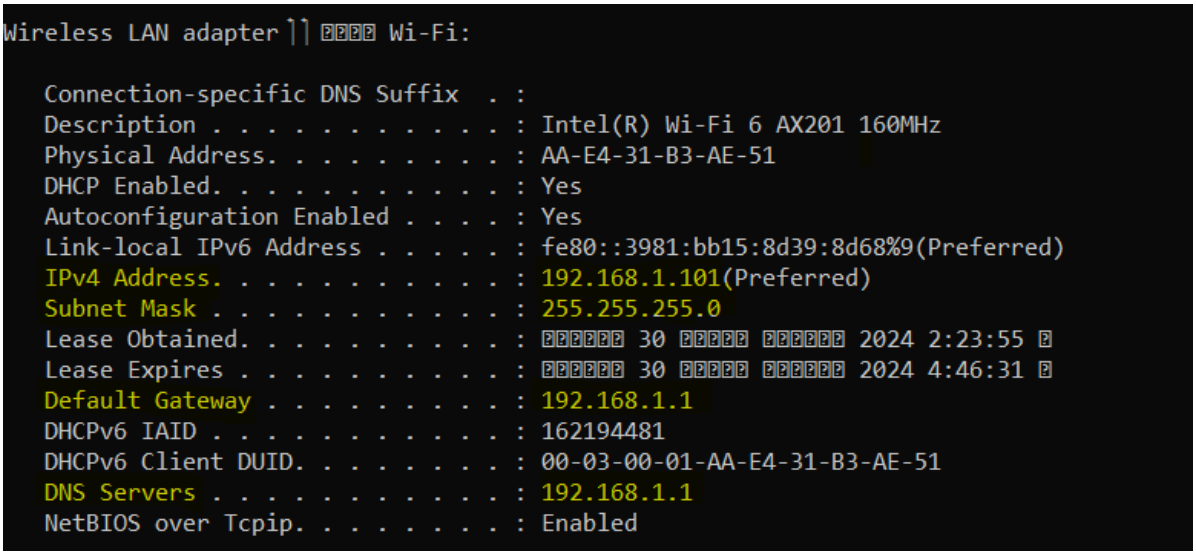


figure 1-output of `ipconfig/all -Task 1`

The following are the retrieved values for the primary network interface:

- IP Address: 192.168.1.101
- Subnet Mask: 255.255.255.0
- Default Gateway: 192.168.1.1
- Domain Name System: (DNS): 192.168.1.1

2. Ping an iPhone within the local network:

```
C:\Users\Ibrah>ping 192.168.1.100

Pinging 192.168.1.100 with 32 bytes of data:
Reply from 192.168.1.100: bytes=32 time=44ms TTL=64
Reply from 192.168.1.100: bytes=32 time=45ms TTL=64
Reply from 192.168.1.100: bytes=32 time=82ms TTL=64
Reply from 192.168.1.100: bytes=32 time=126ms TTL=64

Ping statistics for 192.168.1.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 44ms, Maximum = 126ms, Average = 74ms
```

figure 2- output of ping an iphone - Task 1

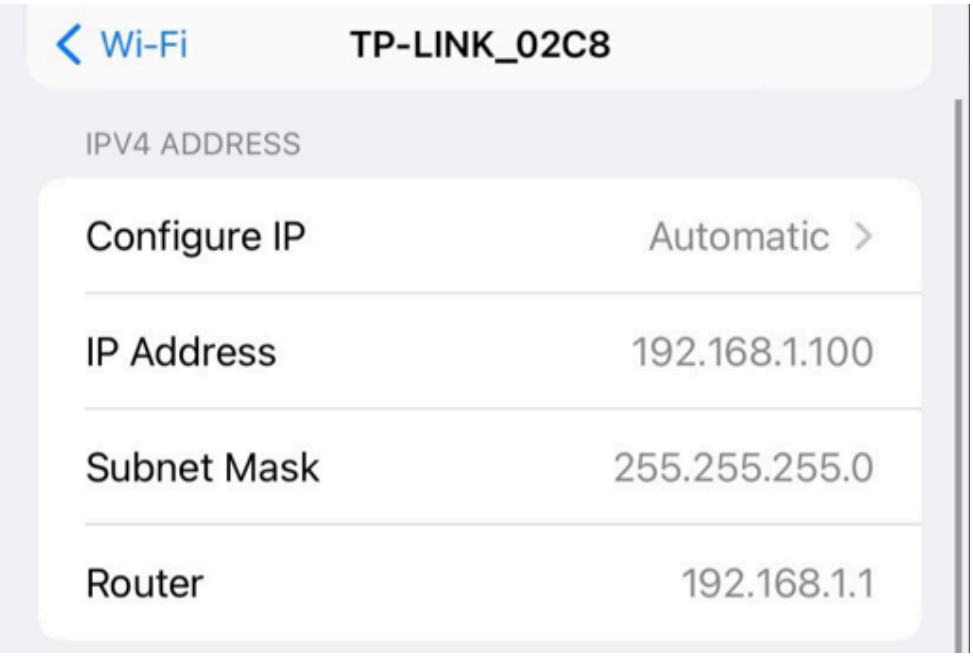


figure 3-iphone IP address - Task 1

The ping results demonstrate successful communication between the laptop and the iphone with the IP address 192.168.1.100 within the local network. The analysis is as follows:

1. **Connectivity**:All ping requests were successfully sent and received, with no packet loss (0%), confirming that the device is reachable and active on the same Wi-Fi network.
2. **Round-Trip Time (RTT)**:
   The time for packets to travel to the target device and back ranged from **44ms** (minimum) to **126ms** (maximum), with an average of **74ms**. This variation in latency is typical for local wireless networks and may be influenced by factors like network traffic or signal strength.
3. **TTL (Time to Live)**:
   The TTL value of **64** indicates direct communication with the device, as expected within a local network, without significant routing.

Notes: -If we attempt to ping a host and get a "request timed out" message, it may indicate that the host is unavailable or that it is preventing any further ping requests.

-If we ping a host and get the response, "destination host unreachable," it means that the path to the target was not found.

3. **Ping discover.engineering.utoronto.ca.:**

```
C:\Users\Ibrah>Ping discover.engineering.utoronto.ca

Pinging discover.engineering.utoronto.ca [23.185.0.2] with 32 bytes of data:
Reply from 23.185.0.2: bytes=32 time=51ms TTL=59
Reply from 23.185.0.2: bytes=32 time=70ms TTL=59
Reply from 23.185.0.2: bytes=32 time=57ms TTL=59
Reply from 23.185.0.2: bytes=32 time=55ms TTL=59

Ping statistics for 23.185.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 51ms, Maximum = 70ms, Average = 58ms
```

figure 4: output of ping discover.engineering.utoronto.ca- Task 1

Based on the results of pinging discover.engineering.utoronto.ca, the IP address 23.185.0.2 was identified. When searching for the location of this IP address [4], it was

found to be in San Francisco, California, USA. This suggests that the response does not originate from Canada. The University of Toronto may be using a third-party hosting service or a Content Delivery Network (CDN) to serve their website,

4. Run tracert on discover.engineering.utoronto.ca:

```
C:\Users\Ibrah>tracert  discover.engineering.utoronto.ca

Tracing route to discover.engineering.utoronto.ca [23.185.0.2]
over a maximum of 30 hops:

  1     4 ms     1 ms     1 ms  192.168.1.1
  2   127 ms     8 ms     7 ms  172.20.1.1
  3     6 ms     6 ms     5 ms  213.6.240.220
  4    54 ms    50 ms    63 ms  10.74.19.21
  5    53 ms    49 ms    48 ms  10.74.19.146
  6    55 ms    53 ms    52 ms  10.74.59.250
  7     *        *        *     Request timed out.
  8    82 ms    51 ms   136 ms  23.185.0.2

Trace complete.
```

figure 5- output of tracert discover.engineering.utoronto.ca - Task1

The tracert results for discover.engineering.utoronto.ca show that the server at IP 23.185.0.2 was reached in **8 hops**. The first few hops indicate routers within our local network and ISP, with low response times.

Hop 7 shows a "Request timed out," which is common as some routers block traceroute requests. The final hop to the server shows response times between **51ms and 136ms**, indicating a relatively short network distance from our location. This suggests the server is efficiently reachable.

5. **Nslookup discover.engineering.utoronto.ca:**

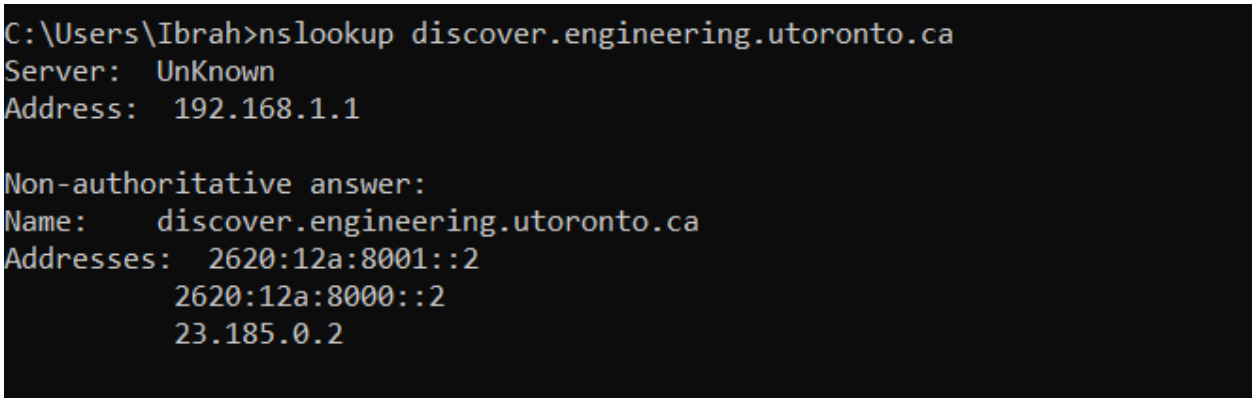


figure 6- output of nslookup discover.engineering.utoronto.ca - Task1

The nslookup command was used to retrieve DNS information for discover.engineering.utoronto.ca.:

- The server used for the lookup is 192.168.1.1, which is usually our local router or DNS server.
- The non-authoritative answer means the information came from a cached source rather than directly from the main DNS server for the domain.
- The IP addresses for discover.engineering.utoronto.ca include:
    - Two IPv6 addresses (2620:12a:8001::2 and 2620:12a:8000::2).
    - One IPv4 address (23.185.0.2), which is the most common address type used on the internet.

*This shows the addresses that can be used to access the website.

6. **Telnet discover.engineering.utoronto.ca:**

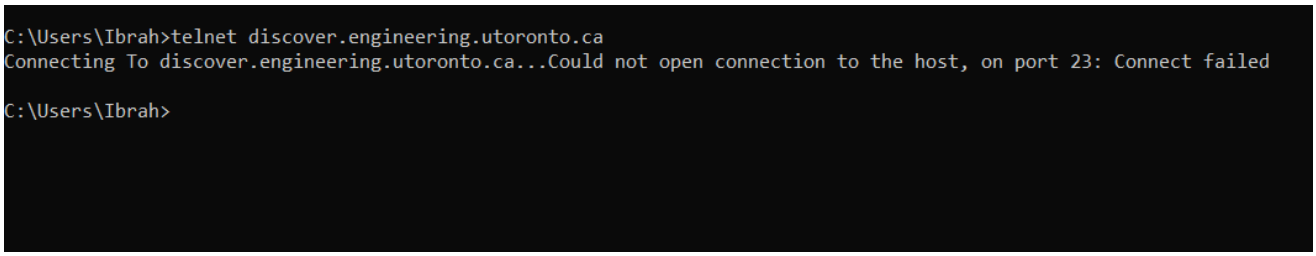


figure 7- output of telnet discover.engineering.utoronto.ca - Task1

When we tried to telnet discover.engineering.utoronto.ca, it failed and we received this error message **"Could not open connection to the host, on port 23: Connect failed"** ,it indicates

that Telnet was unable to establish a connection to discover.engineering.utoronto.ca on the default port 23. This could be due to several reasons: Telnet Service Not Available on Port 23 ,Firewall Blocking Telnet,Wrong Port.

So we used different port (port=80),

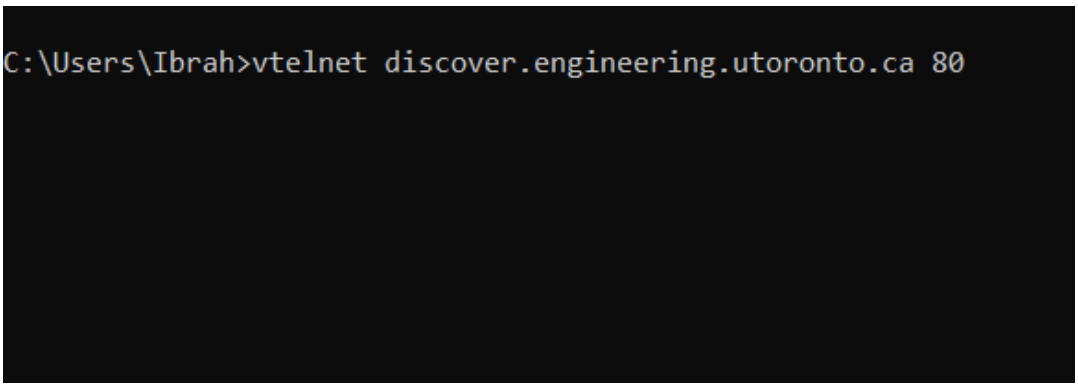


figure 8-telnet command with different port - Task1

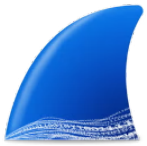
figure 9-output of telnet discover.engineering.utoronto.ca 80 - Task1

To test connectivity to the server discover.engineering.utoronto.ca, we attempted to use Telnet. After specifying the hostname and port, we were able to establish a connection, but the screen remained blank. This indicates that the connection to the server was successful, but the server did not send any immediate response.

In such cases, the server may be waiting for a specific input, such as an HTTP request, before sending data back. This is common when connecting to web servers, as they require commands

like GET / HTTP/1.1 to retrieve webpage content. Telnet is useful for testing simple connections, but it doesn't automatically handle specific communication protocols like HTTP or HTTPS.

Section C:



A network protocol analyzer called Wireshark is used to record and examine packet-level communication. Among other protocols, it offers insights into DNS query-response cycles.[2]

figure 10: Wireshark

The hostname we choice: www.google.com

- After downloading and opining the wireshark we went to the terminal and wrote: "nslookup www.google.com"

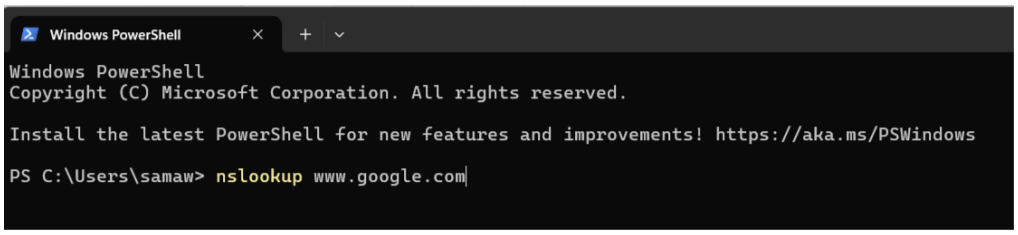
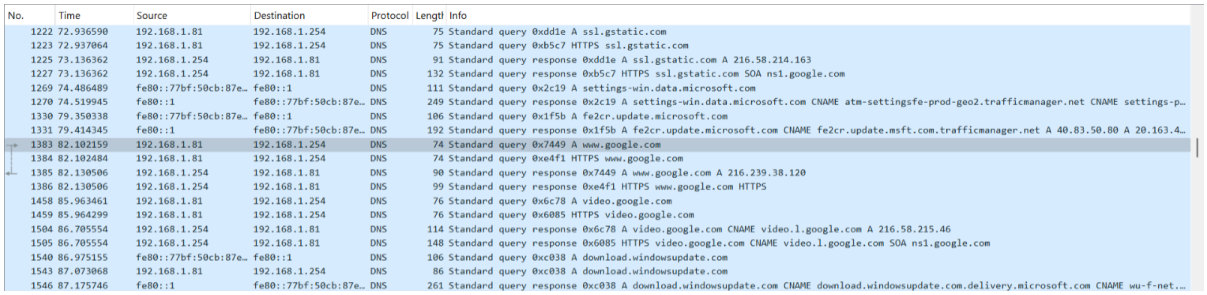


figure 11:terminal 1 - Task 1

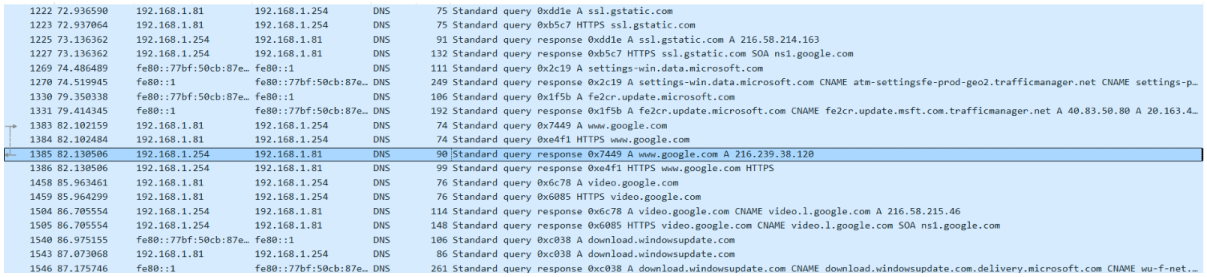
- DNS Query (We had a Filter for DNS Traffic):



No.	Time	Source	Destination	Protocol	Length	Info
1222	72.936590	192.168.1.81	192.168.1.254	DNS	75	Standard query 0xdd1e A ssl.gstatic.com
1223	72.937064	192.168.1.81	192.168.1.254	DNS	75	Standard query 0xb5c7 HTTPS ssl.gstatic.com
1225	73.136362	192.168.1.254	192.168.1.81	DNS	91	Standard query response 0xdd1e A ssl.gstatic.com A 216.58.214.163
1227	73.136362	192.168.1.254	192.168.1.81	DNS	132	Standard query response 0xb5c7 HTTPS ssl.gstatic.com SOA ns1.google.com
1269	74.486489	fe80::77bf:50cb:87e…	fe80::1	DNS	111	Standard query 0x2c19 A settings-win.data.microsoft.com
1270	74.519945	fe80::1	fe80::77bf:50cb:87e…	DNS	249	Standard query response 0x2c19 A settings-win.data.microsoft.com CNAME atm-settingsfe-prod-geo2.trafficmanager.net CNAME settings-p…
1330	79.350338	fe80::77bf:50cb:87e…	fe80::1	DNS	106	Standard query 0x1f5b A fe2cr.update.microsoft.com
1331	79.414345	fe80::1	fe80::77bf:50cb:87e…	DNS	192	Standard query response 0x1f5b A fe2cr.update.microsoft.com CNAME fe2cr.update.msft.com.trafficmanager.net A 40.83.50.80 A 20.163.4…
1383	82.102159	192.168.1.81	192.168.1.254	DNS	74	Standard query 0x7449 A www.google.com
1384	82.102484	192.168.1.81	192.168.1.254	DNS	74	Standard query 0xe4f1 HTTPS www.google.com
1385	82.130506	192.168.1.254	192.168.1.81	DNS	90	Standard query response 0x7449 A www.google.com A 216.239.38.120
1386	82.130506	192.168.1.254	192.168.1.81	DNS	99	Standard query response 0xe4f1 HTTPS www.google.com HTTPS
1458	85.963461	192.168.1.81	192.168.1.254	DNS	76	Standard query 0x6c78 A video.google.com
1459	85.964299	192.168.1.81	192.168.1.254	DNS	76	Standard query 0x6085 HTTPS video.google.com
1504	86.705554	192.168.1.254	192.168.1.81	DNS	114	Standard query response 0x6c78 A video.google.com CNAME video.l.google.com A 216.58.215.46
1505	86.705554	192.168.1.254	192.168.1.81	DNS	148	Standard query response 0x6085 HTTPS video.google.com CNAME video.l.google.com SOA ns1.google.com
1540	86.975155	fe80::77bf:50cb:87e…	fe80::1	DNS	106	Standard query 0xc038 A download.windowsupdate.com
1543	87.073068	192.168.1.81	192.168.1.254	DNS	86	Standard query 0xc038 A download.windowsupdate.com
1546	87.175746	fe80::1	fe80::77bf:50cb:87e…	DNS	261	Standard query response 0xc038 A download.windowsupdate.com CNAME download.windowsupdate.com.delivery.microsoft.com CNAME wu-f-net.…

figure 12:DNS Query - Task 1

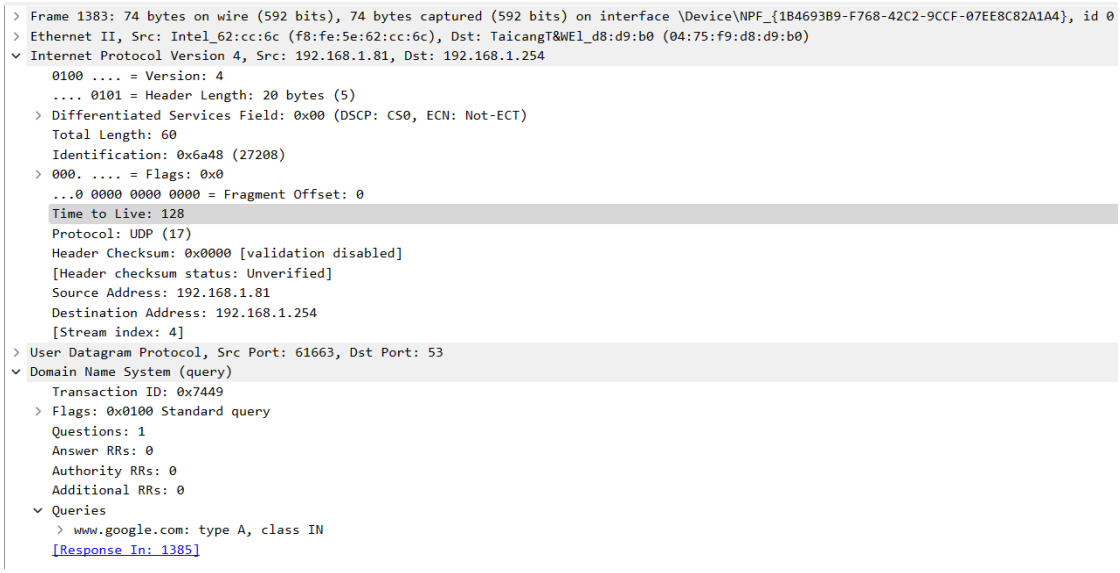
- DNS Response (We had a Filter for DNS Traffic):



1222	72.936590	192.168.1.81	192.168.1.254	DNS	75	Standard query 0xdd1e A ssl.gstatic.com
1223	72.937064	192.168.1.81	192.168.1.254	DNS	75	Standard query 0xb5c7 HTTPS ssl.gstatic.com
1225	73.136362	192.168.1.254	192.168.1.81	DNS	91	Standard query response 0xdd1e A ssl.gstatic.com A 216.58.214.163
1227	73.136362	192.168.1.254	192.168.1.81	DNS	132	Standard query response 0xb5c7 HTTPS ssl.gstatic.com SOA ns1.google.com
1269	74.486489	fe80::77bf:50cb:87e…	fe80::1	DNS	111	Standard query 0x2c19 A settings-win.data.microsoft.com
1270	74.519945	fe80::1	fe80::77bf:50cb:87e…	DNS	249	Standard query response 0x2c19 A settings-win.data.microsoft.com CNAME atm-settingsfe-prod-geo2.trafficmanager.net CNAME settings-p…
1330	79.350338	fe80::77bf:50cb:87e…	fe80::1	DNS	106	Standard query 0x1f5b A fe2cr.update.microsoft.com
1331	79.414345	fe80::1	fe80::77bf:50cb:87e…	DNS	192	Standard query response 0x1f5b A fe2cr.update.microsoft.com CNAME fe2cr.update.msft.com.trafficmanager.net A 40.83.50.80 A 20.163.4…
1383	82.102159	192.168.1.81	192.168.1.254	DNS	74	Standard query 0x7449 A www.google.com
1384	82.102484	192.168.1.81	192.168.1.254	DNS	74	Standard query 0xe4f1 HTTPS www.google.com
1385	82.130506	192.168.1.254	192.168.1.81	DNS	90	Standard query response 0x7449 A www.google.com A 216.239.38.120
1386	82.130506	192.168.1.254	192.168.1.81	DNS	99	Standard query response 0xe4f1 HTTPS www.google.com HTTPS
1458	85.963461	192.168.1.81	192.168.1.254	DNS	76	Standard query 0x6c78 A video.google.com
1459	85.964299	192.168.1.81	192.168.1.254	DNS	76	Standard query 0x6085 HTTPS video.google.com
1504	86.705554	192.168.1.254	192.168.1.81	DNS	114	Standard query response 0x6c78 A video.google.com CNAME video.l.google.com A 216.58.215.46
1505	86.705554	192.168.1.254	192.168.1.81	DNS	148	Standard query response 0x6085 HTTPS video.google.com CNAME video.l.google.com SOA ns1.google.com
1540	86.975155	fe80::77bf:50cb:87e…	fe80::1	DNS	106	Standard query 0xc038 A download.windowsupdate.com
1543	87.073068	192.168.1.81	192.168.1.254	DNS	86	Standard query 0xc038 A download.windowsupdate.com
1546	87.175746	fe80::1	fe80::77bf:50cb:87e…	DNS	261	Standard query response 0xc038 A download.windowsupdate.com CNAME download.windowsupdate.com.delivery.microsoft.com CNAME wu-f-net.…

figure 13: DNS Response - Task 1

- Key Wireshark Fields to Highlight:



```
> Frame 1383: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{1B4693B9-F768-42C2-9CCF-07EE8C82A1A4}, id 0
> Ethernet II, Src: Intel_62:cc:6c (f8:fe:5e:62:cc:6c), Dst: TaicangT&WEl_d8:d9:b0 (04:75:f9:d8:d9:b0)
v Internet Protocol Version 4, Src: 192.168.1.81, Dst: 192.168.1.254
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x6a48 (27208)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: UDP (17)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.81
    Destination Address: 192.168.1.254
    [Stream index: 4]
> User Datagram Protocol, Src Port: 61663, Dst Port: 53
v Domain Name System (query)
    Transaction ID: 0x7449
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    > www.google.com: type A, class IN
    [Response In: 1385]
```

figure 14:Wireshark Highlights - Task 1

1. **Version:** the internet protocol addressing version is IPV4.
2. **Header Length:** the total length of the header is 20 bytes.
3. **TTL:** The packet can travel through 128 routers.

4. **Source Address:** the local IP address of my PC = 192.168.1.81.
5. **Destination Address:** the IP address of the destination = 192.168.1.254.
6. **Source Port:** the source port of the packets = 61663.
7. **Destination Port:** the destination port to the packets = 53.

Task 2- Web Server:

This is a demo for Task2: Task2 Demo

This is a phone demo for Task2: Task2PhoneDemo

This is the source code for Task2: Task2 Source Code

Theory and Procedure:

HTTP protocol:

The HTTP protocol is based on the request-response paradigm. When clients (browsers) ask for resources, servers reply with error codes or content. Client-server communication is guided by status codes such as 200 OK, 404 Not Found, and 307 Temporary Redirect.[1]

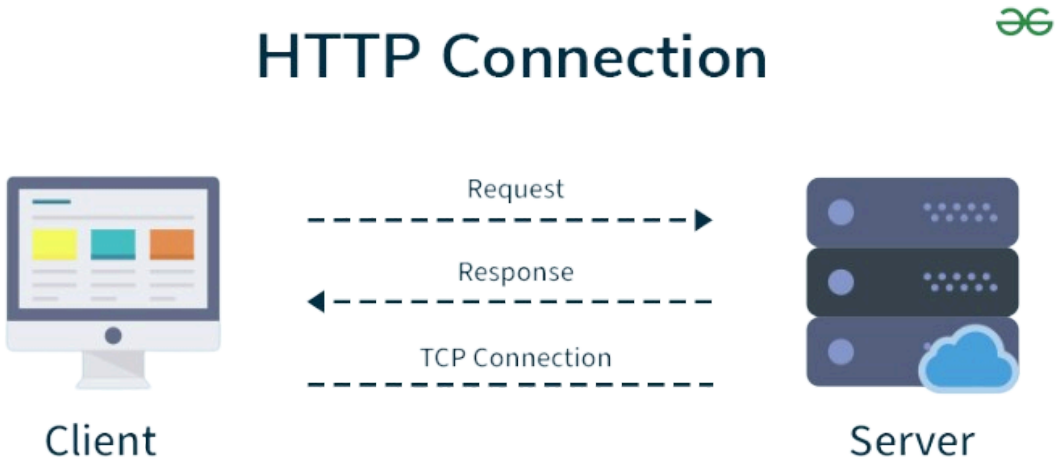


figure 15:HTTP protocol - Task 2

Socket programming:

essential for facilitating data transmission by establishing communication over a network between a client (web browser) and a server (web server). In this project, a server that listens for incoming HTTP requests and replies with the relevant content, like HTML or CSS, is created using socket programming. The usage of the TCP/IP protocol, in which IP manages data routing and TCP guarantees dependable, connection-oriented communication, is one of the key ideas involved. The server binds to a particular local IP address and port (e.g., 5698) to listen for requests, and the socket is defined using constants like SOCK_STREAM (for TCP) and AF_INET (for IPv4). [3]

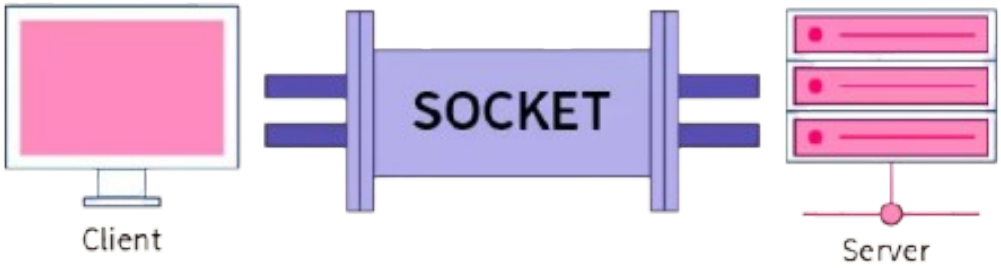


figure 16:Socket programming - Task 2

Web Server Overview:

Using socket programming, we are building a web server that can serve several web pages and listen on port 5698. HTML, CSS, PNG, MP4, and other content types should all be supported by the server, which should also be able to respond dynamically to requests for files. The content will be handled by the server in both Arabic and English, and it will have a function that allows users to request particular photographs or movies depending on networking topics.

The following HTML pages will be served:

- Main English Webpage (main_en.html)
- Supporting Material Page (supporting_material_en.html)
- Arabic Versions (main_ar.html and supporting_material_ar.html)

The server will respond with the appropriate content type (e.g., text/html, text/css, image/png) and handle redirections when necessary (e.g., for unavailable files).

Section A - Main English Webpage:

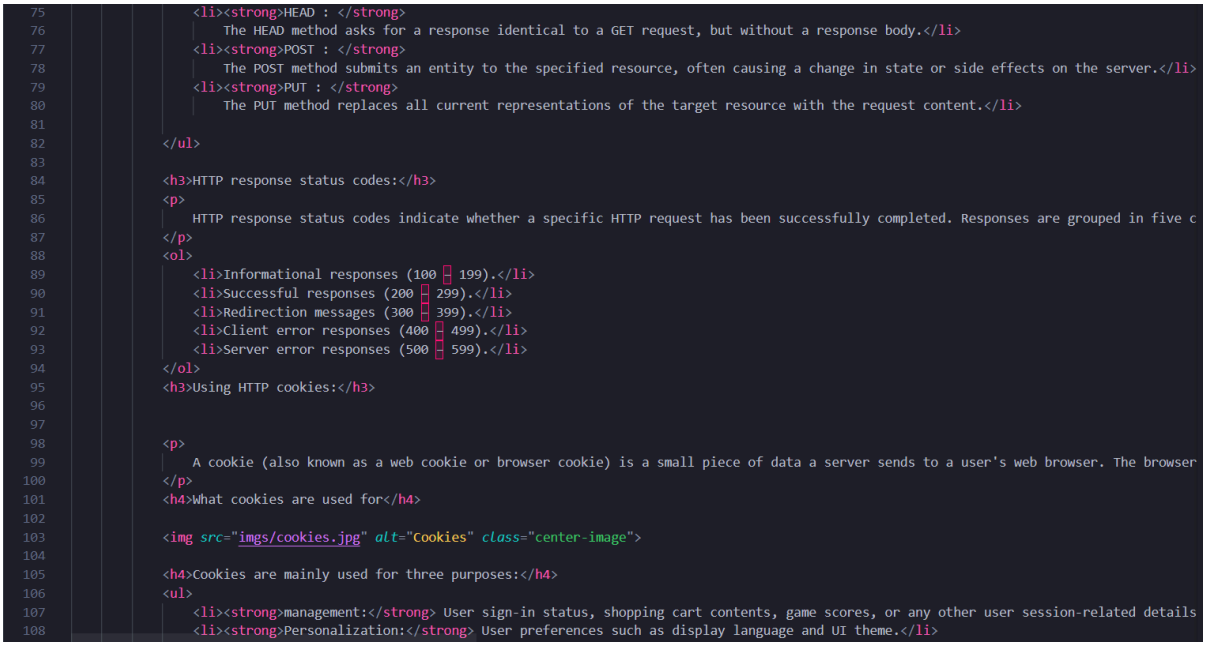
- **HTML/CSS Code:**
  - **main_en.html:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ENCS3320-Webserver</title>
    <link rel="stylesheet" href="/css/styles_main_en.css">
</head>
<body>
    <header>
        <h1>Welcome to ENCS3320 - Computer Networks Webserver</h1>
    </header>
    <a class="arabic-link" href="/main_ar.html">||ARABIC||</a>

    <section class="team-section">
        <h2>Team Members</h2>
        <div class="team-container">
            <div class="team-box">
                <img src="imgs/img1.png" alt="Shahd Khalaf">
                <h3>Shahd Khalaf</h3>
                <p class="id">ID: 1210545</p>
                <p>
                    As a dedicated computer science student, I specialize in Java and C, with a keen interest in solving real-world problems through
                    My internship in full-stack development allowed me to contribute to a project called <strong>Insurance Management System</strong
                    Outside of my studies, I am passionate about drawing, which enhances my creativity, attention to detail, and ability to think ou

                </p>
            </div>
            <div class="team-box">
                <img src="imgs/img3.png" alt="Majd Hamarsheh">
                <h3>Majd Hamarsheh</h3>
                <p class="id">ID: 1212036</p>
                <p>
                    I am a fourth-year computer science student at Birzeit University, with a strong interest in technology, innovation, and AI/mach
                    I am skilled in several programming languages, including Java and C.I thrive in team settings and value creativity and leadershi
                    which have contributed to the success of group projects like developing an <strong>Android app for renting cars</strong> as part
                    Outside of academics, I am committed to volunteer work.
                </p>
            </div>
```

figure 17:main_en.html - Task 2

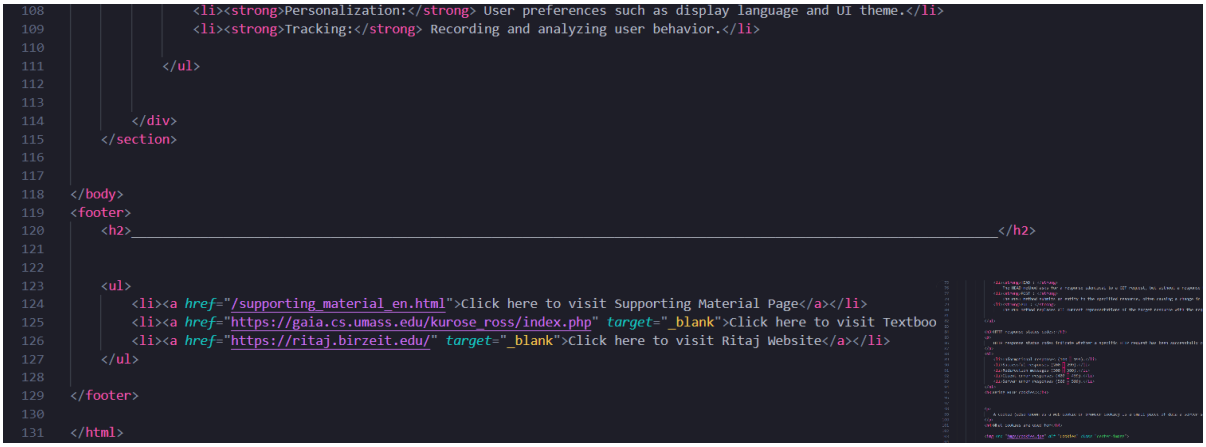
```html
            <div class="team-box">
                <img src="imgs/img2.png" alt="Sama Wahidee">
                <h3>Sama Wahidee</h3>
                <p class="id">ID: 1211503</p> <
                <p>
                    I am a computer science student with strong skills in Java, C, and JavaScript, and a deep passion for technology and problem-sol
                    I recently completed an internship in .NET development, where I contributed to building a <strong>Task Management System</strong
                    I am also passionate about volunteering for causes that make a positive impact on society.
                </p>
            </div>
        </div>
    </section>
    <section class="concept-section">
        <h2 class="concept-title">HTTP: An Overview</h2>
        <div class="concept-box">
            <img src="imgs/http1.jpg" alt="HTTP Overview" class="concept-image">
            <h3 class="highlight-title">What is HTTP?</h3>
            <h4>
                HTTP: hypertext transfer protocol
            </h4>

            <p>
                HTTP is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between
            </p>
            <p>
                HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a
            </p>

            <h3>HTTP request methods:</h3>
            <p>
                HTTP defines a set of request methods to indicate the purpose of the request and what is expected if the request is successful. Alth
            </p>
            <ul>
                <li><strong>GET : </strong>
                    The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should not
```

figure 18:main_en.html - Task 2



```
                <li><strong>HEAD : </strong>
                    The HEAD method asks for a response identical to a GET request, but without a response body.</li>
                <li><strong>POST : </strong>
                    The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.</li>
                <li><strong>PUT : </strong>
                    The PUT method replaces all current representations of the target resource with the request content.</li>

            </ul>

            <h3>HTTP response status codes:</h3>
            <p>
                HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five c
            </p>
            <ol>
                <li>Informational responses (100 – 199).</li>
                <li>Successful responses (200 – 299).</li>
                <li>Redirection messages (300 – 399).</li>
                <li>Client error responses (400 – 499).</li>
                <li>Server error responses (500 – 599).</li>
            </ol>
            <h3>Using HTTP cookies:</h3>


            <p>
                A cookie (also known as a web cookie or browser cookie) is a small piece of data a server sends to a user's web browser. The browser
            </p>
            <h4>What cookies are used for</h4>

            <img src="imgs/cookies.jpg" alt="Cookies" class="center-image">

            <h4>Cookies are mainly used for three purposes:</h4>
            <ul>
                <li><strong>management:</strong> User sign-in status, shopping cart contents, game scores, or any other user session-related details
                <li><strong>Personalization:</strong> User preferences such as display language and UI theme.</li>
```

figure 19:main_en.html - Task 2



```
                <li><strong>Personalization:</strong> User preferences such as display language and UI theme.</li>
                <li><strong>Tracking:</strong> Recording and analyzing user behavior.</li>

            </ul>


        </div>
    </section>


</body>
<footer>
    <h2>_____</h2>


    <ul>
        <li><a href="/supporting_material_en.html">Click here to visit Supporting Material Page</a></li>
        <li><a href="https://gaia.cs.umass.edu/kurose_ross/index.php" target="_blank">Click here to visit Textboo
        <li><a href="https://ritaj.birzeit.edu/" target="_blank">Click here to visit Ritaj Website</a></li>
    </ul>

</footer>

</html>
```

figure 20:main_en.html - Task 2

- **styles_main_en.css**

```css
/* General Page Styling */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #ffeef2; /* Light pink background */
    color: #e63946; /* Neutral text color */
}

/* Header Styling */
header {
    background-color: #f77f98; /* Light red for the header */
    color: white;
    text-align: center;
    padding: 20px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2); /* Subtle shadow for depth */
}

/* Team Section Styling */
.team-section {
    padding: 40px;
    text-align: center;
}

h4 {
    font-style: italic;
}

h4 {
    color: #e63946;
    font-style: italic;
}

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

figure 21: styles_main_en.css1 - Task 2

```css
    flex-wrap: wrap;
    gap: 20px;
    justify-content: center;
}

.team-box {
    background-color: white;
    border-radius: 10px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    padding: 20px;
    width: 400px; /* Increase the width of the box */
    text-align: center;
    transition: transform 0.2s;
    display: inline-block; /* Prevent the box from stretching vertically */
    overflow: hidden; /* Ensure no overflowing content */
}

.team-box:hover {
    transform: scale(1.05); /* Slightly enlarge box on hover */
}

/* Image Styling */
.team-box img {
    width: 150px;
    height: 150px;
    object-fit: cover; /* Ensures the image fills the circle without distortion */
    border-radius: 50%; /* Makes the image circular */
    margin-bottom: 15px;
    max-width: 100%; /* Prevent stretching */
    max-height: 100%; /* Prevent stretching */
}

/* ID Styling */
.team-box .id {
```

figure 22: styles_main_en.css - Task 2

```
    color: #f77f98; /* Blue color for the ID text */
    font-weight: bold; /* Make the ID stand out */
}

/* Paragraph Styling */
.team-box p {
    font-size: 0.9em;
    line-height: 1.4;
    color: #666; /* Subtle text color for descriptions */
}

.team-box h3 {
    margin: 10px 0;
    font-size: 1.2em;
    color: #e63946; /* Matches the header for consistency */
}

/* Concept Section Styling */
.concept-section {
    padding: 40px;
    margin: 20px auto;
    text-align: center;
    max-width: 1200px; /* Increased max width to make the section wider */
}

/* Title outside the box */
.concept-title {
    color: #e63946;
    margin-bottom: 20px;
    font-size: 1.8em;
    text-align: center;
}

/* Box styling */
```

figure 23: styles_main_en.css - Task 2

```
.concept-box {
    background-color: #fff; /* White background for contrast */
    border-radius: 10px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1); /* Subtle shadow for depth */
    padding: 20px;
    text-align: left;
    margin: 0 auto;
    max-width: 1200px; /* Increased max-width */
    width: 100%; /* Ensure the box takes up the full width within the max-width */
}

/* Image inside the box */
.concept-image {
    float: right; /* Align the image to the right */
    margin: 0 0 20px 20px; /* Add spacing around the image */
    max-width: 35%; /* Further reduced image width for even more space for text */
    height: auto;
    border-radius: 8px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1); /* Adds depth to the image */
}

/* Text inside the box */
.concept-box p {
    font-size: 1em;
    line-height: 1.6;
    color: #666;
    margin-bottom: 20px;
}

/* Subheadings inside the box */
.concept-box h3 {
    color: #f77f98; /* A complementary shade to maintain harmony */
    margin-top: 20px;
}

```

figure 24: styles_main_en.css - Task 2

```css
/* Lists inside the box */
.concept-box ul, .concept-box ol {
    margin: 20px 0;
    padding-left: 20px;
}

.concept-box ul li, .concept-box ol li {
    margin: 10px 0;
    font-size: 0.95em;
    color: #666; /* Darker for readability */
}

/* Highlight title styling */
.highlight-title {
    color: #0073e6; /* Blue color for emphasis */
    font-weight: bold; /* Make the text bold */
    /* Slightly larger font size */
    margin-bottom: 10px; /* Add spacing below the title */
}
.center-image {
    display: block;
    margin-left: auto;
    margin-right: auto;
    max-width: 100%; /* Optional: Ensures the image stays responsive */
    height: auto;    /* Optional: Maintains aspect ratio */
}


/* Styling for Arabic link */
.arabic-link {
    float: left;
    margin-top: 10px;
    font-size: 16px;
    color: #e63946;
    text-decoration: none;
}
```

figure 25: styles_main_en.css - Task 2

```css
}


.arabic-link:hover {
    color: #f77f98; /* Change color on hover */
    text-decoration: underline;
}

```

figure 26:: styles_main_en.css - Task 2

- **Web Server Interface:**
  - **main_en.html:**

    The following interface appears when the web browser reaches http://127.0.0.1:5698/main_en.html.

    The title "Welcome to ENCS3320 - Computer Networks Webserver" is shown at the top of this page, which serves as the web server's primary entrance point. The following elements make up the page's organization:

    1. **Team Member Information:** The team members' names, IDs, and photos are shown along with other details about their projects, abilities, and interests.

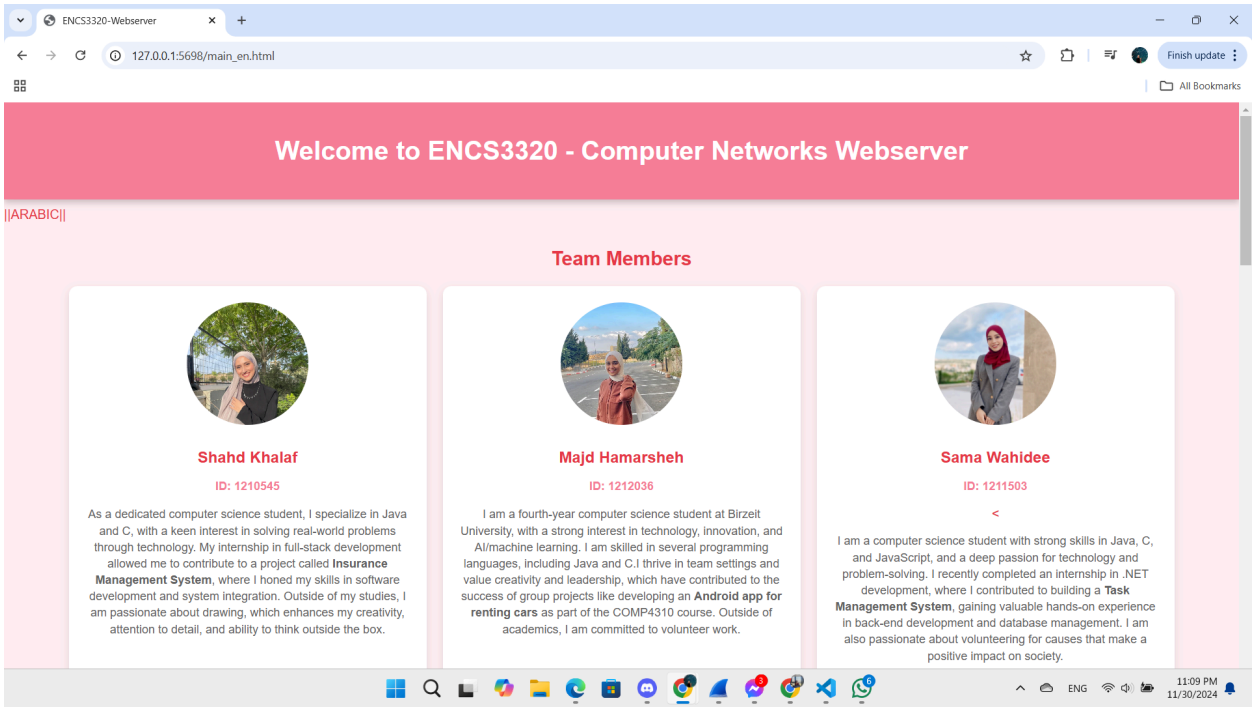


figure 27:Web Server Interface - Task 2

2. **Course Content:** Including prepared text, graphics, and lists, a topic from the course material is presented in an effective manner.[5]

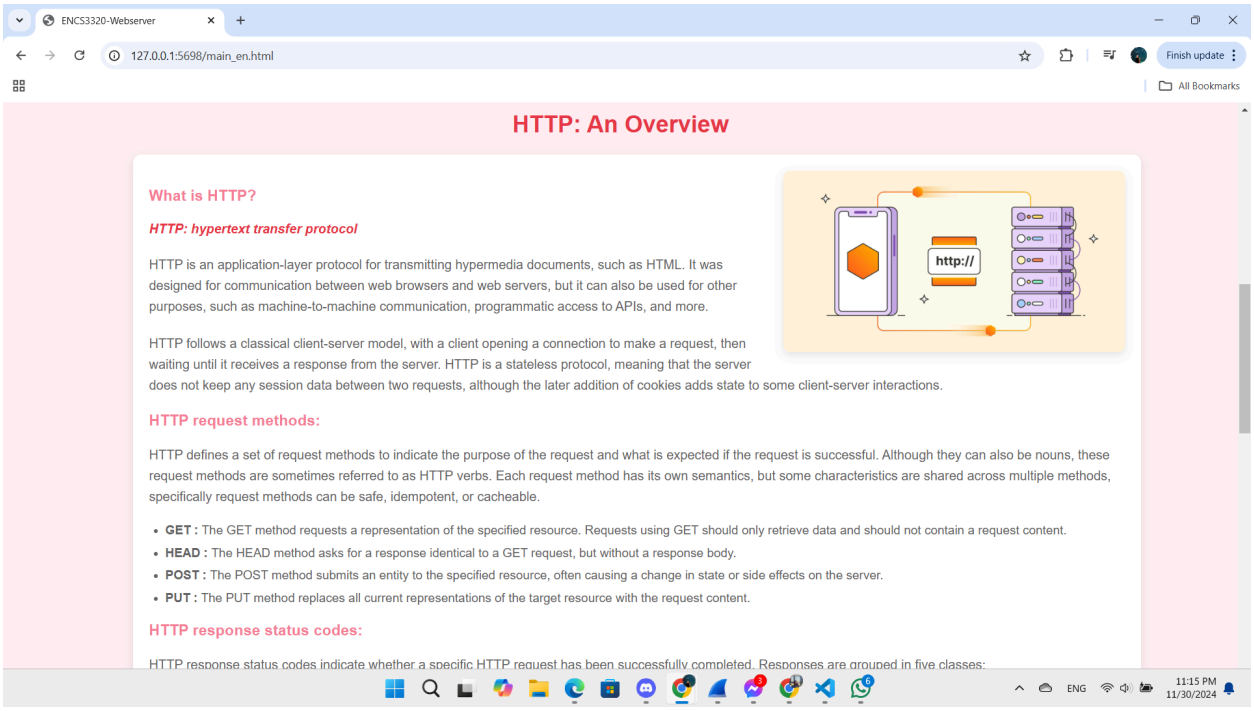


figure 28:Web Server Interface - Task 2

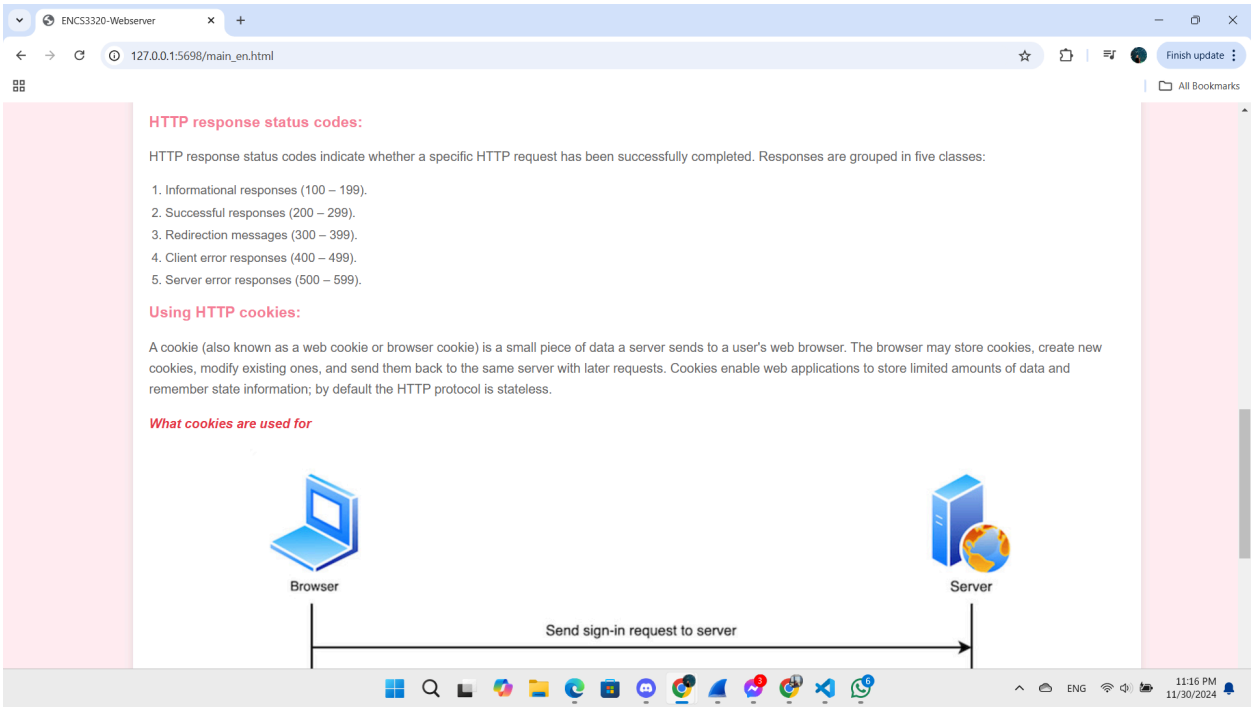


figure 29:Web Server Interface - Task 2

3. **Links to Resources:** A link to a supporting local HTML file (supporting_material_en.html) as well as functional links to external resources (such as the Ritaj portal and the textbook website).

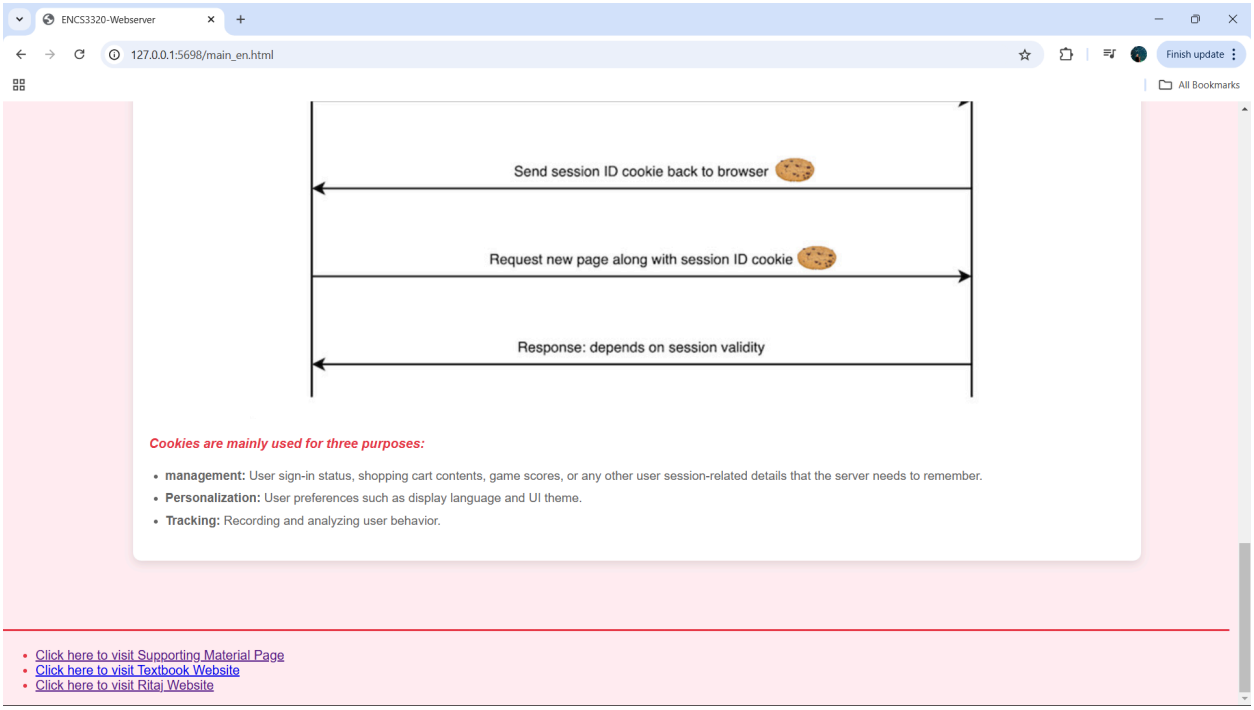


figure 30:Web Server Interface - Task 2

The server is set up to process several types of HTTP requests and, depending on the URL, reply with the relevant content. The implemented situations and the accompanying server behavior are shown below, along with images to support them:

1. **http://localhost:5698/**

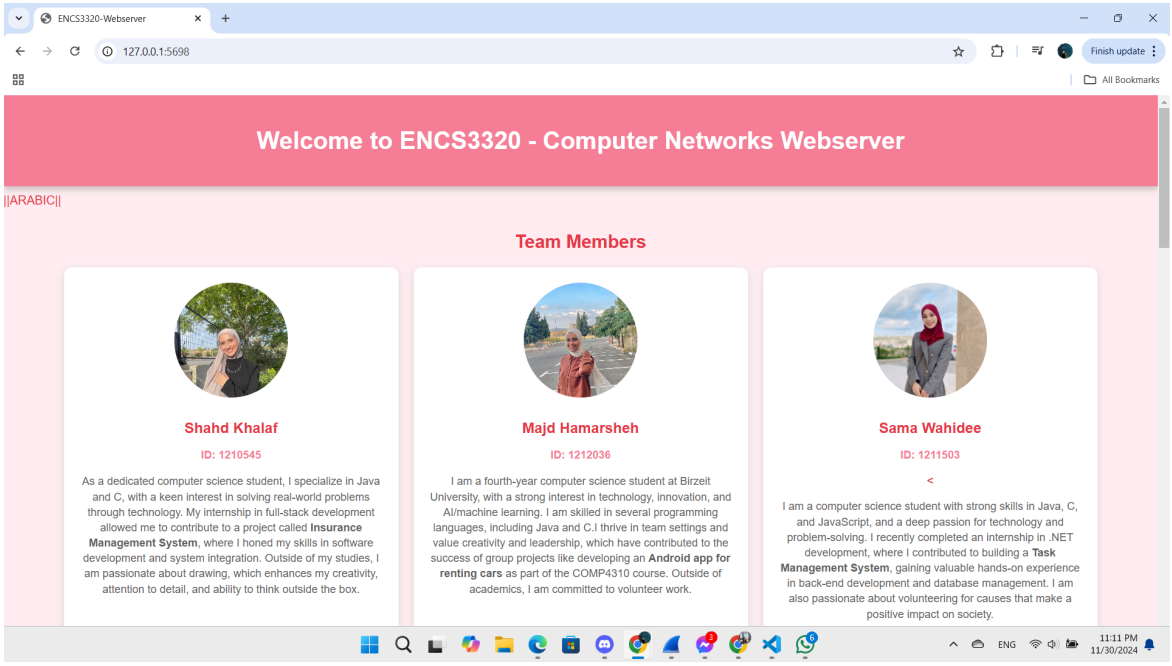


figure 31:Web Server Interface - Task 2

2. **http://localhost:5698/en**

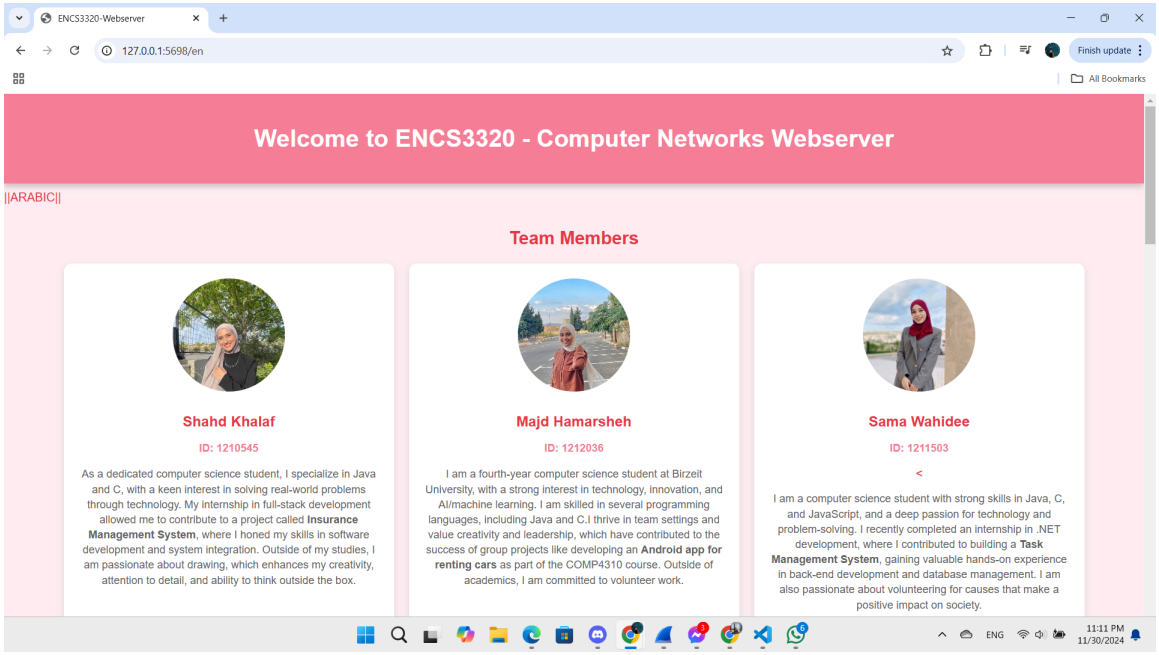


figure 32:Web Server Interface - Task 2

3. **http://localhost:5698/index.html**

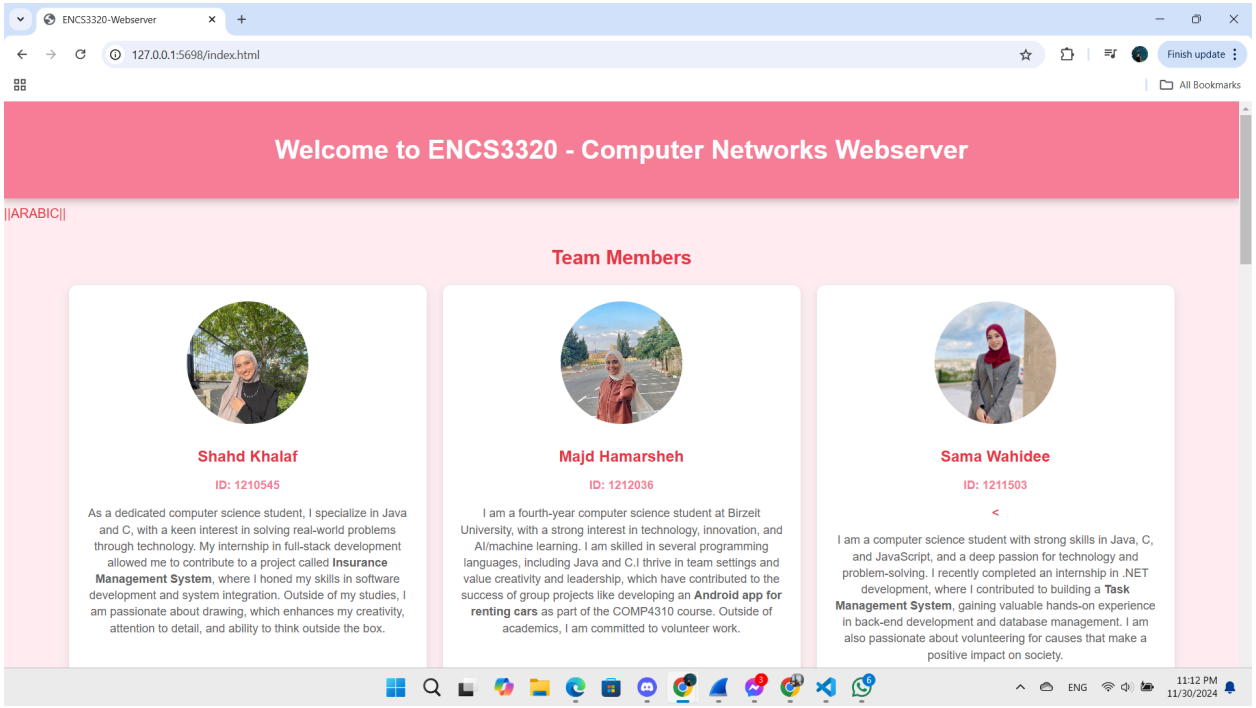


figure 33:Web Server Interface - Task 2

- **Web Server Code:**

  By reading the requested file, identifying its content type (such as HTML or PNG), and creating a suitable HTTP response, the generate_response function dynamically responds to HTTP requests. By producing a 404 Not Found page and adding client IP and port details to the error message, it handles errors. To guarantee reliable handling, it creates a stylized fallback error page for any missing files or the error.html.

```python
import os
import socket
from urllib.parse import unquote, parse_qs

# Configuration
HOST = '127.0.0.1'
PORT = 5698
BASE_DIR = './html'  # Directory for HTML and other static files
IMGS_DIR = './imgs'  # Directory for images

# Function to generate the HTTP response
def generate_response(file_path, content_type, status="200 OK", client_info=None):
    try:
        # Resolve the file path
        full_path = os.path.join(os.getcwd(), file_path)

        # If it's an image or video, check the file exists
        if content_type != 'text/html' and not os.path.exists(full_path):
            status = "404 Not Found"
            return generate_response("html/error.html", "text/html", status=status, client_info=client_info)

        # Open the file in binary mode if it's not an HTML file
        if content_type != 'text/html':
            with open(full_path, 'rb') as file:  # Open in binary mode for image or other non-text files
                content = file.read()
        else:
            with open(full_path, 'r', encoding='utf-8') as file:  # Open in text mode for HTML files
                content = file.read()

        # For 404 errors, inject client information if provided
        if status == "404 Not Found" and client_info:
            content = content.replace("{{client_ip}}", client_info[0])
            content = content.replace("{{client_port}}", str(client_info[1]))

        # Build the HTTP response
```

figure 34: Web Server - Task 2

```python
        response = f"HTTP/1.1 {status}\r\n"
        response += f"Content-Type: {content_type}\r\n\r\n"

        # Return binary response for images or other binary files
        if content_type != 'text/html':
            return response.encode('utf-8') + content

        # Return text response for HTML files
        return response.encode('utf-8') + content.encode('utf-8')

    except FileNotFoundError:
        # Fallback error response if `error.html` is missing
        fallback_message = f"""
        <html>
        <head><title>Error 404</title></head>
        <body style="background-color:#ffeef2; text-align:center; padding:50px;">
        <h1 style="color:#FF0000;">The file is not found</h1>
        <p style="color:#f77f98;">Client IP: {client_info[0]}</p>
        <p style="color:#f77f98;">Port: {client_info[1]}</p>
        </body>
        </html>
        """
        response = f"HTTP/1.1 404 Not Found\r\nContent-Type: text/html\r\n\r\n{fallback_message}"
        return response.encode('utf-8')
```

figure 35: Web Server - Task 2

The get_content_type function maps well-known file extensions, such as.html,.css, and.png, to their corresponding MIME types in order to determine the HTTP Content-Type depending on the file's extension. To securely handle binary files, it falls back to application/octet-stream for unsupported or unknown extensions.

```python
# Function to get the content type based on file extension
def get_content_type(file_path):
    # Determine the content type based on the file extension
    extension = os.path.splitext(file_path)[1].lower()
    return {
        '.html': 'text/html',
        '.css': 'text/css',
        '.png': 'image/png',
        '.jpg': 'image/jpeg',
        '.jpeg': 'image/jpeg',
        '.gif': 'image/gif',
        '.mp4': 'video/mp4',
        '.avi': 'video/avi',
        '.mov': 'video/quicktime',
    }.get(extension, 'application/octet-stream')  # Default to binary for unknown types
```

figure 36: Web Server - Task 2

Incoming HTTP requests are parsed by the handle_request function to identify the proper answer. It delivers static files (like CSS and images), predefined pages (like main_en.html and main_ar.html), and dynamically looks for files in the imgs/ directory. Malformed requests yield a 400 Bad Request response, whereas missing files send a 404 Not Found page.

```python
# Function to handle requests
def handle_request(request, client_address):
    # Parse the request URL
    lines = request.split('\n')
    if len(lines) > 0:
        request_line = lines[0]
        parts = request_line.split(' ')
        if len(parts) > 1:
            path = parts[1].lstrip('/')  # Remove leading '/' for paths
            print(f"Requested path: {path}")

            # Serve main English page
            if path in ['', 'en', 'index.html', 'main_en.html']:
                return generate_response("html/main_en.html", "text/html")

            # Serve supporting material page (English)
            elif path == 'supporting_material_en.html':
                return generate_response("html/supporting_material_en.html", "text/html")

            # Serve supporting material page (Arabic)
            elif path == 'supporting_material_ar.html':
                return generate_response("html/supporting_material_ar.html", "text/html")

            # Serve main Arabic page
            elif path in ['ar', 'main_ar.html']:
                return generate_response("html/main_ar.html", "text/html")

            # Check if the file exists in the imgs directory for images
            elif path.startswith('imgs/'):
                file_name = path.split('/')[1]  # Extract the image name from the path
                full_file_path = os.path.join(IMGS_DIR, file_name)

                if os.path.exists(full_file_path):
                    content_type = get_content_type(path)
                    return generate_response(path, content_type)
                else:
                    # If the file does not exist, return a 404
                    return generate_response("html/error.html", "text/html", status="404 Not Found", client_info=client_address)
```

figure 37: Web Server - Task 2

```
            # Serve static files (CSS, images)
            elif path.startswith('css/') or path.startswith('imgs/'):
                content_type = get_content_type(path)
                return generate_response(path, content_type)

            # If the file is not found, redirect to error.html
            else:
                return generate_response("html/error.html", "text/html", status="404 Not Found", client_info=client_address)

    # Default case for malformed requests
    return "HTTP/1.1 400 Bad Request\n\nInvalid request"
```

figure 38: Web Server - Task 2

In addition to handling requests and listening for new client connections, the start_server function initializes the server. It uses the handle_request function to process HTTP requests, accepts client connections, and makes sure the required directories are present. Clients receive responses in either text or binary format.

```
# Function to start the server
def start_server():
    # Ensure required directories exist
    os.makedirs(BASE_DIR, exist_ok=True)
    os.makedirs(IMGS_DIR, exist_ok=True)

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(("", PORT))
    server_socket.listen(100)
    print(f"Server running at http://{HOST}:{PORT}/")

    while True:
        client_socket, addr = server_socket.accept()  # Accept a new client connection
        print(f"Connection received from {addr}")

        # Receive the request from the client
        request = client_socket.recv(1024).decode('utf-8')
        print("HTTP Request:")
        print(request)  # Log the request details

        # Handle the request and generate a response
        response = handle_request(request, addr)

        # Send the response back to the client
        # Check if the response is bytes; send directly if it is
        if isinstance(response, bytes):
            client_socket.sendall(response)
        else:  # Otherwise, encode it as utf-8 and send
            client_socket.sendall(response.encode('utf-8'))

        client_socket.close()

# Start the server
if __name__ == '__main__':
    start_server()
```

figure 39: Web Server - Task 2

When a client requests a file that is not on the server—for example, because of an incorrect URL or a missing resource—the 404 error page is displayed. When the server is unable to locate the requested file, it returns a "404 Not Found" status, displaying the error message and, for debugging purposes, the client's IP address and port.

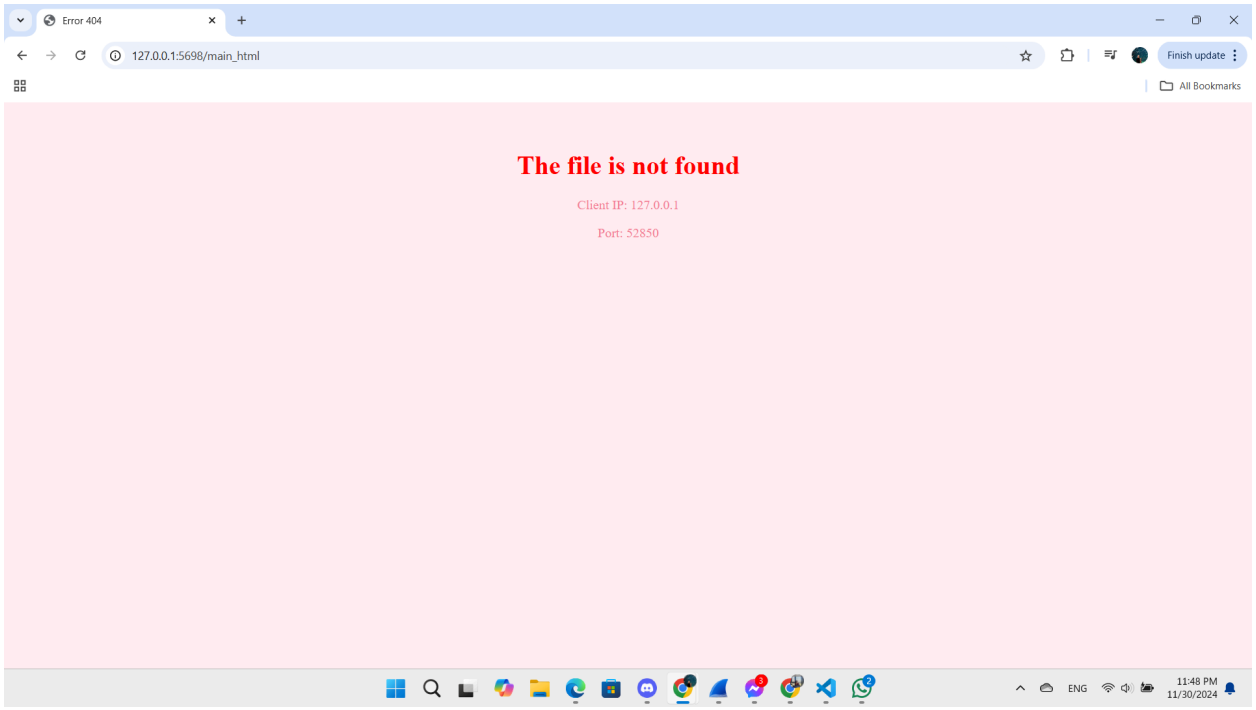


figure 40: Web Server Error - Task 2

Section B - Supporting Material Page:

- **Code:**
  - **supporting_material_en.html:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Supporting Materials</title>
    <link rel="stylesheet" href="/css/style_en.css">
</head>
<body>
<a class="top-right-link" href="supporting_material_ar.html"> || ARABIC || </a>

<div class="left-half">
    <!-- Form to input file name -->
    <div class="image-form">
        <form id="fileForm">
            <h2>Request an Image or Video</h2>
            <input type="text" id="fileName" class="image-input" placeholder="Enter the file name (e.g., http_protocol)">
            <button type="submit" class="submit-btn">Submit</button>
        </form>
    </div>

    <!-- Container to display the image or video -->
    <div class="file-container" id="fileContainer"></div>
</div>

<div class="right-half">
    <div class="center-text">
        <button class="button" onclick="window.location.href='main_en.html'">Go Back To The Main Page</button>
    </div>
</div>

<script>
    function handleFileRequest(event) {
        event.preventDefault(); // Prevent default form submission
        const fileName = document.getElementById('fileName').value.trim();
```

figure 41:supporting_material_en.html - Task 21

```
        if (fileName) {
            const serverUrl = `http://127.0.0.1:5698/imgs/${encodeURIComponent(fileName)}`;  // Adjusted URL format

            // Check if the file exists using fetch
            fetch(serverUrl, { method: 'GET' })
                .then(response => {
                    const fileContainer = document.getElementById('fileContainer');
                    const fileExtension = fileName.split('.').pop().toLowerCase();

                    if (response.status === 200) {
                        // Serve the image or video
                        if (['png', 'jpg', 'jpeg', 'gif'].includes(fileExtension)) {
                            fileContainer.innerHTML = `<img src="${serverUrl}" alt="${fileName}">`;
                        } else if (['mp4', 'avi', 'mov'].includes(fileExtension)) {
                            fileContainer.innerHTML = `<video controls><source src="${serverUrl}" type="video/${fileExtension}">Your browser does no
                        } else {
                            fileContainer.innerHTML = '<p style="color: red;">Unsupported file type.</p>';
                        }
                    } else if (response.status === 404) {
                        // Redirect if the file is not found
                        const searchUrl = fileName.endsWith('.mp4') || fileName.endsWith('.avi')
                            ? `https://www.youtube.com/results?search_query=${encodeURIComponent(fileName)}`
                            : `https://www.google.com/search?q=${encodeURIComponent(fileName)}&tbm=isch`;

                        window.location.href = searchUrl;
                    }
                })
                .catch(error => console.error('Error:', error));
        } else {
            alert('Please enter a valid file name.');
```

figure 42:supporting_material_en.html - Task 2

```
        }
    }

    document.getElementById('fileForm').addEventListener('submit', handleFileRequest);
</script>

</body>
</html>
```

figure 43:supporting_material_en.html - Task 2

- **style_en.css**

```
body {
    font-family: 'Poppins', sans-serif;
    background-color: #ffeef2; /* Soft pink background */
    color: #4a4a4a;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
.top-right-link {
    position: absolute;
    top: 20px;
    right: 20px;
    color: #f77f98;
    text-decoration: none;
    font-size: 25px;
}
/* Layout styling */
.left-half, .right-half {
    width: 50%;
    padding: 20px;
    box-sizing: border-box;
}

.left-half {
    background-color: #fff; /* White background for form */
    border-radius: 20px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    margin-right: 10px;
}

.right-half {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

figure 44:style_en.css - Task 2

```css
/* Form styling */
.image-form {
    text-align: center;
    margin-bottom: 20px;
}

.image-form h2 {
    font-size: 1.5rem;
    color: #e63946; /* Cute pink heading */
}

.image-input {
    width: 80%;
    padding: 10px;
    border: 2px solid #f8b4c8;
    border-radius: 10px;
    margin-bottom: 10px;
    outline: none;
    font-size: 1rem;
}

.image-input:focus {
    border-color: #e63946;
    box-shadow: 0 0 10px #f8b4c8;
}

.submit-btn {
    padding: 10px 20px;
    background-color: #e63946; /* Vibrant pink */
    color: white;
    border: none;
    border-radius: 15px;
    font-size: 1rem;
    cursor: pointer;
    transition: 0.3s;
}

.submit-btn:hover {
```

figure 45:style_en.css - Task 2

```css
    background-color: #f77f98; /* Softer pink */
    transform: scale(1.1);
}

/* Image/Video container styling */
.file-container {
    text-align: center;
    margin-top: 20px;
}

.file-container img {
    max-width: 100%;
    border-radius: 15px;
    margin-top: 20px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

/* Button styling */
.center-text .button {
    padding: 15px 25px;
    background-color: #f77f98; /* Light pink */
    color: white;
    border: none;
    border-radius: 20px;
    font-size: 1.2rem;
    font-weight: bold;
    cursor: pointer;
    text-align: center;
    transition: 0.3s;
}

.center-text .button:hover {
    background-color: #e63946;
    transform: scale(1.1);
}
```

figure 46: style_en.css - Task 2

- **Web Server Interface:**

Users can request a picture or video on computer networking topics covered in the Main English Webpage (main_en.html) via this page. In an input box, users type the name of the desired file (video or image), and the server offers the necessary redirection based on the request.

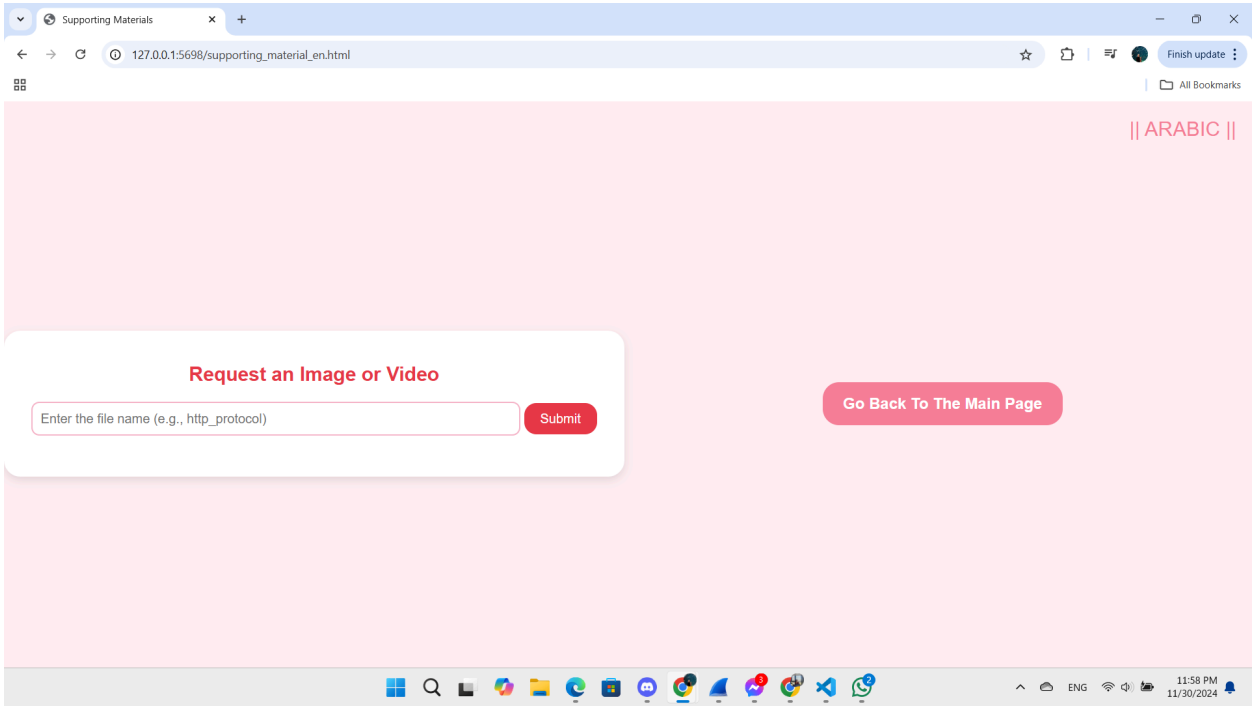


figure 47: Web Server Interface - Task 2

1. The user request an image that exist on the server:

   The server will serve the image directly to the client

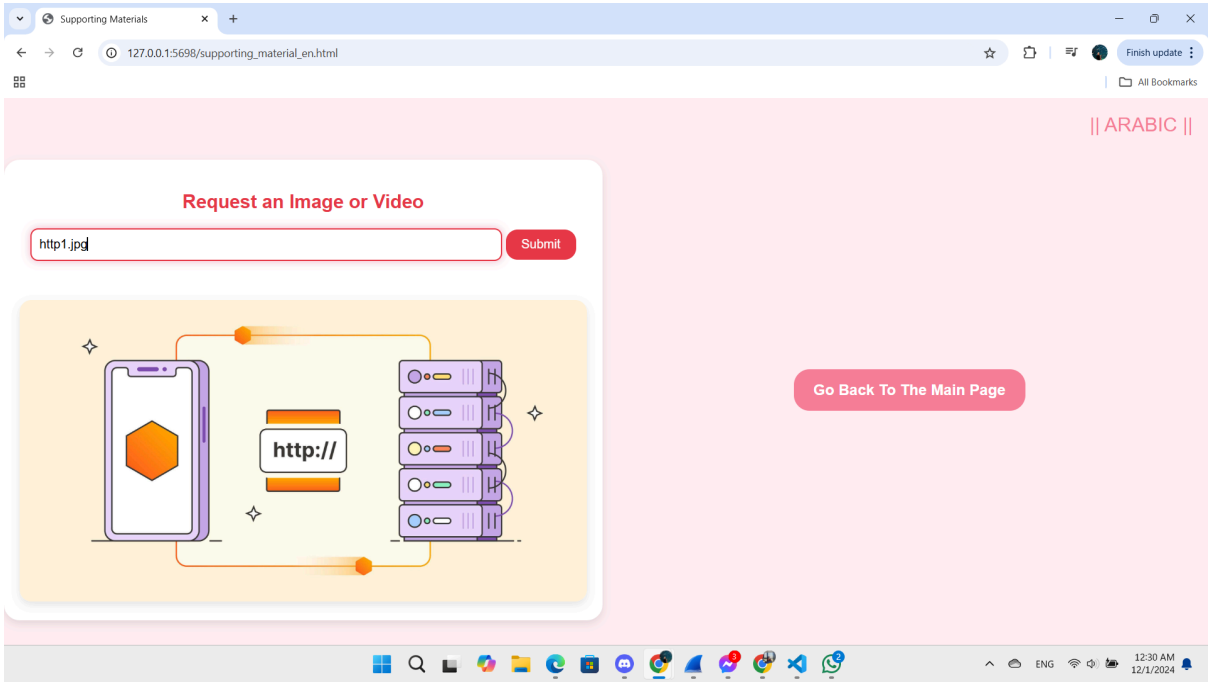


figure 48: Web Server Interface - Task 2

- **Data from the HTTP Request:**
  A. **Requested Path:** imgs/http1.jpg (client requests an image).
  B. **Client Connection:** 127.0.0.1:53320 (local machine, port 53320).
  C. **HTTP Request**: GET /imgs/http1.jpg HTTP/1.1 (client requests the image).
  D. **Accept:** Accepting image formats: image/avif, image/webp, image/png.
  E. **Referrer:** http://127.0.0.1:5698/supporting_material_en.html (page requesting the image).

```
Requested path: imgs/http1.jpg
Connection received from ('127.0.0.1', 53320)
HTTP Request:
GET /imgs/http.jpg HTTP/1.1
Host: 127.0.0.1:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:5698/supporting_material_en.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8
```

figure 49: HTTP Request - Task 2

2. The user request an image that does not exist on the server:

   The server will redirect to a Google search URL that filters results to images based on the user's input

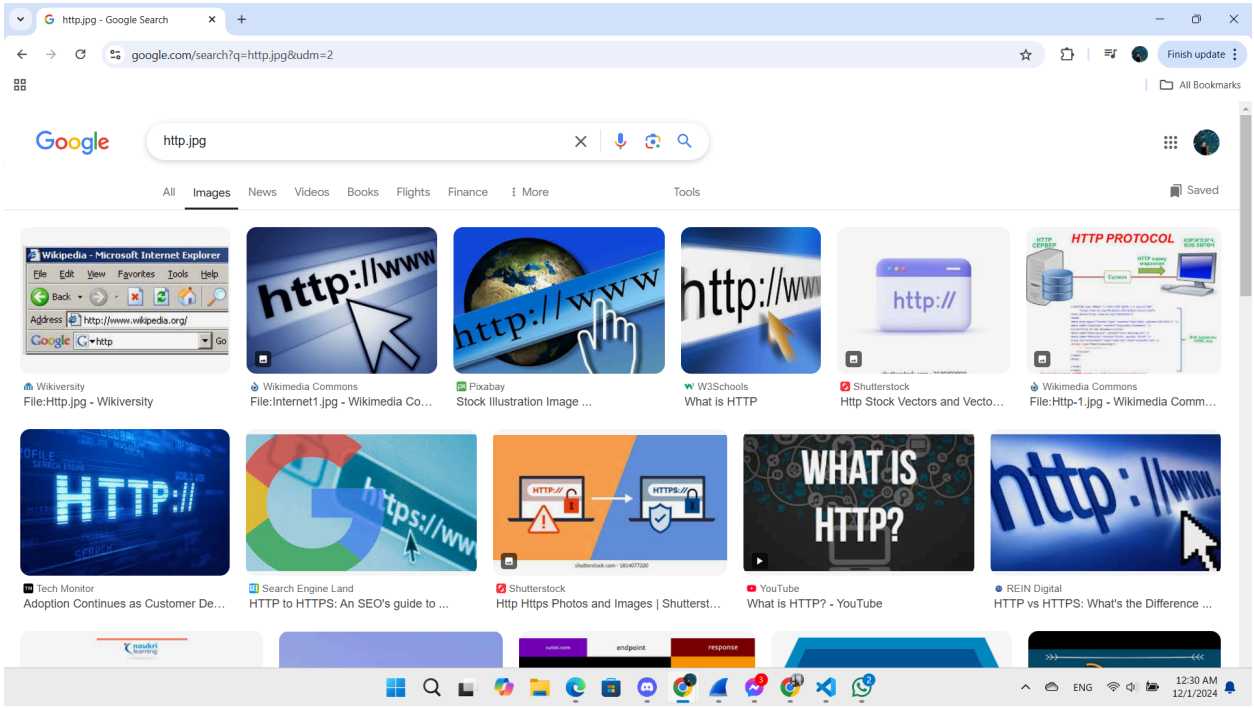


figure 50: HTTP Response - Task 2

- **Data from the HTTP Request:**
    A. **Requested Path:** imgs/http.jpg (client requests an image).
    B. **Client Connection:** 127.0.0.1:53766 (local machine, port 53766).
    C. **HTTP Request**: GET /imgs/http.jpg HTTP/1.1 (client requests the image).
    D. **Accept:** Accepting all types of content (*/*).
    E. **Referrer:** http://127.0.0.1:5698/supporting_material_en.html (page initiating the request).

```
Requested path: imgs/Task3.mp4
Connection received from ('127.0.0.1', 53766)
HTTP Request:
GET /favicon.ico HTTP/1.1
Host: 127.0.0.1:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
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://127.0.0.1:5698/supporting_material_en.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8
```

figure 51: HTTP Request - Task 2

3. The user request a video that does not exist on the server:

    The server will redirect to a YouTube search URL that filters results to videos based on the user's input

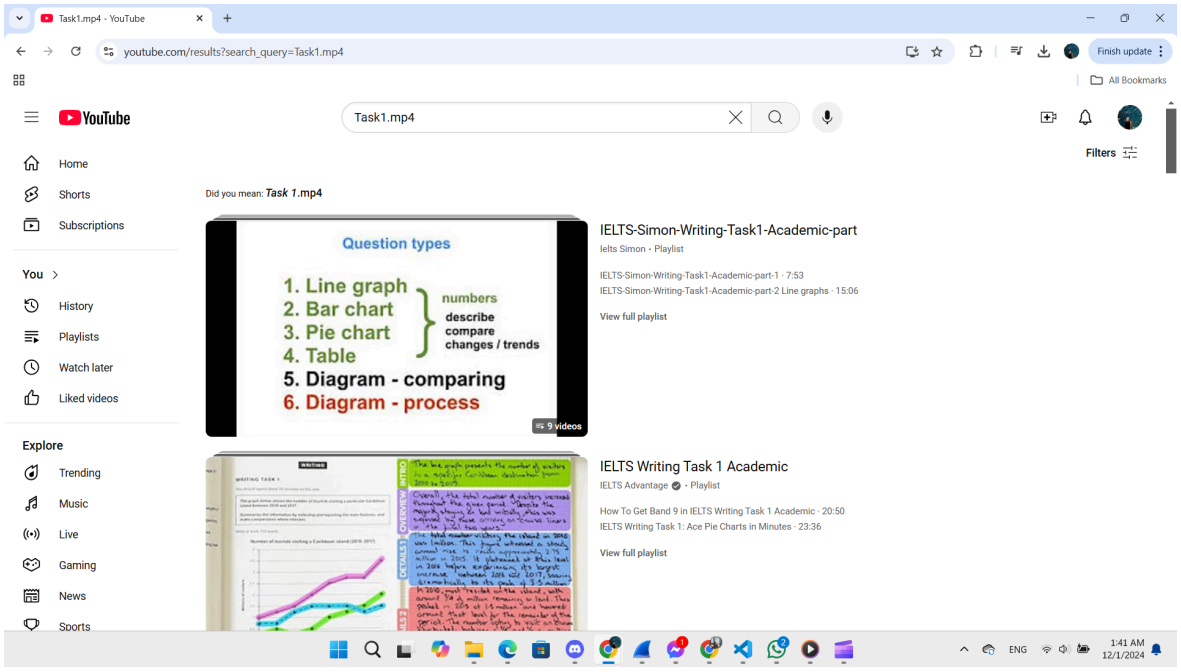


figure 52: HTTP Response - Task 2

- **Data from the HTTP Request:**
    A. **Requested Path:** imgs/Task3.mp4 (client requests a video).
    B. **Client Connection**: 127.0.0.1:55818 (local machine, port 55818).
    C. **HTTP Request:** GET /imgs/Task3.mp4 HTTP/1.1 (client requests the video file).
    D. **Accept:** Accepting all types of content (*/*).
    E. **Referrer:** http://127.0.0.1:5698/supporting_material_en.html (page initiating the request).

```
Requested path: favicon.ico
Connection received from ('127.0.0.1', 55818)
HTTP Request:
GET /imgs/Task1.mp4 HTTP/1.1
Host: 127.0.0.1:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:5698/supporting_material_en.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8


Requested path: imgs/Task1.mp4
```

figure 53: HTTP Request - Task 23

4. The user request a video that exist on the server:

    The server will serve the video directly to the client

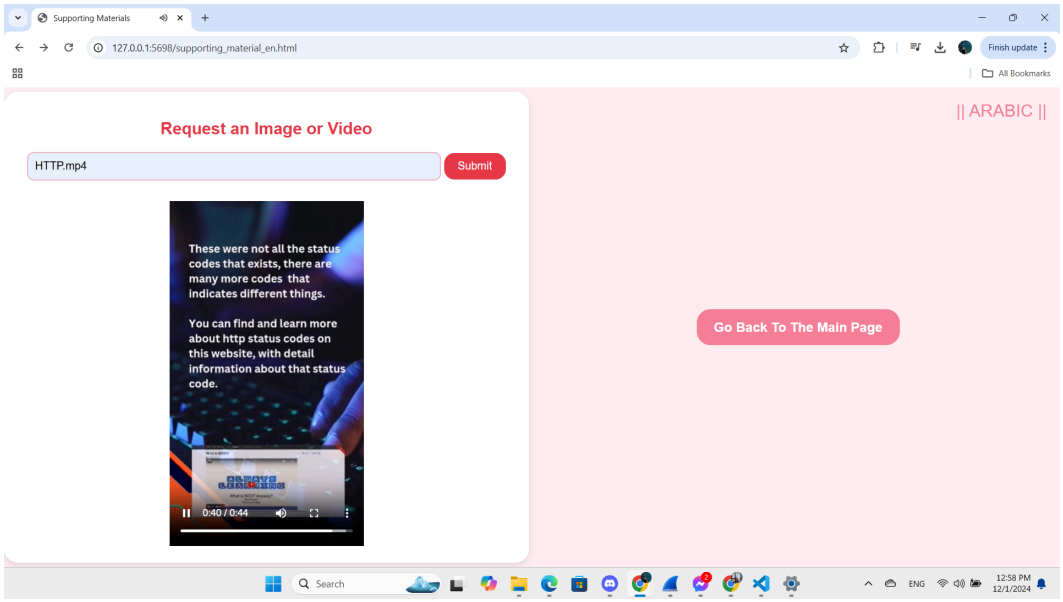


figure 54: HTTP Response - Task 2

- **Data from the HTTP Request:**
    A. **Requested Path:** imgs/http.mp4 (client requests a video file).
    B. **Client Connection:** 127.0.0.1:55788 (local machine, port 55788).
    C. **HTTP Request:** GET /imgs/http.mp4 HTTP/1.1 (client requests the video file).
    D. **Accept:** Accepting all types of content (*/*).
    E. **Referrer:** http://127.0.0.1:5698/supporting_material_en.html (page initiating the request)

figure 55: HTTP Request - Task 2

Section C - Arabic Versions:

The Main English Webpage and Supporting Material Page were translated into Arabic to accommodate both Arabic and English readers. Main_ar.html and supporting_material_ar.html are the translated versions of the English pages main_en.html and supporting_material_en.html.

- **Supporting Materia Arabic version::**
  - **Code:**
    - **supporting_material_ar.html:**

```html
<!DOCTYPE html>
<html lang="ar">
<head>
    <meta charset="utf-8">
    <title>مواد داعمة</title>
    <link rel="stylesheet" href="/css/style_ar.css">
</head>
<body>

    <a class="top-right-link" href="supporting_material_en.html"> || English || </a>

    <div class="left-half">
        <!-- Form to input file name -->
        <div class="image-form">
            <form id="fileForm">
                <h2>طلب صورة أو فيديو</h2>
                <input type="text" id="fileName" class="image-input" placeholder="أدخل اسم الملف (مثل: http_protocol)">
                <button type="submit" class="submit-btn">إرسال</button>
            </form>
        </div>

        <!-- Container to display the image or video -->
        <div class="file-container" id="fileContainer"></div>
    </div>

    <div class="right-half">
        <div class="center-text">
            <button class="button" onclick="window.location.href='main_ar.html'">العودة إلى الصفحة الرئيسية</button>
        </div>
    </div>

    <script>
        function handleFileRequest(event) {
            event.preventDefault(); // Prevent default form submission
            const fileName = document.getElementById('fileName').value.trim();

            if (fileName) {
                const serverUrl = `http://127.0.0.1:5698/imgs/${encodeURIComponent(fileName)}`;  // Correct URL format

```

figure 56: supporting_material_ar.html - Task 2

```
                // Check if the file exists using fetch
                fetch(serverUrl, { method: 'GET' })
                    .then(response => {
                        const fileContainer = document.getElementById('fileContainer');
                        const fileExtension = fileName.split('.').pop().toLowerCase();

                        if (response.status === 200) {
                            // Serve the image or video
                            if (['png', 'jpg', 'jpeg', 'gif'].includes(fileExtension)) {
                                fileContainer.innerHTML = `<img src="${serverUrl}" alt="${fileName}">`;
                            } else if (['mp4', 'avi', 'mov'].includes(fileExtension)) {
                                fileContainer.innerHTML = `<video controls><source src="${serverUrl}" type="video/${fileExtension}">Your browser doe
                            } else {
                                fileContainer.innerHTML = '<p style="color: red;">نوع الملف غير مدعوم.</p>';
                            }
                        } else if (response.status === 404) {
                            // Redirect if the file is not found
                            const searchUrl = fileName.endsWith('.mp4') || fileName.endsWith('.avi')
                                ? `https://www.youtube.com/results?search_query=${encodeURIComponent(fileName)}`
                                : `https://www.google.com/search?q=${encodeURIComponent(fileName)}&tbm=isch`;

                            window.location.href = searchUrl;
                        }
                    })
                    .catch(error => console.error('خطأ:', error));
            } else {
                alert('من فضلك أدخل اسم الملف بشكل صحيح.');
            }
        }

        document.getElementById('fileForm').addEventListener('submit', handleFileRequest);
    </script>

</body>
</html>
```

figure 57: supporting_material_ar.html - Task 2

- **style_ar.cssl:**

```
/* General Page Styling */
body {
    font-family: 'Tajawal', sans-serif; /* Arabic-friendly font */
    background-color: #ffeef2; /* Light pink background */
    color: #4a4a4a;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    direction: rtl; /* Right to Left text direction */
    text-align: right; /* Right-aligned text */
}

/* Top-right link */
.top-right-link {
    position: absolute;
    top: 20px;
    right: 20px;
    color: #f77f98;
    text-decoration: none;
    font-size: 25px;
}

/* Layout styling */
.left-half, .right-half {
    width: 50%;
    padding: 20px;
    box-sizing: border-box;
}

.left-half {
    background-color: #fff; /* White background for form */
    border-radius: 20px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    margin-left: 10px;
}

```

figure 58: style_ar.css - Task 2

```css
.right-half {
    display: flex;
    justify-content: center;
    align-items: center;
}

/* Form styling */
.image-form {
    text-align: center;
    margin-bottom: 20px;
}

.image-form h2 {
    font-size: 1.5rem;
    color: #e63946; /* Cute pink heading */
}

.image-input {
    width: 80%;
    padding: 10px;
    border: 2px solid #f8b4c8;
    border-radius: 10px;
    margin-bottom: 10px;
    outline: none;
    font-size: 1rem;
}

.image-input:focus {
    border-color: #e63946;
    box-shadow: 0 0 10px #f8b4c8;
}

.submit-btn {
    padding: 10px 20px;
    background-color: #e63946; /* Vibrant pink */
    color: white;
    border: none;
    border-radius: 15px;
    font-size: 1rem;
```

figure 59: style_ar.css - Task 2

```css
    cursor: pointer;
    transition: 0.3s;
}

.submit-btn:hover {
    background-color: #f77f98; /* Softer pink */
    transform: scale(1.1);
}

/* Image/Video container styling */
.file-container {
    text-align: center;
    margin-top: 20px;
}

.file-container img {
    max-width: 100%;
    border-radius: 15px;
    margin-top: 20px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

/* Button styling */
.center-text .button {
    padding: 15px 25px;
    background-color: #f77f98; /* Light pink */
    color: white;
    border: none;
    border-radius: 20px;
    font-size: 1.2rem;
    font-weight: bold;
    cursor: pointer;
    text-align: center;
    transition: 0.3s;
}
```

figure 60: style_ar.css - Task 2

- **Web Server Interface :**

The supporting materials are also available in Arabic on this website, with the format and content modified for Arabic speakers.

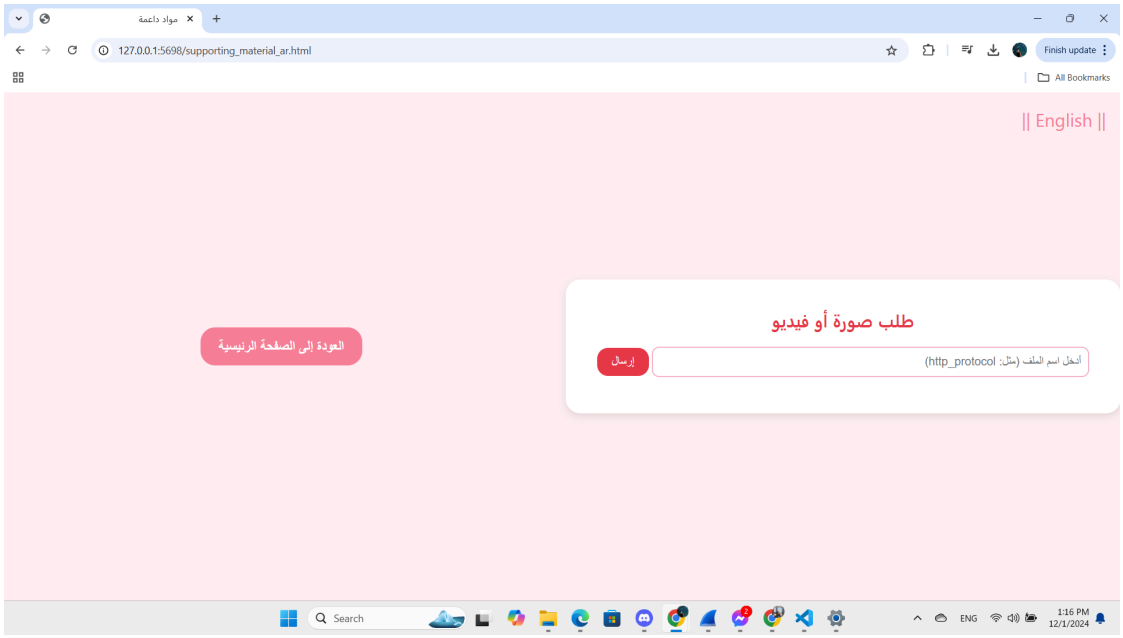


figure 61:Web Server Interface - Task 2

- **Data from the HTTP Request:**
  A. **Requested Path:** css/style_ar.css (client requests the Arabic CSS stylesheet).
  B. **Client Connection**: 127.0.0.1:55890 (local machine, port 55890).
  C. **HTTP Request**: GET /css/style_ar.css HTTP/1.1 (client requests the CSS file).
  D. **Accept**: Accepting text/css for CSS files and all other content (*/*).
  E. **Referrer:** http://127.0.0.1:5698/supporting_material_ar.html (the page requesting the CSS).

```
Requested path: supporting_material_ar.html
Connection received from ('127.0.0.1', 55980)
HTTP Request:
GET /css/style_ar.css HTTP/1.1
Host: 127.0.0.1:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: text/css,*/*;q=0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: style
Referer: http://127.0.0.1:5698/supporting_material_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8


Requested path: css/style_ar.css
```

figure 62: HTTP Request - Task 2

- Same as supporting_material_ar.html this designed to allow users to request images or videos by entering a search term in an input form. Depending on the requested file type, the server either serves the requested content (if available) or redirects the user to external search engines

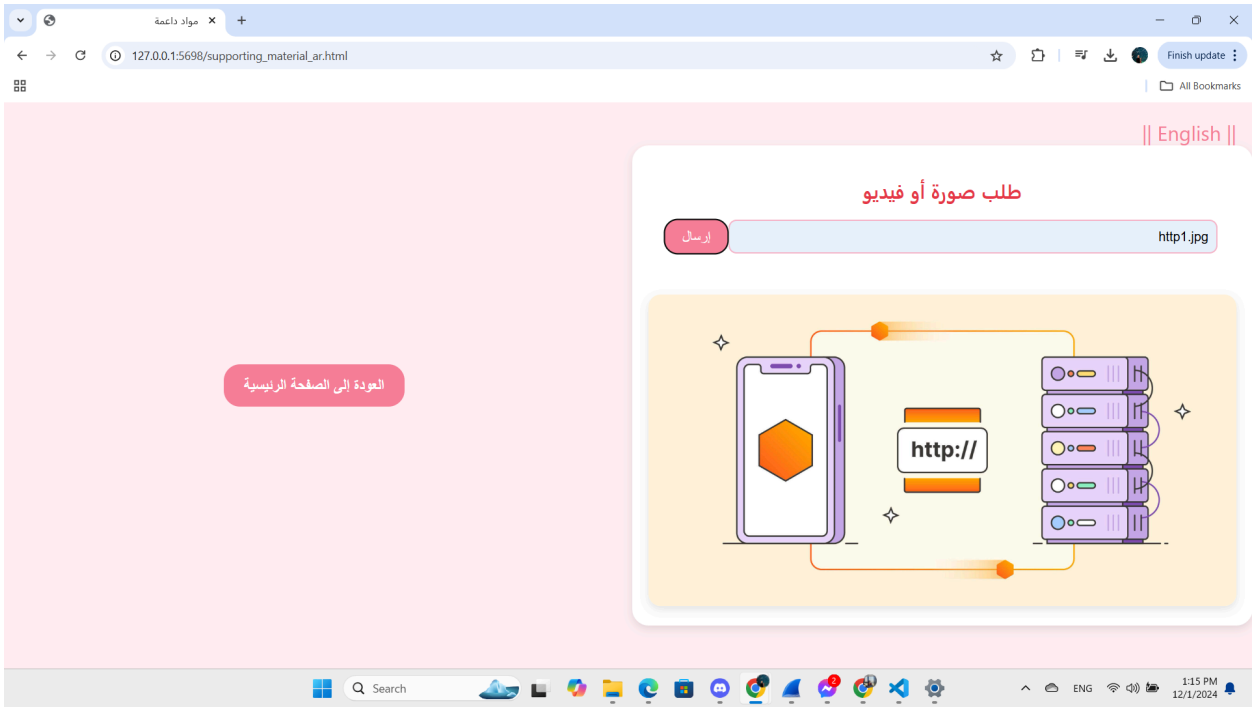


figure 63:Web Server Interface - Task 2

1. The user request an image that exist on the server:
   The server will serve the image directly to the client

- **Data from the HTTP Request:**
    A. **Requested Path:** imgs/http1.jpg (client requests an image).
    B. **Client Connection**: 127.0.0.1:55974 (local machine, port 55974).
    C. **HTTP Request**: GET /imgs/http1.jpg HTTP/1.1 (client requests the image).
    D. **Accept:** Accepting image formats like image/avif, image/webp, and image/png.
    E. **Referrer:** http://127.0.0.1:5698/supporting_material_ar.html (page requesting the image).

```
Requested path: imgs/http1.jpg
Connection received from ('127.0.0.1', 55974)
HTTP Request:
GET /imgs/http1.jpg HTTP/1.1
Host: 127.0.0.1:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
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://127.0.0.1:5698/supporting_material_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8


Requested path: imgs/http1.jpg
```

figure 64: HTTP Request - Task 2

2. The user request a video that exist on the server:
   The server will serve the video directly to the client

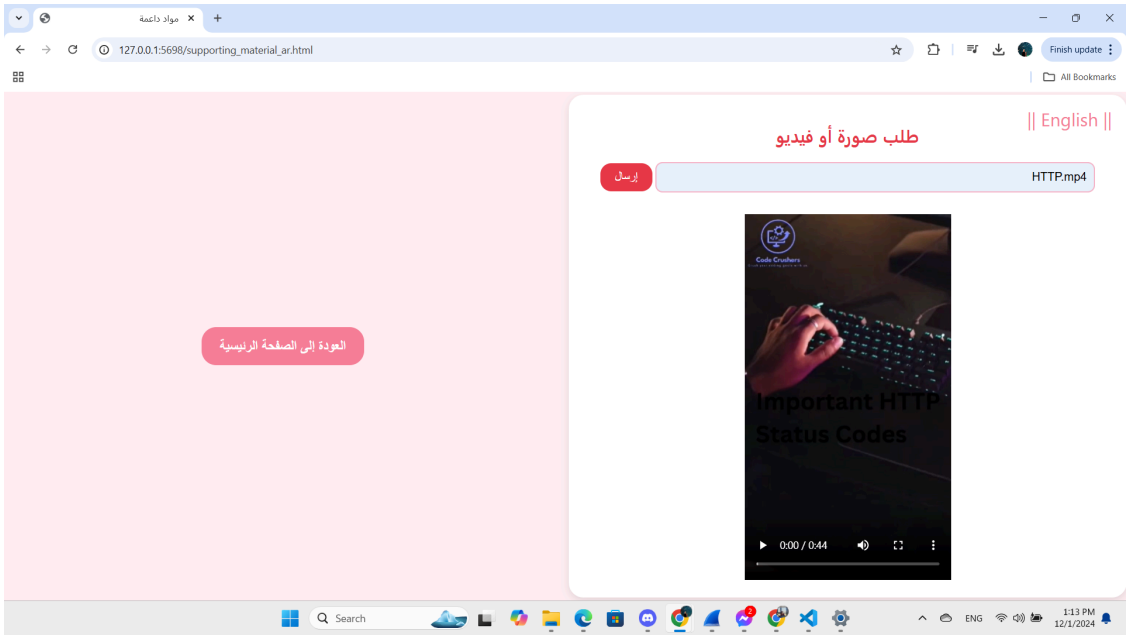


figure 65::Web Server Interface - Task 2

- **Data from the HTTP Request:**

  A. **Requested Path**: imgs/HTTP.mp4 (client requests a video file).

  B. **Client Connection:** 127.0.0.1:55951 (local machine, port 55951).

  C. **HTTP Request:** GET /imgs/HTTP.mp4 HTTP/1.1 (client requests the video file).

  D. **Accept**: Accepting all types of content (*/*).

  E. **Referrer**: http://127.0.0.1:5698/supporting_material_ar.html (the page requesting the video).

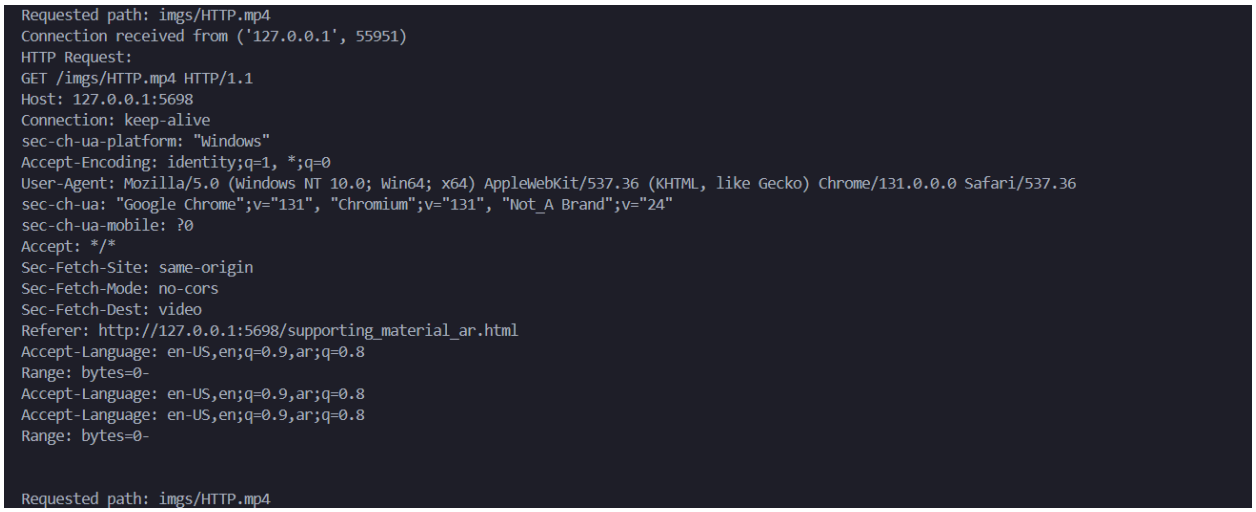


figure 66: HTTP Request - Task 2

3. The user request a video that does not exist on the server:

   The server will redirect to a YouTube search URL that filters results to videos based on the user's input

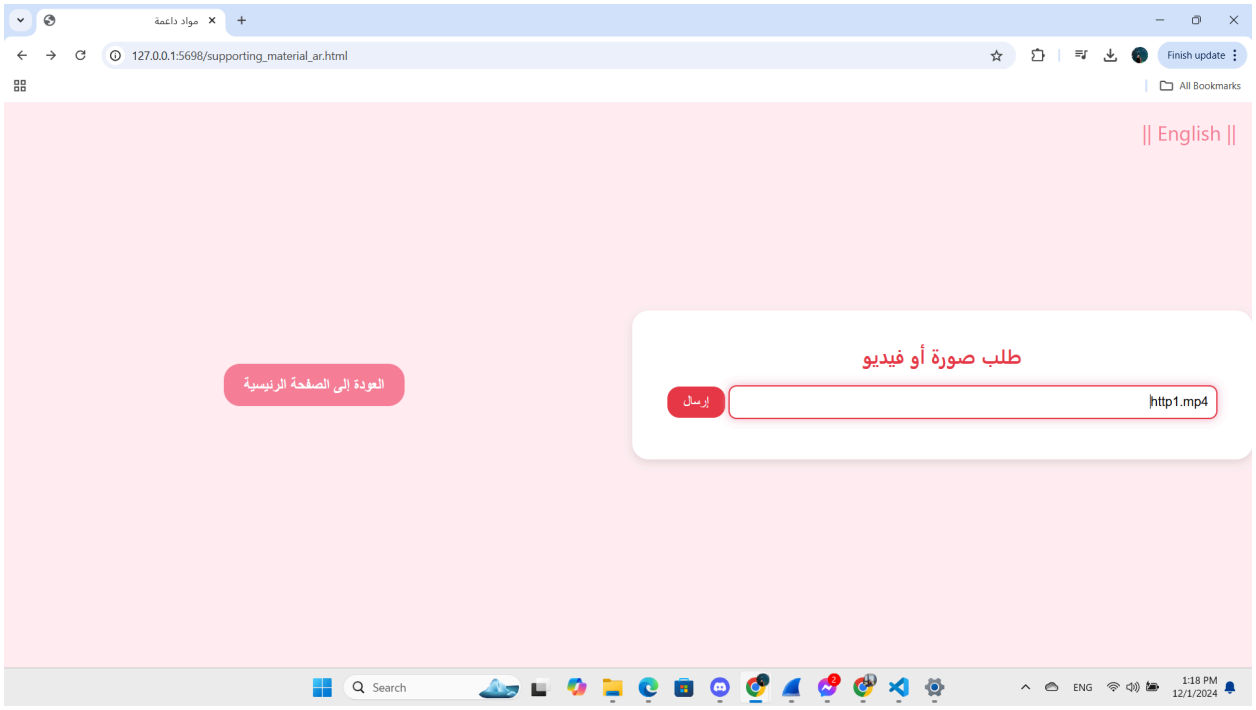


figure 67:Web Server Interface - Task 2

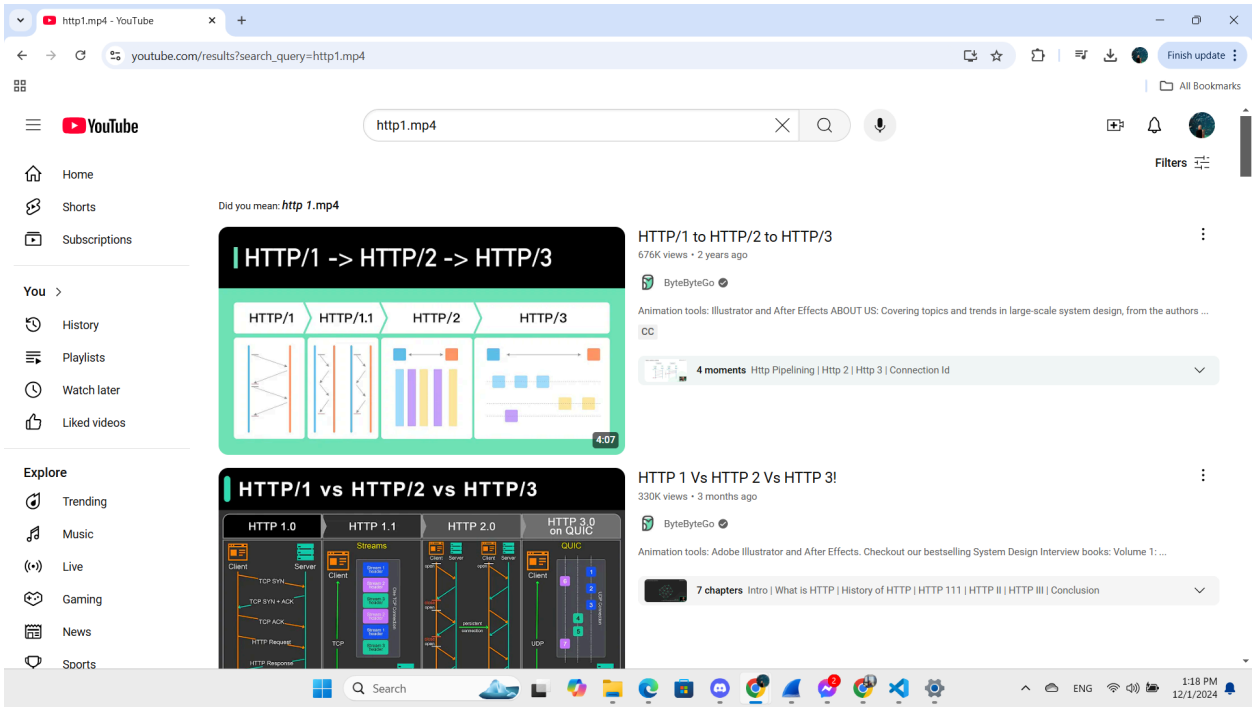


figure 68: HTTP Response - Task 2

- **Data from the HTTP Request:**
    A. **Requested Path:** imgs/http1.mp4 (client requests a video file).
    B. **Client Connection**: 127.0.0.1:56018 (local machine, port 56018).
    C. **HTTP Request:** GET /imgs/http1.mp4 HTTP/1.1 (client requests the video file).
    D. **Accept:** Accepting all types of content (*/*).
    E. **Referrer**: http://127.0.0.1:5698/supporting_material_ar.html (the page requesting the video).

```
Requested path: imgs/http1.mp4
Connection received from ('127.0.0.1', 56018)
HTTP Request:
GET /imgs/http1.jpg HTTP/1.1
Host: 127.0.0.1:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:5698/supporting_material_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8
```

figure 69: HTTP Request - Task 2

4. The user request an image that does not exist on the server:

   The server will redirect to a Google search URL that filters results to images based on the user's input

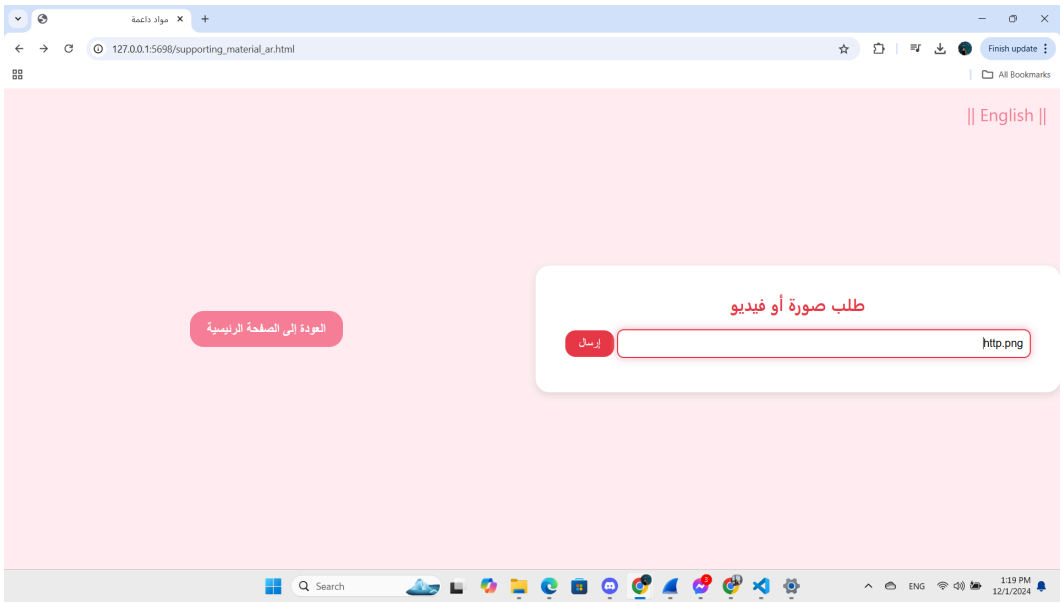


figure 70::Web Server Interface - Task 2

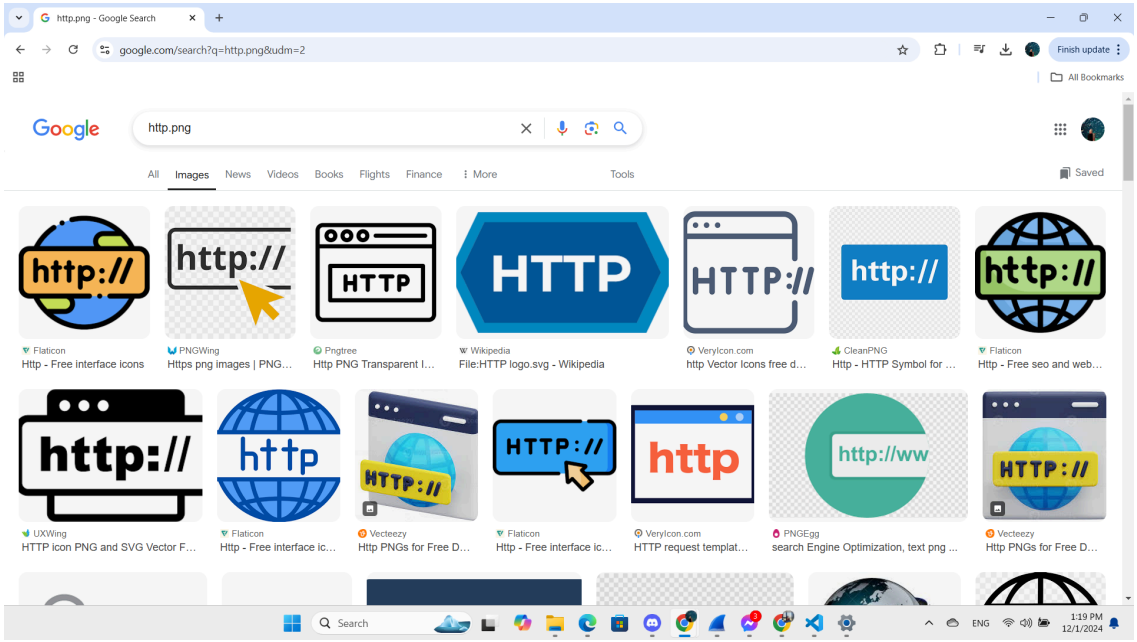


figure 71: HTTP Response - Task 2

- **Data from the HTTP Request:**

A. **Requested Path:** imgs/http.png (client requests an image file).

B. **Client Connection:** 127.0.0.1:56026 (local machine, port 56026).

C. **HTTP Request:** GET /imgs/http.png HTTP/1.1 (client requests the image).

D. **Accept:** Accepting all types of content (*/*).

E. **Referrer:** http://127.0.0.1:5698/supporting_material_ar.html.

```
Requested path: css/style_ar.css
Connection received from ('127.0.0.1', 56026)
HTTP Request:
GET /imgs/http.png HTTP/1.1
Host: 127.0.0.1:5698
Connection: keep-alive
sec-ch-ua-platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36
sec-ch-ua: "Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
sec-ch-ua-mobile: ?0
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:5698/supporting_material_ar.html
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9,ar;q=0.8

Requested path: imgs/http.png
```

figure 72: HTTP Request - Task 2

- **Main Page Arabic version:**
  - **Code**
    - **main_ar.html:**

```
<!DOCTYPE html>
<html lang="ar">
    <body dir="rtl">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ENCS3320-خادم الويب</title>
    <link rel="stylesheet" href="/css/styles_main_ar.css">
</head>
<body>
    <header>
    <h1>مرحباً بكم في ENCS3320 - خادم الويب لشبكات الحاسوب</h1>

    </header>
    <a class="english-link" href="/main_en.html">||ENGLISH||</a>

    <section class="team-section">
        <h2>أعضاء الفريق</h2>
        <div class="team-container">
            <div class="team-box">
                <img src="imgs/img1.png" alt="Shahd Khalaf">
                <h3>شهد خلف</h3>
                <p class="id">1210545 </p>
                <p>
                    هتمام كبير بحل مشكلات العالم الحقيقي من خلال التكنولوجيا .C و Java طالبة علم حاسوب في السنة الرابعة،متخصصة في لغات البرمجة
                    كامل الأنظمة. Insurance Management System المعاهمة في مشروع يسمى development لقد أتاح لي التدريب في مجال ال full-stack
                    خارج دراستي، أنا شغوفة بالرسم، مما يعزز إبداعي والاهتمام بالتفاصيل والقدرة على التفكير خارج الصندوق.
                </p>
            </div>
            <div class="team-box">
                <img src="imgs/img3.png" alt="Majd Hamarsheh">
                <h3>مجد حمارشة</h3>
                <p class="id">1212036</p>
                <p>
                    الية علم حاسوب في السنة الرابعة في جامعة بيرزيت، ولدي اهتمام كبير بالتكنولوجيا والابتكار والتعلم الآلي و الذكاء الاصطناعي
                    أقدر الإبداع والقيادة .C و Java أتقن العمل في العديد من لغات البرمجة، بما في ذلك
                    والتي تساهم في نجاح المشاريع الجماعية مثل تطوير تطبيق اندرويد لتأجير السيارات كجزء من فصاق COMP4310.
                    خارج المجال الأكاديمي، أنا مهتمة بالعمل التطوعي.
```

figure 73:main_ar.html - Task 2

```
                </p>
            </div>
            <div class="team-box">
                <img src="imgs/img2.png" alt="Sama Wahidee">
                <h3>سما وحيدي</h3>
                <p class="id">1211503 </p>د
                <p>
                    أنا طالبة علم حاسوب وأتمتع بمهارات قوية في Java و C وJavaScript. ولدي شغف عميق بالتكنولوجيا وحل المشكلات.
                    واكتسبت خبرة عملية قيمة في back-e ل Task Management System . حيث ساهمت في بناء مشروع .NET. لقد أكملت مؤخرًا تدريبًا في تطوير
                    أنا أيضًا مهتمة بالعمل التطوعي في القضايا التي لها تأثير إيجابي على المجتمع.
                </p>
            </div>
        </div>
    </section>
    <section class="concept-section">
        <h2 class="concept-title">نظرة عامة عن : HTTP</h2>
        <div class="concept-box">
            <img src="imgs/http1.jpg" alt="HTTP Overview" class="concept-image">
            <h3 class="highlight-title">ما هو ال HTTP ؟ </h3>
            <h4>
                HTTP:
                بروتوكول نقل النص التشعبي
            </h4>

            <p>
                HTTP هو بروتوكول طبقة تطبيق لنقل مستندات الوسائط التشعبية، مثل HTML. والمزيد ، والوصول البرمجي إلى واجهات برمجة التطبيقات،
            </p>
            <p>
                HTTP
                ت خادم العميل HTTP يتبع نموذج خادم العميل الكلاسيكي، حيث يفتح العميل اتصالاً لتقديم طلب، ثم ينتظر حتى يتلقى استجابة من الخادم
            </p>

            <h3>طرق طلب HTTP:</h3>
            <p>
```

figure 74: main_ar.html - Task 2

figure 75:main_ar.html - Task 2

figure 76:main_ar.html - Task 2

- **styles_main_ar.css:**

```css
/* General Page Styling */
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #ffeef2; /* Light pink background */
    color: #e63946; /* Neutral text color */
    direction: rtl;

}

/* Header Styling */
header {
    background-color: #f77f98; /* Light red for the header */
    color: white;
    text-align: center;
    padding: 20px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2); /* Subtle shadow for depth */
}

/* Team Section Styling */
.team-section {
    padding: 40px;
    text-align: center;
}

h4 {
    font-style: italic;
}

h4 {
    color: #e63946;
    font-style: italic;
}

.team-container {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
```

figure 77:styles_main_ar.css - Task 2

```css
    justify-content: center;
}

.team-box {
    background-color: white;
    border-radius: 10px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    padding: 20px;
    width: 400px;
    text-align: center;
    transition: transform 0.2s;
    display: inline-block;
    overflow: hidden;
}

.team-box:hover {
    transform: scale(1.05);
}

/* Image Styling */
.team-box img {
    width: 150px;
    height: 150px;
    object-fit: cover;
    border-radius: 50%;
    margin-bottom: 15px;
    max-width: 100%;
    max-height: 100%;
}

/* ID Styling */
.team-box .id {
    color: #f77f98; /* Blue color for the ID text */
    font-weight: bold;
}

/* Paragraph Styling */
.team-box p {
```

figure 78:styles_main_ar.css - Task 2

```
    font-size: 0.9em;
    line-height: 1.4;
    color: #666; /* Subtle text color for descriptions */
}

.team-box h3 {
    margin: 10px 0;
    font-size: 1.2em;
    color: #e63946; /* Matches the header for consistency */
}

/* Concept Section Styling */
.concept-section {
    padding: 40px;
    margin: 20px auto;
    text-align: center;
    max-width: 1200px; /* Increased max width to make the section wider */
}

/* Title outside the box */
.concept-title {
    color: #e63946;
    margin-bottom: 20px;
    font-size: 1.8em;
    text-align: center;
}

/* Box styling */
.concept-box {
    background-color: #fff; /* White background for contrast */
    border-radius: 10px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1); /* Subtle shadow for depth */
    padding: 20px;
    text-align: right;
    margin: 0 auto;
    max-width: 1200px; /* Increased max-width */
    width: 100%; /* Ensure the box takes up the full width within the max-width */
}
```

figure 79:styles_main_ar.css - Task 2

```

/* Image inside the box */
.concept-image {
    float: left; /* Align the image to the right */
    margin: 0 0 20px 20px; /* Add spacing around the image */
    max-width: 35%; /* Further reduced image width for even more space for text */
    height: auto;
    border-radius: 8px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1); /* Adds depth to the image */
}

/* Text inside the box */
.concept-box p {
    font-size: 1em;
    line-height: 1.6;
    color: #666;
    margin-bottom: 20px;
}

/* Subheadings inside the box */
.concept-box h3 {
    color: #f77f98; /* A complementary shade to maintain harmony */
    margin-top: 20px;
}

/* Lists inside the box */
.concept-box ul, .concept-box ol {
    margin: 20px 0;
    padding-right: 20px;
}

.concept-box ul li, .concept-box ol li {
    margin: 10px 0;
    font-size: 0.95em;
    color: #666; /* Darker for readability */
}

/* Highlight title styling */
.highlight-title {
```

figure 80:styles_main_ar.css - Task 2

```
155     color: #0073e6; /* Blue color for emphasis */
156     font-weight: bold;
157     margin-bottom: 10px;
158 }
159 .center-image {
160     display: block;
161     margin-left: auto;
162     margin-right: auto;
163     max-width: 100%;
164     height: auto;
165 }
166
167
168 /* Styling for Arabic link */
169 .english-link {
170     float: left;
171     margin-top: 10px;
172     font-size: 16px;
173     color: #e63946;
174     text-decoration: none;
175 }
176
177
178 .english-link:hover {
179     color: #f77f98;
180     text-decoration: underline;
181 }
182
```

figure 81:styles_main_ar.css - Task 2

- **Web Server Interface :**

  The following interface appears when the web browser reaches http://127.0.0.1:5698/main_ar.html. The title "Welcome to ENCS3320 - Computer Networks Webserver" is shown at the top of the page, which serves as the web server's primary entrance point. This page is fully translated into Arabic for Arabic-speaking users. The following elements make up the page's organization:

  1. **Team Member Information:** The team members' names, IDs, and photos are displayed along with other details about their projects, skills, and interests, all presented in Arabic to ensure accessibility for Arabic-speaking users.

figure 82:Web Server Interface - Task 2

  2. **Course Content**:A topic from the course material is presented effectively, including prepared text, graphics. This section includes key concepts from the first chapters of the textbook, with content formatted in a clear and accessible manner for Arabic-speaking users

figure 83:Web Server Interface - Task 2

figure 84:Web Server Interface - Task 2

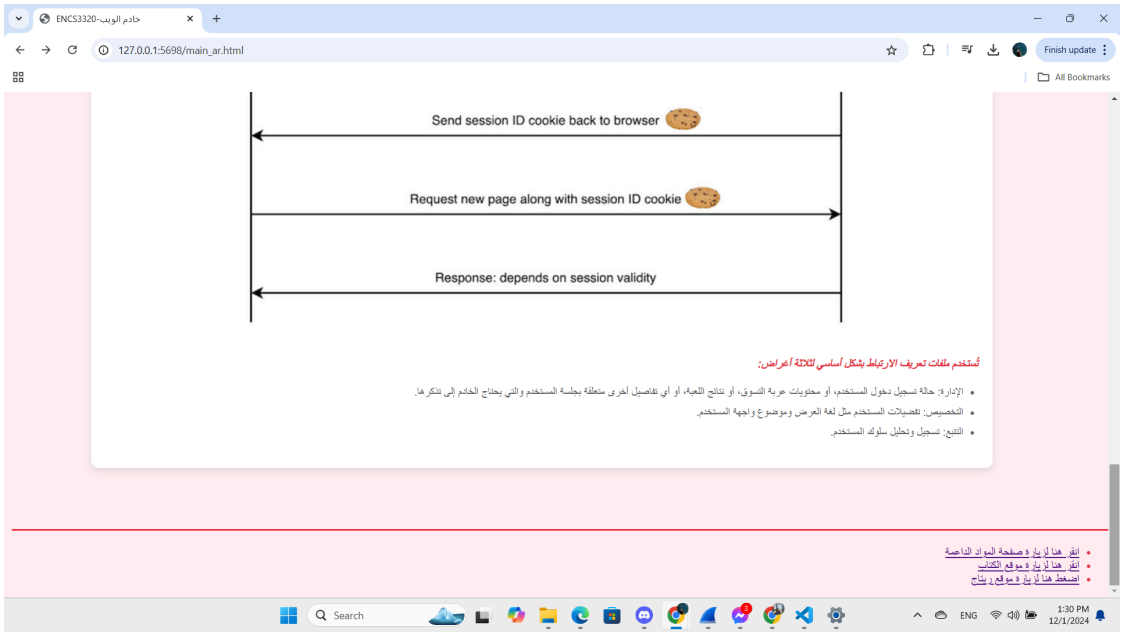


figure 85:Web Server Interface - Task 2

The server is set up to process several types of HTTP requests and, depending on the URL, reply with the relevant content. The implemented situations and the accompanying server behavior are shown below, along with images to support them:

1. http://localhost:5698/ar

figure 86:Web Server Interface - Task 2

Task 3

UDP Client-Server Trivia Game Overview:

using UDP socket programming, we developed an interactive multiplayer trivia game that allows clients to connect to a server and compete in answering trivia questions. The server listens for client connections and manages the game flow, including broadcasting questions, collecting answers, tracking scores, and announcing winners. The game supports multiple rounds, and the server handles communication between clients in real time. The server also ensures fair game play by awarding points based on the speed and correctness of answers. This task highlights the use of **UDP (User Datagram Protocol)** for fast, efficient, and low-latency communication in a client-server model, making it ideal for real-time multiplayer games.

Theory and Procedure:

UDP vs. TCP:

For the trivia game, **UDP** (User Datagram Protocol) was chosen over **TCP** because of its low-latency, connectionless nature, which is ideal for real-time applications. UDP allows fast transmission of data without the overhead of establishing and maintaining a connection, making it suitable for time-sensitive interactions like answering trivia questions. Unlike TCP, UDP does not guarantee delivery, but for this game, occasional lost packets (like missed answers) are acceptable in exchange for speed.

Client-Server Model:

The **client-server model** is used to structure communication between the server and multiple clients. The **server** handles the game by managing client connections, broadcasting trivia questions, collecting answers, and updating scores. **Clients** (the players) connect to the server, receive questions, and submit answers. The server is responsible for synchronizing the game flow and ensuring that all players are on the same page.

Key Networking Concepts:

- **UDP Packets**: The game uses **UDP packets** to transmit small units of data (such as questions and answers) between the server and clients quickly.
- **IP Addresses and Ports**: Each client and the server are uniquely identified using their **IP addresses** and **port numbers**, ensuring that communication occurs between the correct devices.

- **Broadcasting Messages**: The server broadcasts trivia questions and updates to all connected clients simultaneously, keeping the game synchronized across all players.

Server Responsibilities:

The server plays a central role in managing the game and ensuring smooth gameplay:

- **Client Connection Management**: The server listens on **port 5689** for incoming connections. Clients are identified by their **IP addresses** and **port numbers**, and the server maintains a list of active clients. A minimum of two clients is required to start a round.
- **Round Initialization**: Once enough clients are connected, the server broadcasts a **welcome message** to notify players that the round will begin.
- **Question Broadcast**: The server randomly selects questions and broadcasts them to all clients with a **60-second preparation period** before each question.
- **Answer Collection and Scoring**: Clients submit answers within a set time (e.g., 90 seconds). The server checks the answers, awards points, and accepts only the first answer submitted.
- **Score Tracking and Winner Announcement**: After each question, the server broadcasts the correct answer and updates the scores. At the end of each round, the server announces the leading player.
- **Round Pauses**: The server introduces short **pauses** between rounds, giving players time to review the leaderboard and prepare for the next round.

Client Responsibilities:

The client's role in the trivia game is to connect to the server, receive game data, and submit answers:

- **Connection Management**: The client connects to the server using the provided **IP address** and **port 5689**, submitting a **username** for the leaderboard.
- **Receiving Notifications**: The client listens for notifications from the server, including **round start**, **questions**, and **score updates**.
- **Answer Submission**: After receiving a question, the client has a **90-second window** to submit an answer to the server. The client receives feedback on whether their answer was correct.

Game Flow and Client-Server Interaction:

This is the source code for Task3: [Task3 Source Code]

1. **Server Start and Listening on Port 5689:**

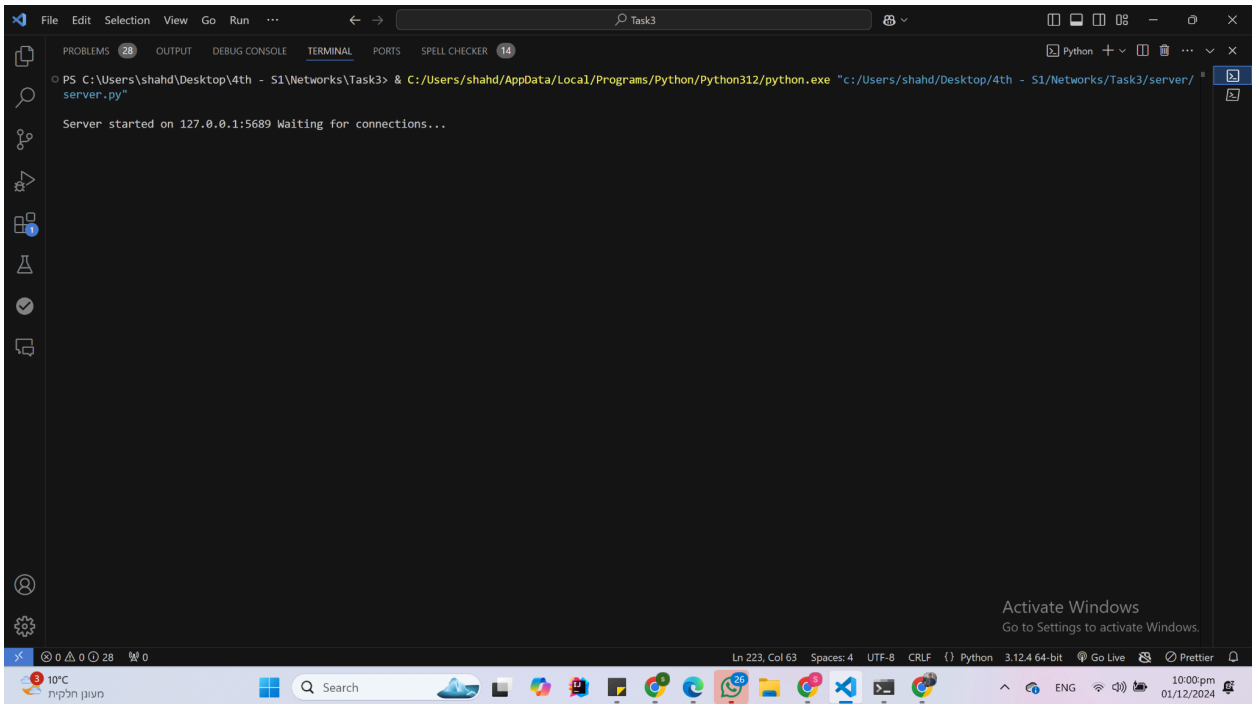


figure 87: server listen on port 5689 - Task 3

The server initializes by creating a UDP socket and binding it to the local host (127.0.0.1) and port number (5689). It enters a listening state, waiting to accept incoming client connections. This stage ensures that the server is prepared to interact with clients and coordinate the trivia game.

2. **Server Handling Multiple Clients Connections for Game Start (3 Clients):**

In this step, the server successfully manages connections from three clients, ensuring all participants are ready before starting the game. Each client is identified by their IP address and port number. After a client connects and submits their name, the server displays the game rules to them. Once at least two clients are connected, the server begins a 90-second waiting period to allow additional clients to join. During this time, the server keeps all connected clients informed of the status. After the waiting period or when three clients have joined, the server finalizes the participant list and prepares to initiate the trivia game, ensuring synchronization and readiness for the upcoming rounds.

=> First client:

- The server receives a connection request from the first client.
- It prompts the client to enter their name and saves their details.
- After registering the client, the server sends a welcome message and displays the game rules to the client.
- The server informs the client that it is waiting for additional players before starting the game.

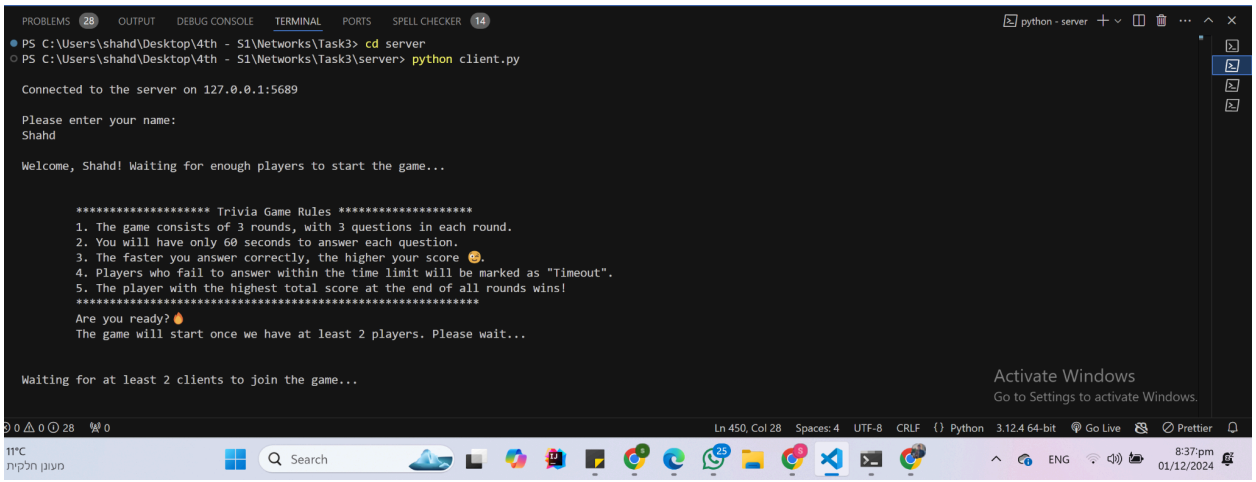


figure 88: first client joined the game - Task 3

=> Second client:

- When the second client connects, the server prompts them for their name and registers their information.
- Like the first client, the second client receives a welcome message, the **game rules as shown in the figure 89**, and an update that the server is now ready to start but will wait for more players.
- Once two clients are connected, the server initiates a 90-second waiting period for additional players to join.
- During this time, the server continues listening for new connections while keeping the current clients updated about the status.

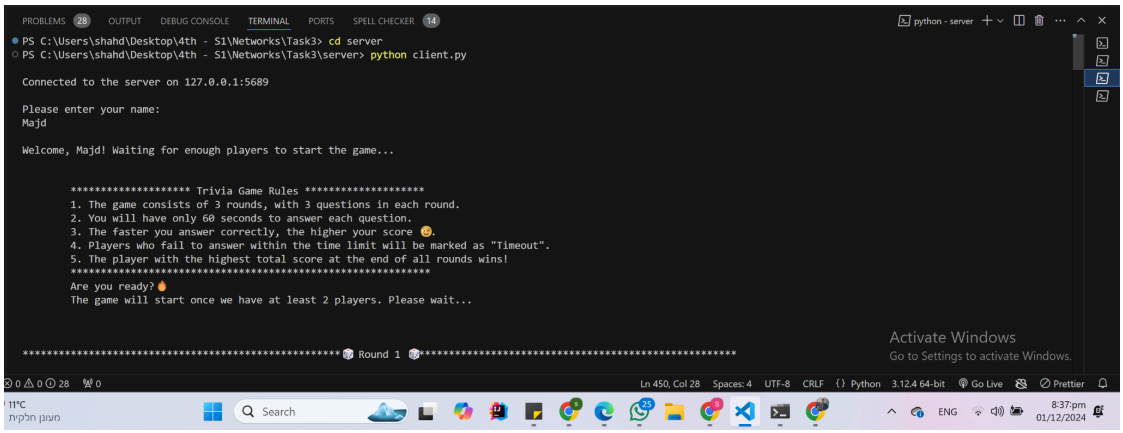


figure 89: Second client Joined The game - Task 3

Third client:

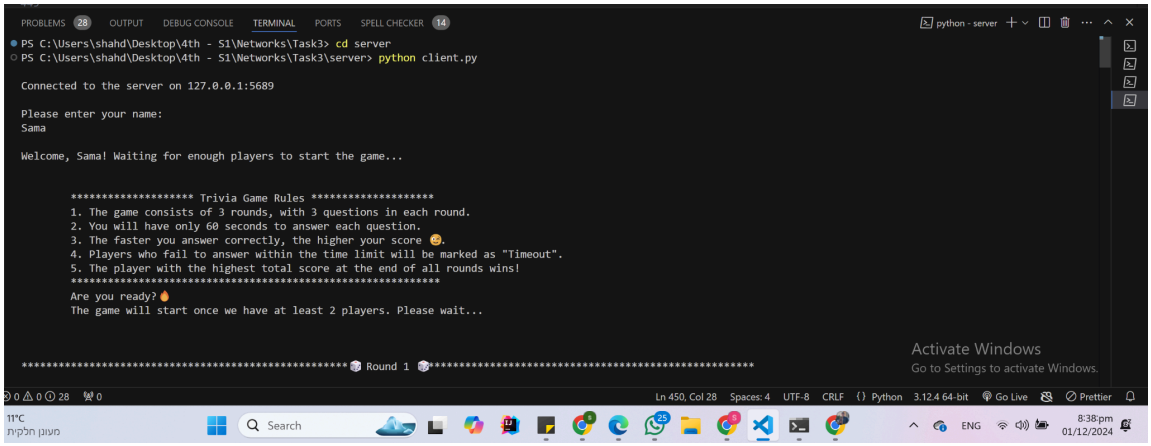


figure 90: third client joined the game - Task 3

3. **Game Start and First Round Progression**

Once the waiting period ends and the game starts, the server transitions into the first round. The following sequence describes how the server handles this phase:

3.1. Broadcasting a Randomly Selected Question:

At the beginning of each question, the server selects a random unique question that hasn't been asked before. This question is broadcasted simultaneously to all active players (at the same time).

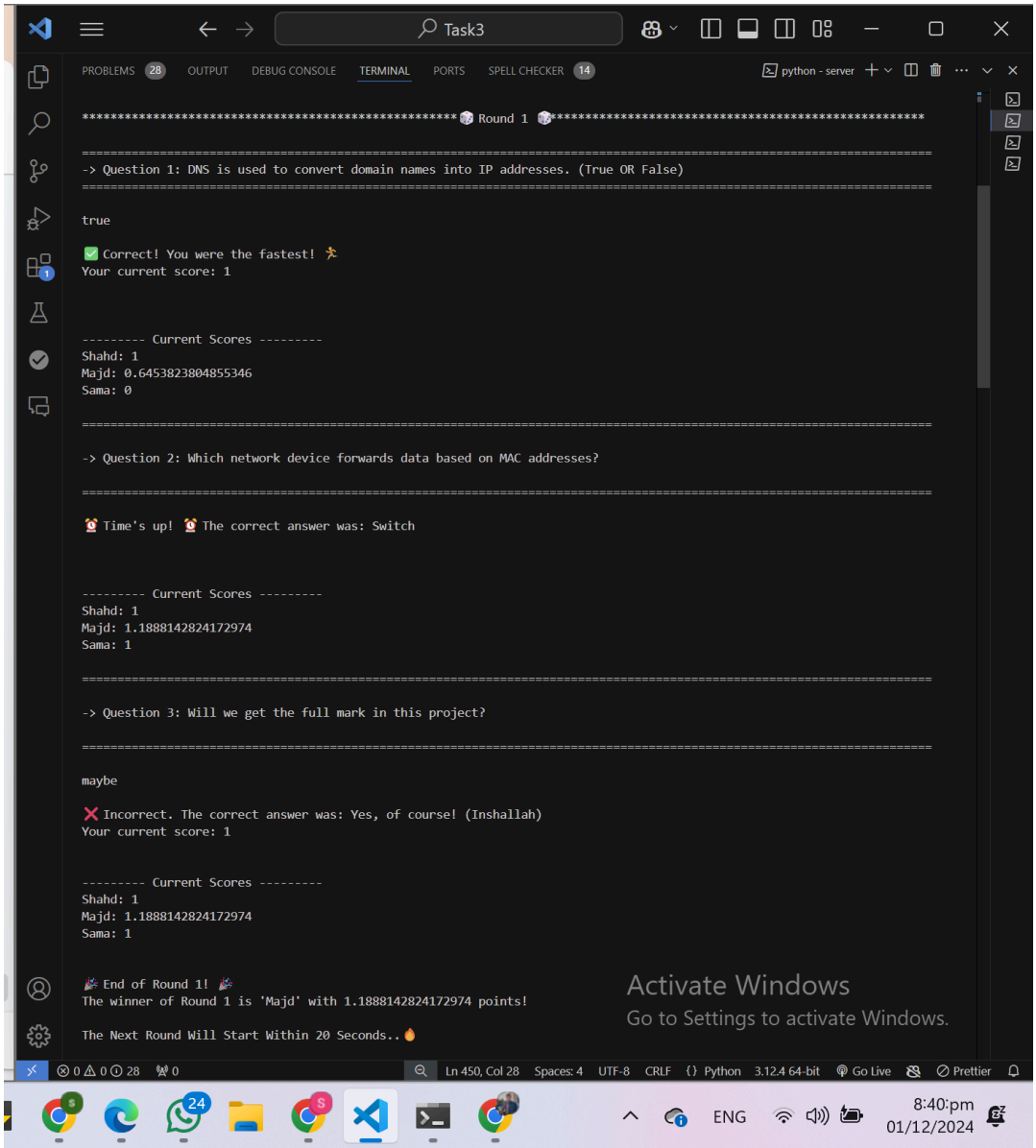


figure 91: Game Start and First Round Progression - Task 3

3.2. Minute Response Timer:

Each client is given 60 seconds to submit their response. During this time, the server listens for answers from all connected players. Players who respond faster and correctly are scored higher, while those who answer incorrectly or fail to respond within the time limit receive no points.

3.3. Handling Responses:

Correct Answers: Players submitting correct answers within the time limit earn points based on their response time, with faster responses receiving higher scores.

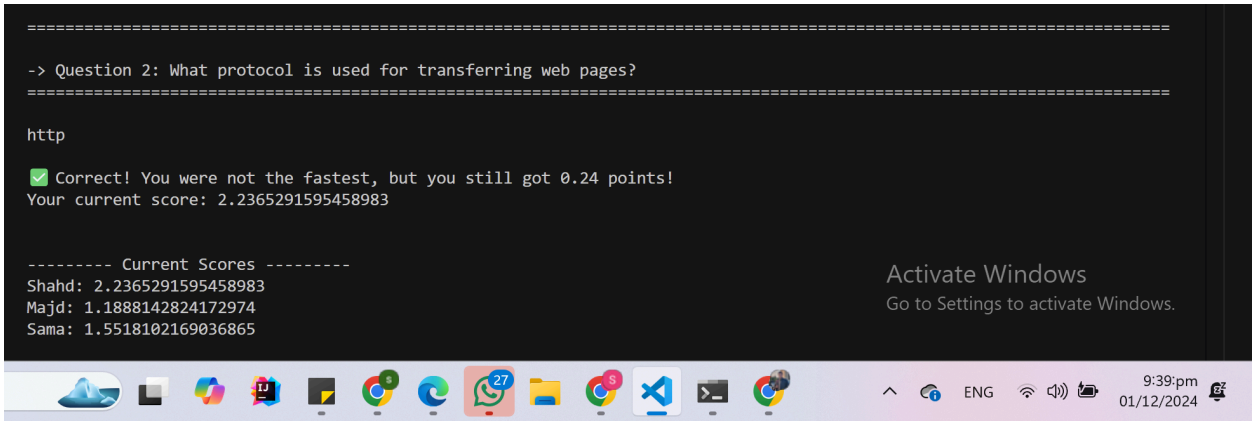


figure 92: handling correct answers - Task 3

Wrong Answers: Players who submit incorrect answers are informed of the mistake and provided with the correct answer.

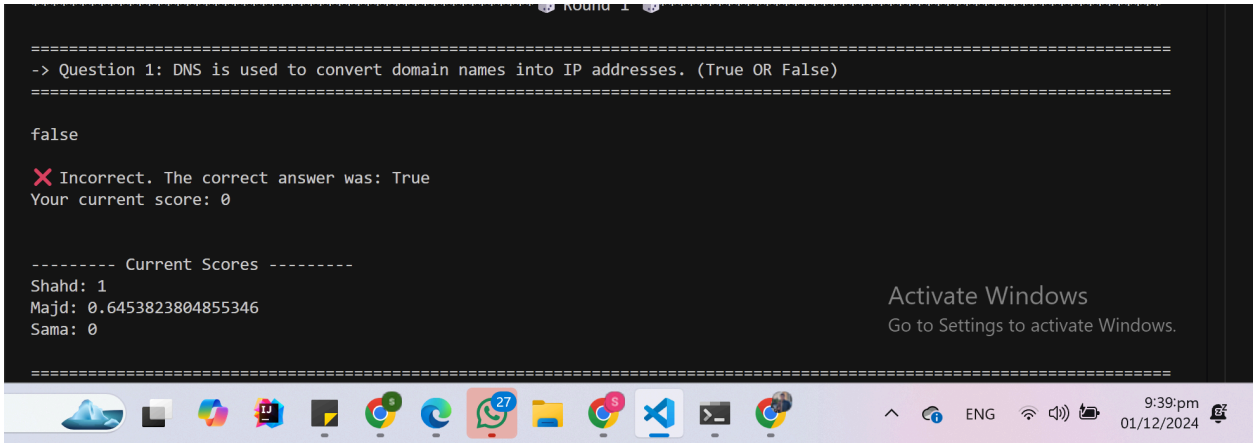


figure 93: Handling wrong answers - Task3

Timeouts: Players who fail to respond within the time limit are notified that the time is up and provided with the correct answer.

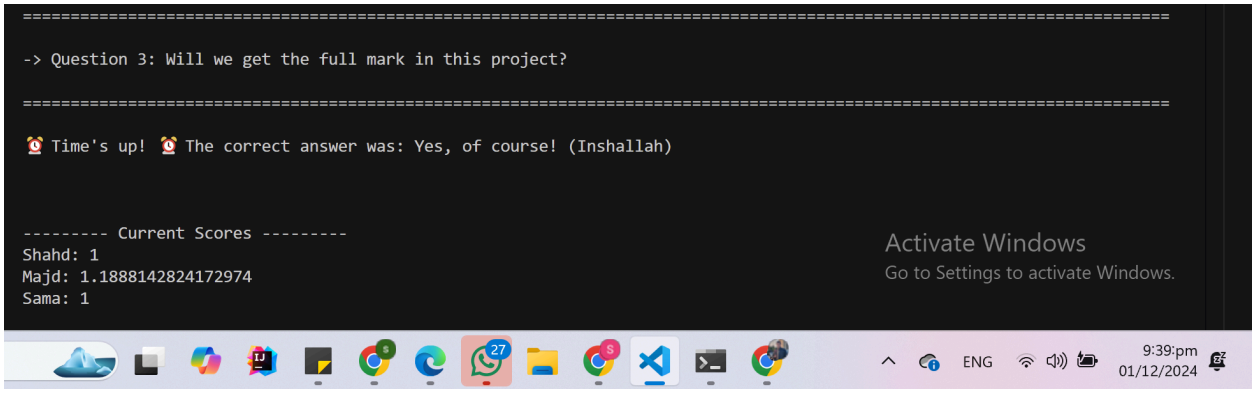


figure 94: Handling Time Up Case - Task 3

3.4. End of Round Announcement And Transition to the Next Round:

After all questions for the current round are completed, the server evaluates and announces the winner of the round. If there is a tie, it highlights the top-scoring players. The server waits for 20 seconds before starting the next round. This short break ensures players have time to prepare for the upcoming questions. During this time, the server broadcasts a message indicating that the next round will begin shortly.

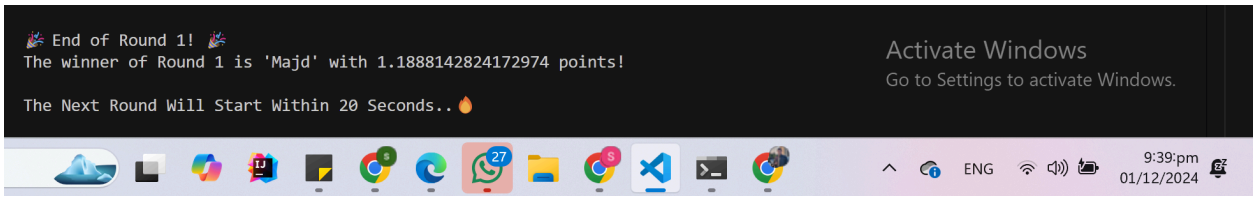


figure 95:End of Round Announcement - Task 3

3.5. Scoring Based on Response Speed

- In this scenario, two clients answer the same question correctly but at different times. The score for each client is determined based on their speed of response.

In this screenshot, we see Client 1's response to the trivia question. Client 1 answered correctly and was the first to respond among all active participants. Since Client 1 was the fastest to submit a correct answer, they receive the full 1 point for the round, and feedback is sent showing the answer was correct and they were the fastest, as shown in figure 96.

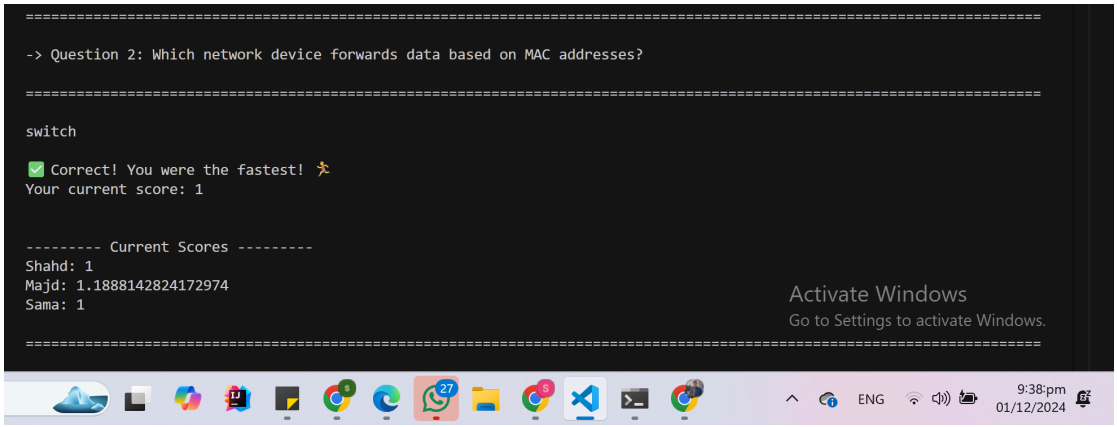


figure 96: Client Answered Question First Case - Task 3

In this second screenshot, Client 2 answers correctly but later than Client 1, who was the fastest. Due to the delay, Client 2's score is reduced by 10% for each second after the fastest response. For example, if Client 2 took 2 seconds longer, their score would be reduced by 20%. Feedback is sent showing the correct answer, but with a reduced score.as shown in figure 97.

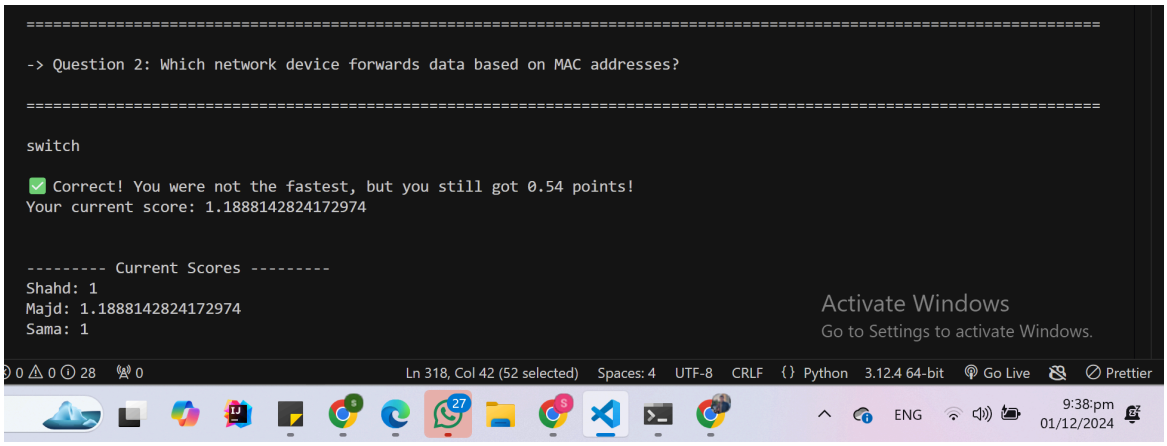


figure 97: Client Answered Correct But Not The First Case - Task 3

3.5. Handling Multiple Submissions: Only the First Answer Counts

In this screenshot, Client 3 submitted multiple answers for the same trivia question. The first answer they submitted was "oo" which was incorrect, so it was mClient 3 submitted the correct answer "HTTP," but this second answer was not considered for scoring. Only the first answer is used to determine the score, so despite the correct second submission, the score remains unchanged based on the first incorrect response. Feedback is sent to the client, showing their first answer as incorrect and updating the score accordingly.

arked as wrong. The system then informed Client 3 of the correct answer. Later,

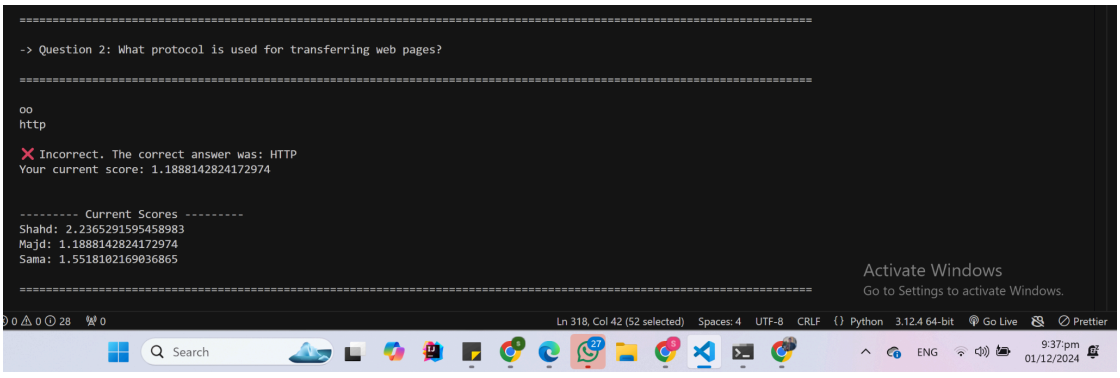


figure 98: Handling Multiple Submissions - Task 3

4. Final Round: Announcing the Winner and Handling No Winner Case

At the end of the final round, the player with the highest total score is announced as the winner. In case of a tie, multiple winners are declared. If no player scores points, there is no winner. After displaying the results, the server sends a final message and waits 20 seconds before closing the connection, ensuring a smooth game conclusion.

4.1. Case 1: A Single Winner is Announced

In this case, at the end of the final round, we calculate the scores from all the rounds to determine the player with the maximum total score. If one player has the highest score, they are declared the winner

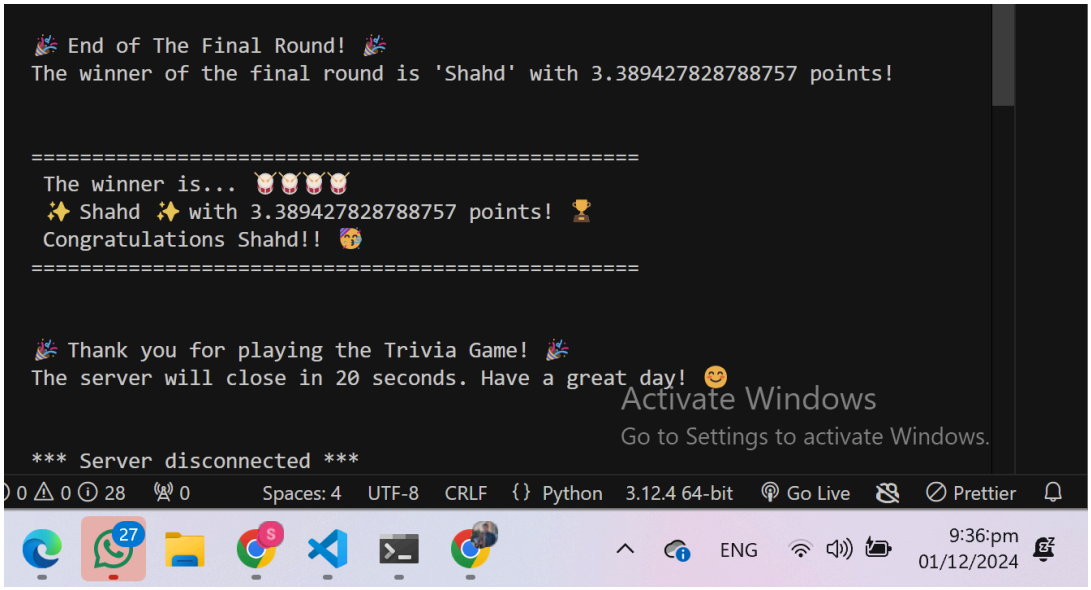


figure 99: A Single Winner is Announced - Task 3

4.2. Case 2: A Tie Between Two or More Players

In this scenario, two or more players tie with the same maximum score at the end of the final round. In such a case, the tie is recognized, and feedback is sent indicating the tied players. The final message may state something like, "We have a tie between Client 1 and Client 2, both scoring the highest points."

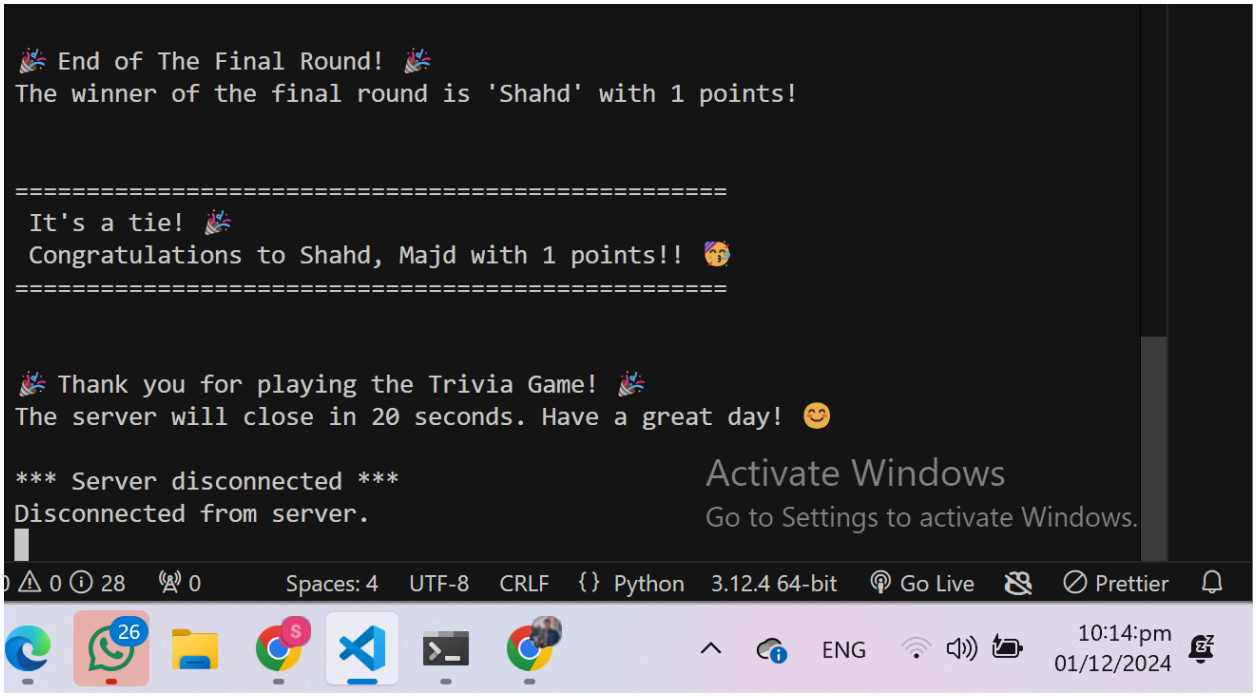


figure 100: A Tie Between Two or More Players - Task 3

Server Terminal Output

In this screenshot, we see a message on the server side indicating that enough clients have joined the game and that a new round will begin. The server will print a message confirming that it has started and is actively listening on a specific port. Additionally, for each new client connection, a message will display the client's IP address and port number to confirm their successful connection to the game.

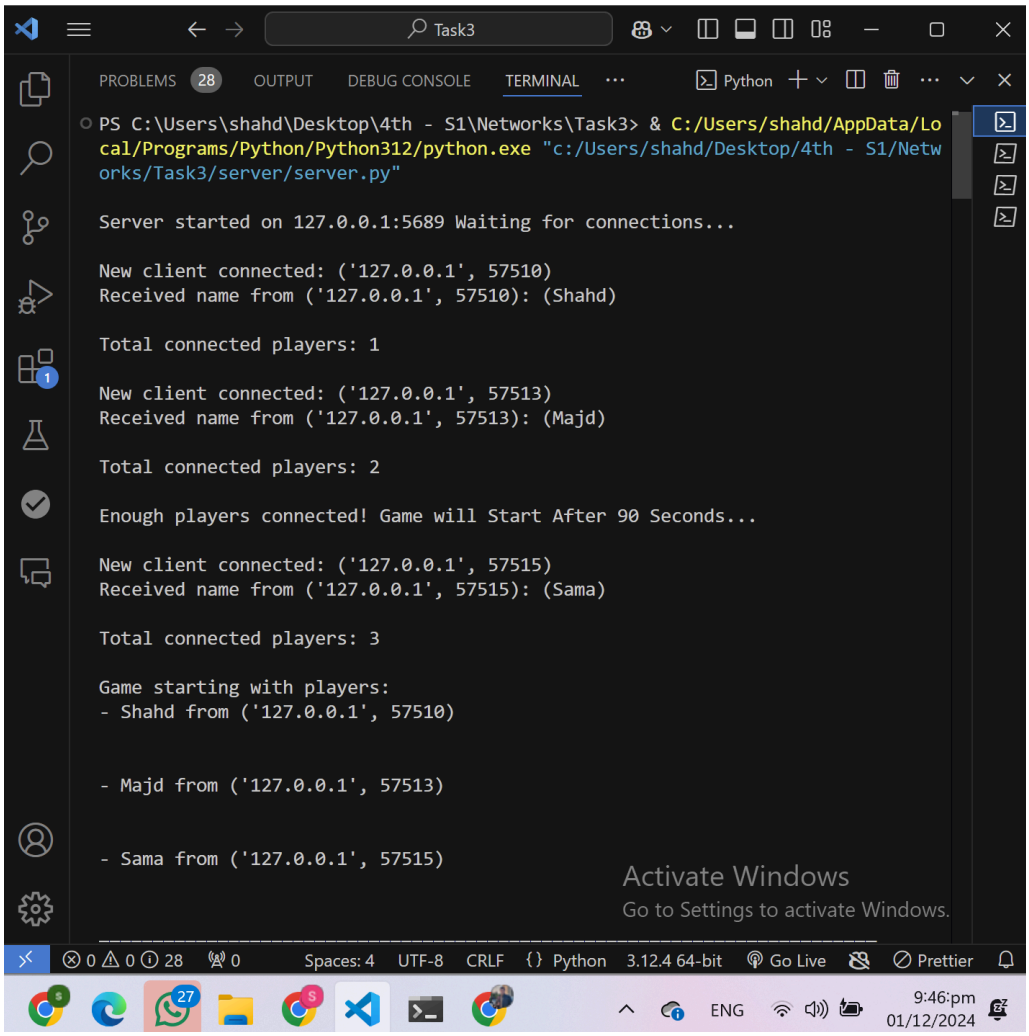


figure 101: server actively listening on a 5689 port. & confirming each new client connection - Task 3

In this screenshot, we see a message on the server side indicating that enough clients have joined the game and a new round will begin.

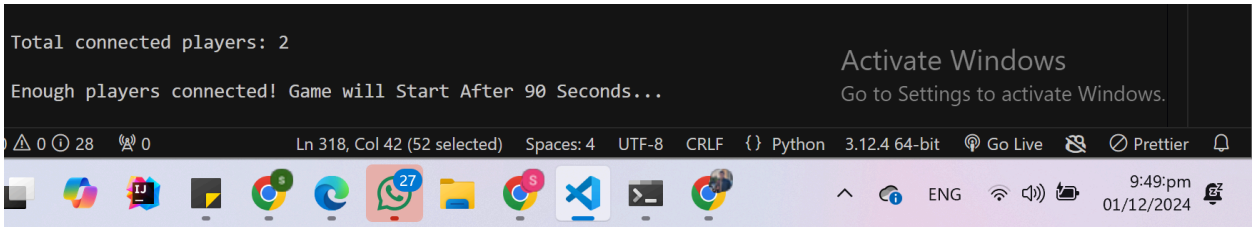


figure 102: message on the server side that enough clients joined - Task 3

In this screenshot, we see the server displaying the question number and text for each question. As answers are received from the clients, the server also displays each client's response along with their identifier (IP and port), and whether the answer is correct or incorrect. This provides transparency and allows both the players and server to track the progress of the game in real-time.

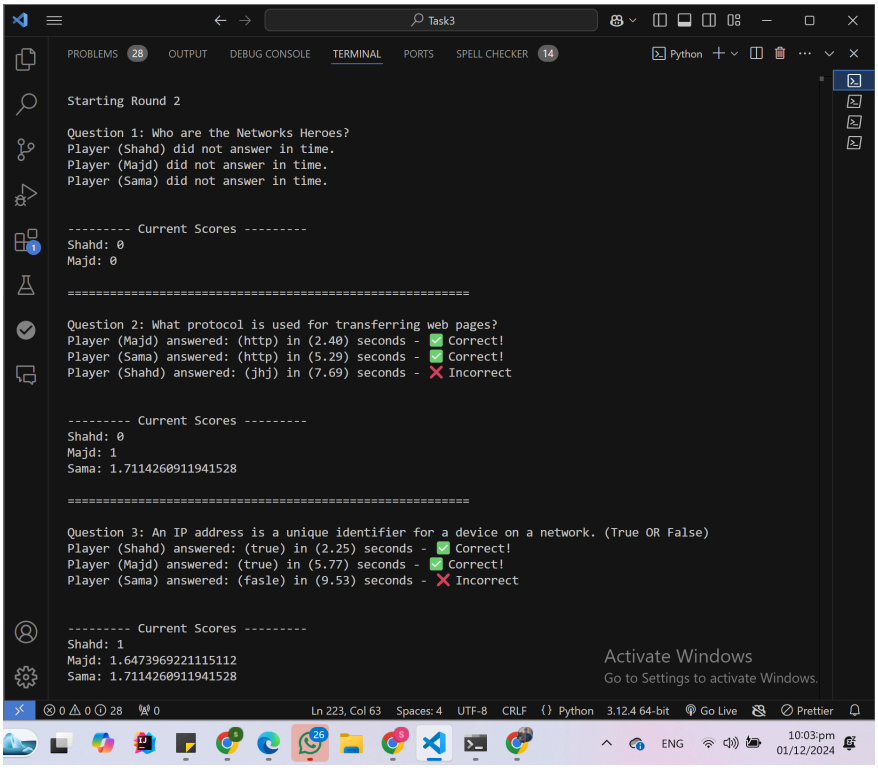


figure 103: question displaying in server side and clients answers with answer result - Task 3

In this screenshot, we see a message on the server side at the end of a round and announcing the winner of the round.

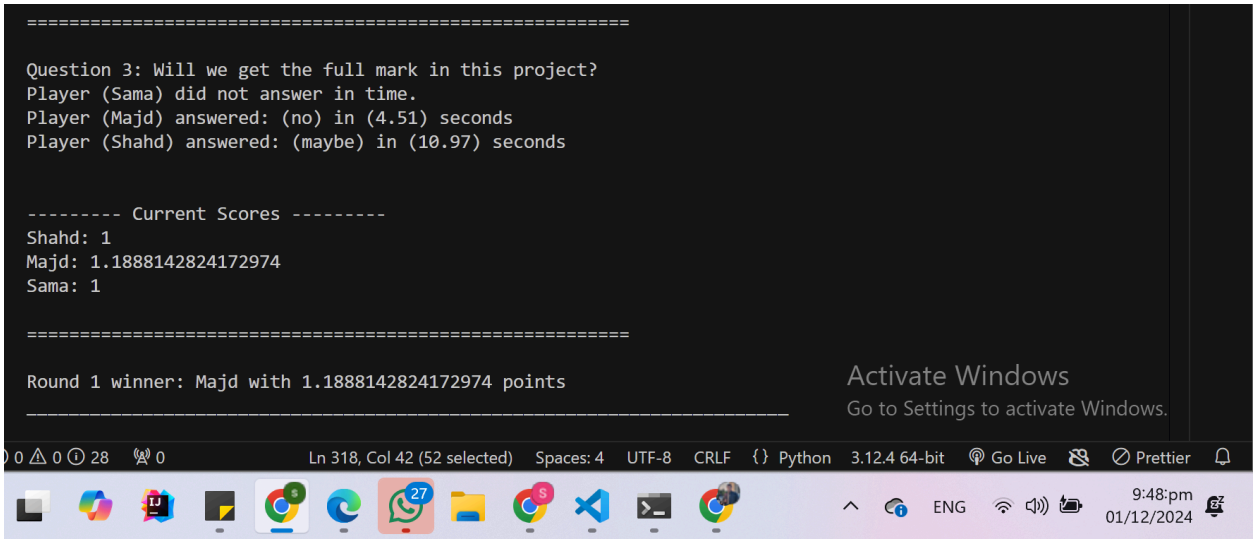


figure 104: message on the server side at the end of a round & announcing round winner - Task 3

In this screenshot i Indicate the countdown in server side (20 sec) before starting the next round.

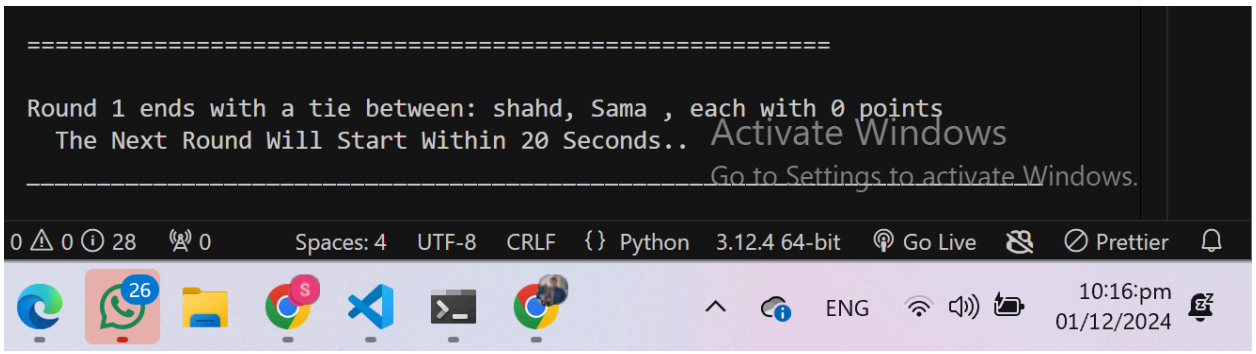


figure 105: message in server side countdown in server side before starting the next round - Task 3

In this screenshot, we see a message on the server side to handle if the server is stopped, by displaying a message confirming the shutdown.

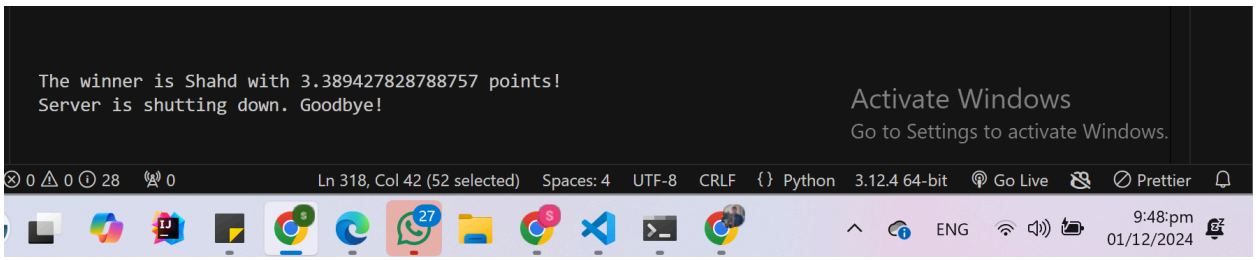


figure 106: server shutting down message - Task 3

Task 3 - Server Side Code

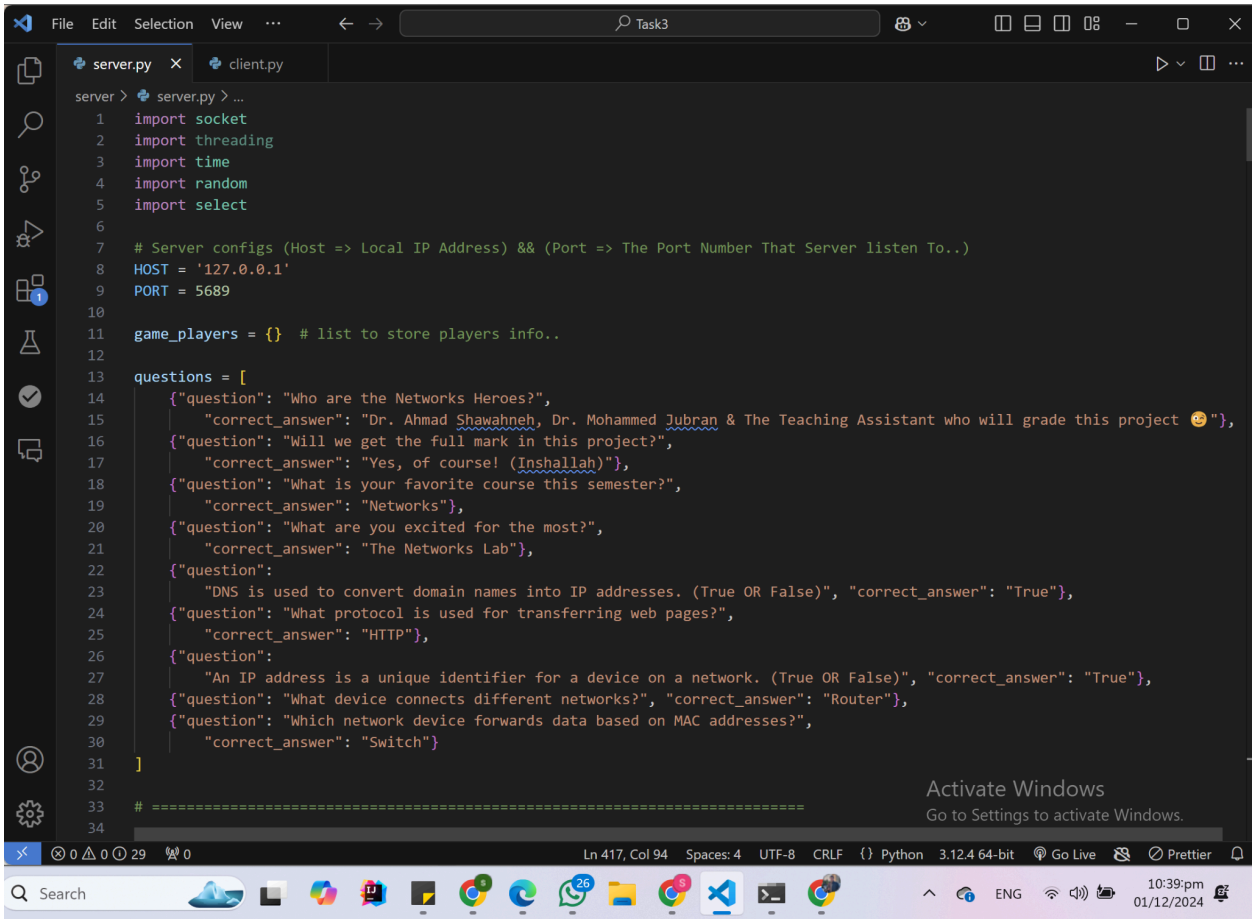
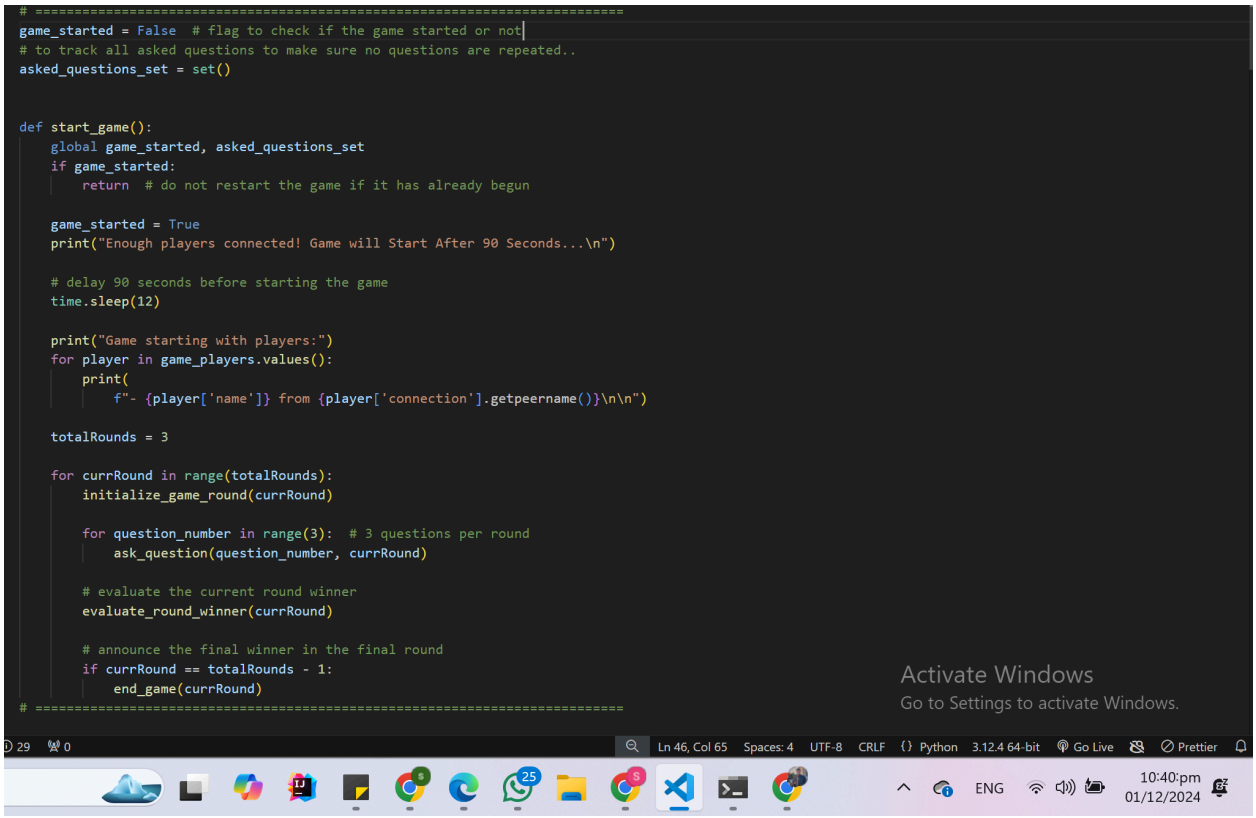


figure 107: server configs & list of question definition code -Task 3

```
def broadcast_message(message):  # msg to all connected active players
    for player in game_players.values():
        try:
            server_socket.sendto(message.encode(), player['address'])
        except Exception as e:
            print(f"**ERROR** -> Error sending message to {player['name']}: {e}")
            del game_players[player['address']]
```

figure 108: method to broadcast messages to clients - Task 3



```
# ========================================================================
game_started = False  # flag to check if the game started or not
# to track all asked questions to make sure no questions are repeated..
asked_questions_set = set()


def start_game():
    global game_started, asked_questions_set
    if game_started:
        return  # do not restart the game if it has already begun

    game_started = True
    print("Enough players connected! Game will Start After 90 Seconds...\n")

    # delay 90 seconds before starting the game
    time.sleep(12)

    print("Game starting with players:")
    for player in game_players.values():
        print(
            f"- {player['name']} from {player['connection'].getpeername()}\n\n")

    totalRounds = 3

    for currRound in range(totalRounds):
        initialize_game_round(currRound)

        for question_number in range(3):  # 3 questions per round
            ask_question(question_number, currRound)

        # evaluate the current round winner
        evaluate_round_winner(currRound)

        # announce the final winner in the final round
        if currRound == totalRounds - 1:
            end_game(currRound)
# ========================================================================
```


figure 109: method to start game and send questions to clients - Task 3

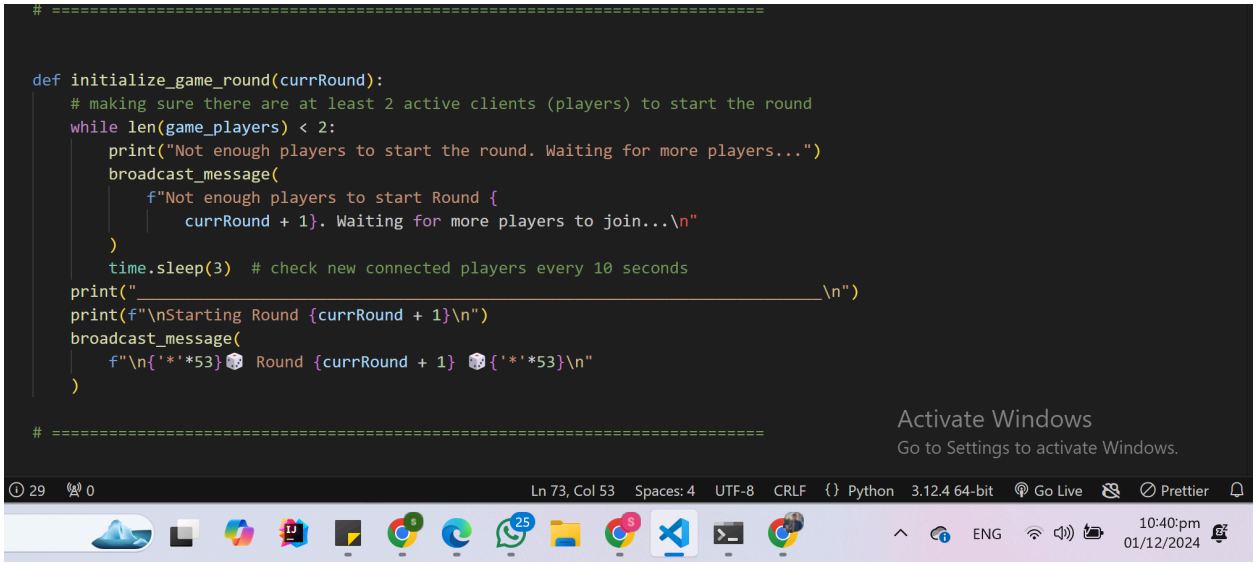


figure 110: method to initialize the game set up -Task 3

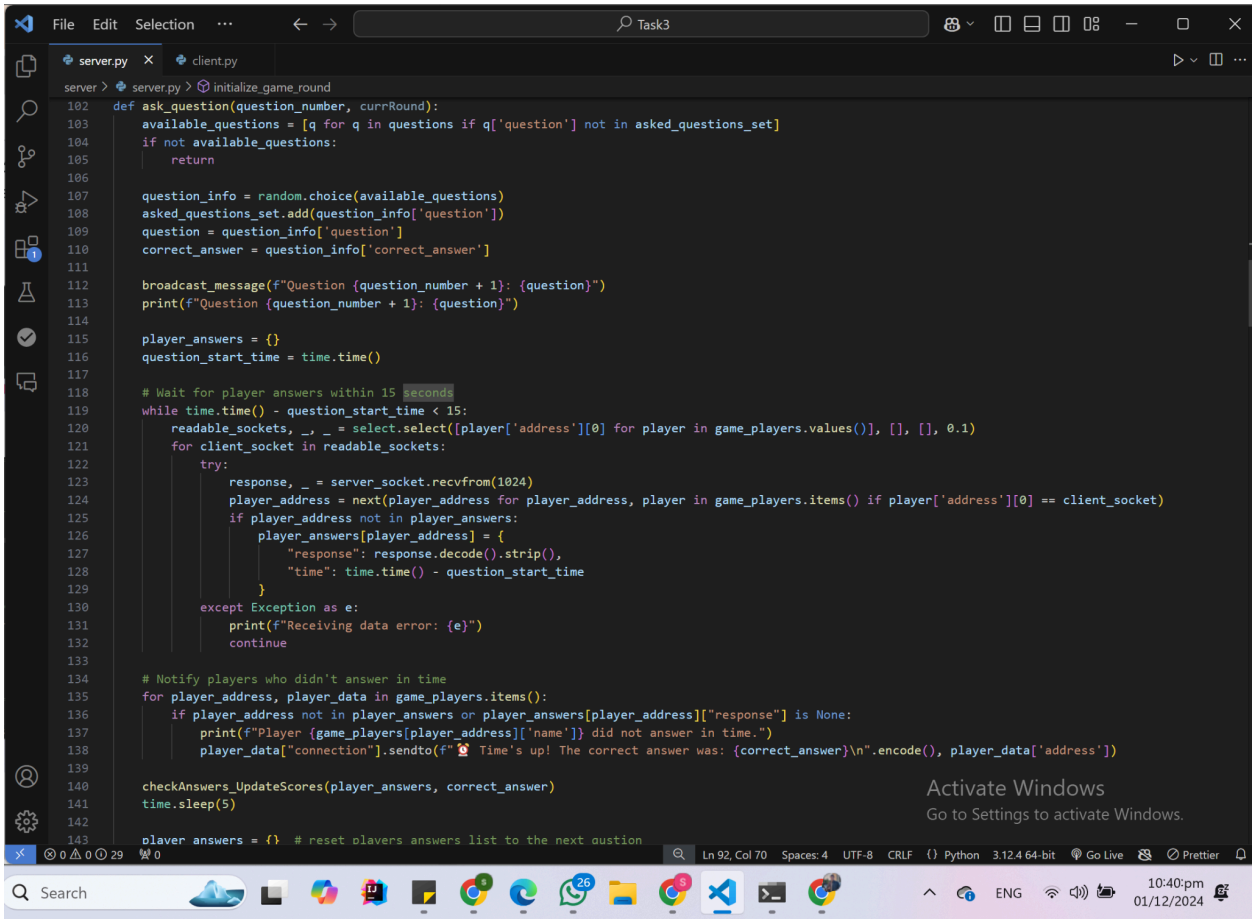
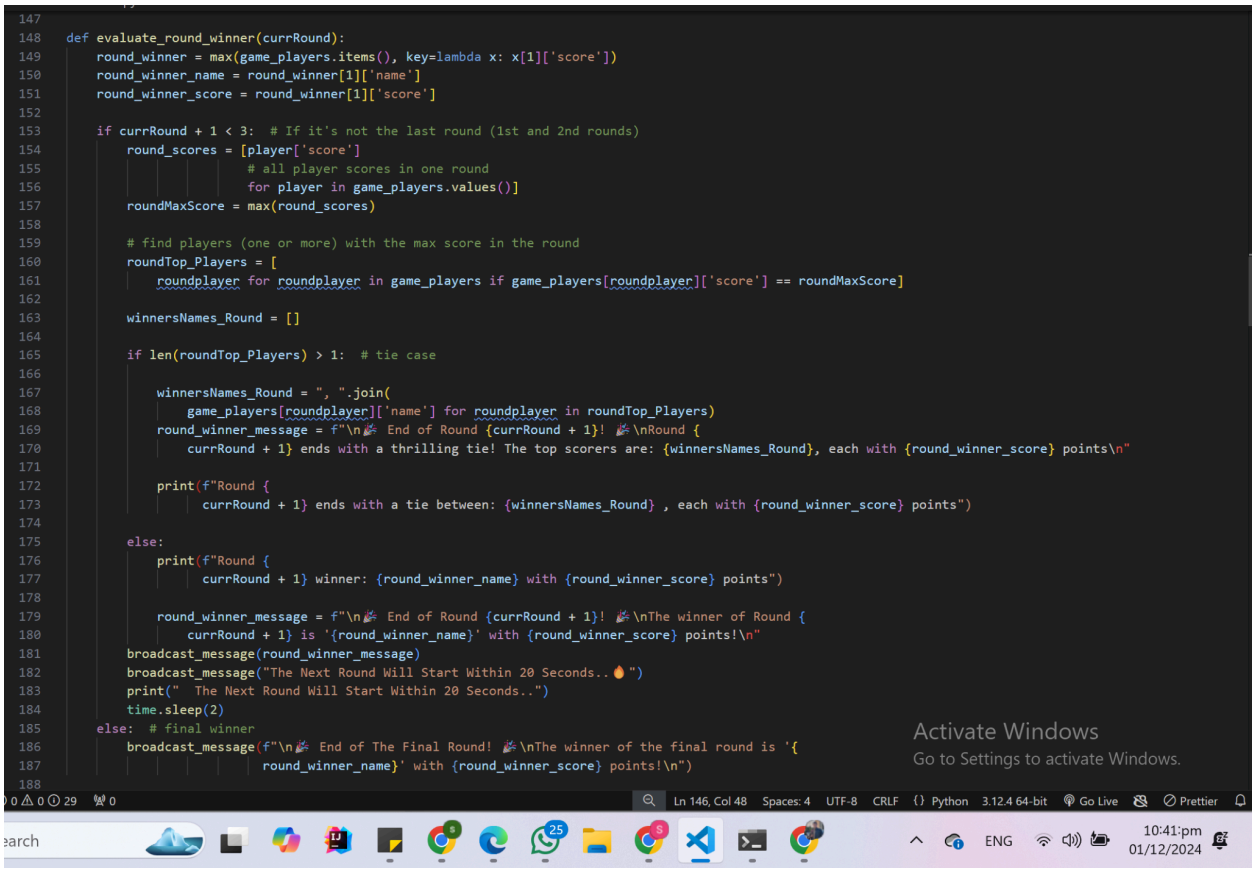
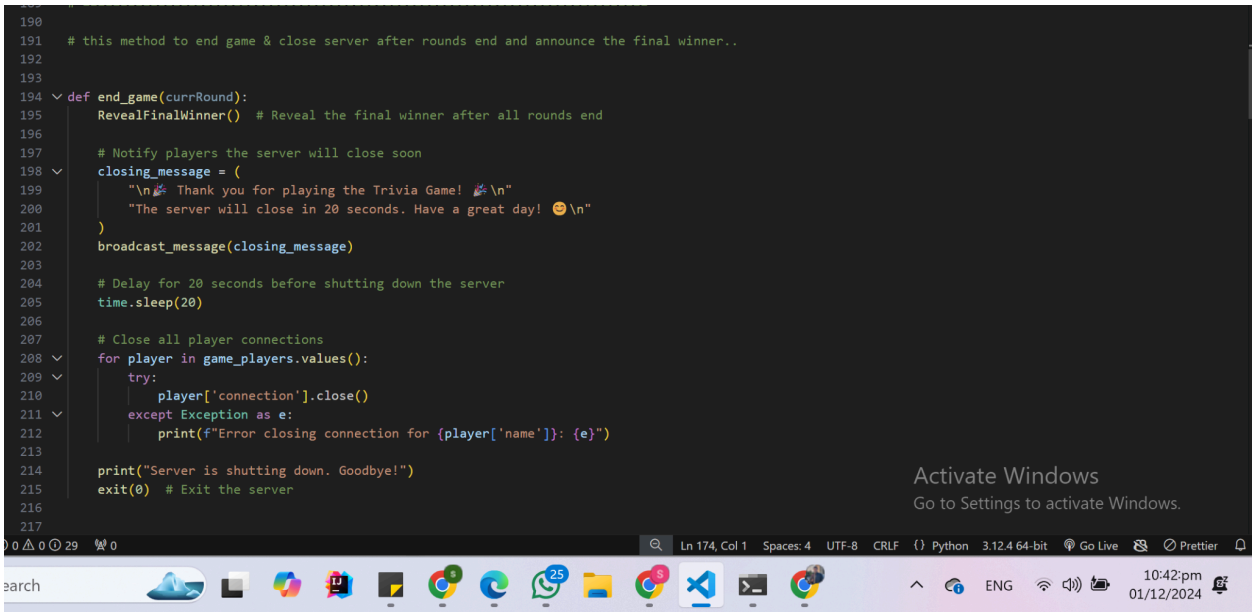


figure 111: method to ask random unique questions to client - Task 3



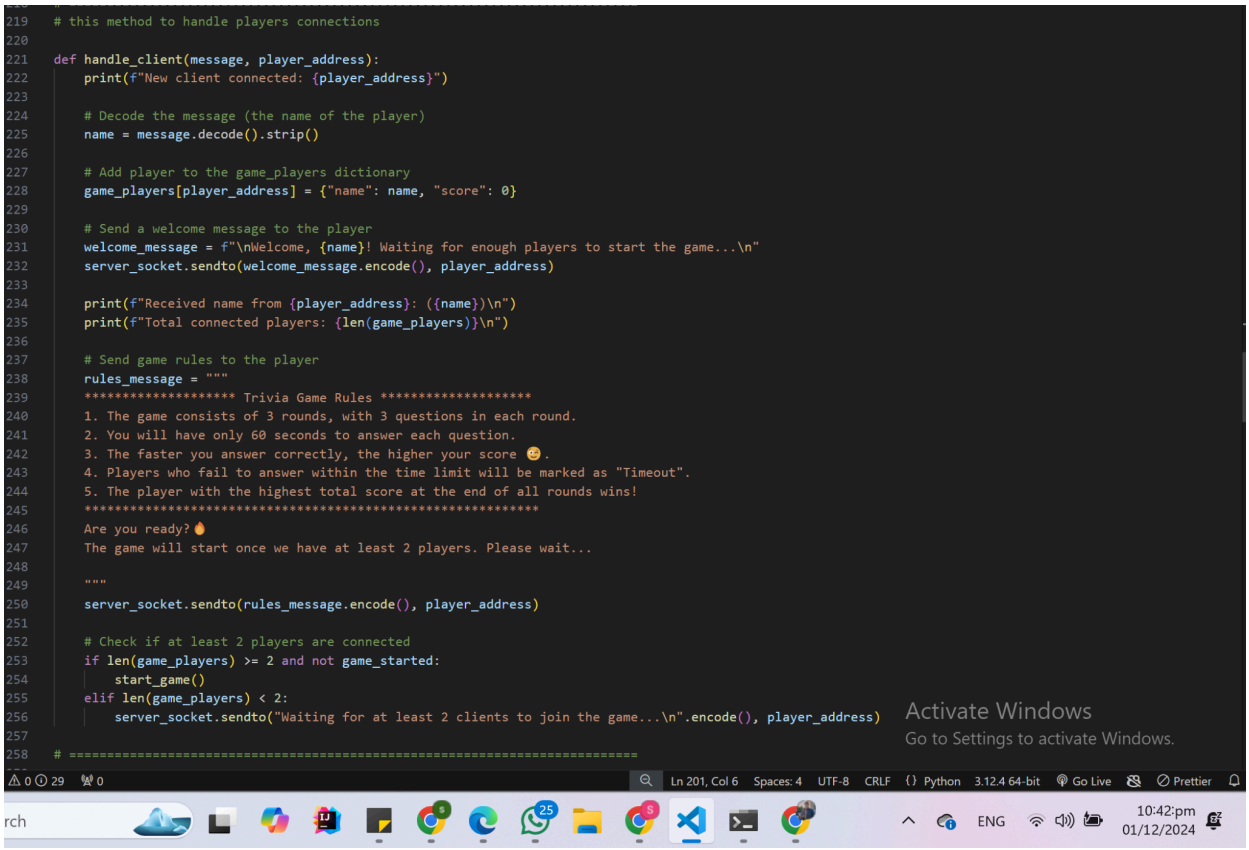
```
def evaluate_round_winner(currRound):
    round_winner = max(game_players.items(), key=lambda x: x[1]['score'])
    round_winner_name = round_winner[1]['name']
    round_winner_score = round_winner[1]['score']

    if currRound + 1 < 3:  # If it's not the last round (1st and 2nd rounds)
        round_scores = [player['score']
                        # all player scores in one round
                        for player in game_players.values()]
        roundMaxScore = max(round_scores)

        # find players (one or more) with the max score in the round
        roundTop_Players = [
            roundplayer for roundplayer in game_players if game_players[roundplayer]['score'] == roundMaxScore]

        winnersNames_Round = []

        if len(roundTop_Players) > 1:  # tie case

            winnersNames_Round = ", ".join(
                game_players[roundplayer]['name'] for roundplayer in roundTop_Players)
            round_winner_message = f"\n🎉 End of Round {currRound + 1}! 🎉\nRound {
                currRound + 1} ends with a thrilling tie! The top scorers are: {winnersNames_Round}, each with {round_winner_score} points\n"

            print(f"Round {
                  currRound + 1} ends with a tie between: {winnersNames_Round} , each with {round_winner_score} points")

        else:
            print(f"Round {
                  currRound + 1} winner: {round_winner_name} with {round_winner_score} points")

            round_winner_message = f"\n🎉 End of Round {currRound + 1}! 🎉\nThe winner of Round {
                currRound + 1} is '{round_winner_name}' with {round_winner_score} points!\n"
        broadcast_message(round_winner_message)
        broadcast_message("The Next Round Will Start Within 20 Seconds..🔥")
        print("  The Next Round Will Start Within 20 Seconds..")
        time.sleep(2)
    else:  # final winner
        broadcast_message(f"\n🎉 End of The Final Round! 🎉\nThe winner of the final round is '{
                          round_winner_name}' with {round_winner_score} points!\n")
```


figure 112: method to evaluate round winner - Task 3



```python
# this method to end game & close server after rounds end and announce the final winner..


def end_game(currRound):
    RevealFinalWinner()  # Reveal the final winner after all rounds end

    # Notify players the server will close soon
    closing_message = (
        "\n🎉 Thank you for playing the Trivia Game! 🎉\n"
        "The server will close in 20 seconds. Have a great day! 😊\n"
    )
    broadcast_message(closing_message)

    # Delay for 20 seconds before shutting down the server
    time.sleep(20)

    # Close all player connections
    for player in game_players.values():
        try:
            player['connection'].close()
        except Exception as e:
            print(f"Error closing connection for {player['name']}: {e}")

    print("Server is shutting down. Goodbye!")
    exit(0)  # Exit the server
```


figure 113: end game and close server method - Task 3



```python
# this method to handle players connections

def handle_client(message, player_address):
    print(f"New client connected: {player_address}")

    # Decode the message (the name of the player)
    name = message.decode().strip()

    # Add player to the game_players dictionary
    game_players[player_address] = {"name": name, "score": 0}

    # Send a welcome message to the player
    welcome_message = f"\nWelcome, {name}! Waiting for enough players to start the game...\n"
    server_socket.sendto(welcome_message.encode(), player_address)

    print(f"Received name from {player_address}: ({name})\n")
    print(f"Total connected players: {len(game_players)}\n")

    # Send game rules to the player
    rules_message = """
    ****************** Trivia Game Rules ******************
    1. The game consists of 3 rounds, with 3 questions in each round.
    2. You will have only 60 seconds to answer each question.
    3. The faster you answer correctly, the higher your score 😊.
    4. Players who fail to answer within the time limit will be marked as "Timeout".
    5. The player with the highest total score at the end of all rounds wins!
    ********************************************************
    Are you ready?🔥
    The game will start once we have at least 2 players. Please wait...

    """
    server_socket.sendto(rules_message.encode(), player_address)

    # Check if at least 2 players are connected
    if len(game_players) >= 2 and not game_started:
        start_game()
    elif len(game_players) < 2:
        server_socket.sendto("Waiting for at least 2 clients to join the game...\n".encode(), player_address)

# ========================================================================
```


figure 114: welcoming and handle client method - Task 3

```python
# this method used to send the same question to all active players in the game at same time..
def send_to_all_players(message):
    for player in game_players.values():
        player['connection'].send(message.encode())
```

figure 115: method to send data to active clients at same time - Task 3

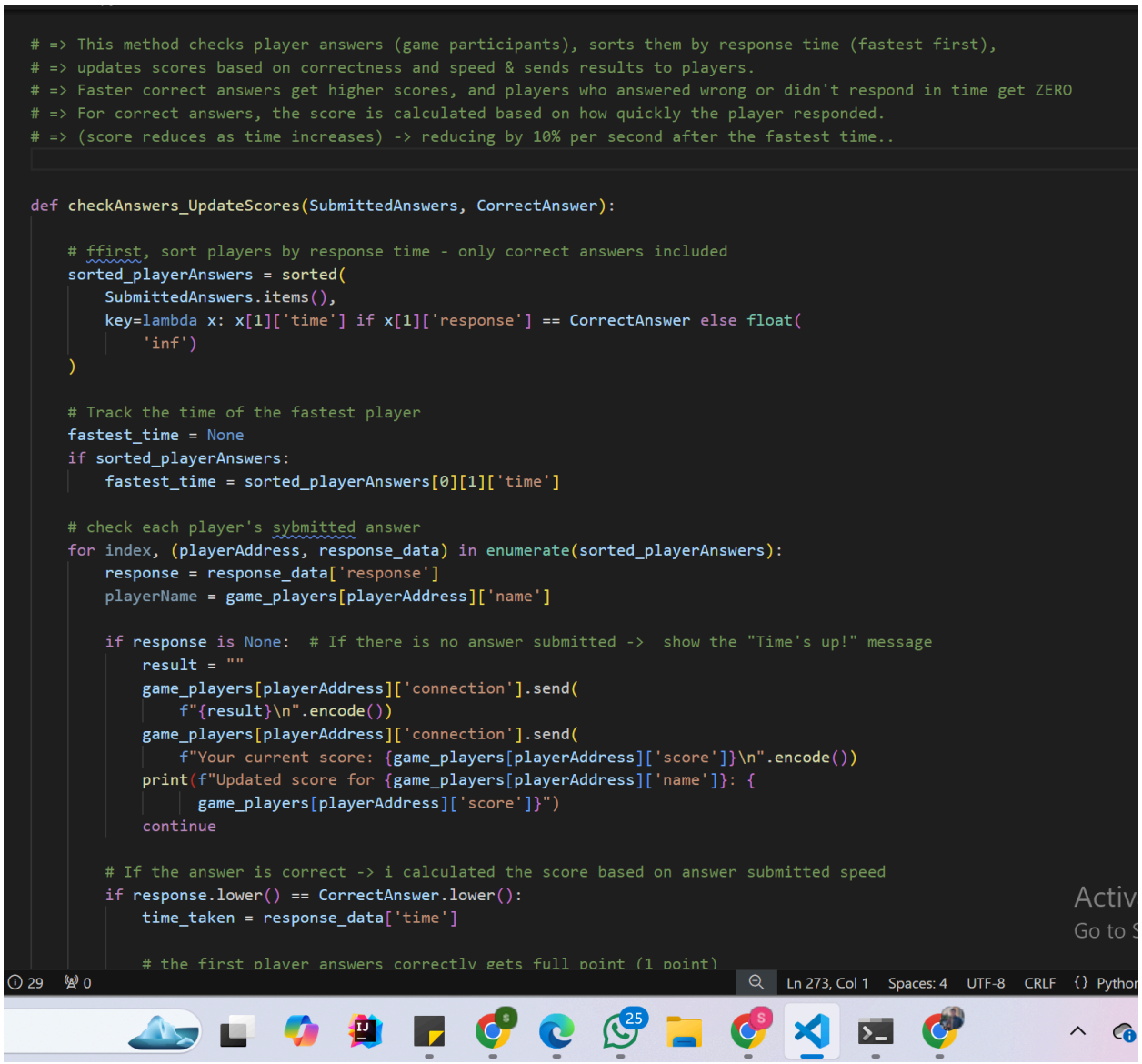


figure 116: method to check answers and update scores - Task 3

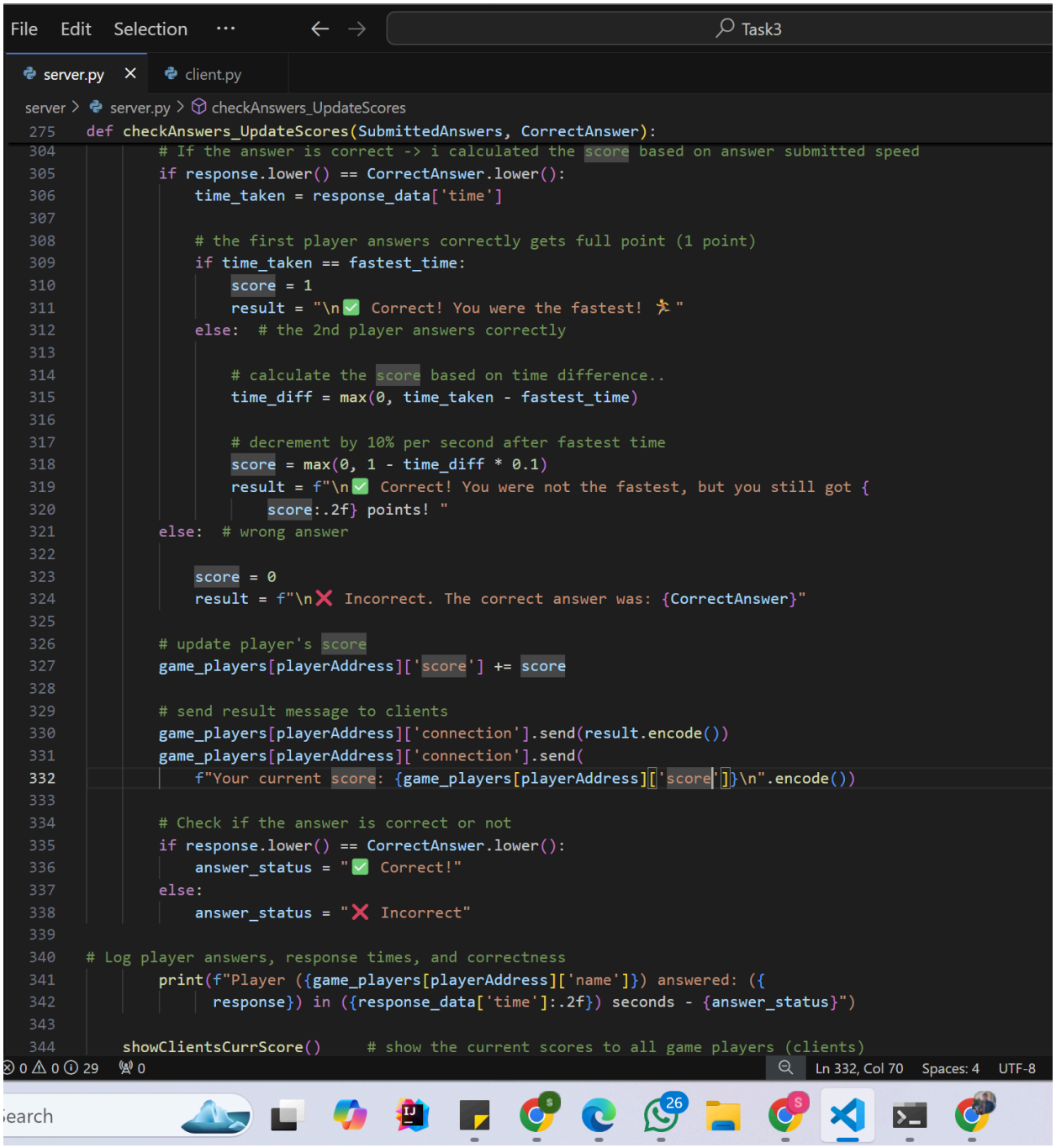


figure 117: method to check answers , update scores & send feedbacks - Task 3

```python
def showClientsCurrScore():
    score_message = "\n\n--------- Current Scores ---------\n"
    for player in game_players.values():
        score_message += f"{player['name']}: {player['score']}\n"
    broadcast_message(score_message)
    print(score_message)
    print("============================================================\n")
# ============================================================================
```

figure 118: show current score method - Task 3

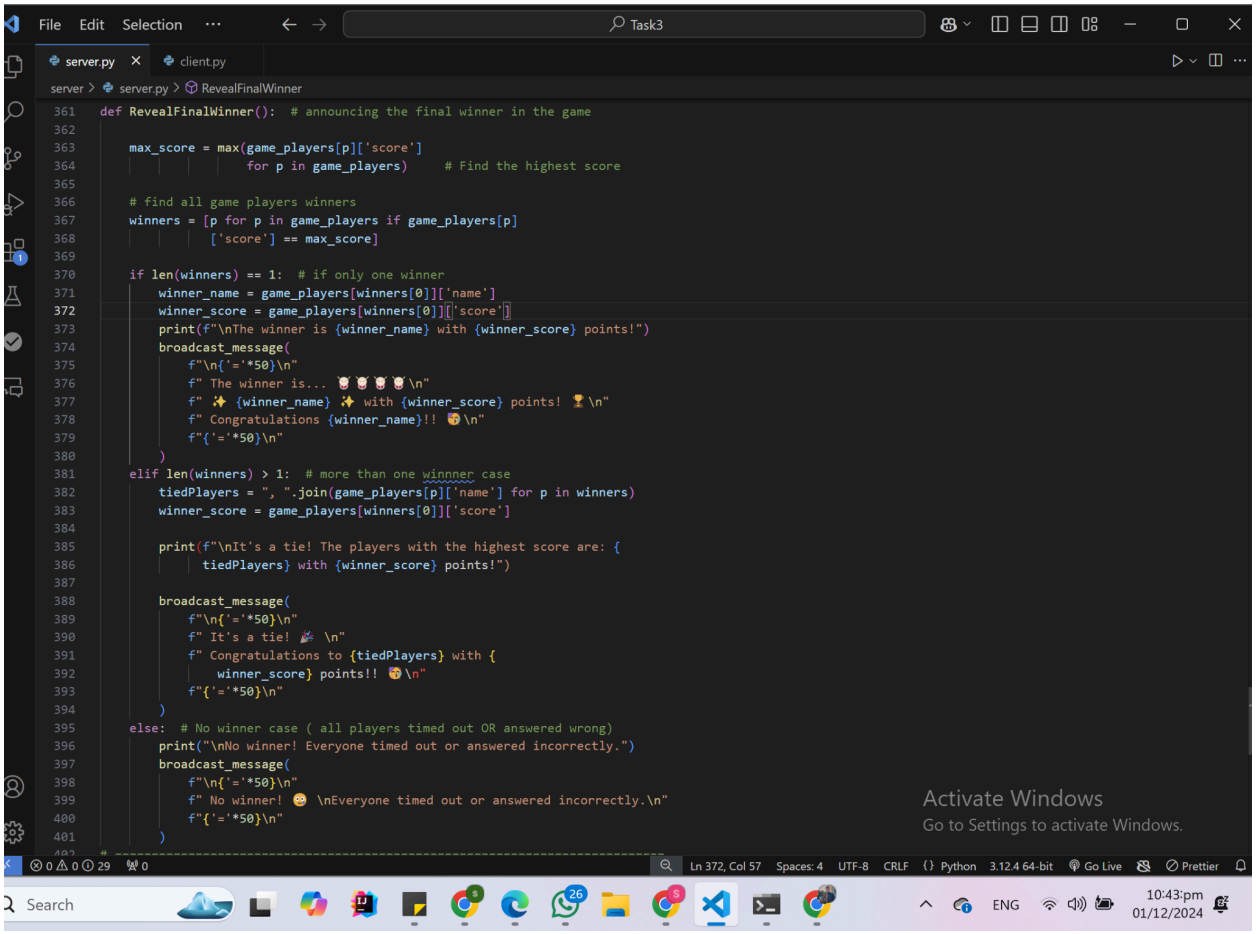


figure 119: calculate final score and announce winner method - Task 3

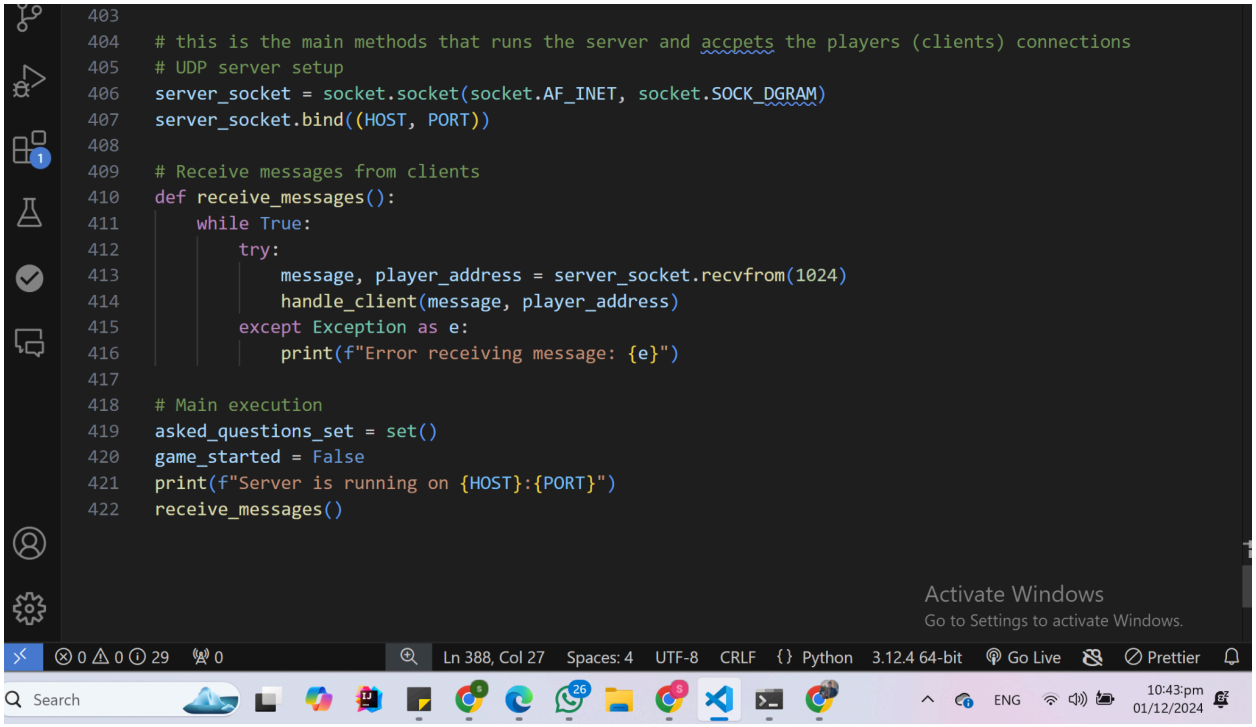


figure 120: main method to set up UDP connection and start server - Task 3

Task 3 - Client Side Code

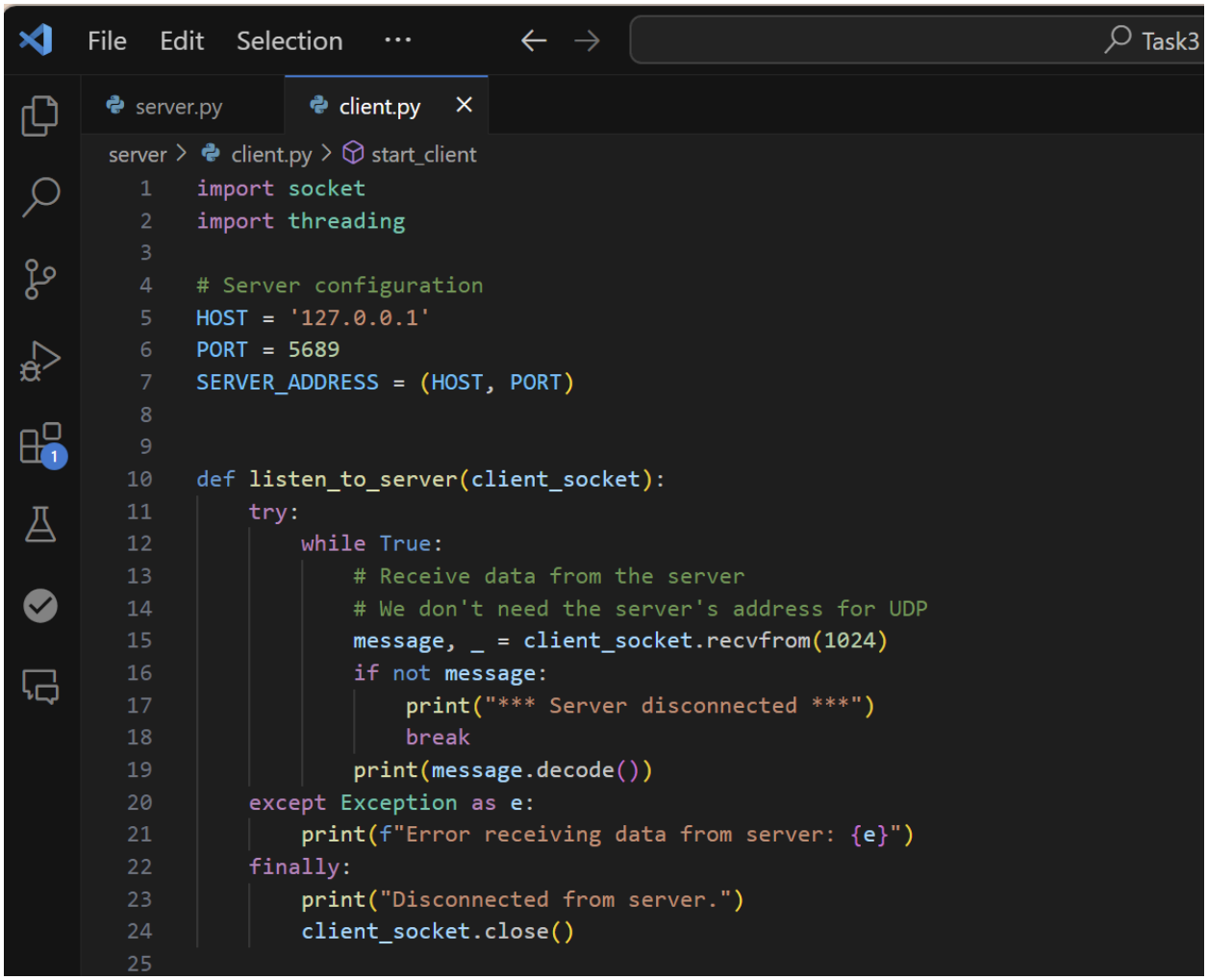


figure 121: Server Configs Client Side Code - Task 3

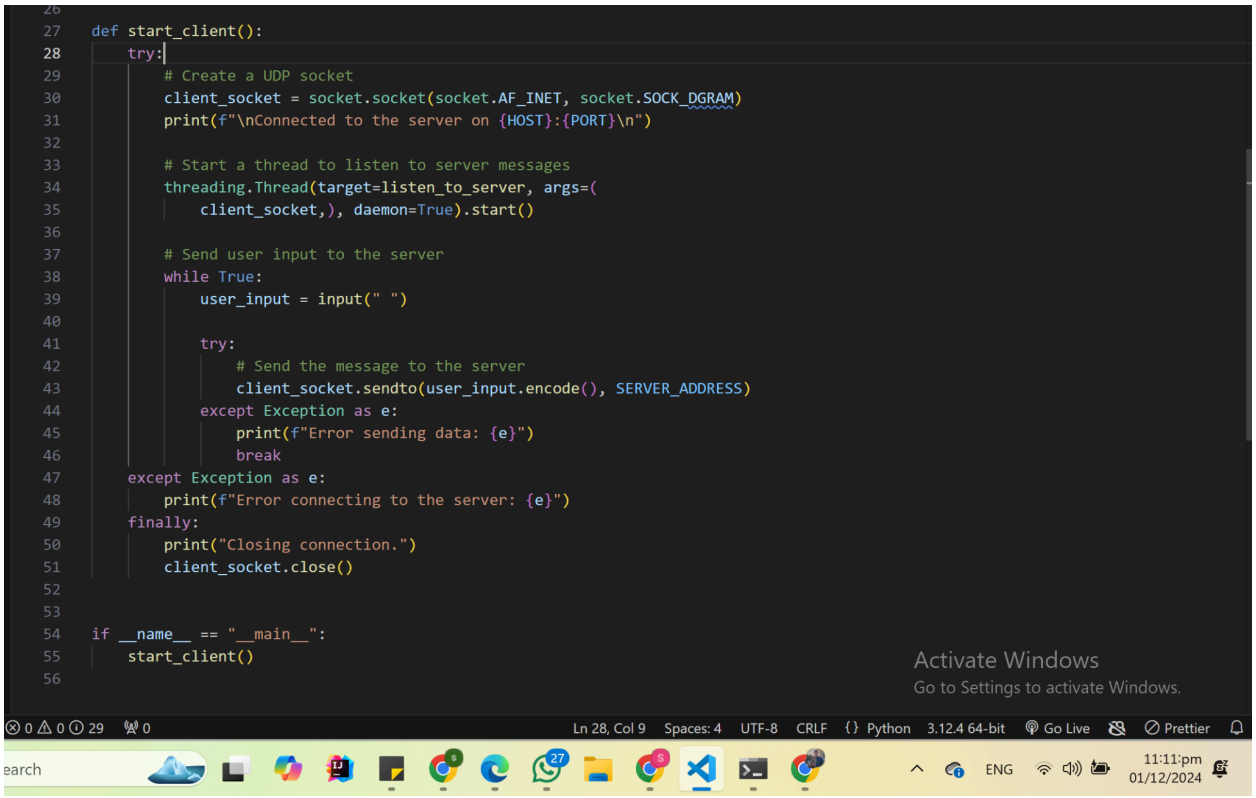


figure 122: Start Client Connection Code CLient Side - Task 3

Teamwork:

Team Member	ID	Tasks Completed
Shahd Khalaf	1210545	• Task 1, section A. • Task 3. • Wrote the report parts for: 1.Task 1(A). 2.Task 3.
Majd Hamarsheh	1212036	• Task 1, section B. • Task 2, section A+C. • Wrote the report parts for: 1.Task 1(B). 2.Task 3.
Sama Wahidee	1211503	• Task 1, section C. • Task 2, section B+C. • Wrote the report parts for: Task 1(C). Task 2.

References:

1. Geeksforgeeks HTTP
2. Wireshark
3. Geeksforgeeks Socket programming
4. Abuseipdb
5. Developer mozilla US/docs Web HTTP
6. Helpful resources
7. IPCONFIG command
8. Ping Command
9. TRACERT Command
10. TelNet Command
11. Nslookup Command