Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Networks ~ ENCS2110

Project 1 (Socket Programming ~ REPORT)

Instructor: Dr. Mohammad Jubran

Prepared by:

Roaa Hamoudah – 1210738 (Section: 5)

Ameer Yasen – 1210187 (Section: 5)

Ata Musleh – 1211245 (Section: 5)

Date: 01/12/2024

Contents

Table of figures:

1 Part 1 Results

Task 1 – Network Commands and Wireshark

a): In your own words, provide a brief (two-sentence maximum) explanation of each of the following commands: (i) ipconfig, (ii) ping, (iii) tracert, (iv) telnet, and (v) nslookup.

(i) ipconfig: (Internet Protocol Configuration) A command line utility that is used to display and manage the IP address assigned to the machine. In Windows, typing ipconfig without any parameters displays the computer's currently assigned IP, subnet mask, and default gateway addresses.

(ii) Ping: a utility used diagnostically to ensure that a host computer the user is trying to reach is operating and measures the round-trip time (RTT)between devices by sending packets and recording the time taken for their return. Typically, it sends multiple packets to ensure accurate measurements.

(iii) Tracert: also known as traceroute, traces and maps the journey of data packets from a source to a destination, revealing each intermediate hop and the time taken for the packets to reach each stop along the way by incrementing the TTL (time to live) from source to destination. It also enables to locate where the data was unable to be sent along, known as points of failure.

(iv) Telnet: a client/server application protocol that provides access to virtual terminals of remote systems on local area networks or the Internet.

(V) nslookup: short for nameserver lookup, is a tool used to find the corresponding IP address or domain name system (DNS) record for a given hostname. This command also supports reverse DNS lookups by inputting the IP addresses of the domains to be looked up. (Essentially, Nslookup facilitates the translation between hostnames and IP addresses in a network environment.

b) Ensure your computer is connected to the internet, then perform the following actions:

1.b.1 Run the ipconfig /all command on your computer and identify the IP address, subnet mask, default gateway, and Domain Name System (DNS) server addresses for your primary network interface.

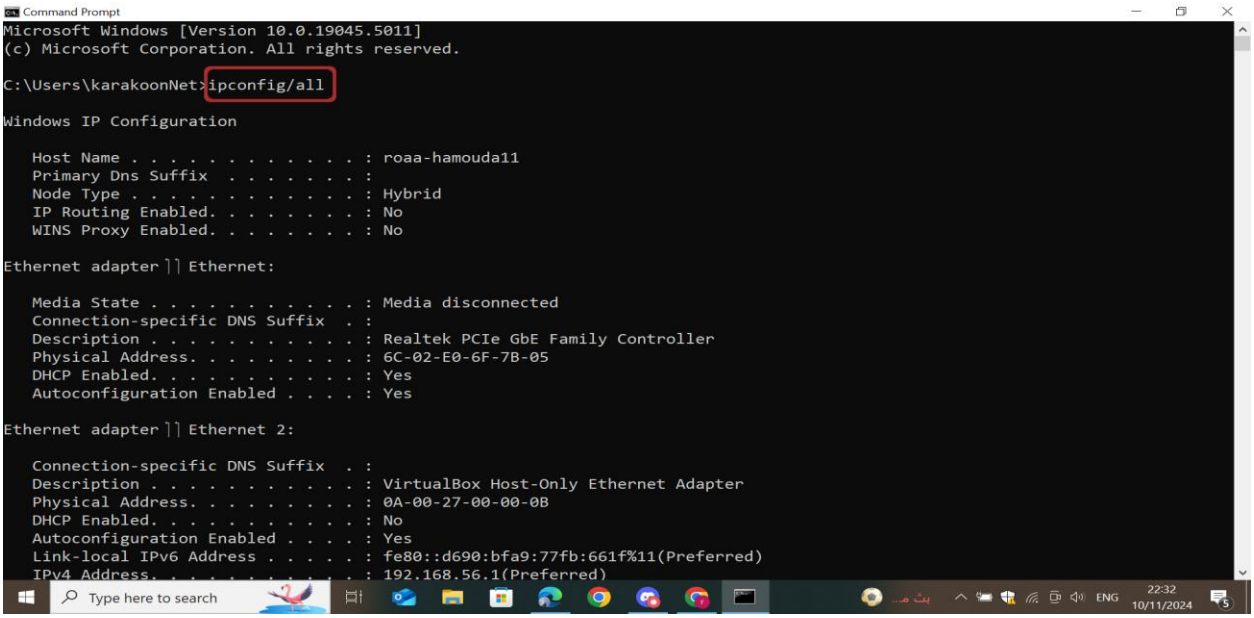


Figure 1 ipconfig

1) IP address:

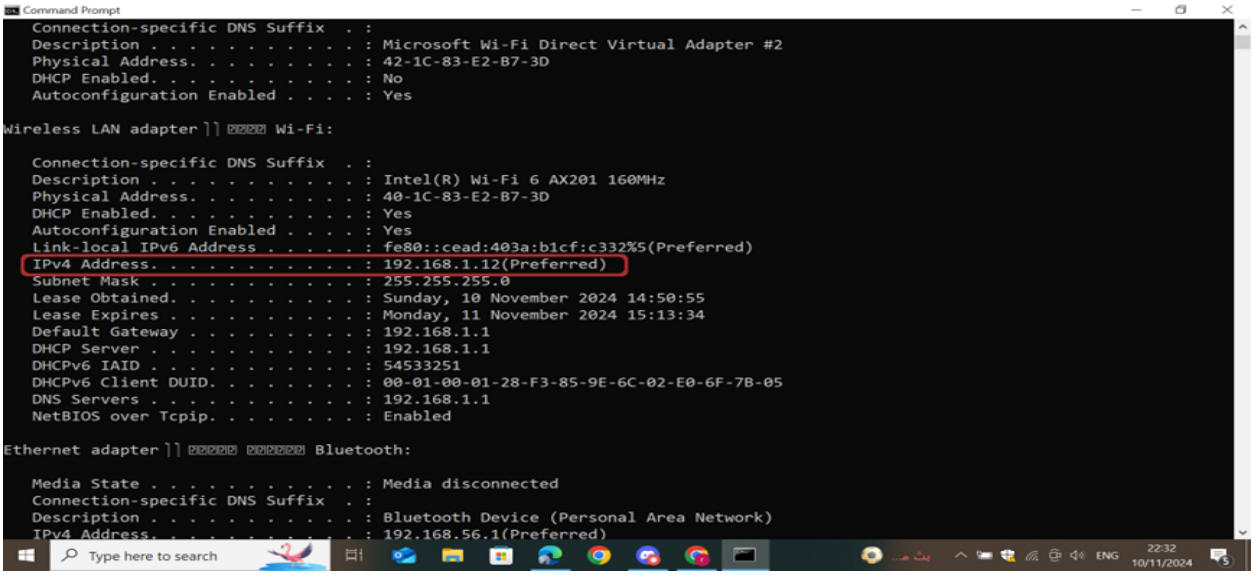


Figure 2 IP address

2) Subnet Mask:

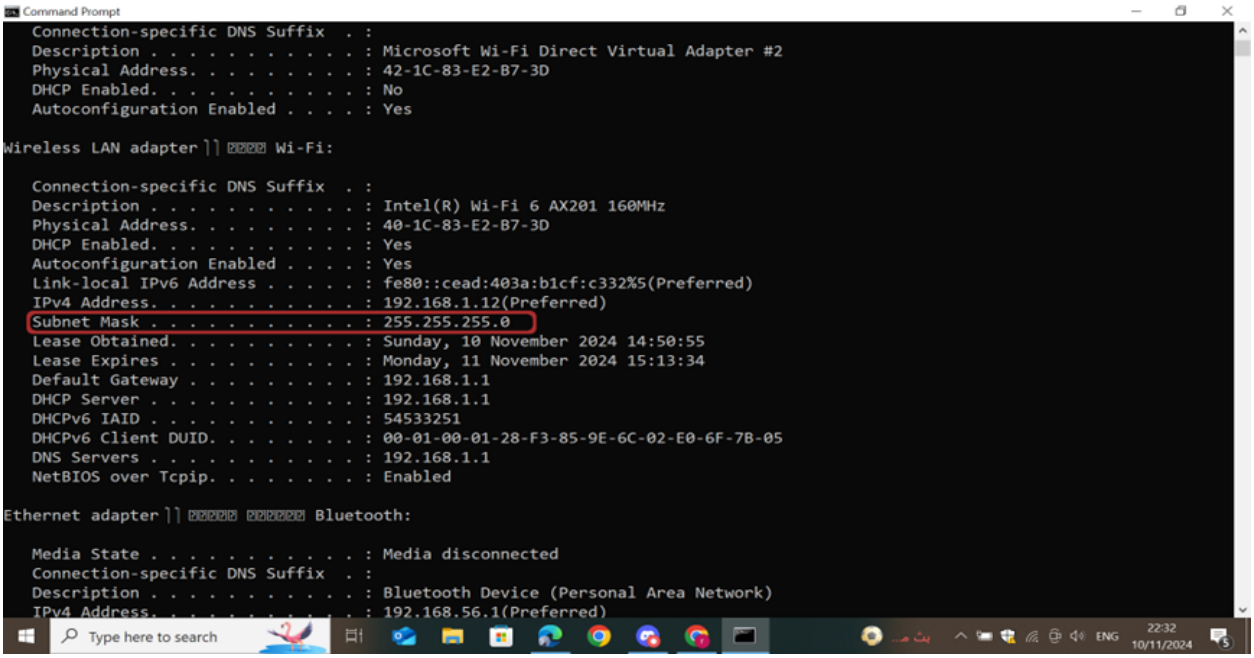


Figure 3 Subnet Mask

3) Default Gateway:

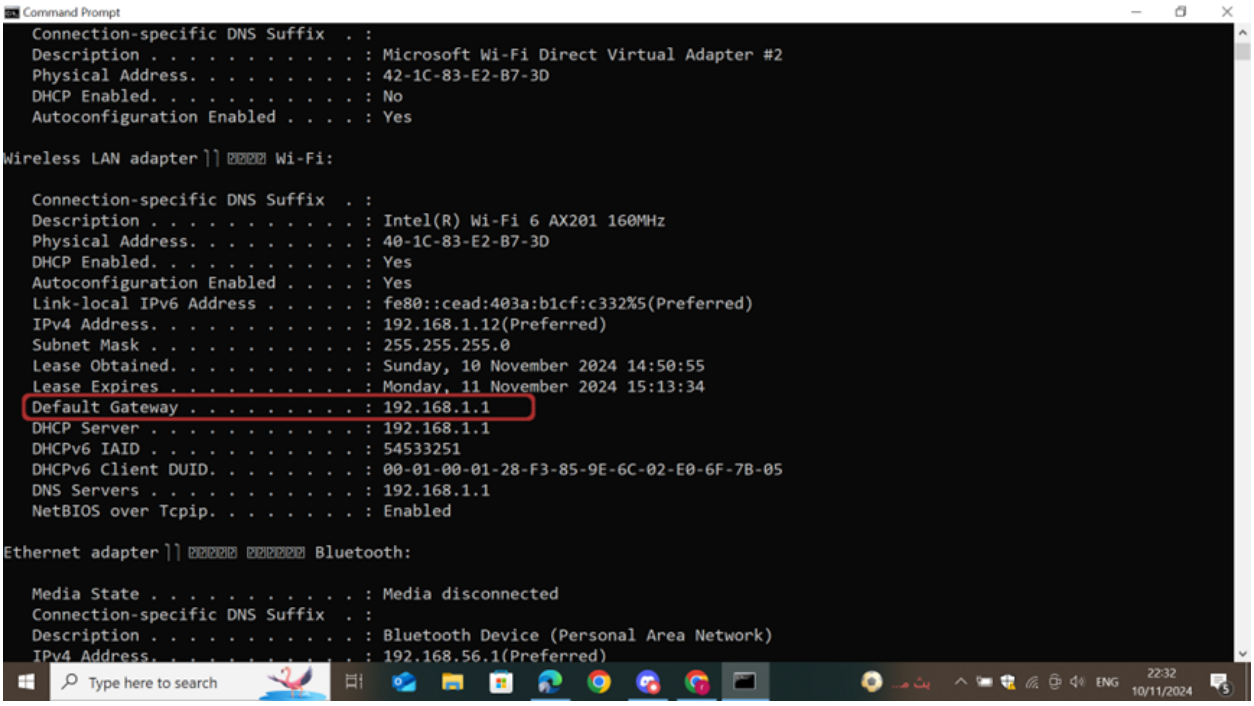


Figure 4 Default Gateway

4) Domain Name System (DNS) server addresses:

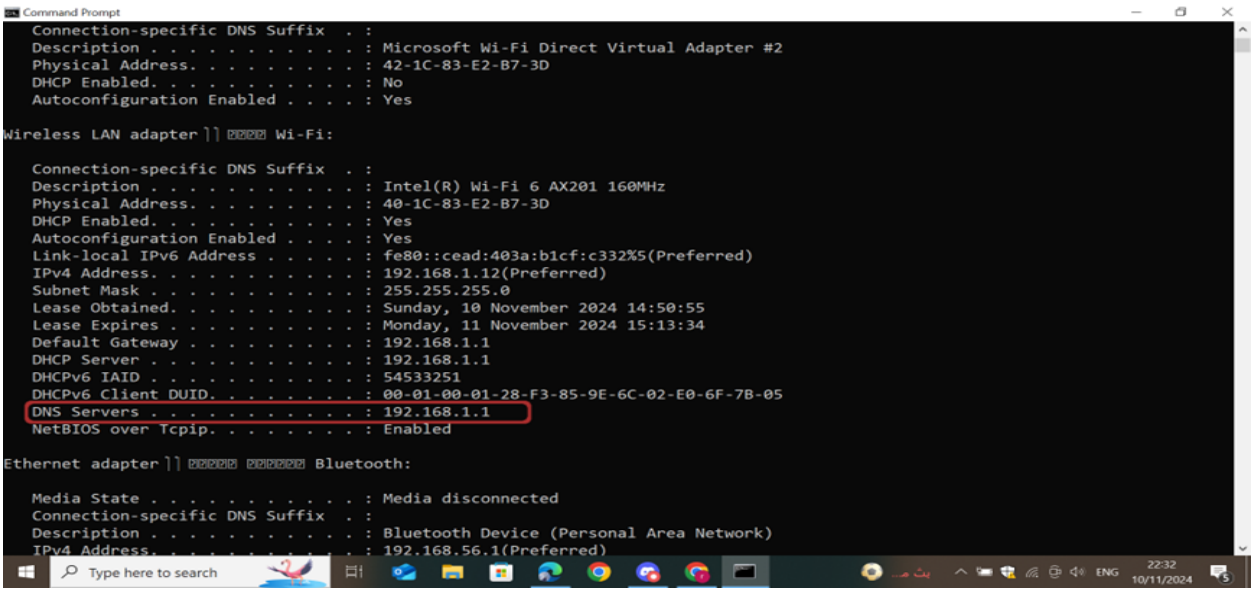


Figure 5) Domain Name System (DNS)

1.b.2 Ping a device within your local network (e.g., from your laptop to a smartphone on the same Wi-Fi network)

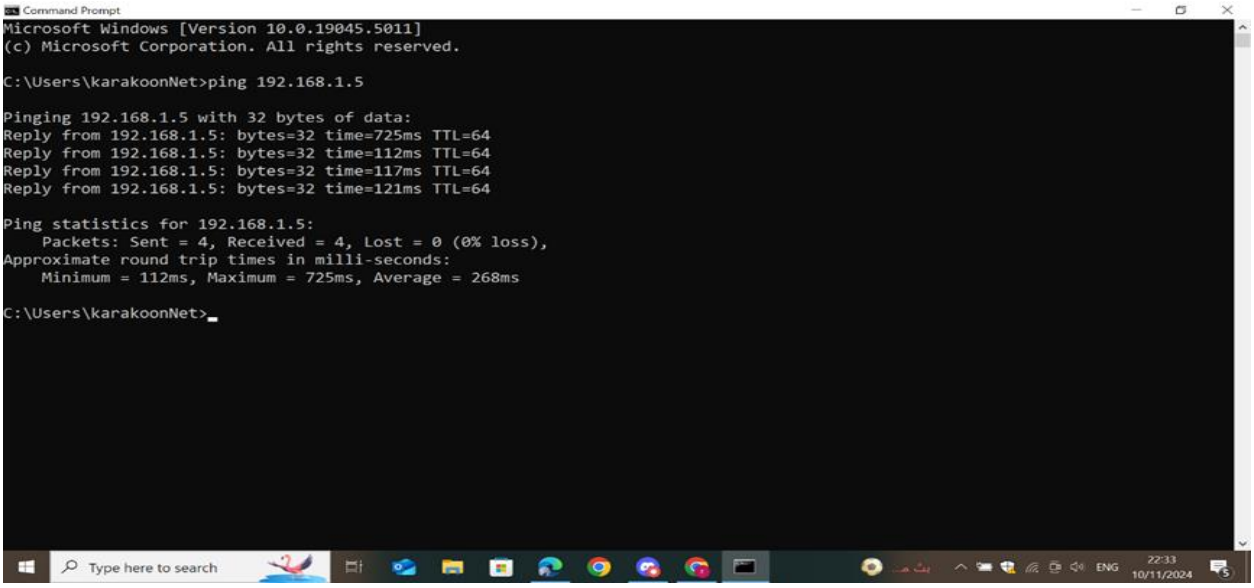


Figure 6 local network device Ping

The packet is specified with a size of 32 bytes and each request will contain 32 bytes of data payload. The line reply indicates that the destination device has received the request packet successfully, and a reply has been received from the specified IP address.

Number of packets sent and received is 4. Statics indicate that all four packets were successfully received from the device. Each line shows the details of every packet including the round trip time and TTL and no packet loss (percentage is 0%). Which means our connection was stable and responsive.

1.b.3 Ping discover.engineering.utoronto.ca. Based on the results, briefly explain whether you believe the response originates from Canada

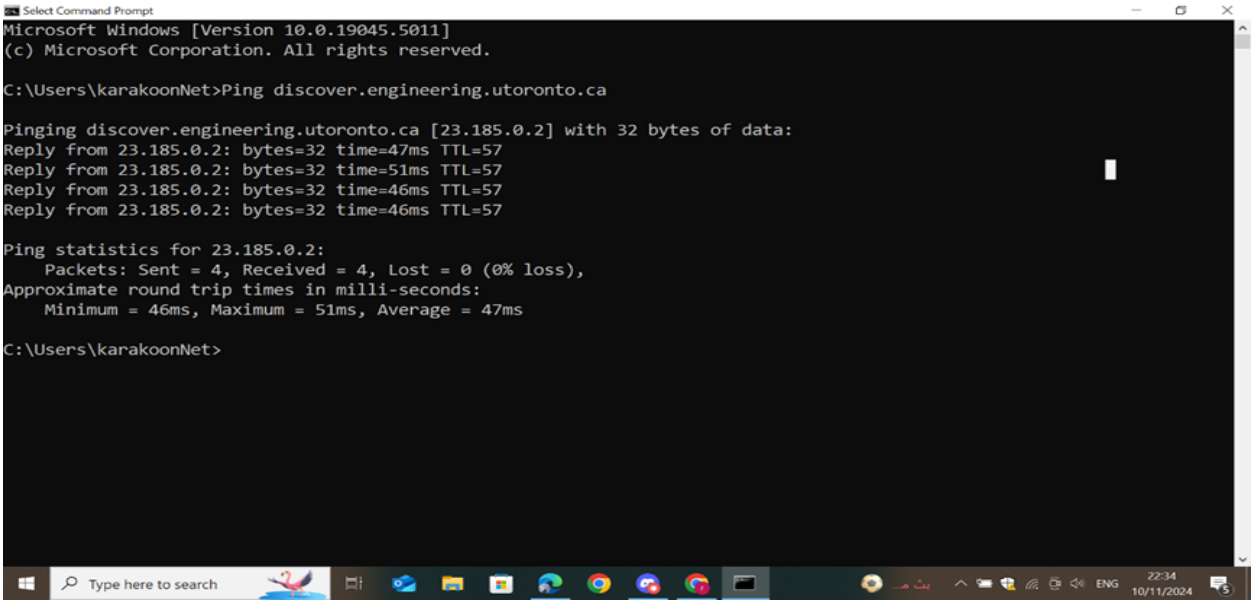


Figure 7 Ping discover.engineering.utoronto.ca.

The packet is specified with a size of 32 bytes and each request will contain 32 bytes of data payload. The line reply indicates that the destination device has received the request packet successfully, and a reply has been received from the specified IP address. TTL has an initial value of 57.

The output indicates that there was no loss, meaning that each packet request has had a corresponding packet response (This suggests a responsive and stable network connection to the specified domain). Number of packets sent and received is 4.

Based on the results, briefly explain whether you believe the response originates from Canada:

The response looks like it is from Canada.

The hostname discover.engineering.utoronto.ca strongly suggests a connection to the University of Toronto's Faculty of Applied Science and Engineering, which is located in Toronto, Canada. The .ca top-level domain is specifically designated for Canada.

Low Latency/TTL: While not definitive proof, the relatively low latency and initial TTL of 57 are consistent with a server located relatively nearby If the server were located on a different continent, you'd typically expect higher latency and a lower TTL due to the greater number of hops the packets would have to traverse.

But when we used the IP address online tool we found that the server is located in San Francisco California

Even though the hostname discover.engineering.utoronto.ca strongly suggests a Canadian location, the University of Toronto likely uses a Content Delivery Network (CDN) or cloud hosting provider with servers located around the world. CDNs improve website performance and availability by serving content from servers closer to users. In this case, the request was routed to a server in San Francisco, likely for better performance.

1.b.4 Run tracert on discover.engineering.utoronto.ca

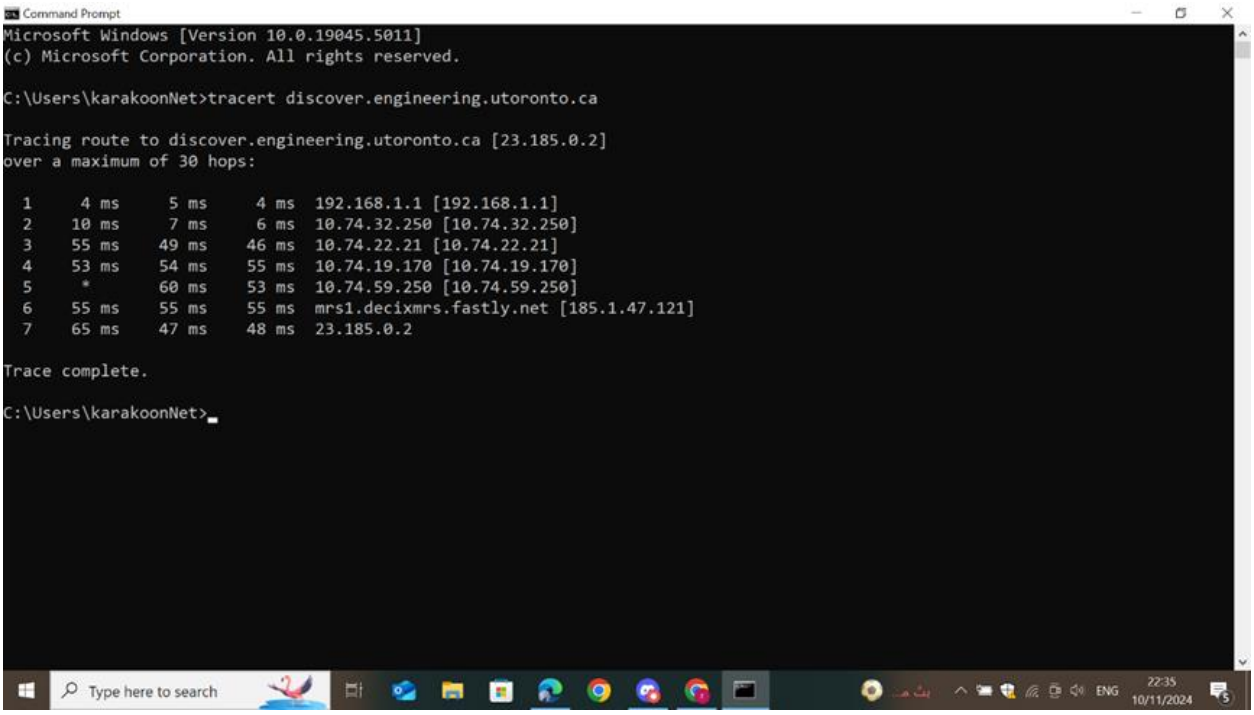


Figure 8 tracert on discover.engineering.utoronto.ca

A maximum number of hops or network devices the command will attempt to reach before stopping the trace is 30 hops. Each hop in the output shows the hop number, the round-trip time in milliseconds (ms) for packets to reach that hop and return, and the IP address of the router at that hop.

The first IP Address is my network IP address (192.168.1.1), and the last IP address is usually associated with the destination IP Address if the trace is successfully completed. The first hop is the local router at 192.168.1.1

Asterisks (*) means traceroute couldn’t access that router or unreachable due to the router’s configuration if it was set to reject ICMP packets.

“Trace completed” message indicates that the command has finished tracing the route, and the output provides information about the routers traversed from computer to reach discover.engineering.utoronto.ca

1.b.5 Use nslookup to retrieve the DNS information for discover.engineering.utoronto.ca

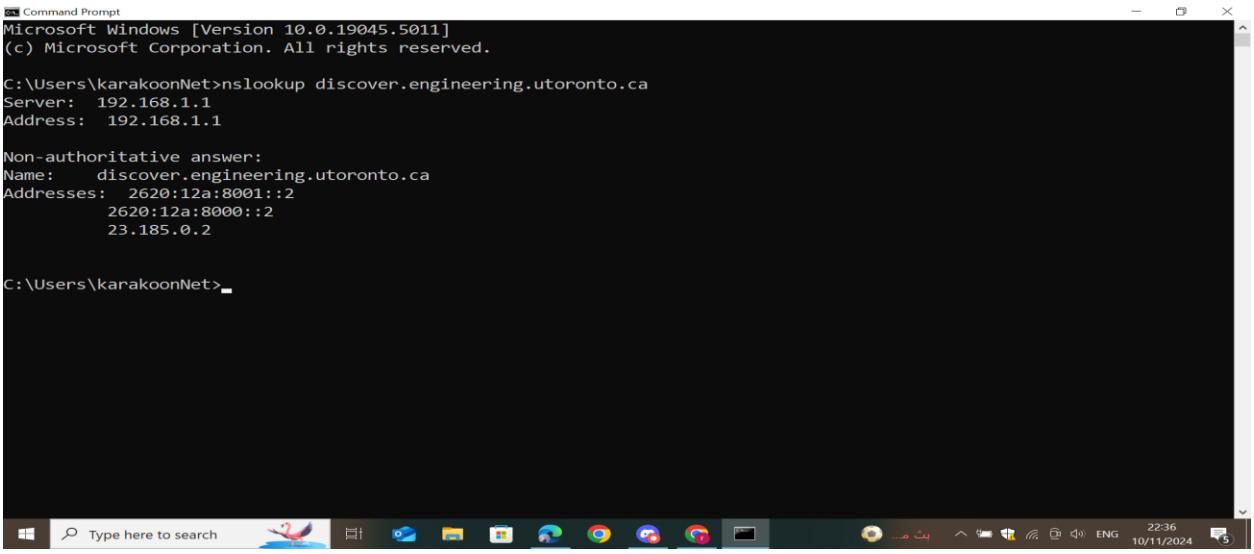


Figure 9 nslookup to retrieve the DNS

The output displays that the DNS server used for the query is at IP address 192.168.1.1. It also provides the IP addresses associated with the domain name "discover.engineering.utoronto.ca" which are "2620:12a:8001::2" and "2620:12a:8000::2" and "23.185.0.2"

The information is non-authoritative, meaning it's not directly from the authoritative DNS server for the domain but is obtained from a caching or intermediary DNS server.

1.b.6 Attempt to connect with telnet to discover.engineering.utoronto.ca

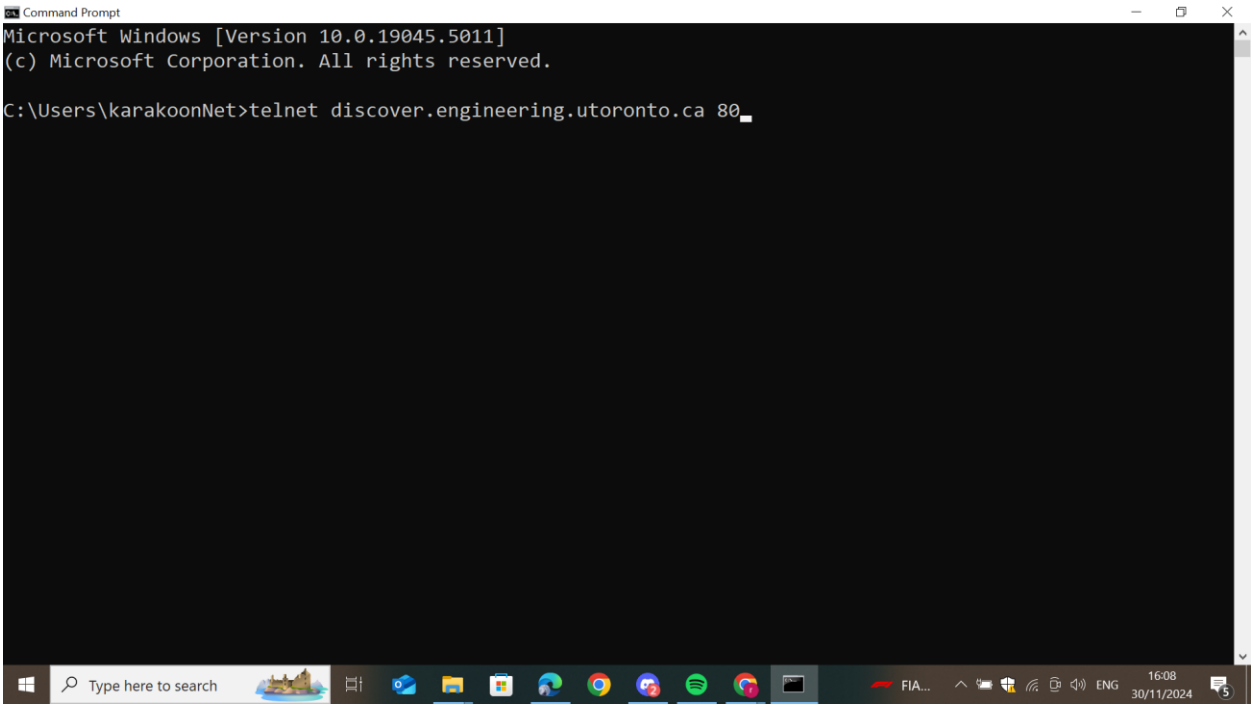
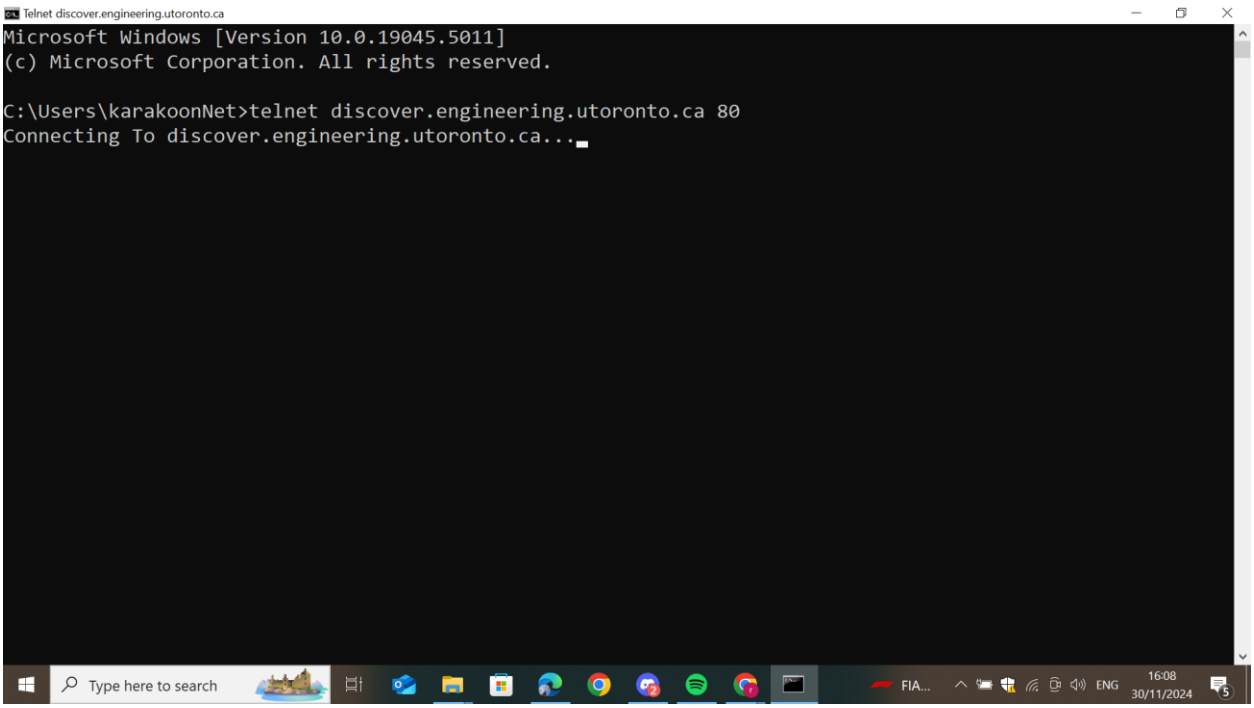
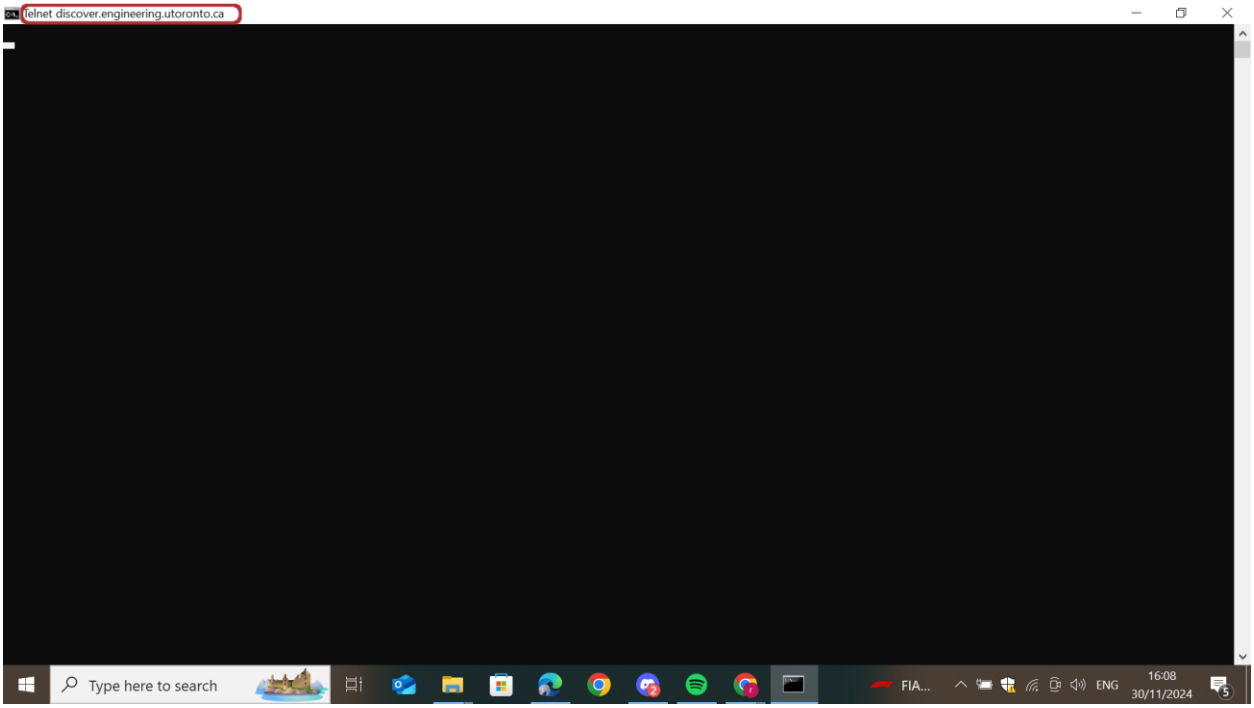
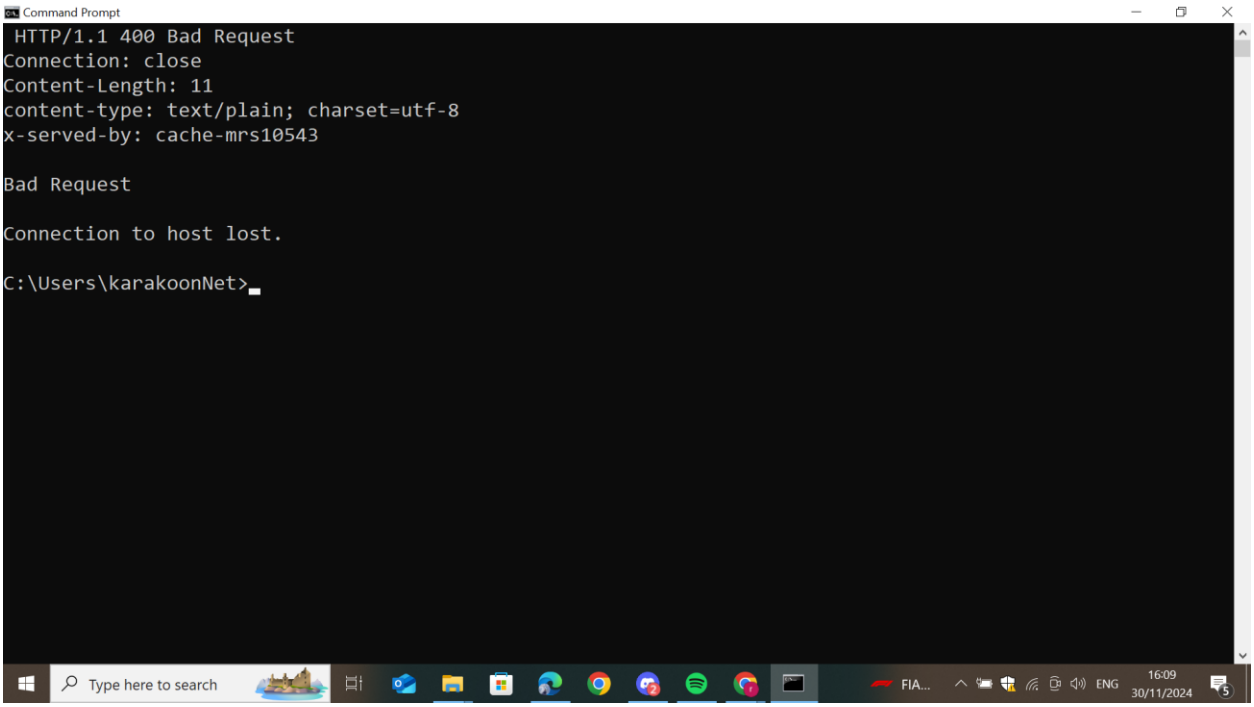


Figure 10 connect with telnet to discover.engineering.utoronto.ca




Telnet discover.engineering.utoronto.ca
Type here to search
FIA...
ENG
16:08
30/11/2024


Command Prompt
HTTP/1.1 400 Bad Request
Connection: close
Content-Length: 11
content-type: text/plain; charset=utf-8
x-served-by: cache-mrs10543
Bad Request
Connection to host lost.
C:\Users\karakoonNet>
Type here to search
FIA...
ENG
16:09
30/11/2024

We used the Telnet command to establish a connection to the server at discover.engineering.utoronto.ca on port 80, which is the standard port for HTTP communication.

After connecting, we sent a basic HTTP GET request:

The server responded with:

HTTP/1.1 400 Bad Request

1.b.7 Use the Wireshark packet analyzer to capture a DNS query and reply for any hostname of your choice.

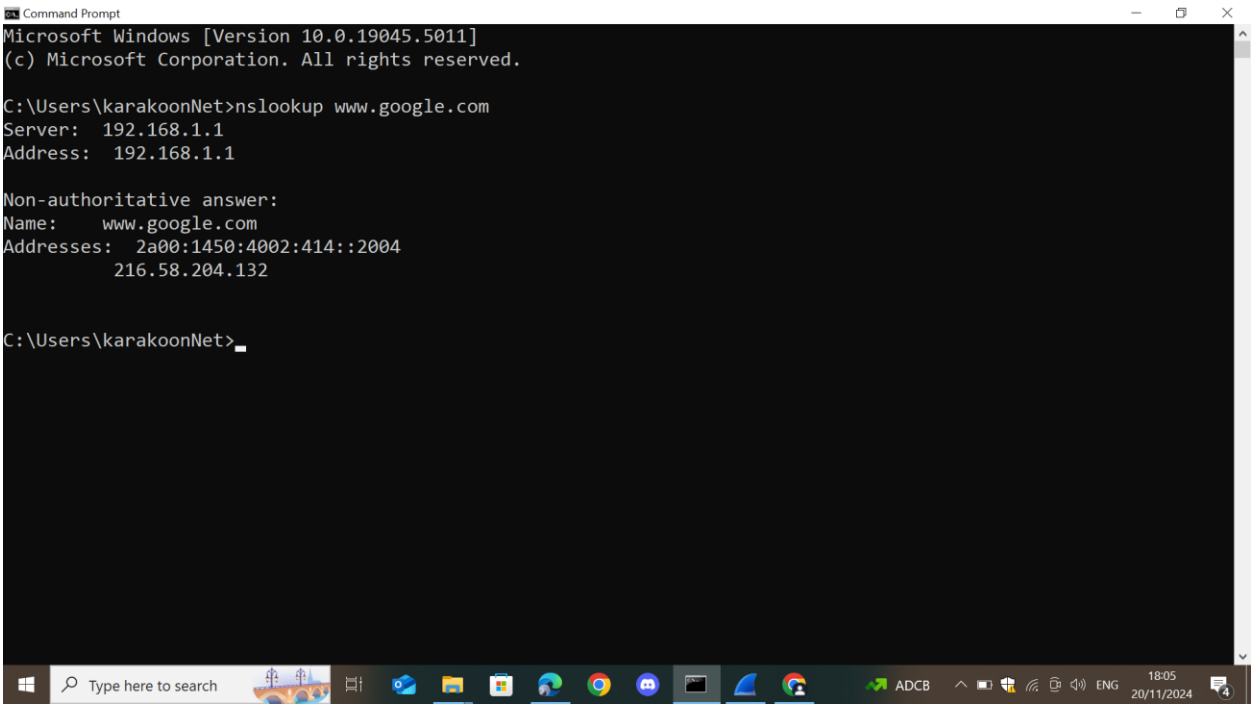
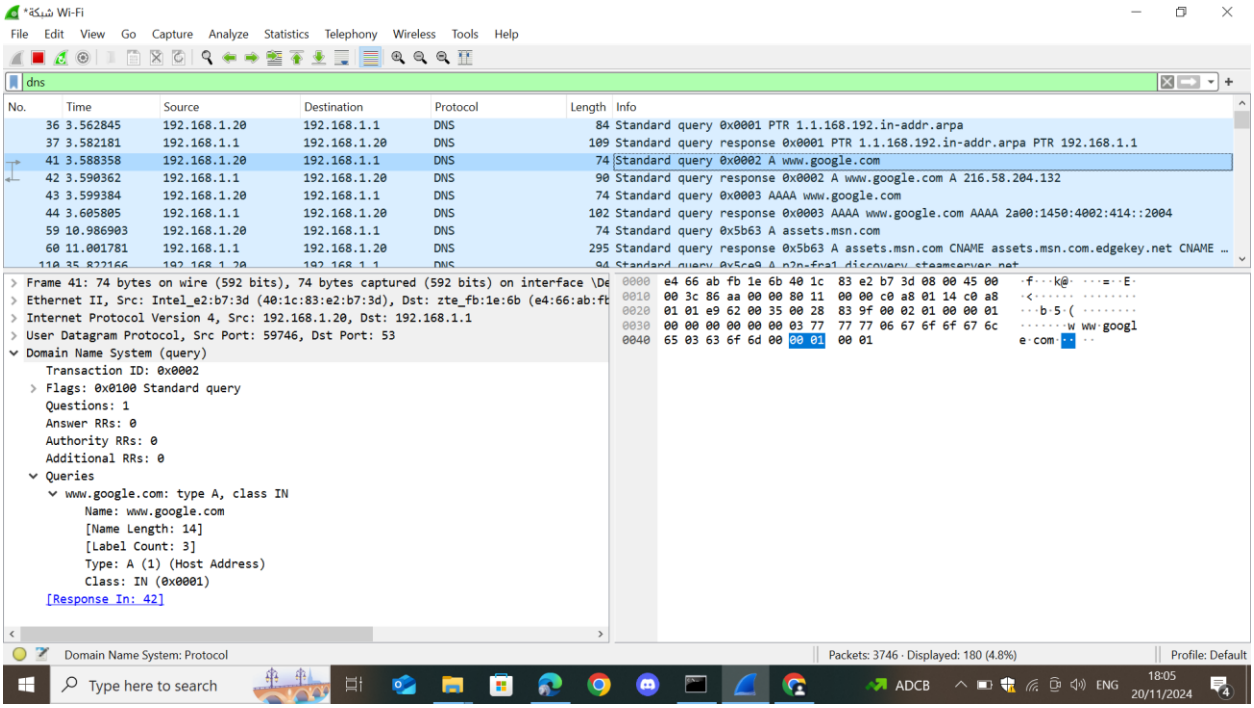
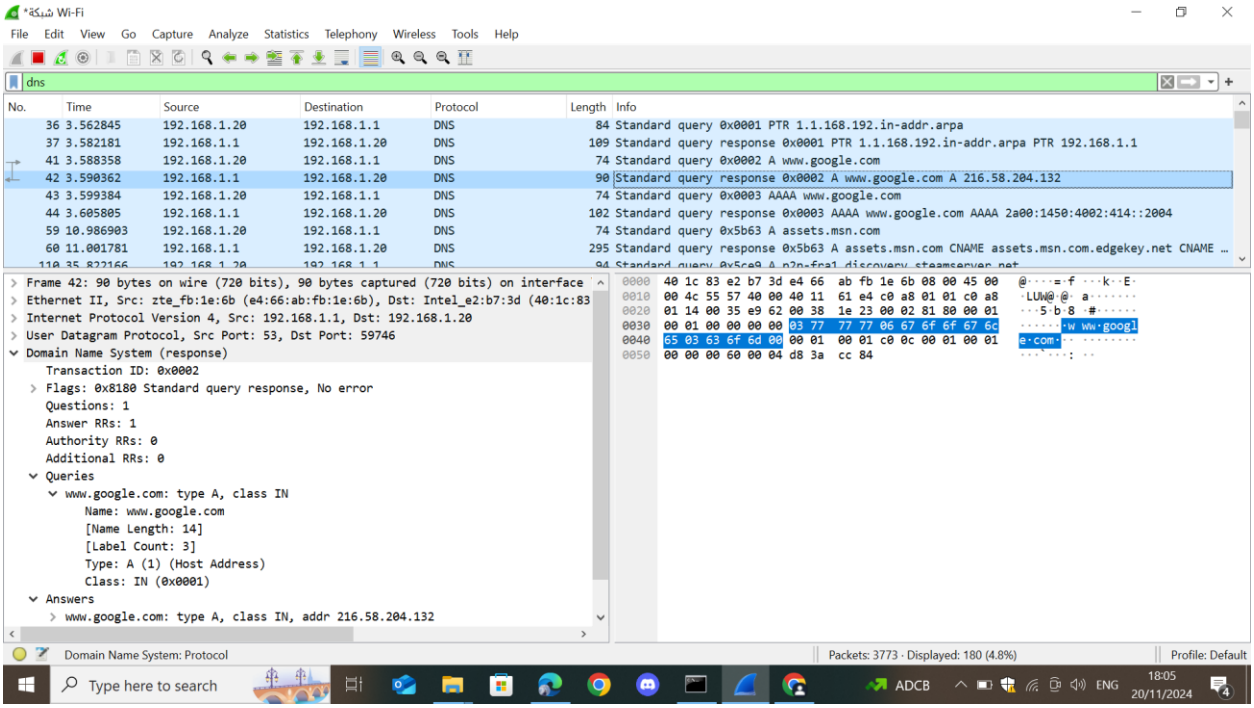


Figure 11 Use the Wireshark packet analyzer to capture a DNS query and reply for any hostname





We used nslookup to send packets and then WireShark to capture the DNS messages.

2 Part2 Web Server Results

Abstract:

This part presents the implementation of a simple HTTP server Python sockets. The server listens for connections and handles HTTP requests from clients. The server's goal is to serve requested files and redirect specific URLs.

Built with the socket module in Python, the server follows a loop-based structure, continuously listening for new connections. Upon accepting a connection, it parses the client's HTTP request to extract the requested URL path. Based on the path, the server determines the file to serve and sets the appropriate content type.

The server supports various file types like HTML, CSS, PNG, and JPG. It handles requests for different language versions of the main page and provides a 404 Not Found response for invalid paths or requests using a predefined "NotFound.html" file.

The implementation showcases error handling for file not found and I/O errors. Detailed status messages are printed during execution for debugging and verification.

2.1 Some screenshots of the web server running:

2.1.1 main_en.html page

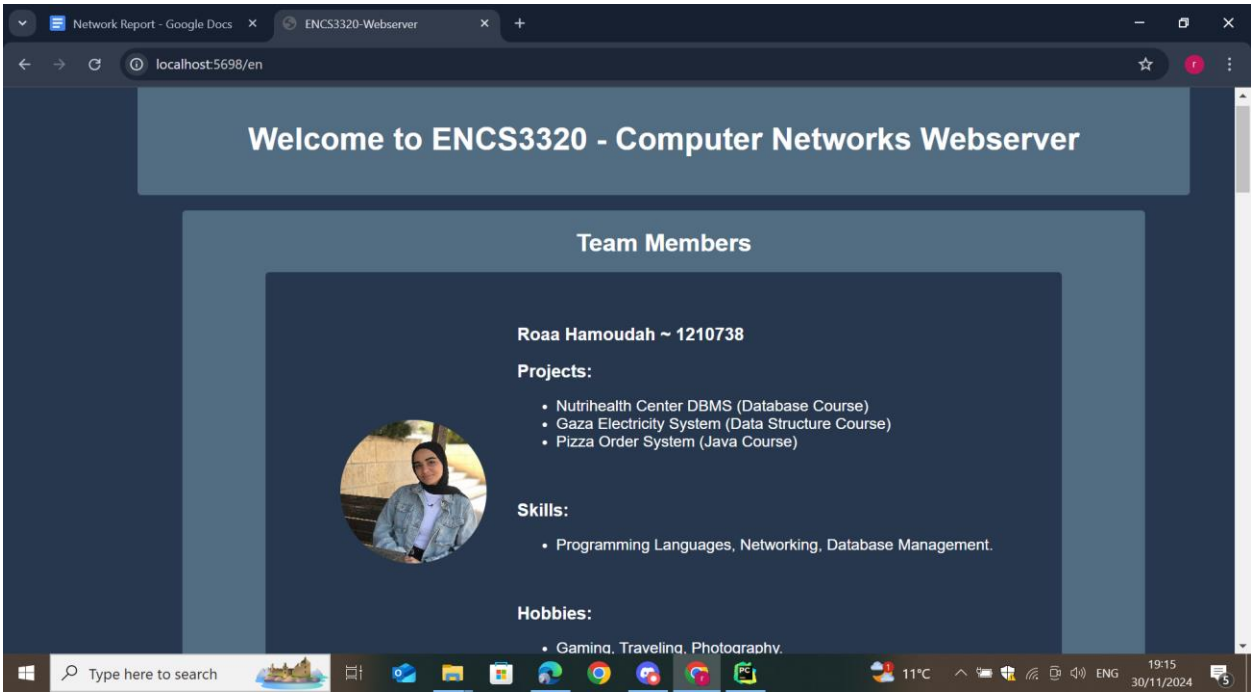
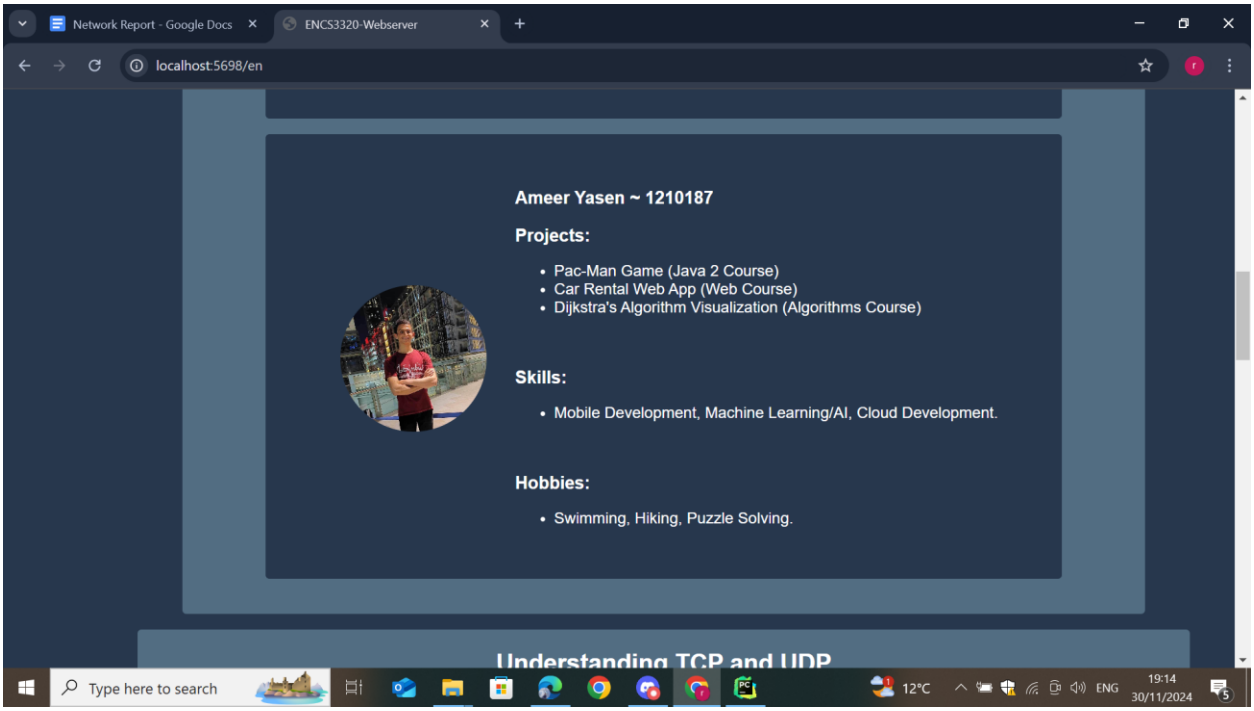


Figure 12 main_en.html page



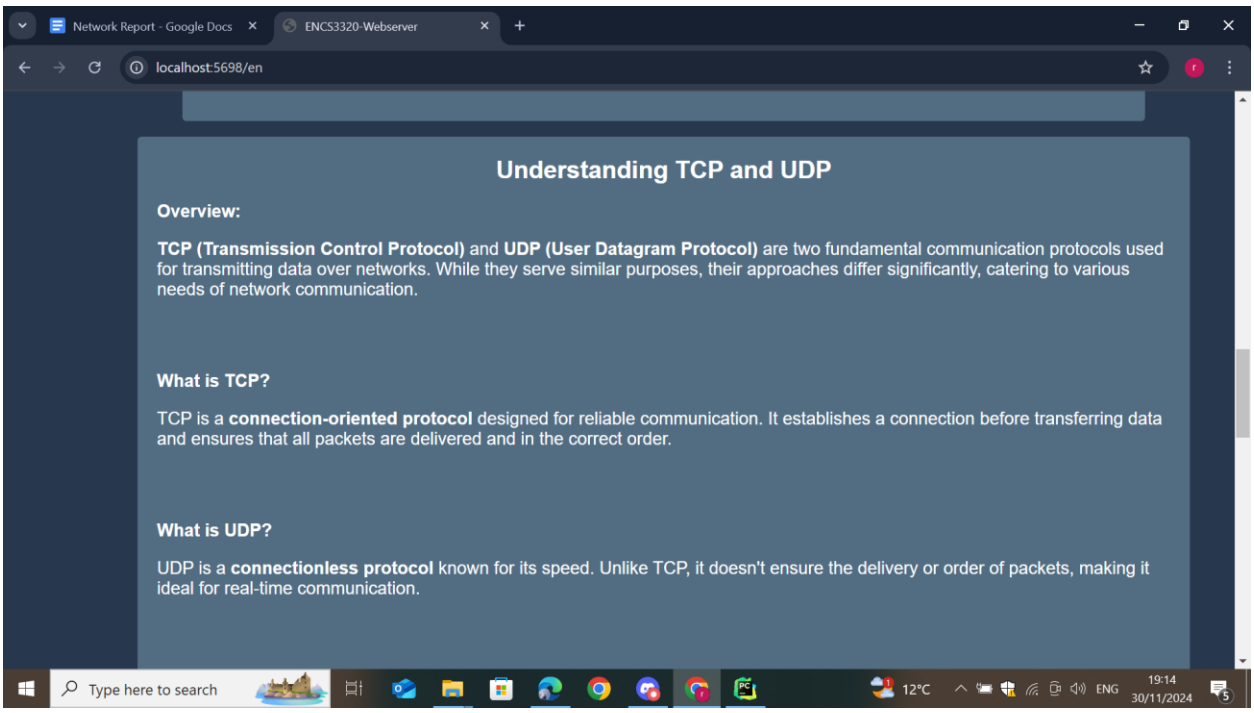
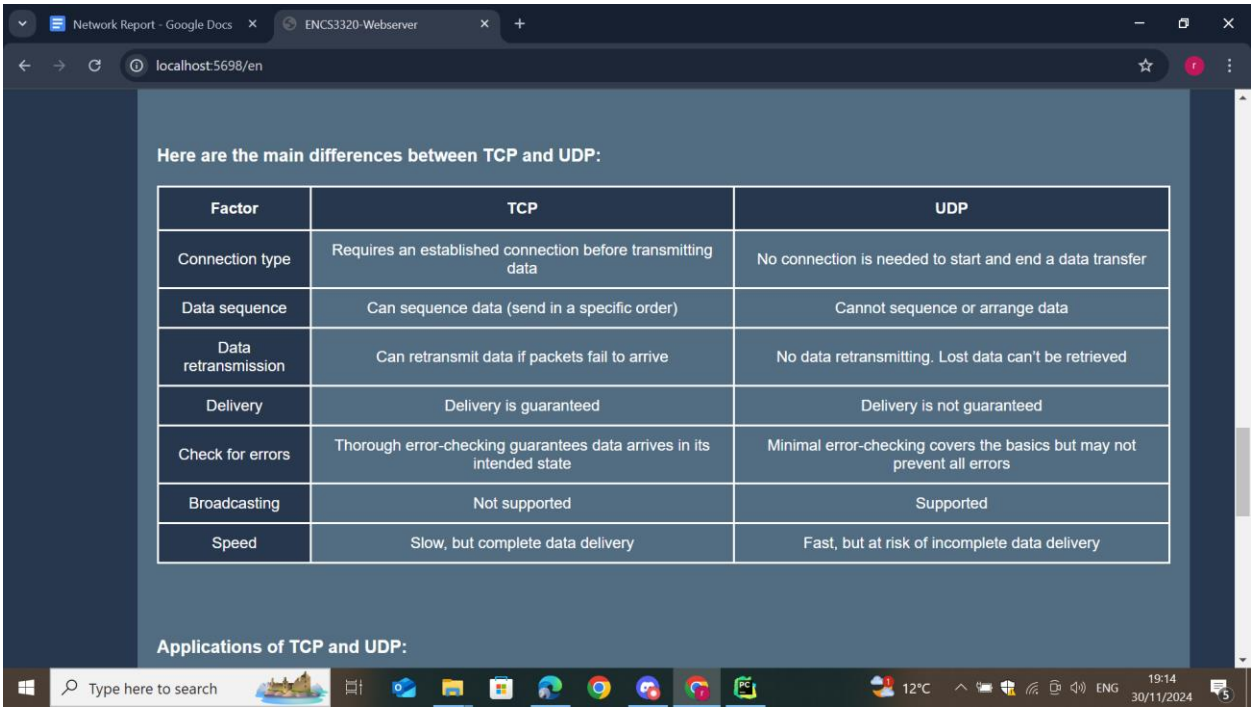


Figure 13 main_en.html page network topic



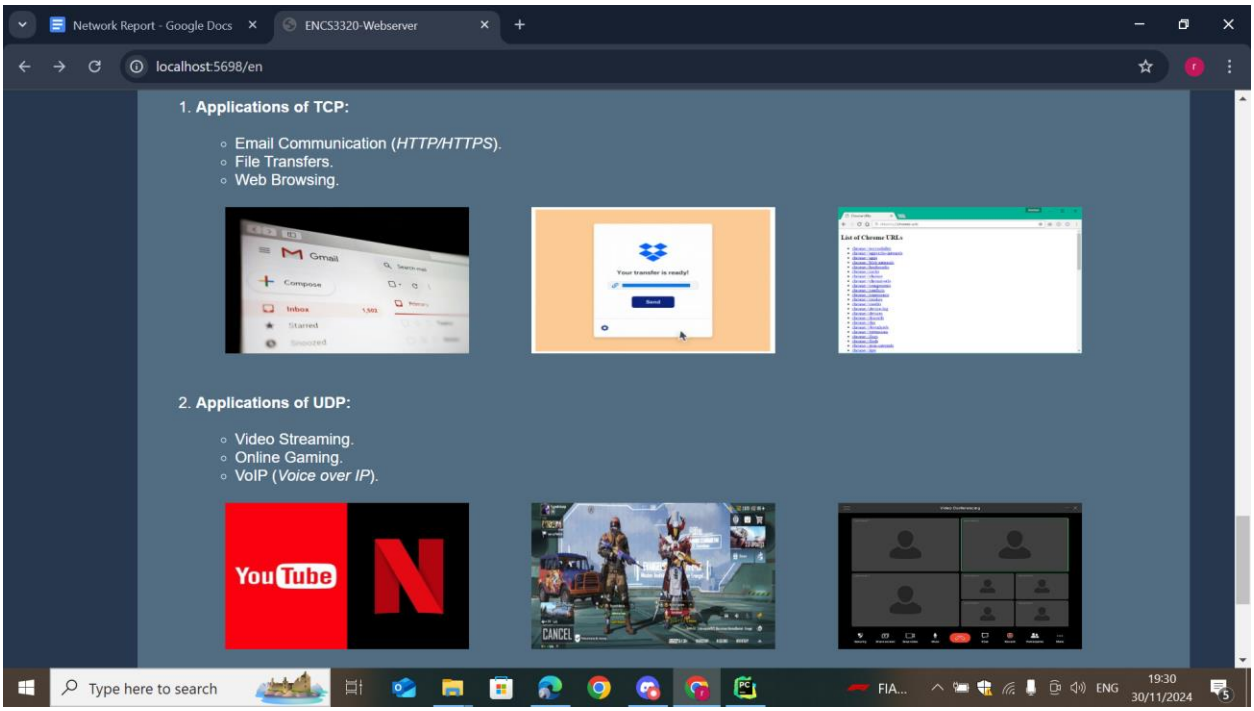


Figure 14 UDP and TCP Applications

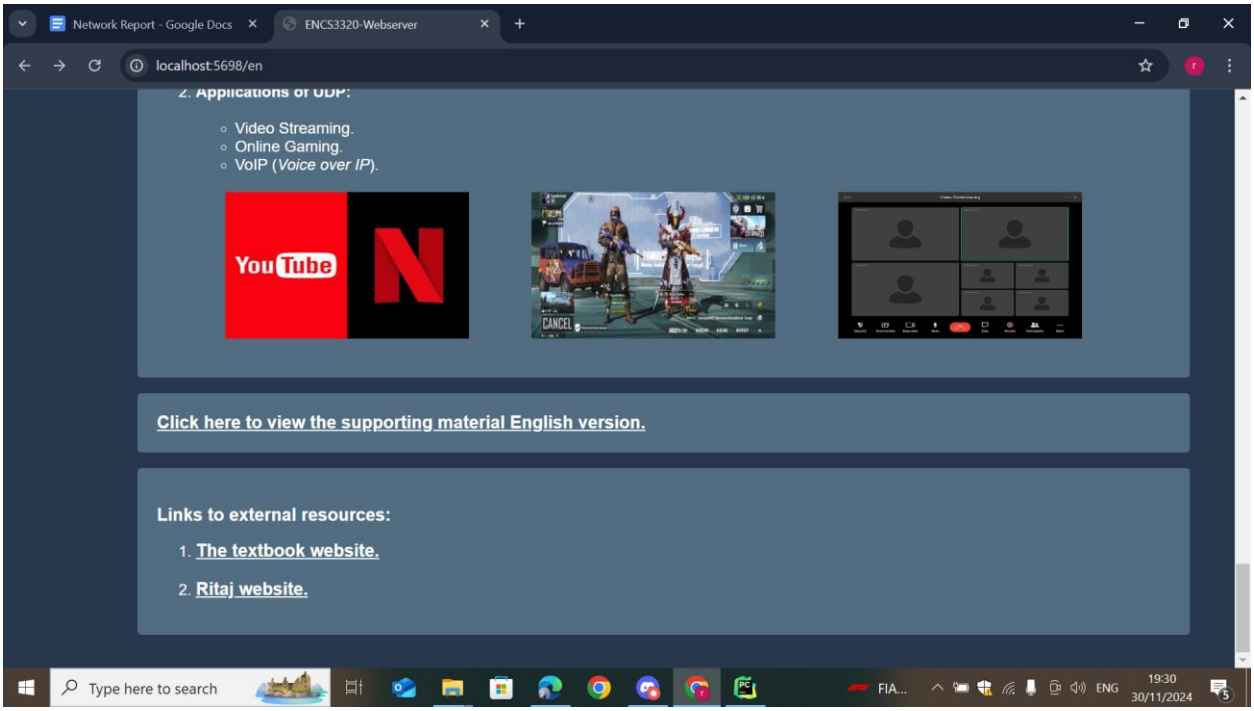
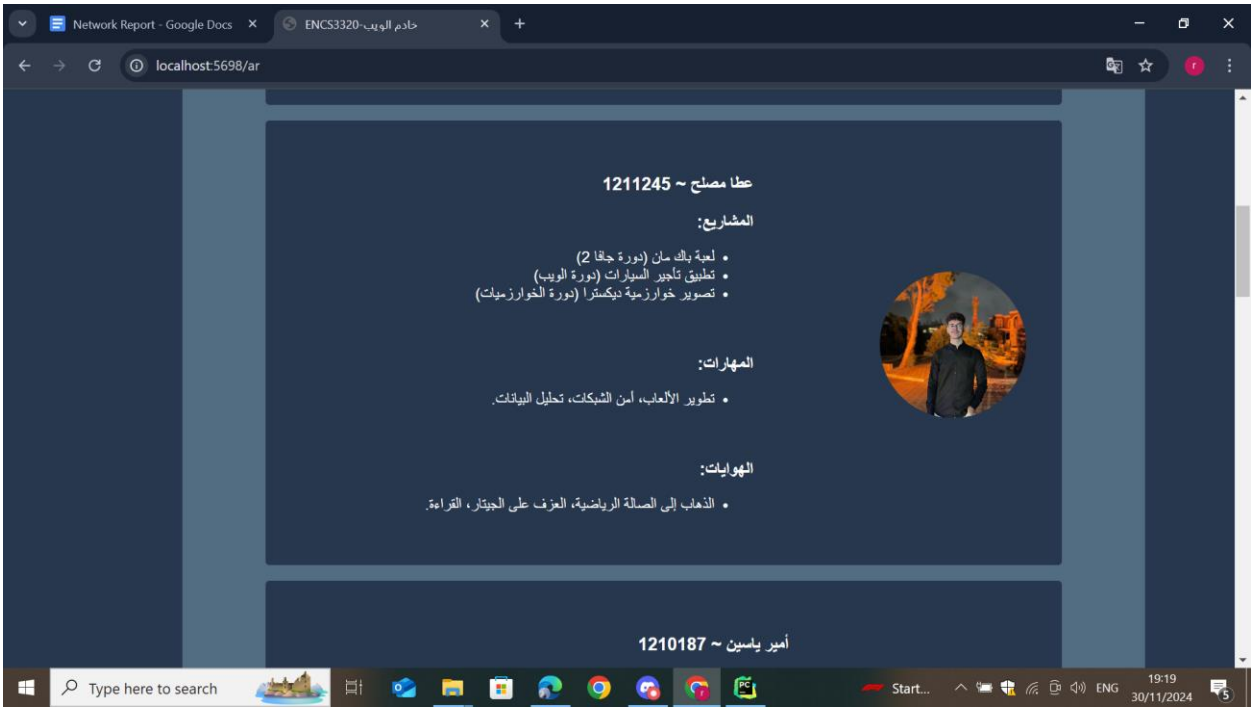
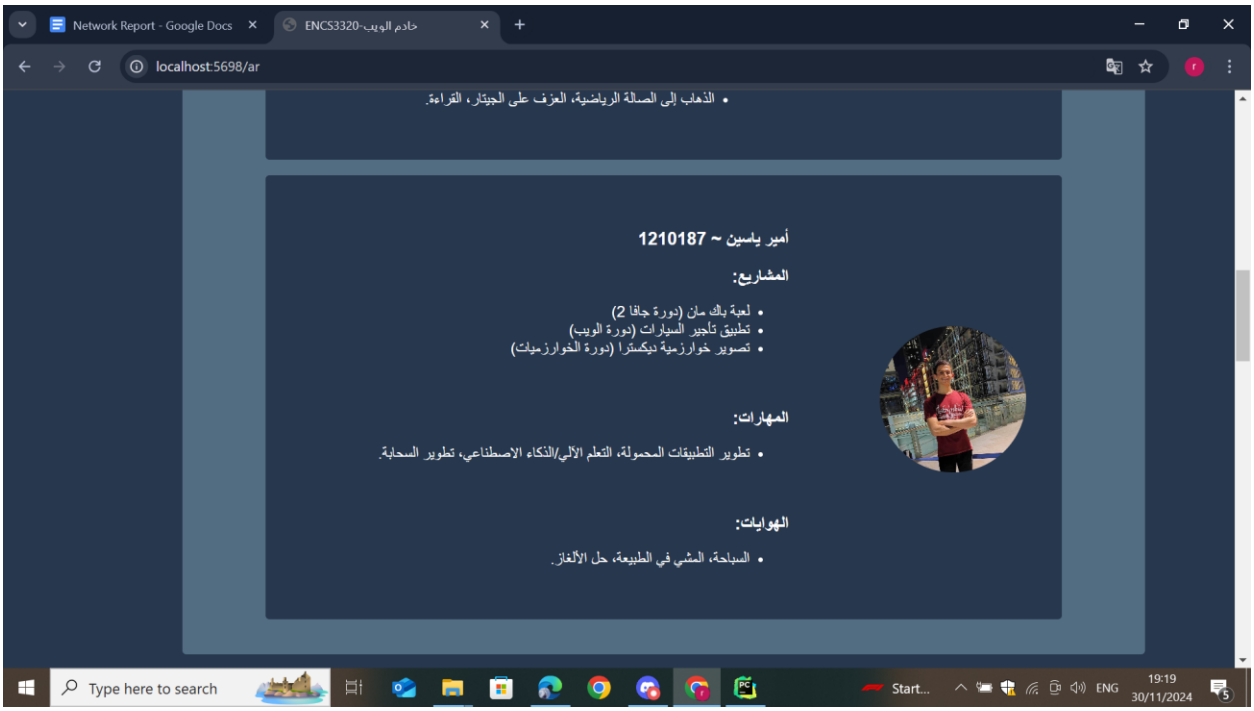


Figure 15 main_en.html page extra resources links

2.1.2 main_ar.html page

Figure 16 main_er.html Page


Network Report - Google Docs
ENCS3320-خادم الويب
localhost:5698/ar
عطا مصلح ~ 1211245
المشاريع:
لعبة باك مان (دورة جافا 2)
تطبيق تأجير السيارات (دورة الويب)
تصوير خوارزمية ديكسترا (دورة الخوارزميات)
المهارات:
تطوير الألعاب، أمن الشبكات، تحليل البيانات.
الهوايات:
الذهاب إلى الصالة الرياضية، العزف على الجيتار، القراءة.
أمير ياسين ~ 1210187
Type here to search
Start...
ENG
19:19
30/11/2024


Network Report - Google Docs
ENCS3320-خادم الويب
localhost:5698/ar
الذهاب إلى الصالة الرياضية، العزف على الجيتار، القراءة.
أمير ياسين ~ 1210187
المشاريع:
لعبة باك مان (دورة جافا 2)
تطبيق تأجير السيارات (دورة الويب)
تصوير خوارزمية ديكسترا (دورة الخوارزميات)
المهارات:
تطوير التطبيقات المحمولة، التعلم الآلي/الذكاء الاصطناعي، تطوير السحابة.
الهوايات:
السباحة، المشي في الطبيعة، حل الألغاز.
Type here to search
Start...
ENG
19:19
30/11/2024


Network Report - Google Docs
ENCS3320-خادم الويب
localhost:5698/ar
فهم بروتوكول TCP و UDP
نظرة عامة:
بروتوكول التحكم في الإرسال (TCP) و بروتوكول بيانات المستخدم (UDP) هما بروتوكولان أساسيان للتواصل يُستخدمان لنقل البيانات عبر الشبكات. على الرغم من أنهما يخدمان أغراضًا مشابهة، إلا أن أسلوبهما يختلف بشكل كبير، مما يلبي احتياجات مختلفة في التواصل الشبكي.
ما هو بروتوكول TCP؟
بروتوكول TCP هو بروتوكول موجه نحو الاتصال مصمم للتواصل الموثوق. يقوم بإنشاء اتصال قبل نقل البيانات ويضمن أن جميع الحزم يتم تسليمها وبالترتيب الصحيح.
ما هو بروتوكول UDP؟
بروتوكول UDP هو بروتوكول غير موجه نحو الاتصال معروف بسرعته. على عكس بروتوكول TCP، فإنه لا يضمن تسليم الحزم أو ترتيبها، مما يجعله مثاليًا للتواصل في الوقت الحقيقي.
إليك الاختلافات الرئيسية بين بروتوكول TCP و UDP:
Type here to search
Start...
ENG
19:19
30/11/2024


Network Report - Google Docs
ENCS3320-خادم الويب
localhost:5698/ar
إليك الاختلافات الرئيسية بين بروتوكول TCP و UDP:
العامل | TCP | UDP
نوع الاتصال | يتطلب اتصالاً مُؤسسًا قبل نقل البيانات | لا حاجة لاتصال لبدء وإنهاء نقل البيانات
ترتيب البيانات | يمكنه ترتيب البيانات (إرسالها بترتيب محدد) | لا يمكنه ترتيب أو تنظيم البيانات
إعادة إرسال البيانات | يمكن إعادة إرسال البيانات إذا فشلت الحزم في الوصول | لا توجد إعادة إرسال للبيانات. البيانات المفقودة لا يمكن استردادها
التسليم | التسليم مضمون | التسليم غير مضمون
التحقق من الأخطاء | التحقق الشامل من الأخطاء يضمن وصول البيانات في حالتها المقصودة | التحقق الأدنى من الأخطاء يغطي الأساسيات ولكن قد لا يمنع جميع الأخطاء
البث | غير مدعوم | مدعوم
السرعة | بطيء، ولكن تسليم البيانات كامل | سريع، ولكن مع خطر تسليم بيانات غير كاملة
تطبيقات بروتوكول TCP و UDP:
1. تطبيقات بروتوكول TCP:
Type here to search
Start...
ENG
19:19
30/11/2024

Network Report - Google Docs | خادم الويب-ENCS3320 | localhost:5698/ar

تطبيقات بروتوكول TCP و UDP:

1. **تطبيقات بروتوكول TCP:**
   - التواصل عبر البريد الإلكتروني (*HTTP/HTTPS*).
   - نقل الملفات.
   - تصفح الويب.


2. **تطبيقات بروتوكول UDP:**
   - بث الفيديو.
   - الألعاب عبر الإنترنت.
   - VoIP (*الصوت عبر بروتوكول الإنترنت*).


Type here to search | Wat... | ENG | 19:32 | 30/11/2024

Network Report - Google Docs | خادم الويب-ENCS3320 | localhost:5698/ar

2. **تطبيقات بروتوكول UDP:**
   - بث الفيديو.
   - الألعاب عبر الإنترنت.
   - VoIP (*الصوت عبر بروتوكول الإنترنت*).


انقر هنا لعرض المواد الداعمة باللغة العربية.

روابط لمصادر خارجية:

1. موقع الكتاب.
2. موقع ريتاج.

Type here to search | 11°C | ENG | 19:32 | 30/11/2024

2.1.3 supporting_material_en.html Page

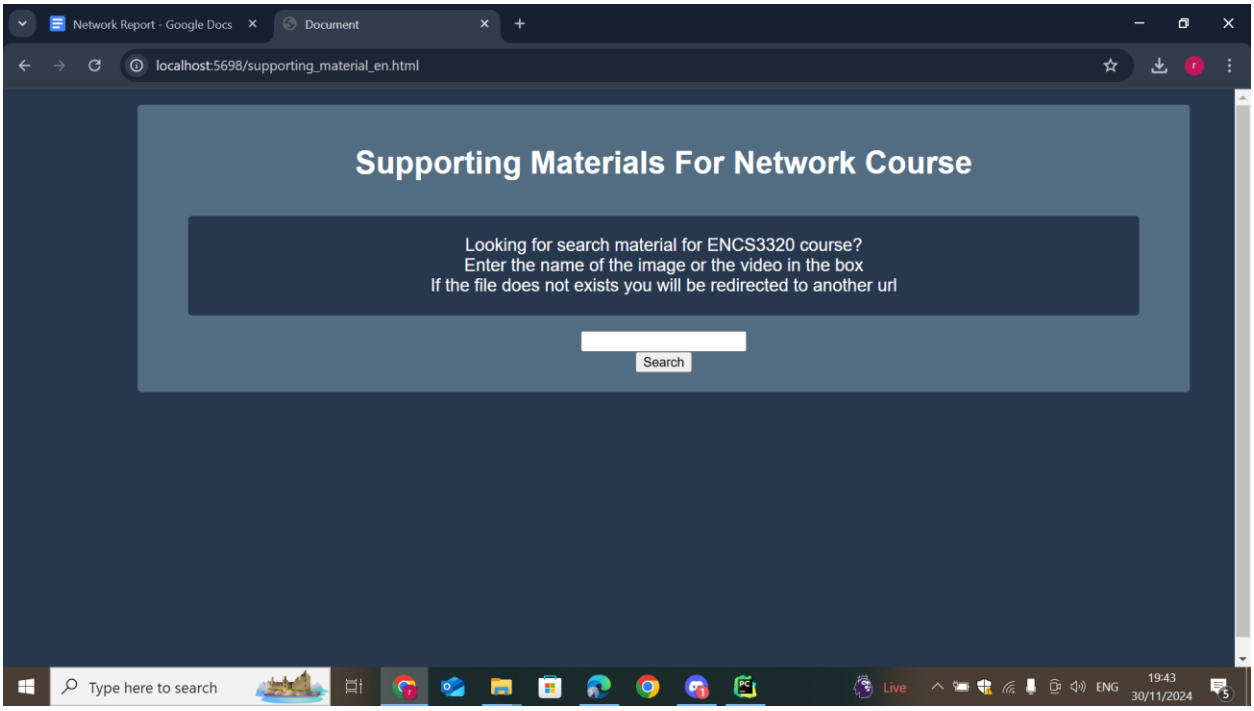


Figure 17 supporting_material_en.html Page

2.1.3.1 Request an existing image

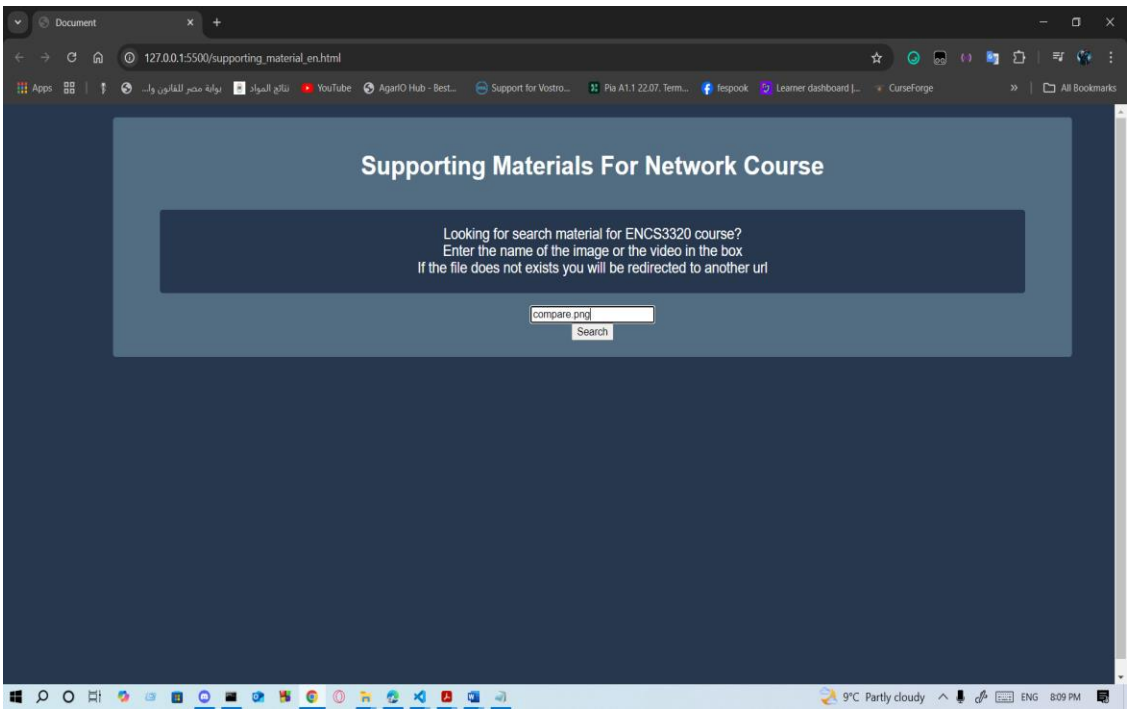
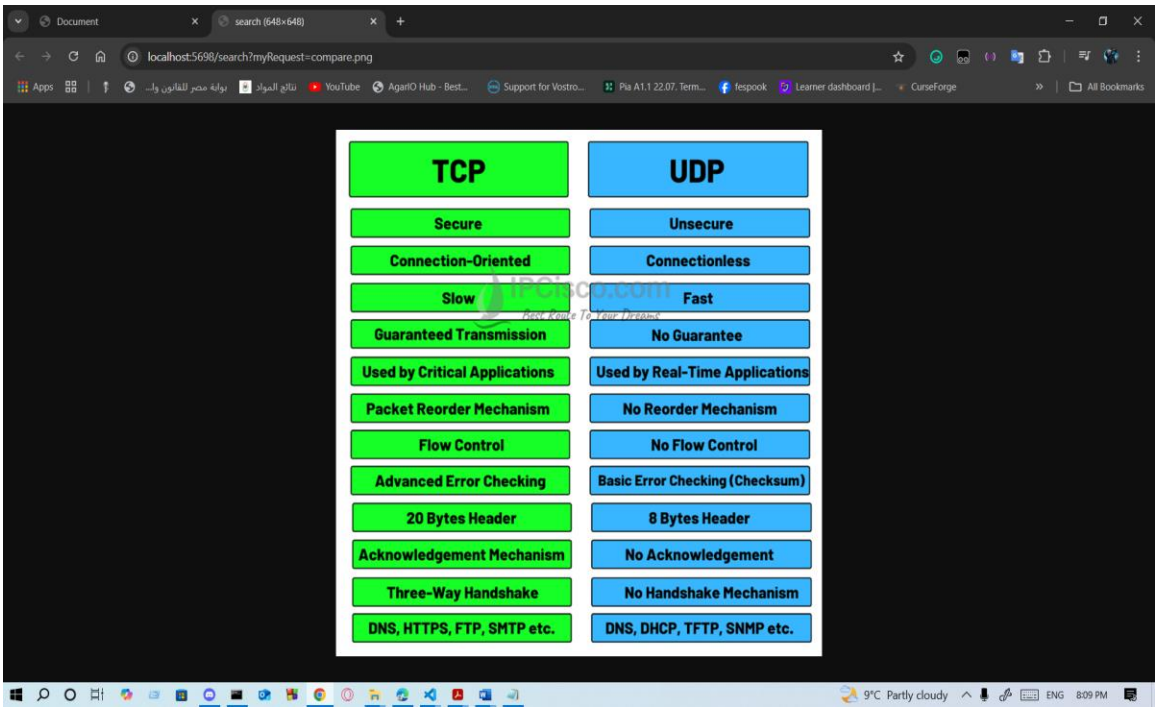


Figure 18 Request an existing image



2.1.3.2 Request a non-existing image

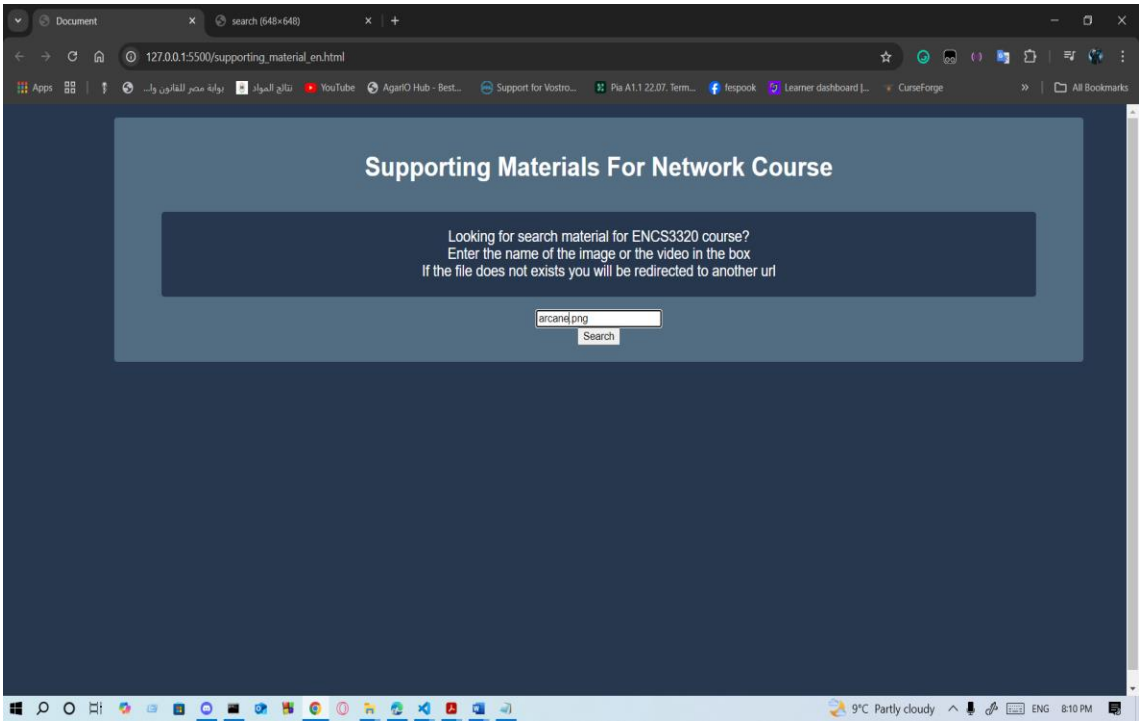
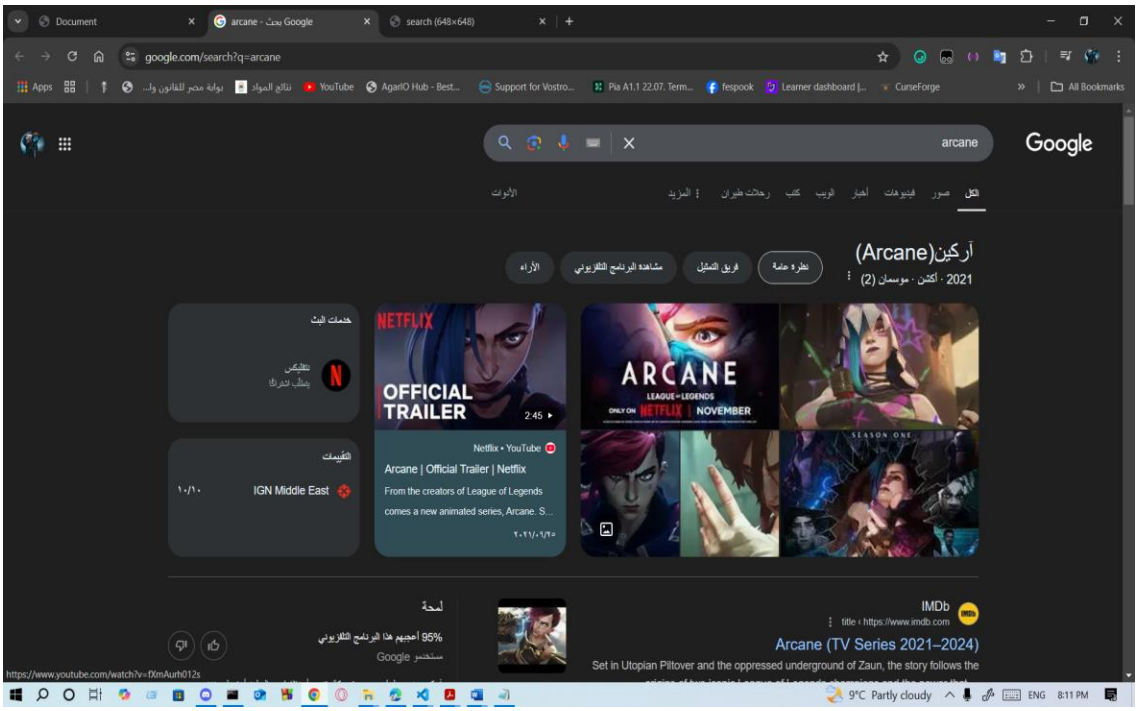


Figure 19 Request a non-existing image



HTTP status code 307 Temporary Redirect indicates that the resource being requested has been temporarily moved to a different URL. The client should resubmit the original request to the new URL provided in the Location header of the response.

In the last image case it should look like this:

```
HTTP/1.1 307 Temporary Redirect
Date: Sun, 1 Dec 2024 4:21:00 GMT
Server: Apache/2.4.52 (Ubuntu)  (Example server)
Location: https://www.google.com/search?q=arcane
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: close
```

Figure 20 HTTP status code 307 Temporary Redirect

That the server response header and the new location redirect to the client browser know that the resource location changed so it sends

another request to the new location that is in the response header. The original request must not be changed when resubmitting the request to the new location. This preserves the intended action of the original request. 307 redirects are often used when a server wants to temporarily move a resource without changing the semantics of the client's interaction with it.

2.1.3.3 Request an existing video

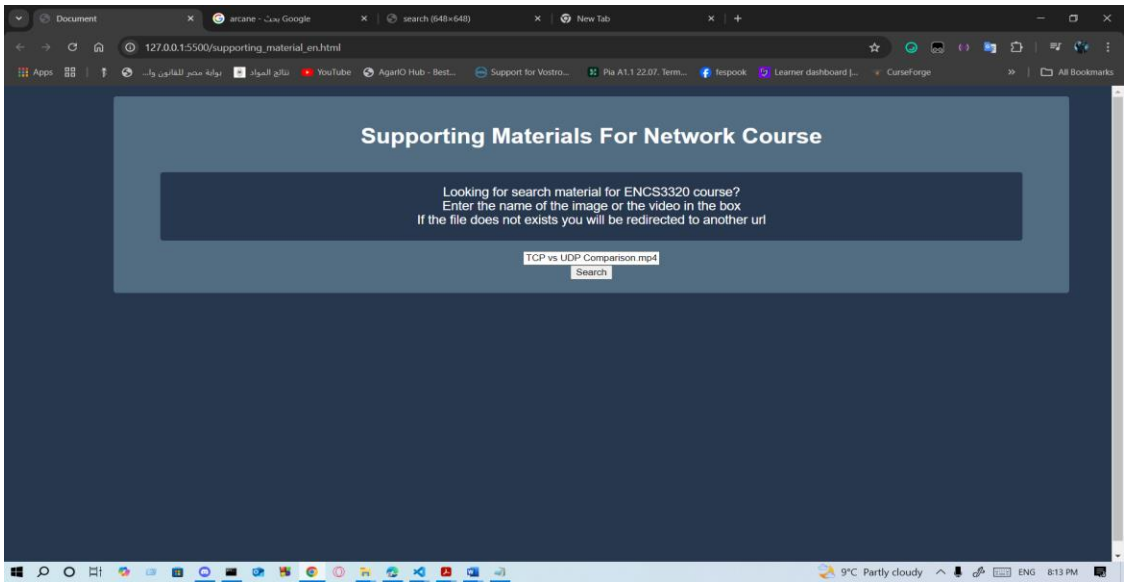
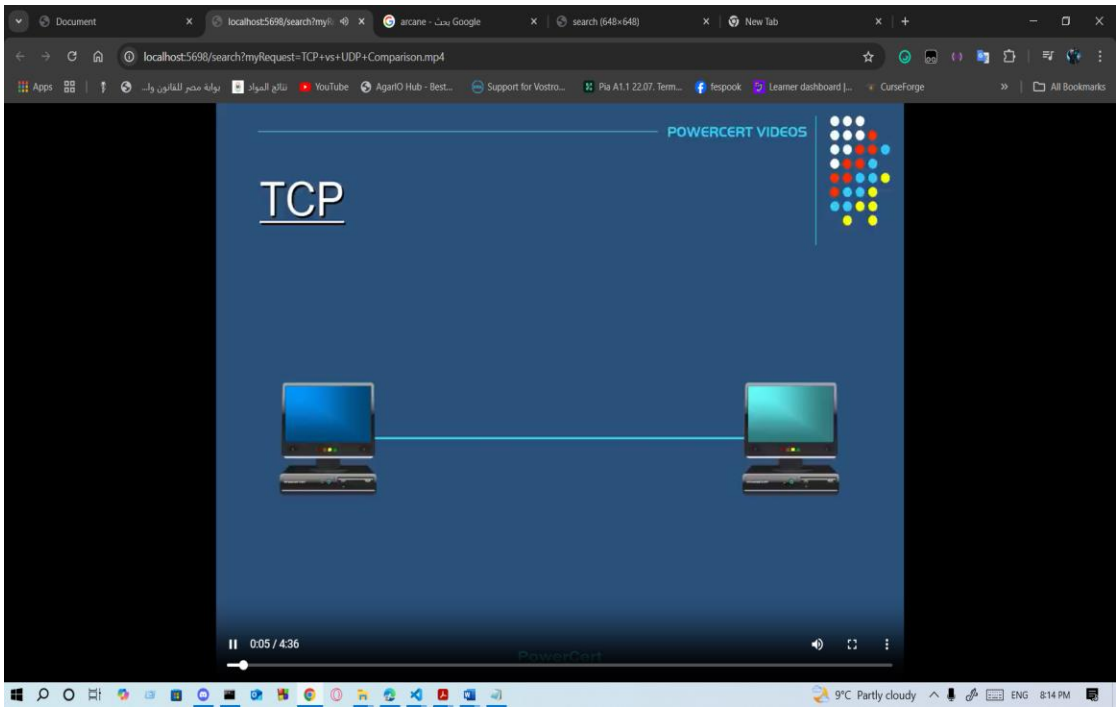


Figure 21 Request an existing video



2.1.3.4 Request a non-existing video

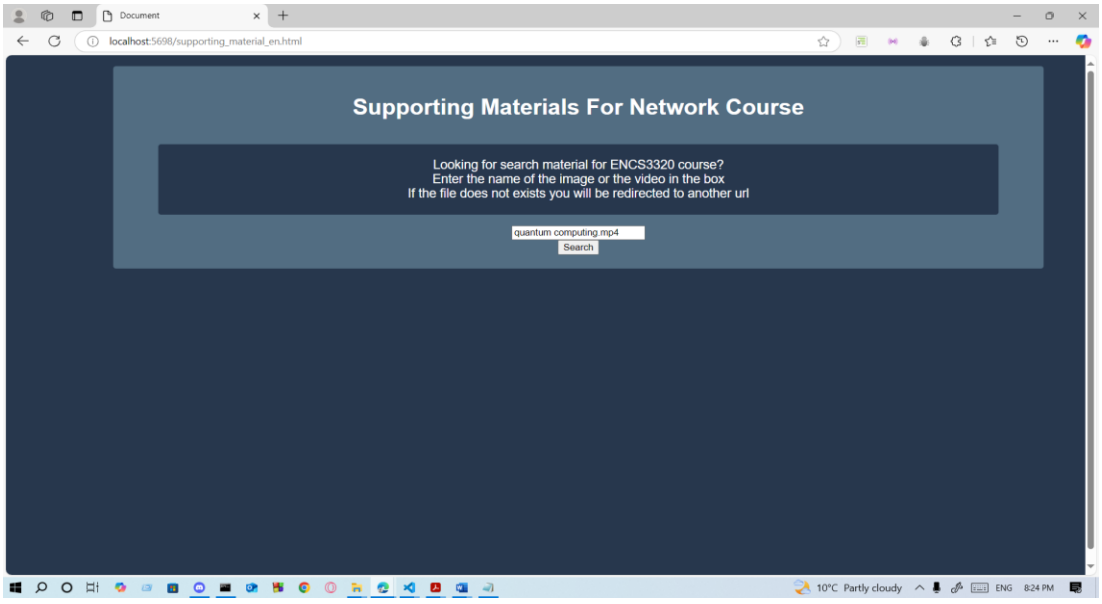
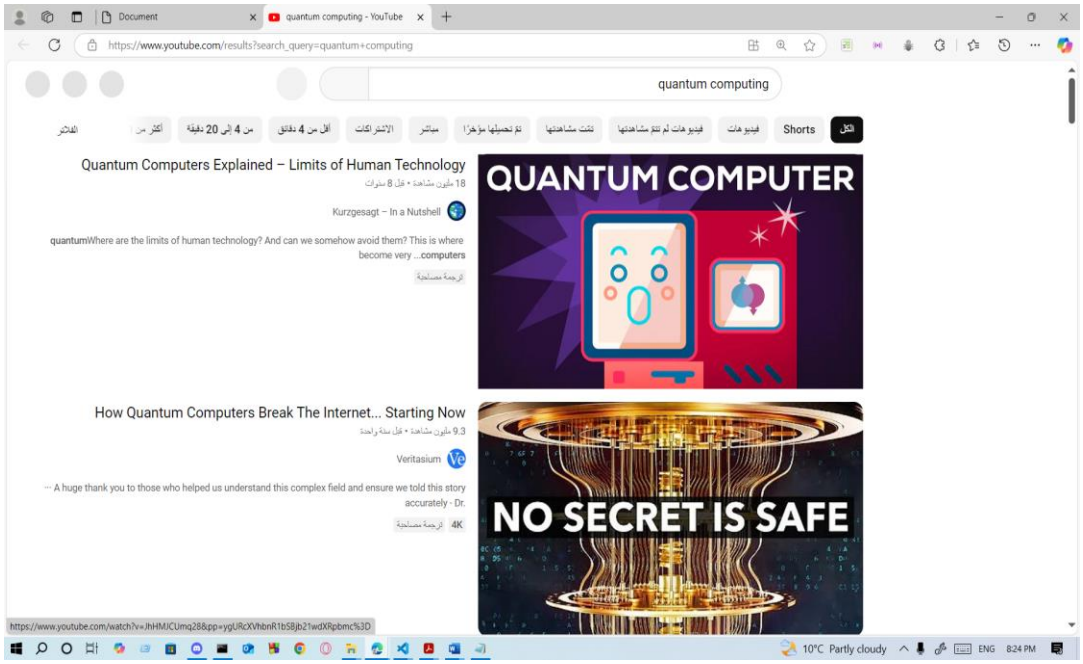


Figure 22 Request a non-existing video



Redirect to a YouTube search URL that filters results to videos based on the user's input.

2.1.4 supporting_material_ar.html Page

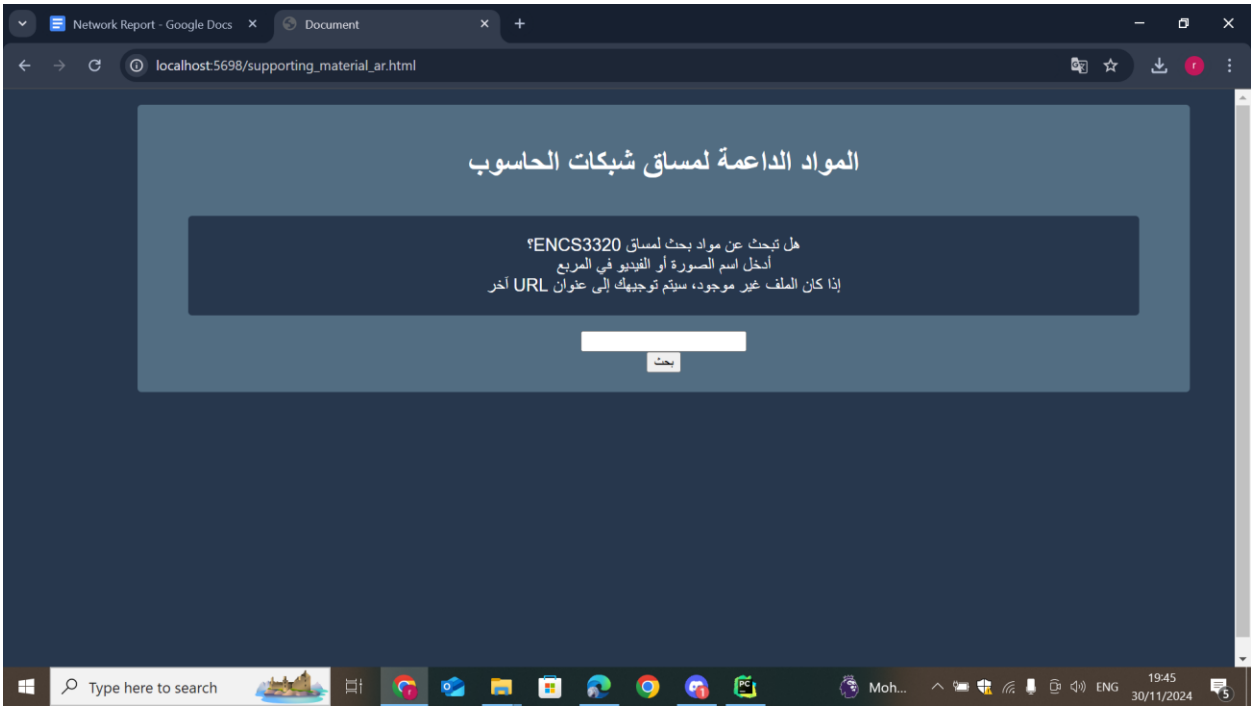


Figure 23 supporting_material_ar.html Page

2.1.5 HTTP Requests printed one the server terminal

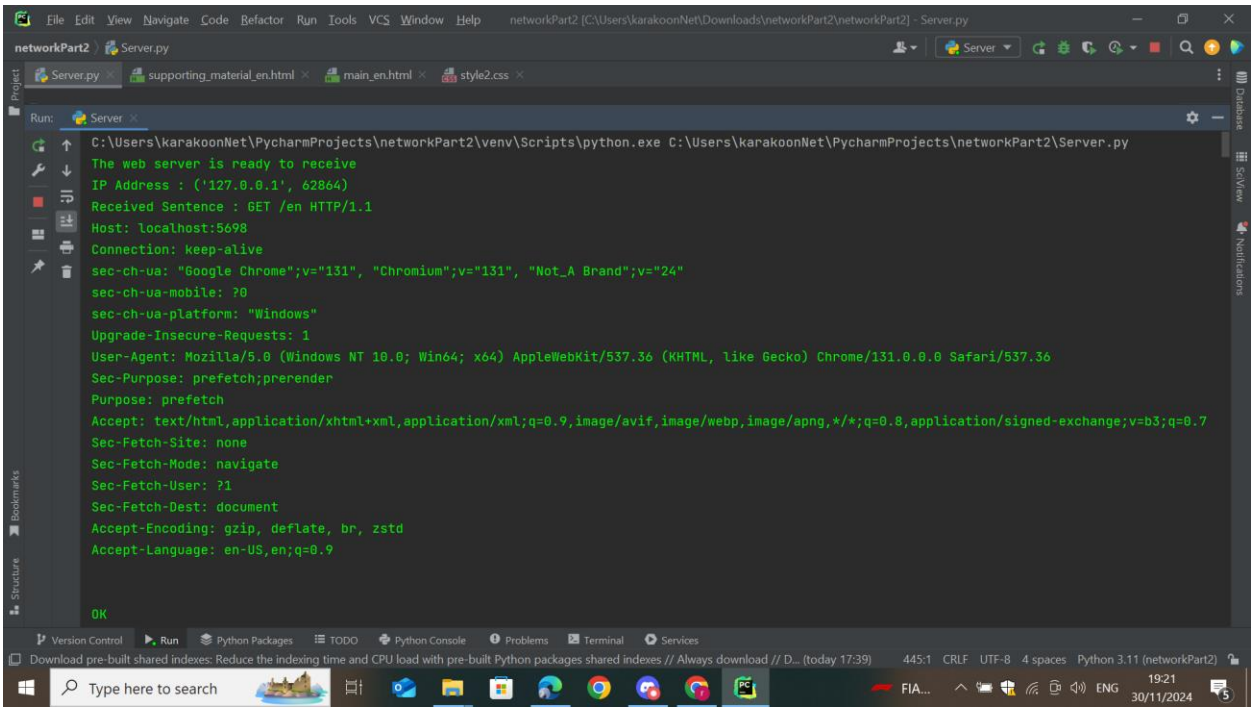
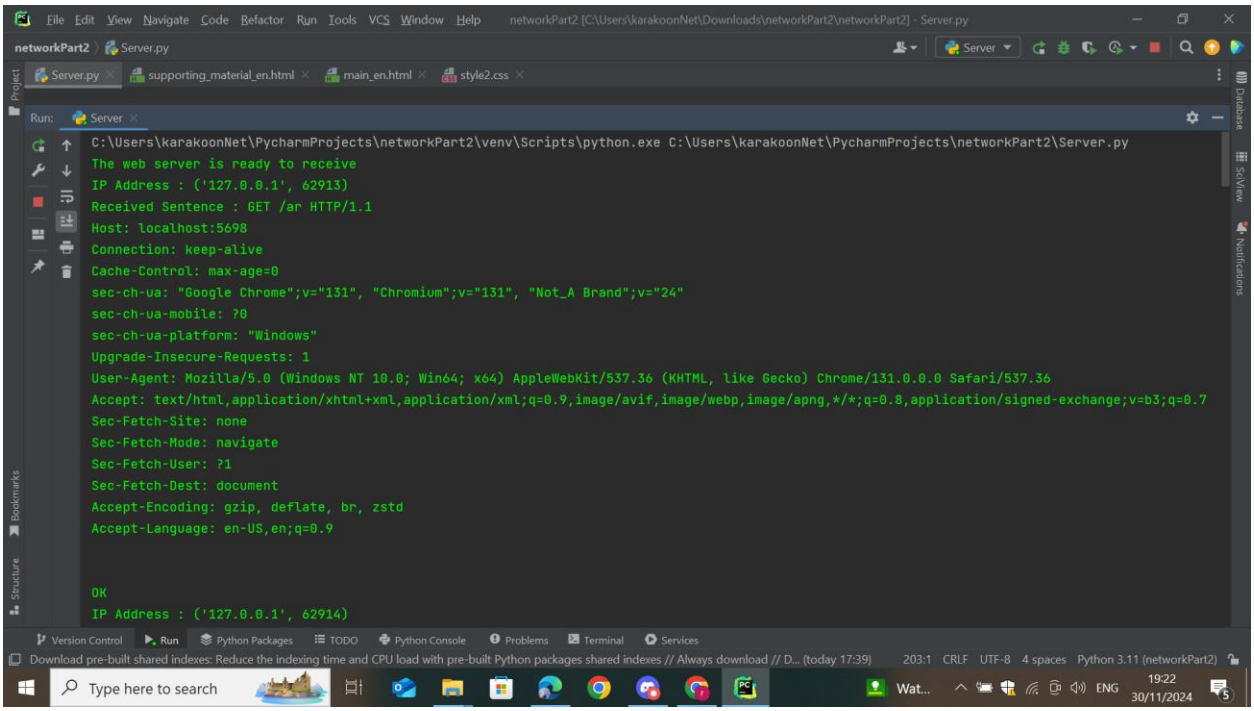
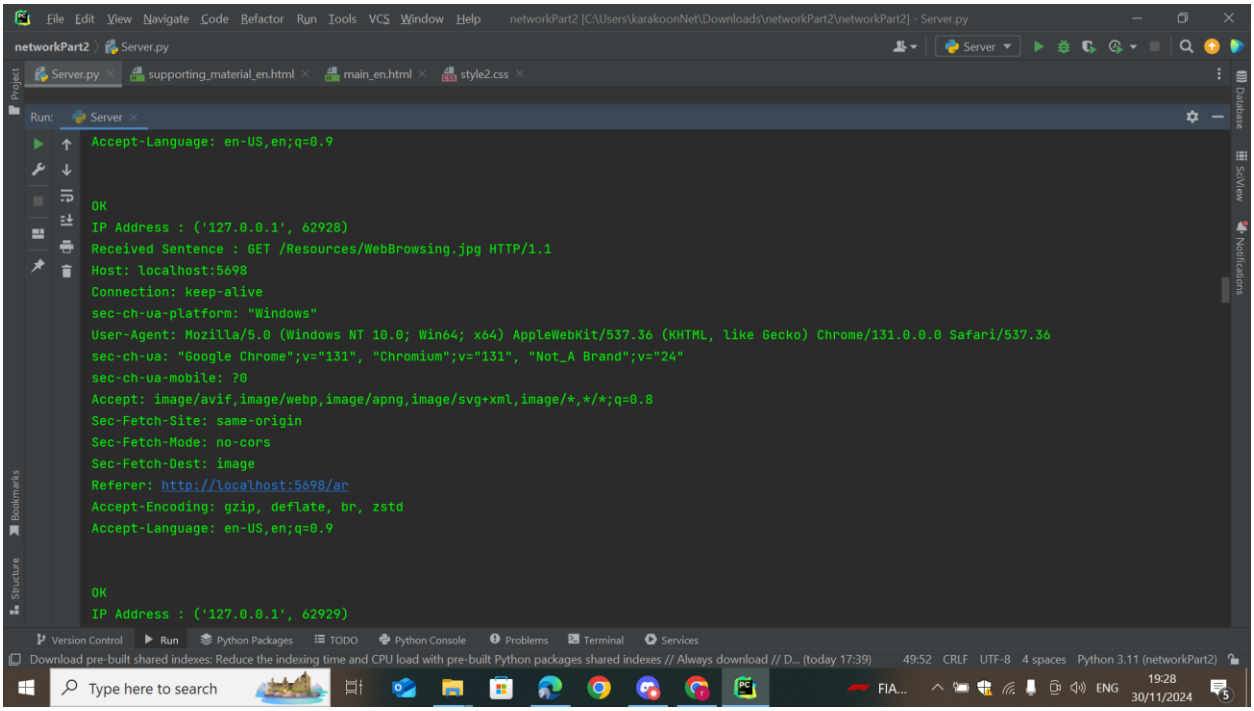
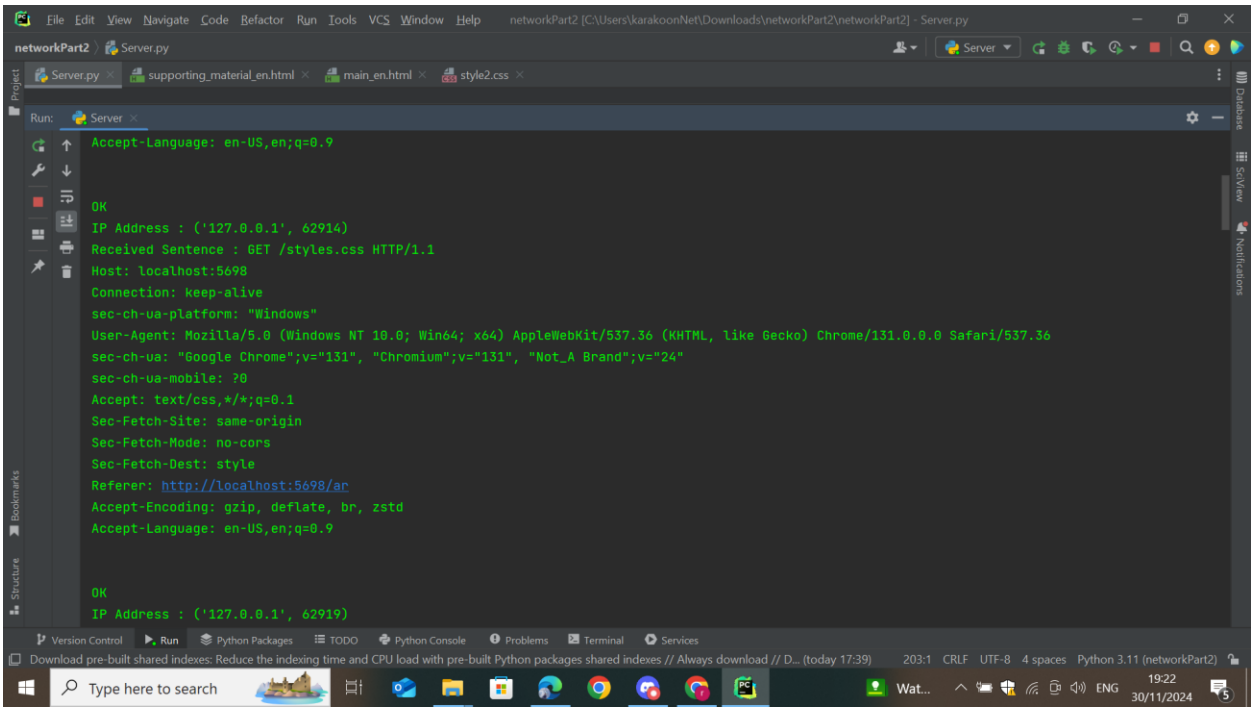


Figure 24 HTTP Requests printed one the server terminal




File Edit View Navigate Code Refactor Run Tools VCS Window Help
networkPart2 [C:\Users\karakoonNet\Downloads\networkPart2\networkPart2] - Server.py
networkPart2 Server.py
Server.py supporting_material_en.html main_en.html style2.css
Run: Server
Accept-Language: en-US,en;q=0.9
OK
IP Address : ('127.0.0.1', 62928)
Received Sentence : GET /Resources/WebBrowsing.jpg HTTP/1.1
Host: localhost: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://localhost:5698/ar
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
OK
IP Address : ('127.0.0.1', 62929)
Version Control Run Python Packages TODO Python Console Problems Terminal Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // D... (today 17:39)
49:52 CRLF UTF-8 4 spaces Python 3.11 (networkPart2)
Type here to search
FIA...
ENG
19:28
30/11/2024


File Edit View Navigate Code Refactor Run Tools VCS Window Help
networkPart2 [C:\Users\karakoonNet\Downloads\networkPart2\networkPart2] - Server.py
networkPart2 Server.py
Server.py supporting_material_en.html main_en.html style2.css
Run: Server
Accept-Language: en-US,en;q=0.9
OK
IP Address : ('127.0.0.1', 62914)
Received Sentence : GET /styles.css HTTP/1.1
Host: localhost: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://localhost:5698/ar
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
OK
IP Address : ('127.0.0.1', 62919)
Version Control Run Python Packages TODO Python Console Problems Terminal Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always download // D... (today 17:39)
203:1 CRLF UTF-8 4 spaces Python 3.11 (networkPart2)
Type here to search
Wat...
ENG
19:22
30/11/2024

2.1.6 http://localhost:5698/Resources/Roaa.png request

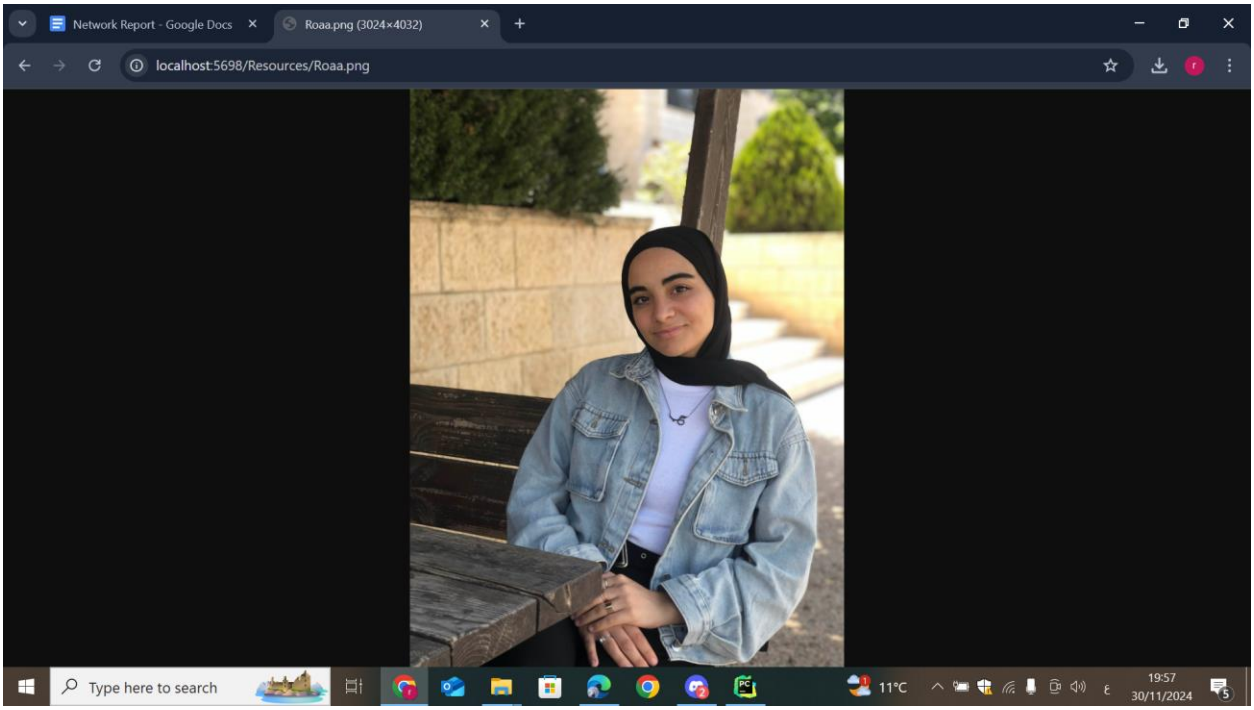
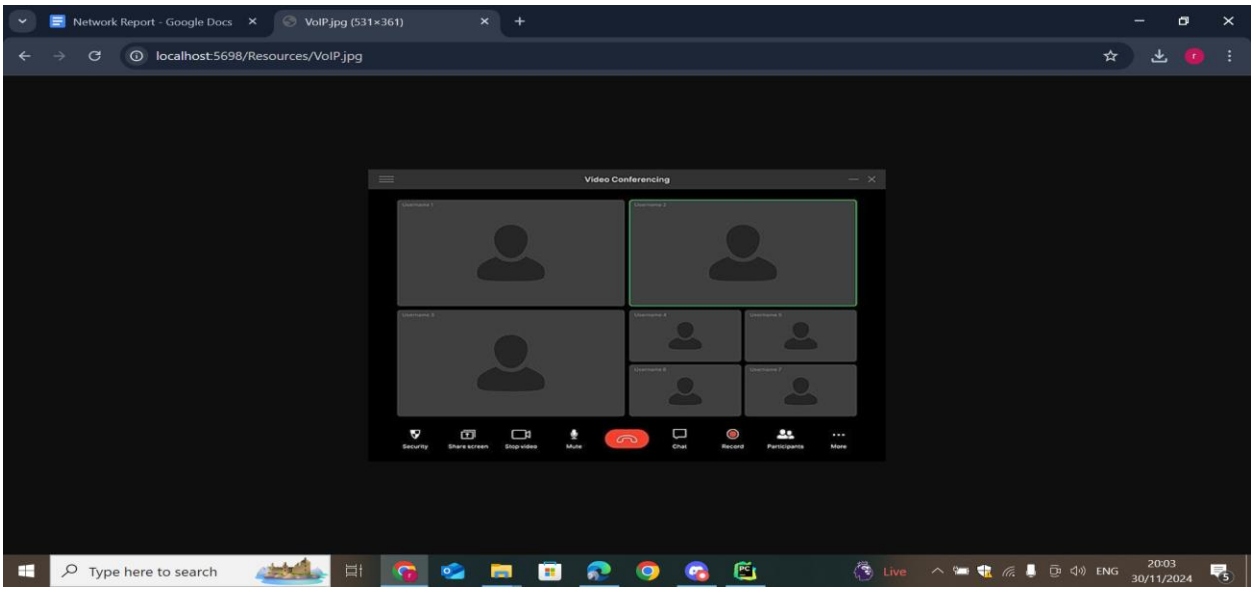
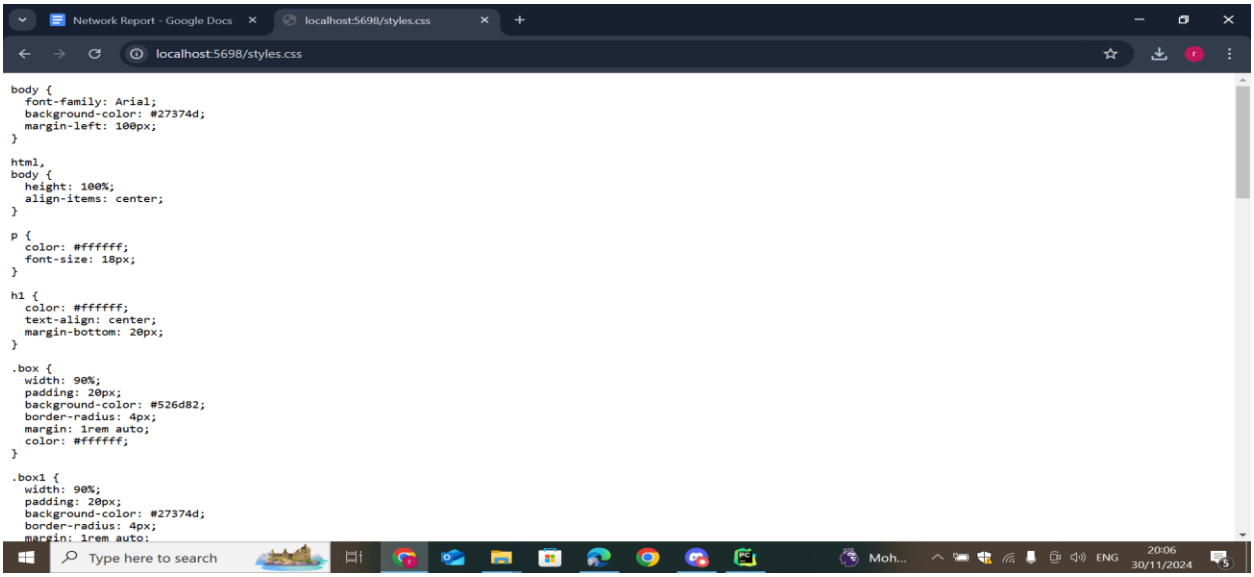


Figure 25 2.1.6

2.1.7 http://localhost:5698/Resources/VoIP.jpg request

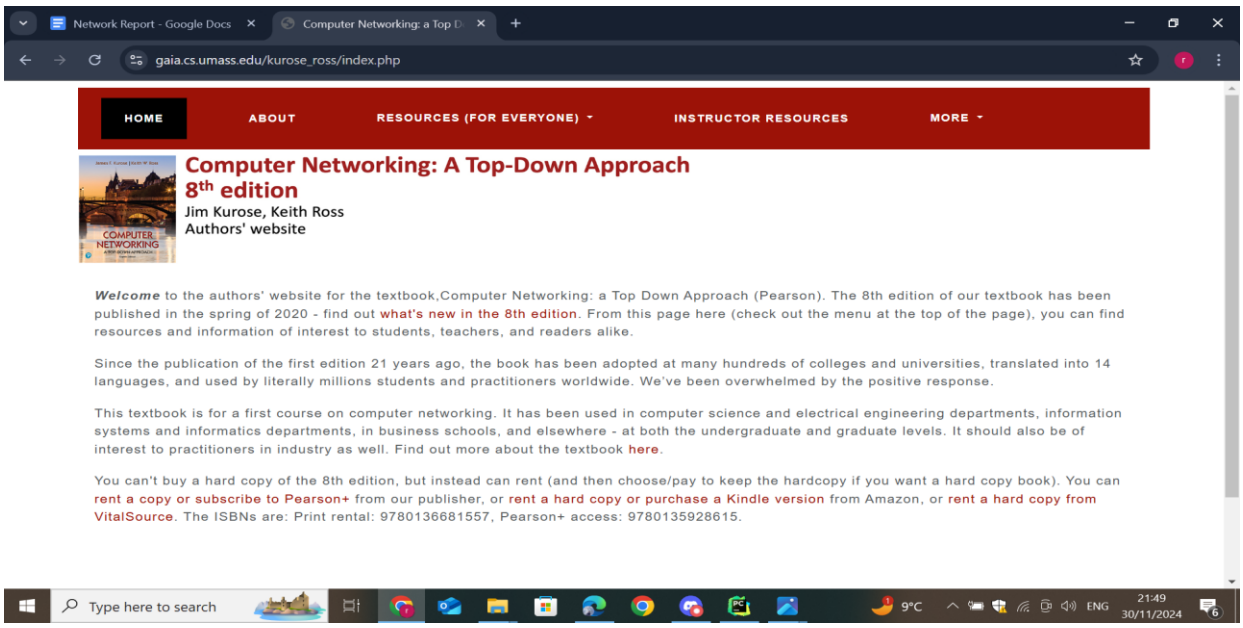


2.1.8 http://localhost:5698/styles.css request

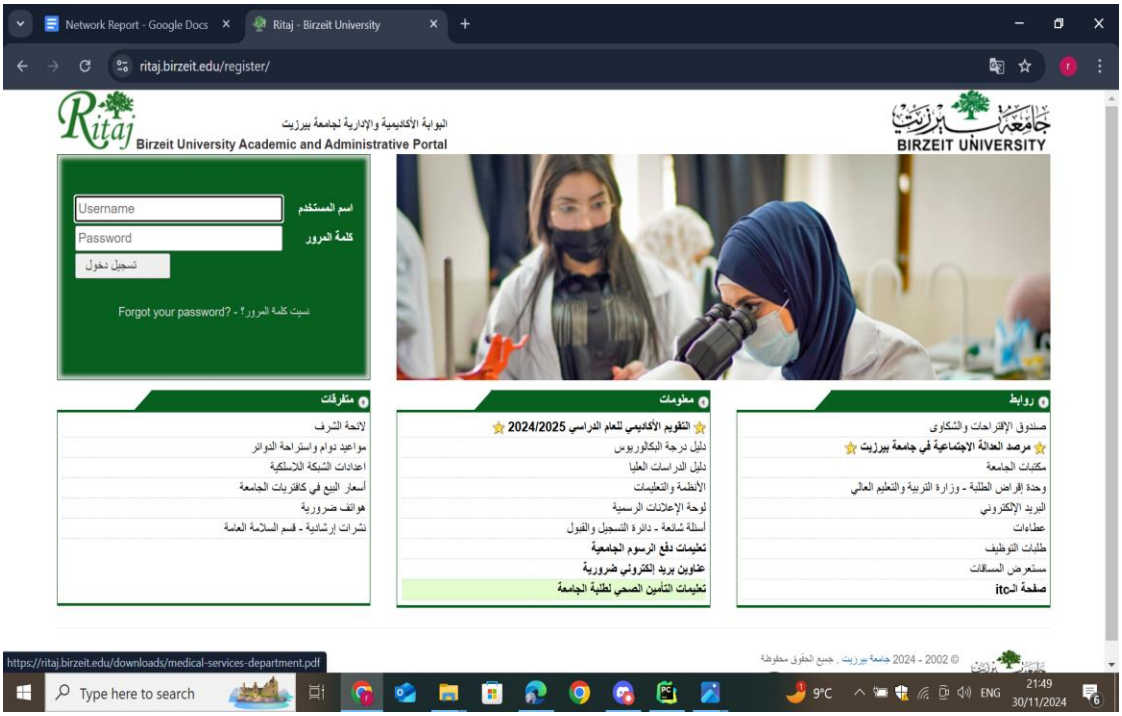


2.1.9 Links to external resources:

2.1.9.1 The textbook website



2.1.9.2 Ritaj website



2.1.10 Error 404 HTML page

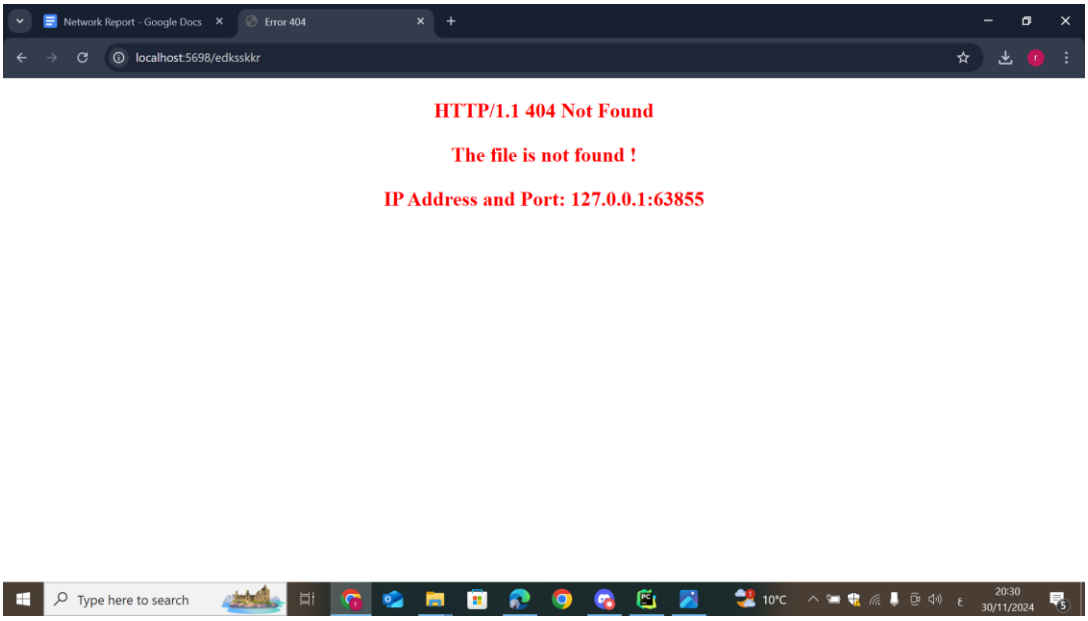


Figure 26 Error 404 HTML page

2.1.10.1 The status line "HTTP/1.1 404 Not Found"

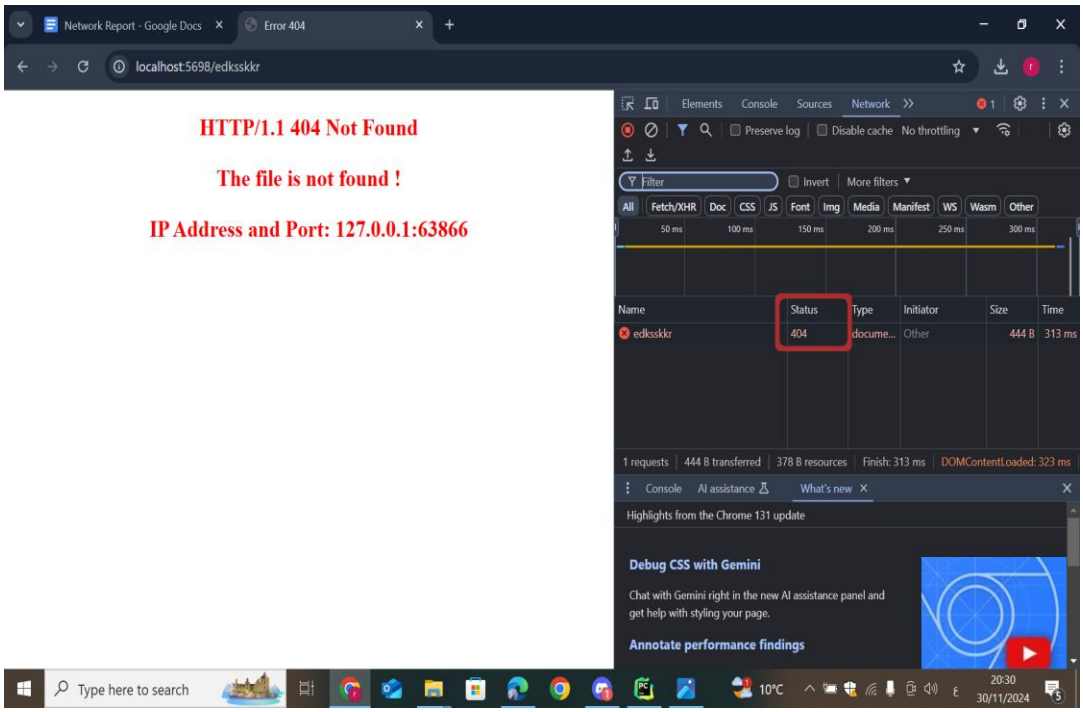


Figure 27 The status line "HTTP/1.1 404 Not Found"

2.1.11 Test the project from an external device

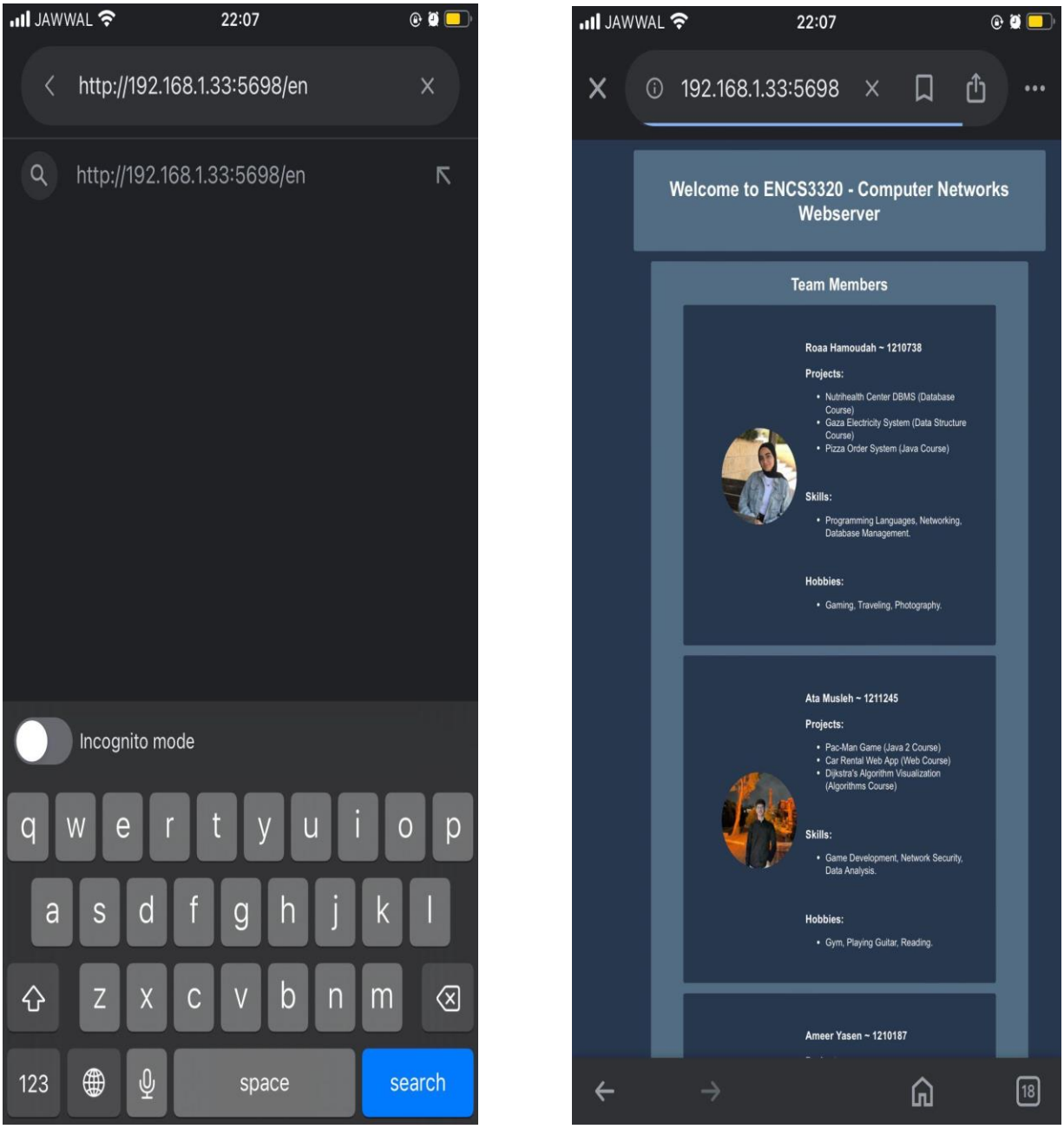
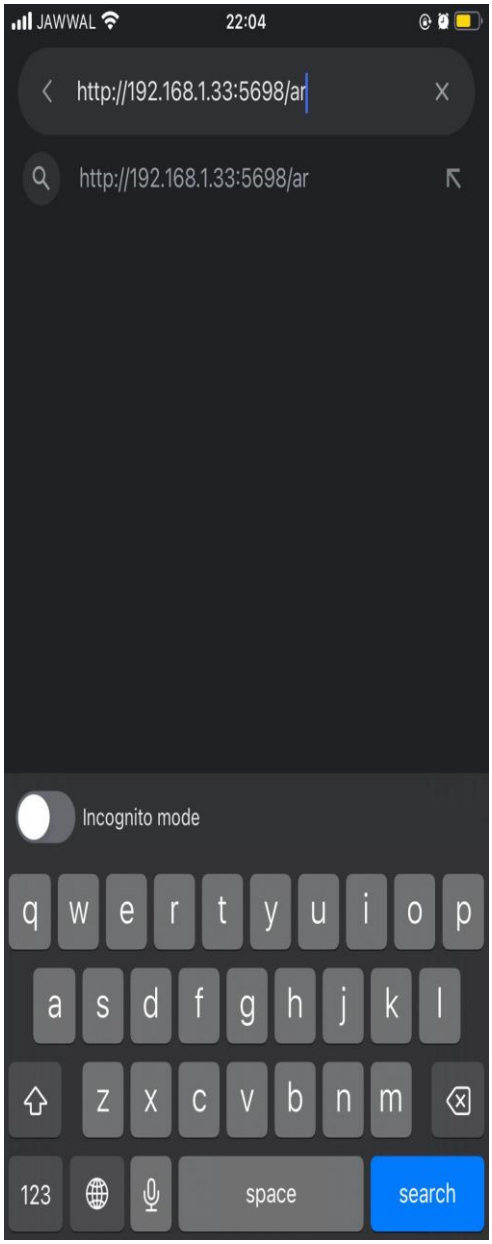
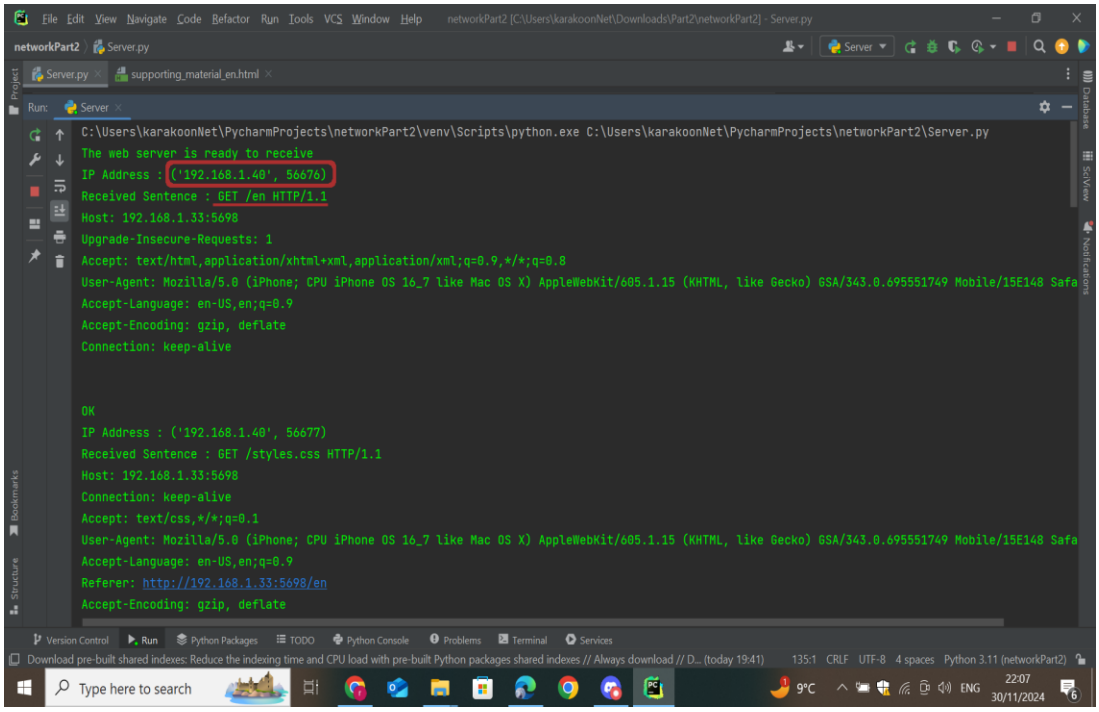


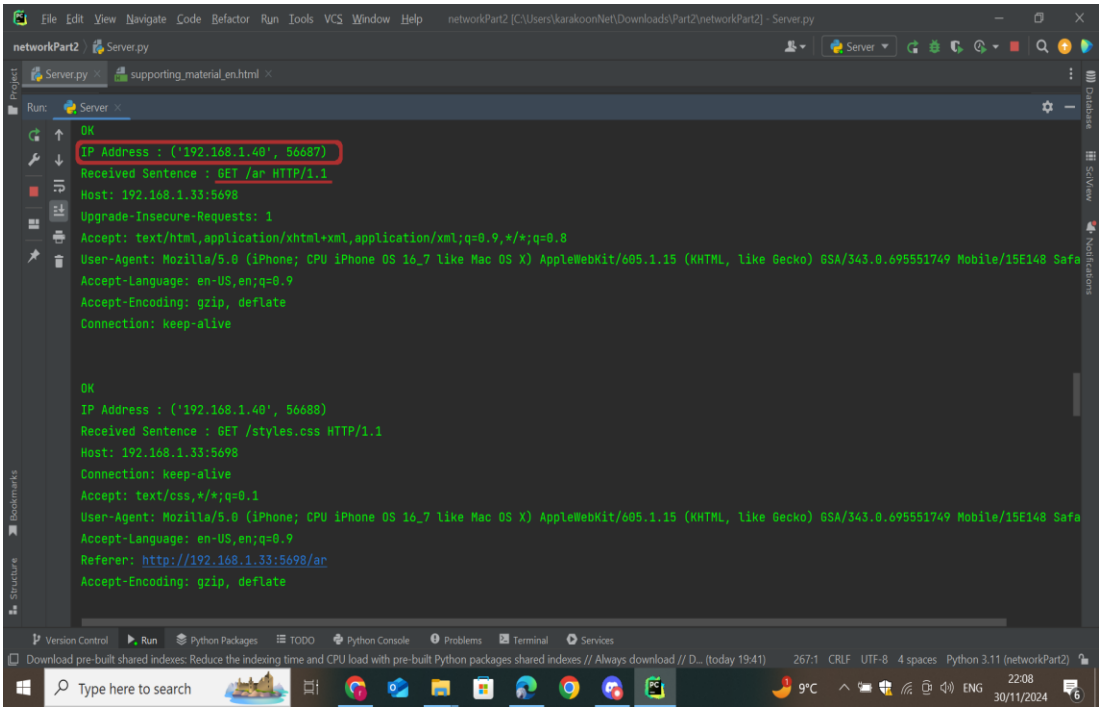
Figure 28 view from external device


JAWWAL
22:04
http://192.168.1.33:5698/ar
http://192.168.1.33:5698/ar
Incognito mode
space
search


JAWWAL
22:04
192.168.1.33:5698
مرحباً بكم في ENCS3320 - خادم الويب لشبكات الكمبيوتر
أعضاء الفريق
رؤى حمودة ~ 1210738
المشاريع:
نظام إدارة قاعدة بيانات مركز اللغة (دورة قاعدة البيانات)
نظام الكهرباء في غزة (دورة هياكل البيانات)
نظام طلب البيتزا (دورة جافا)
المهارات:
لغات البرمجة، الشبكات، إدارة قواعد البيانات.
الهوايات:
الألعاب، السفر، التصوير.
عطا مصلح ~ 1211245
المشاريع:
لعبة باك مان (دورة جافا 2)
تطبيق متجر السيارات (دورة الويب)
تصوير خوارزمية ديكسترا (دورة الخوارزميات)
المهارات:
تطوير الألعاب، أمن الشبكات، تحليل البيانات.
الهوايات:
الذهاب إلى الصالة الرياضية، المزرعة على الجيتار، القراءة.
أمير ياسين ~ 1210187
18

2.1.11.1 Include a Screenshot of the HTTP Request Output





3 Part3 UDP Client-Server Trivia Game

This section shows the implementation and results of the UDP Trivia Game, outlining the design choices, observed behavior, and encountered challenges. The game is designed to allow
multiple clients to connect to a main server and compete by answering random questions in a limited time range.

The trivia game utilizes UDP sockets protocol for communication between the client and server. UDP's connectionless nature allows for simpler implementation and is suitable for this application so many players can join the game and leave without creating multiple connections that always stay connected and take a lot of resources.

.

3.1 Theory and Procedure

The trivia game utilizes UDP sockets protocol for communication between the client and server. UDP’s connectionless nature simplifies the implementation and suits this application where some packet loss is acceptable. Unlike TCP, which requires dedicated connections for each client, UDP's lightweight approach reduces server overhead, allowing for better scalability with a larger number of players joining and leaving dynamically without the overhead of maintaining persistent connections.

3.2 Server Implementation:

The server manages client connections, broadcasts questions, receives and evaluates answers, tracks scores, and declares the winner. Key components include:

Client Management: A dictionary (active_clients) stores active client information, mapping their (IP, port) to their chosen usernames. This dictionary is dynamically updated as players join and leave.

Game Logic: The handle_game_round() function orchestrates the game flow. It selects random questions from a predefined list of questions and then broadcasts them to the players, receives answers within a 15-second timeframe, evaluates the answers, and updates scores based on the scoring logic.

Scoring System: The game implements a dynamic scoring mechanism. Faster correct answers get higher points: 10 points for answers within 5 seconds, 5 points between 5 and 10 seconds, and 2 points for answers after 10 seconds but within the 15-second round window. This adds a time-pressure element to the game.

Broadcasting: The broadcast_message() function efficiently disseminates messages (questions, scores, game status updates) to all connected clients.

Disconnection Handling: The server checks client connections using a short timeout on the sendto() function. This mechanism finds unresponsive clients and removes them from the active_clients list and

the scores dictionary, ensuring the game progresses smoothly even with intermittent client disconnections.

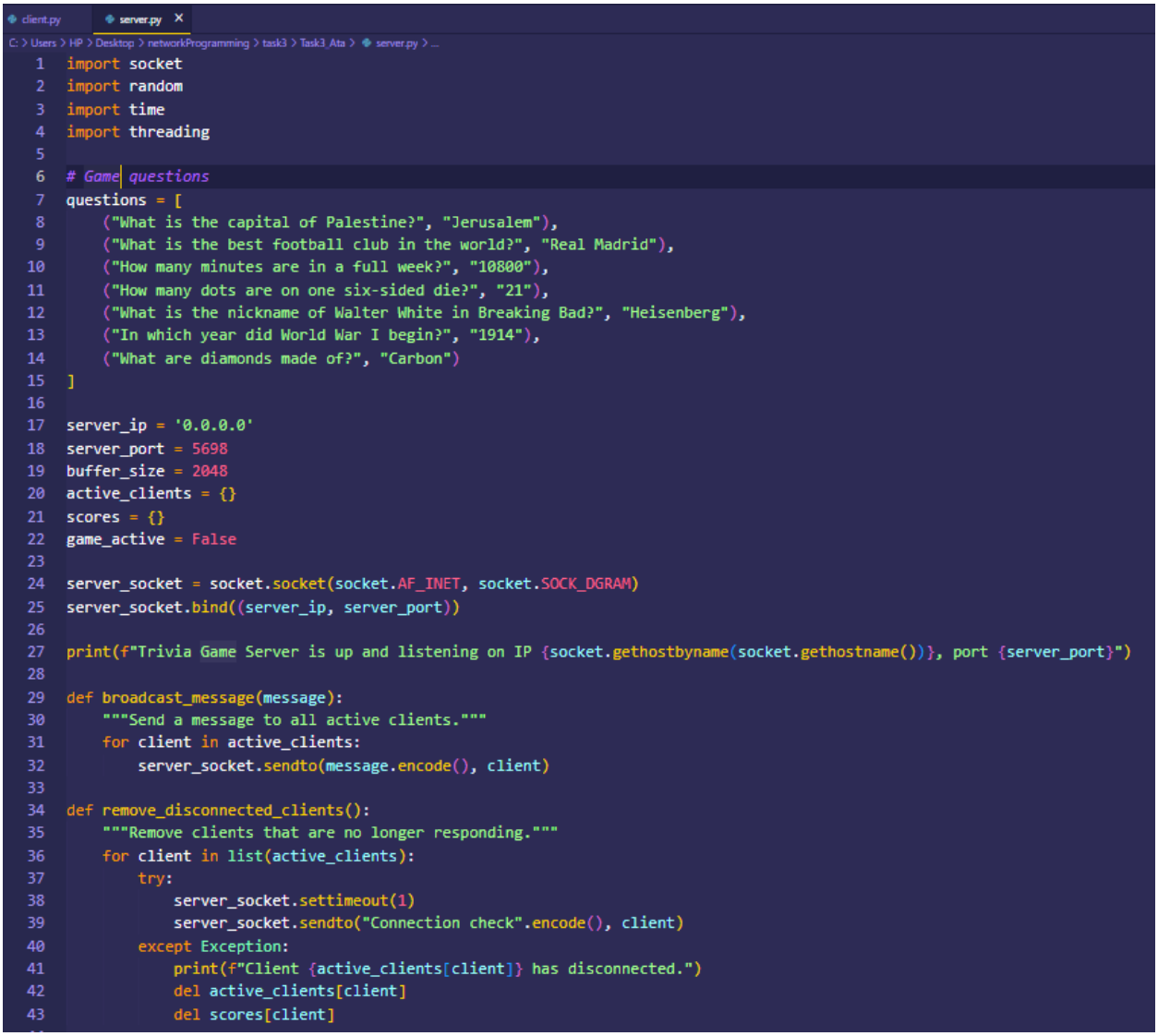
Client Implementation:

The client connects to the server by the ip address for the server and port number, sends a username, receives and displays questions, submits answers within the allocated time, and displays the score updates. It continuously listens for messages from the server, ensuring responsiveness to game events. Upon connection, the client sends its desired username to the server, which then adds it to the active_clients list. If a username is already taken, the client needs to enter a different name.

Description of server.py (found in .zip file):

The server and client source code in Python:

3.2.1 Server.py



```python
import socket
import random
import time
import threading

# Game questions
questions = [
    ("What is the capital of Palestine?", "Jerusalem"),
    ("What is the best football club in the world?", "Real Madrid"),
    ("How many minutes are in a full week?", "10800"),
    ("How many dots are on one six-sided die?", "21"),
    ("What is the nickname of Walter White in Breaking Bad?", "Heisenberg"),
    ("In which year did World War I begin?", "1914"),
    ("What are diamonds made of?", "Carbon")
]

server_ip = '0.0.0.0'
server_port = 5698
buffer_size = 2048
active_clients = {}
scores = {}
game_active = False

server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((server_ip, server_port))

print(f"Trivia Game Server is up and listening on IP {socket.gethostbyname(socket.gethostname())}, port {server_port}")

def broadcast_message(message):
    """Send a message to all active clients."""
    for client in active_clients:
        server_socket.sendto(message.encode(), client)

def remove_disconnected_clients():
    """Remove clients that are no longer responding."""
    for client in list(active_clients):
        try:
            server_socket.settimeout(1)
            server_socket.sendto("Connection check".encode(), client)
        except Exception:
            print(f"Client {active_clients[client]} has disconnected.")
            del active_clients[client]
            del scores[client]
```

Figure 29 Server.py source code

- First import the needed libraries for the game like the socket library for client-server communication and threading for a smoother game experience and making the game logic spirit from the communication

logic then use the time library for time calculation and the random for far game rules and question distribution

- A predefined list of random general knowledge questions with their answers stored in a dictionary (key-value pairs)
- **Declared variables :**

server_ip = **'0.0.0.0'** means that the server will listen for incoming connections on all available network interfaces on the machine.

server_port = 5698: The port number the server will listen on.

buffer_size = 2048: The maximum size of data that can be received at once.

active_clients = {}: A dictionary to store connected clients (address: name) pairs.

scores = {}: A dictionary to keep track of player scores.

game_active = False: A flag to indicate whether a game is currently running.

- Socket Creation and Binding:

```
24  server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25  server_socket.bind((server_ip, server_port))
```

Creates a UDP socket. socket.AF_INET specifies the IPv4 address family and socket.SOCK_DGRAM indicates a UDP socket.
Binds the socket to the specified IP address and port, making the server start listening for incoming connections on that address and port.

- Broadcast message function:

This function sends a message to all currently connected clients.

def broadcast_message(message): Defines the function name and takes a message (a string) as input.

for client in active_clients:: This loop iterates through the keys (client addresses) of the active_clients dictionary. Active_clients is a dictionary where keys are client addresses (IP, port) and values are client names.

server_socket.sendto(message. encode(), client): encode the message to binary and send it to the client using UDP socket server_socket, encoding is UTF-8 by default.

- Remove disconnected clients function:

The remove_disconnected_clients() function identifies and removes unresponsive clients from the game. It loops through a copy of connected clients (copy of active_clients list). For each client, it send "Connection check" message with a 1-second timeout (server_socket.settimeout(1)). If the send fails the client information is removed from both the active_clients and scores dictionaries. A message is printed to the console confirming the disconnection.

```python
def handle_game_round():
    """Conduct a round of trivia, asking questions and handling scoring."""
    global game_active
    game_active = True

    if len(active_clients) < 2:
        print("Not enough players. Waiting for more clients...")
        return

    broadcast_message("Game starting now! Get ready!")
    time.sleep(2)

    round_scores = {client: 0 for client in active_clients}
    used_questions = []
    total_questions = len(questions)

    for i in range(total_questions):
        remove_disconnected_clients()

        available_questions = [q for q in questions if q not in used_questions]
        if not available_questions:
            break

        question, answer = random.choice(available_questions)
        used_questions.append((question, answer))

        print(f"Broadcasting Question {i + 1}/{total_questions}: {question}")
        broadcast_message(f"Question {i + 1}/{total_questions}: {question}")

        start_time = time.time()
        answered_clients = {}
        fastest_client = None
        fastest_time = float('inf')

        while time.time() - start_time < 15:
            try:
                server_socket.settimeout(1)
                message, client_address = server_socket.recvfrom(buffer_size)
                response = message.decode().strip().lower()

                if client_address in active_clients:
                    if client_address not in answered_clients:
                        answered_clients[client_address] = response
                        if response == answer.lower():
                            time_elapsed = time.time() - start_time
                            points = 10 if time_elapsed <= 5 else 5 if time_elapsed <= 10 else 2
                            round_scores[client_address] += points
                            scores[client_address] += points
                            if time_elapsed < fastest_time:
                                fastest_time = time_elapsed
                                fastest_client = client_address
            except socket.timeout:
                continue

        for client in active_clients:
            if client not in answered_clients:
                broadcast_message(f"{active_clients[client]} did not answer in time.")

        broadcast_message(f"Time's up! The correct answer was: {answer}")

        leaderboard = "\n".join([f"{active_clients[client]}: {scores[client]} points" for client in scores])
        broadcast_message("Current Scores:\n" + leaderboard)
        print(f"Leaderboard after Question {i + 1}")
        print(leaderboard)
        time.sleep(5)

    game_active = False
    max_score = max(scores.values(), default=0)
    winners = [active_clients[client] for client in scores if scores[client] == max_score]

    if winners:
        winner_names = ", ".join(winners)
        broadcast_message(f"Game Over! The winners are: {winner_names} with {max_score} points!")
    else:
        broadcast_message("Game Over! No winner this time.")

    final_scores = "\n".join([f"{active_clients[client]}: {scores[client]} points" for client in scores])
    broadcast_message("Final Scores:\n" + final_scores)
    print("Final Scores:")
    print(final_scores)
```

Figure 30 server.py source code 2

- Handling the game round function:

The handle_game_round() function runs a complete trivia round. It first checks for sufficient players (2 or more) and then broadcasts a start message.

```
45  def handle_game_round():
46      """Conduct a round of trivia, asking questions and handling scoring."""
47      global game_active
48      game_active = True
49
50      if len(active_clients) < 2:
51          print("Not enough players. Waiting for more clients...")
52          return
53
54      broadcast_message("Game starting now! Get ready!")
55      time.sleep(2)
56
57      round_scores = {client: 0 for client in active_clients}
58      used_questions = []
59      total_questions = len(questions)
```

The function iterates through available questions, selecting one randomly and sending it to all clients, and checking for disconnected clients before each question. available_questions filter out questions already asked. A random question (question, answer) is selected from the remaining available questions. The chosen question is added to used_questions to prevent repetition. The question is broadcast with its number to all clients.

```
61      for i in range(total_questions):
62          remove_disconnected_clients()
63
64          available_questions = [q for q in questions if q not in used_questions]
65          if not available_questions:
66              break
67
68          question, answer = random.choice(available_questions)
69          used_questions.append((question, answer))
70
71          print(f"Broadcasting Question {i + 1}/{total_questions}: {question}")
72          broadcast_message(f"Question {i + 1}/{total_questions}: {question}")
```

A 15-second timer starts, during this time the server is listening for client answers. server_socket.settimeout(1) sets a 1-second timeout on the socket to prevent blocking indefinitely.
server_socket.recvfrom(buffer_size) receives a client's answer and their address. If the client hasn't already answered, the client's response is recorded in answered_clients. If the response is correct, the time taken is calculated. Points are given based on correctness and response time, with bonuses for faster answers (10 points for answering in less than 5 seconds,5 points for answering within 10 seconds, and two points for any answer after that).
except socket.timeout: Handles timeouts if no message is received within 1 second, allowing the loop to continue listening.

```
73
74          start_time = time.time()
75          answered_clients = {}
76          fastest_client = None
77          fastest_time = float('inf')
78
79          while time.time() - start_time < 15:
80              try:
81                  server_socket.settimeout(1)
82                  message, client_address = server_socket.recvfrom(buffer_size)
83                  response = message.decode().strip().lower()
84
85                  if client_address in active_clients:
86                      if client_address not in answered_clients:
87                          answered_clients[client_address] = response
88                          if response == answer.lower():
89                              time_elapsed = time.time() - start_time
90                              points = 10 if time_elapsed <= 5 else 5 if time_elapsed <= 10 else 2
91                              round_scores[client_address] += points
92                              scores[client_address] += points
93                              if time_elapsed < fastest_time:
94                                  fastest_time = time_elapsed
95                                  fastest_client = client_address
96              except socket.timeout:
97                  continue
```

After the 15-second answer window closes, the game server processes the results of the question. notifies all players who the ones failed to submit an answer within the time limit. Then, the correct answer is displayed to everyone. A leaderboard, displaying the current scores of all players, is broadcasted to all clients and printed on the server console. Finally, a 5-second pause is before the next question begins, allowing players a short break to review the results and prepare for the next round.

```
99          for client in active_clients:
100             if client not in answered_clients:
101                 broadcast_message(f"{active_clients[client]} did not answer in time.")
102
103         broadcast_message(f"Time's up! The correct answer was: {answer}")
104
105         leaderboard = "\n".join([f"{active_clients[client]}: {scores[client]} points" for client in scores])
106         broadcast_message("Current Scores:\n" + leaderboard)
107         print(f"Leaderboard after Question {i + 1}")
108         print(leaderboard)
109         time.sleep(5)
```

The game round is finished. It deactivates the game flag and determines the highest score. A list of winners is displayed by naming players whose scores match the highest score. the winning score is broadcast to

all clients. Otherwise, a "no winner" message is sent. Finally, a formatted string of final scores is created and broadcast to all players, and also printed to the server console for review.

```
    game_active = False
    max_score = max(scores.values(), default=0)
    winners = [active_clients[client] for client in scores if scores[client] == max_score]

    if winners:
        winner_names = ", ".join(winners)
        broadcast_message(f"Game Over! The winners are: {winner_names} with {max_score} points!")
    else:
        broadcast_message("Game Over! No winner this time.")

    final_scores = "\n".join([f"{active_clients[client]}: {scores[client]} points" for client in scores])
    broadcast_message("Final Scores:\n" + final_scores)
    print("Final Scores:")
    print(final_scores)
```

- Main server logic and new client handling:

  The code defines two main functions: handle_new_client and server_main. handle_new_client manages new client connections, storing their usernames and initializing their scores. it checks for duplicate usernames before accepting a new client, preventing conflicts. If a game is already in progress, new clients are informed they will join the next round. Otherwise, if enough players are present, a new game round is initiated in a separate thread using handle_game_round. This threading is crucial for preventing the server from blocking while a game is running, allowing it to continue accepting new connections.

  The server_main function drives the server's operation. It continuously listens for incoming client connections and messages within an infinite loop. When a client connects and sends their username, server_main calls handle_new_client to process the connection. A try-except block handles keyboard interrupts, allowing for graceful server shutdown. Closing the server socket releases the port and resources.

```python
# new Client can join but an error will occure on the server side!!
def handle_new_client(client_address, username):
    active_clients[client_address] = username
    scores[client_address] = 0
    print(f"New client joined: {username} from {client_address}")
    broadcast_message(f"{username} has joined the game.")

    # If a game is in progress, put the new player on the wait list and add them on the next round
    if game_active:
        server_socket.sendto("A game is currently in progress. You'll join the next round.".encode(), client_address)
    else:
        if len(active_clients) >= 2:
            # Start a new game round in a separate thread
            game_thread = threading.Thread(target=handle_game_round)
            game_thread.start()

def server_main():
    global game_active
    while True:
        try:
            message, client_address = server_socket.recvfrom(buffer_size)
            username = message.decode().strip()
            handle_new_client(client_address, username)
        except KeyboardInterrupt:
            print("Shutting down server...")
            break

    server_socket.close()

server_main()
```

Figure 31 server.py source code 3

3.2.2 client.py

client.py

client.py > ...

```python
import socket

buffer_size = 2048

server_ip = input("Enter the server IP address: ")
server_port = int(input("Enter the server port: "))

client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

username = input("Enter your username: ")
client_socket.sendto(username.encode(), (server_ip, server_port))

print(f"Connected to the Trivia Game Server at {server_ip}:{server_port}")

try:
    while True:
        message, _ = client_socket.recvfrom(buffer_size)
        decoded_message = message.decode()
        print("\n" + decoded_message)

        if "Question" in decoded_message:
            answer = input("Your answer (press Enter to skip): ").strip()
            client_socket.sendto(answer.encode(), (server_ip, server_port))

except KeyboardInterrupt:
    print("\nExiting the game...")
finally:
    client_socket.close()
```

Figure 32 Client.py source code

This code implements a basic trivia game client using the UDP sockets protocol. The client first enters the server's IP address and port number, then creates a UDP socket and sends the user's chosen username to the server. It then enters a main loop where it continuously listens for messages from the server. Upon receiving a message, it displays it to the user. If the message contains a question, the client prompts the user for their answer and sends it back to the server. The client handles keyboard

interrupts gracefully, allowing the user to exit the game with Ctrl+C. A final block ensures the socket is always closed properly.

3.2.3 live game experience

The game starts when running the server code, it prints the IP address for the machine and the port number and then starts listening for clients requests to join the game

```
PS C:\Users\HP\Desktop\networkProgramming\task3\Task3_Ata> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe
Enter the server IP address: 192.168.56.1
Enter the server port: 5698
Enter your username: Darth_Vader
Connected to the Trivia Game Server at 192.168.56.1:5698

Darth_Vader has joined the game.
```

First client joined, The client Enters the server ip address and port number and his username in the game, in our case: Darth Vader (from Starwars)

The server response with {username} has joined the game.

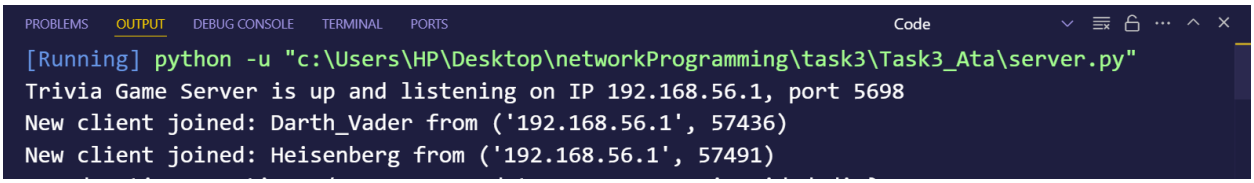
A new client also joined the game with the user name Heisenberg (from Breaking Bad)

The server broadcasts a message that the game will begin, and then the handle round function starts checking if all players are still responding with each round

The server dictates a new thread to run the handle round function so the

game continues receiving clients' requests to join the game while the round is running.

Server terminal output:



First-round server terminal output:

```
New client joined: Heisenberg from ('192.168.56.1', 57491)
Broadcasting Question 1/7: How many dots are on one six-sided die?
Leaderboard after Question 1
Darth_Vader: 0 points
Heisenberg: 0 points
```

Client terminal (Darth_Vader):

```
Question 1/7: How many dots are on one six-sided die?

Your answer (press Enter to skip):

Darth_Vader did not answer in time.

Heisenberg did not answer in time.

Time's up! The correct answer was: 21

Current Scores:
Darth_Vader: 0 points
Heisenberg: 0 points

Connection check
```

The rest of rounds server terminal output (until the game finished):

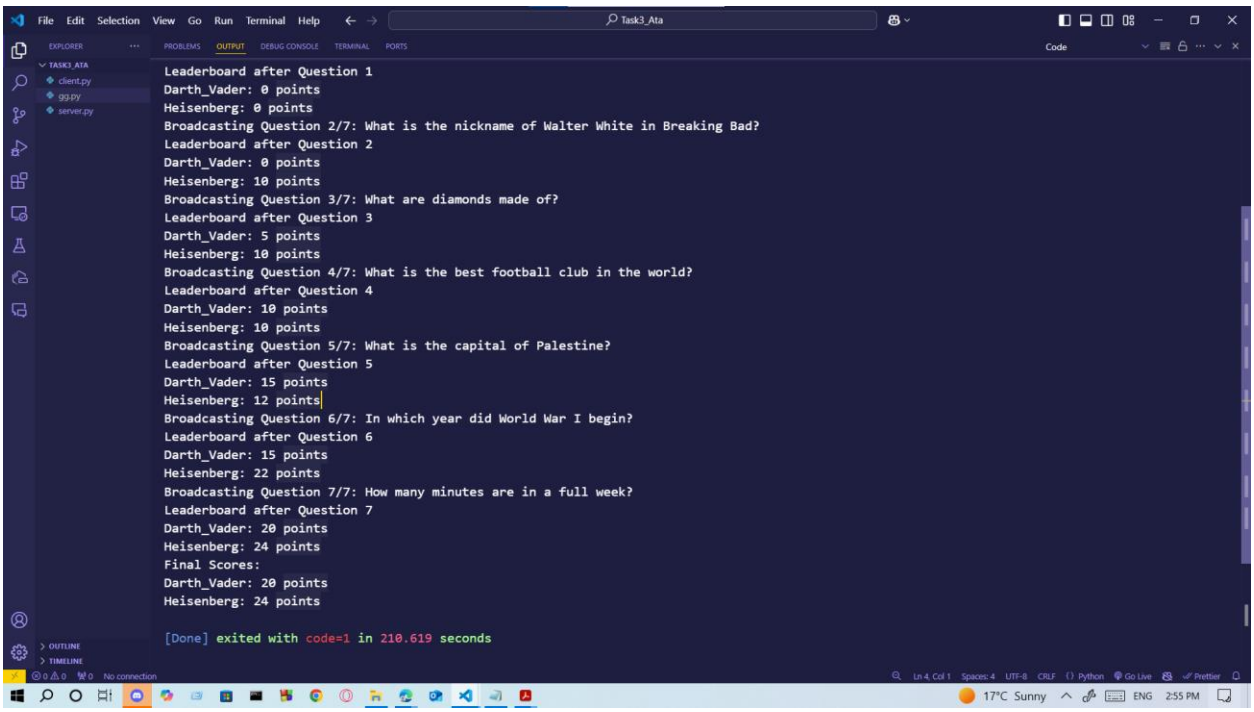


Figure 33 server terminal game output

For Darth_Vader:

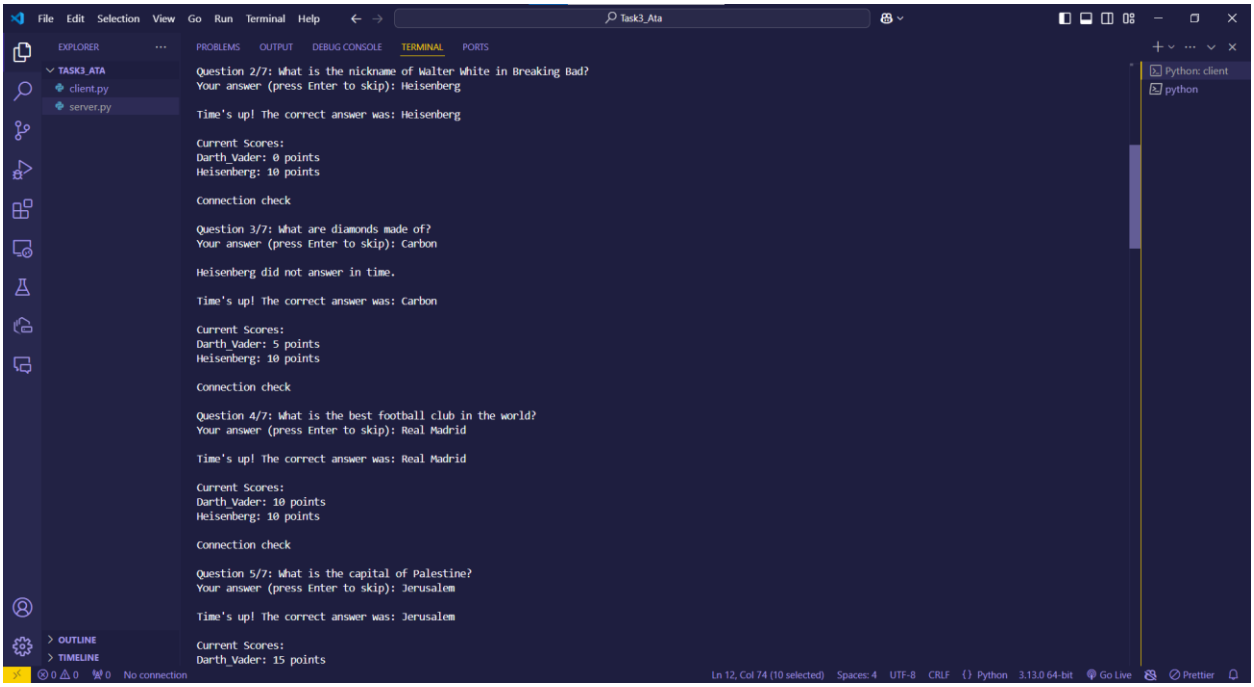
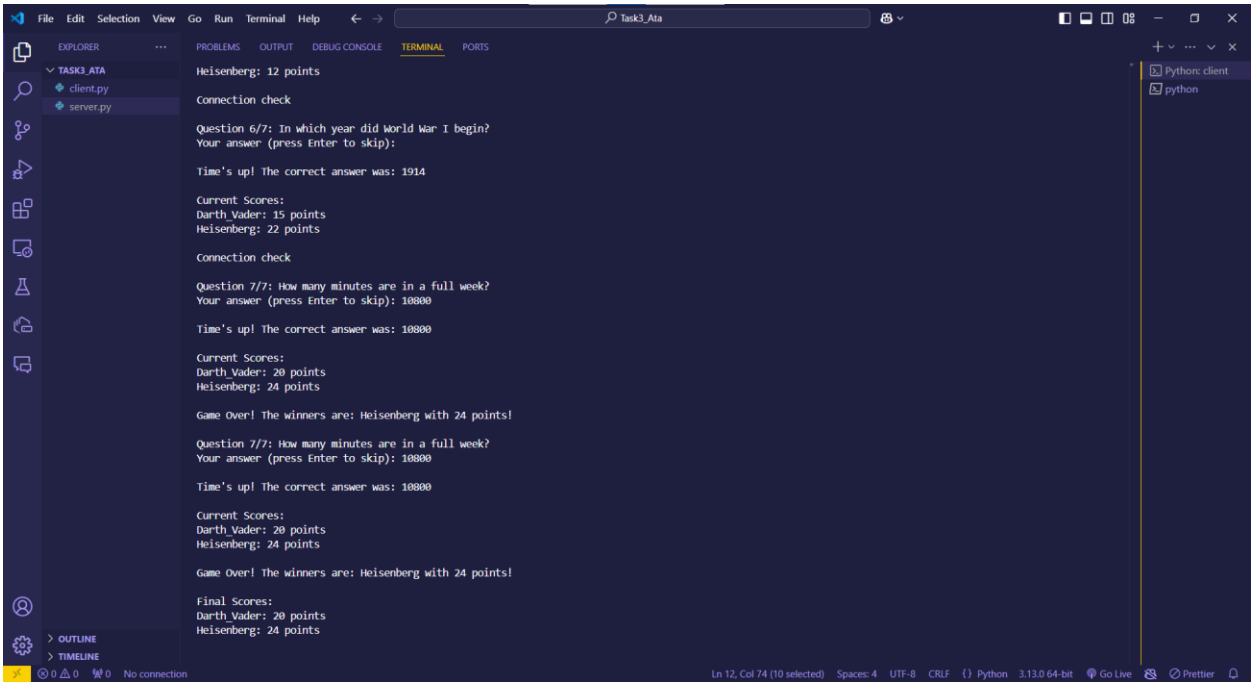


Figure 34 Client terminal output

In question 2 Heisenberg answered the question in less than 5 seconds so he got 10 points after his score was 0 from the last round
But Darth_Vader does not answer and just press ok to skip the question.
In question 3 Darth_Vader answers the question after the first 5 seconds but before 10 seconds pass so he has got 5 points while Heisenberg did not answer and just skipped the question.

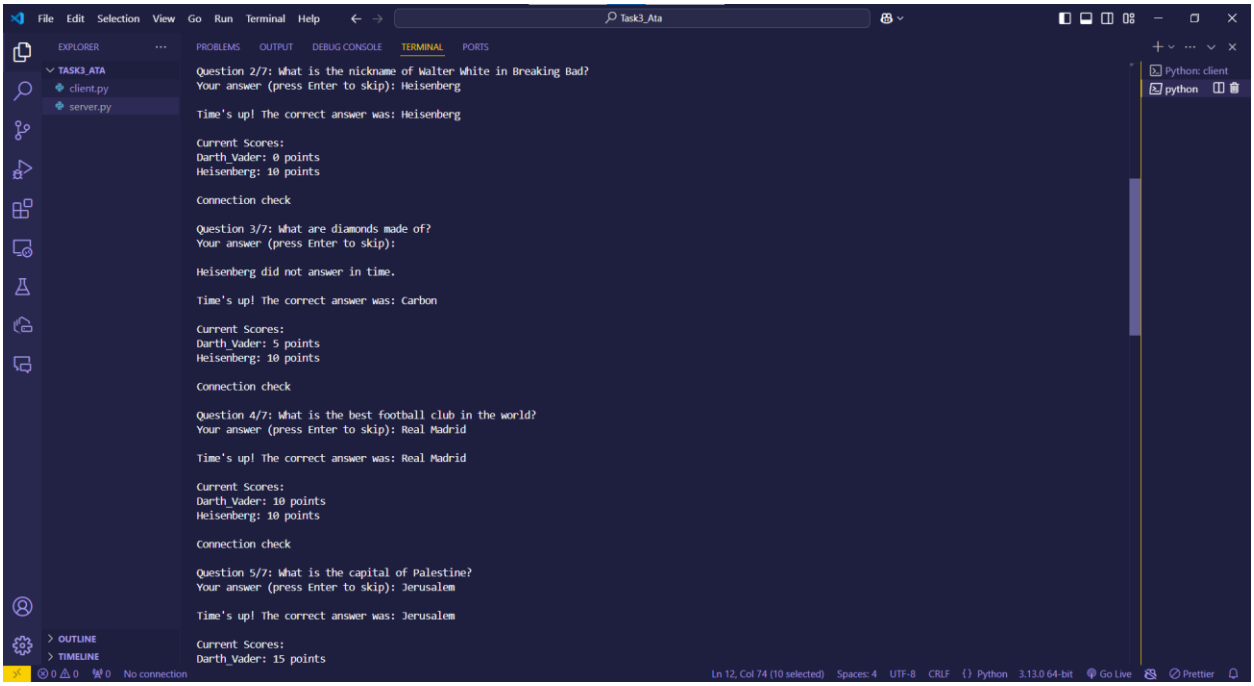
Same thing in question 4, Darth_Vader gets 5 points and nothing for Heisenberg

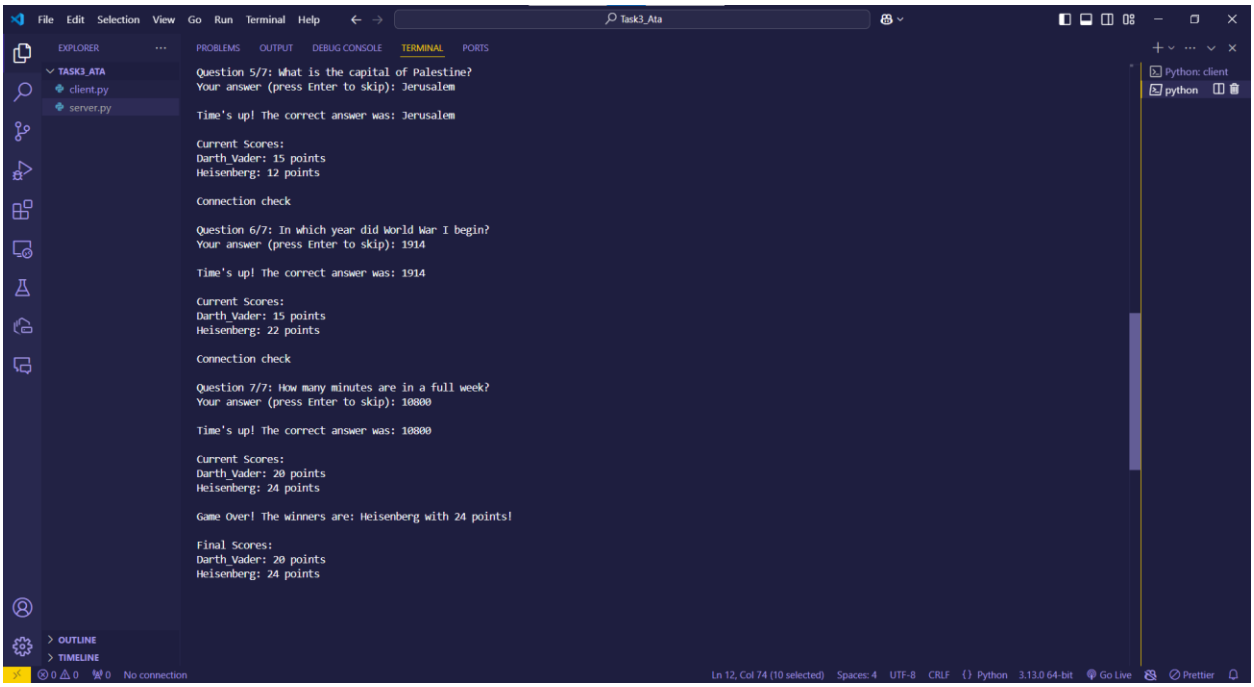
In question 5 Darth_Vader gets 5 points for answering within 10 seconds and Heisenberg gets only two because he answered after 2 seconds!



The rest of the game status from The client Darth_Vader Terminal
Heisenberg won the game after 7 rounds

The Terminal output for Heisenberg:





4 Alternative Solutions, Issues, and Limitations

4.1 Alternative Solutions

A key challenge was handling concurrent client joins mid-game. The initial single-threaded server design blocked new connections while a round was active, as the server couldn't process new requests while managing the game. To resolve this, threading was introduced, handling new connections via handle_new_client concurrently with gameplay. This improved responsiveness but introduced potential race conditions with shared resources like active clients and scores.

4.2 Issues and Challenges

4.2.1 Web Server

- Ensuring cross-browser compatibility can be challenging.
- Error handling is crucial to prevent unexpected Behavior.

4.2.2 Trivia Game

- Network congestion can lead to message loss or delays.

4.3 Limitations

The question database is hardcoded within the server script, restricting the game's flexibility and scalability. Adding or modifying questions requires code changes and server restarts, making content updates cumbersome. The reliance on UDP, while beneficial for responsiveness, inherently risks packet loss, which can lead to missing questions, answers, or score updates, impacting gameplay and fairness. The basic disconnection handling, using timeouts, may not be robust enough in all network environments, potentially leading to delayed removal of inactive players or incorrect score calculations. The threading model, while improving concurrency, lacks robust synchronization mechanisms, creating a vulnerability to race conditions when multiple clients interact simultaneously, particularly during scoring and connection updates. Finally, the server's capacity is limited by its single-process, threaded architecture, hindering scalability for a larger number of players or more complex game features. These limitations could lead to performance bottlenecks or unexpected behavior under heavy load.

Teamwork

Name	ID	Tasks
Roaa	**1210738**	**Task 1:** created the Definitions and tested and interrupted the Commands **Task 2 (Code):** Developed HTML and CSS for main_en.html and main_ar.html and its logic in the server code. **Task 2 (Report):** Authored the first two sections of the report, providing a detailed overview of the project's objectives, literature review, and related technologies. **Task 2 (Code):** Created the error HTML page and its logic.
Ameer	**1210187**	**Task 2 (Supporting Materials):** Developed HTML and CSS for Arabic and English pages (Supporting_Materials_en and Supporting_Materials_ar), implemented server redirection logic (HTTP status 307), and handled resource requests. **Task 2 (Report):** Contributed to Task 2 of the report. **Task 3 (Report):** Contributed to the main report structure and tables.
Ata	**1211245**	**Task 3 (Code):** Implemented both server-side and client-side code. **Contributed to Report:** Helped with resource collection and management and contributed to the report.