Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Networks – ENCS3320

Project #1- Socket Programming

Project Report

Prepared by:

Taymaa Nasser 1222640

Malak Milhem 1220031

Instructor: Dr. Alhareth Zyoud

Section: 2

Date: 11/05/2025

Abstract

This project explores fundamental concepts of network programming through three tasks, each demonstrating client-server communication. The first task involves using network commands and Wireshark to analyze IP configurations and DNS queries. The second task implements a simple web server using TCP sockets, serving static web pages and handling HTTP requests, redirections, and error responses. The final task develops a hybrid TCP/UDP guessing game where players compete to guess a random number. TCP ensures reliable player registration and result announcements, while UDP is used for fast-paced gameplay. This project demonstrates the integration of TCP and UDP protocols, providing practical experience in network socket programming and real-time client-server interaction.

Table of Contents

Table of Figures

1. Theory & Procedure

1.1. Network Commands and DNS Tools

Computer networks operate using interconnected routers that forward packets of data from one device to another. Data sent across networks is broken into smaller units called **packets**, which are individually routed toward the destination and reassembled upon arrival. The process by which these packets travel through a sequence of intermediary devices (such as routers and switches) is referred to as **message segmentation** and **multi-hop routing**. The tools and commands used in this task allow users to observe and interact with these underlying network processes.

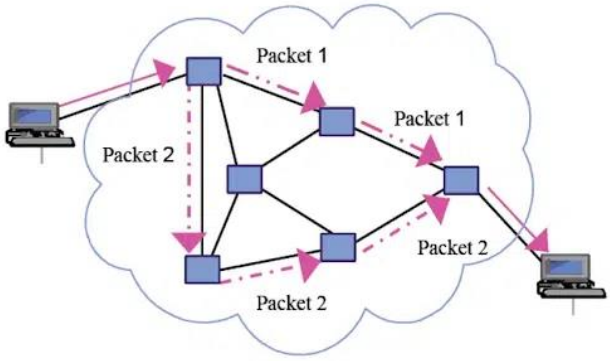


Figure 1: Network packets illustration [1]

Each command used serves a specific diagnostic or informational purpose. Tools like ping and tracert test the reachability and route to remote systems, while ipconfig displays local network configurations. DNS resolution is tested using nslookup, and telnet is used to open direct connections to remote ports, such as HTTP (port 80), for basic testing. Additionally, **Wireshark**, a packet analysis tool, was used to capture and inspect real-time DNS queries and responses at the packet level.

1.1.1. Components & Tools Used

- **ipconfig**: Displays all current IP configuration data including IPv4/IPv6 addresses, subnet mask, default gateway, and DNS server information. This helps verify local network setup and troubleshoot connectivity issues.
- **ping**: Sends ICMP Echo Request packets to a specified IP or domain name and listens for replies. It is used to test basic connectivity and measure round-trip time between devices.
- **tracert**: Reveals the route taken by packets across the internet by showing each router between the source and the destination, along with the delay incurred at each hop.
- **telnet**: Allows the user to open a raw connection to a specified port on a remote server, useful for testing services like HTTP or SMTP. It can also be used to manually send HTTP requests and observe raw server responses.
- **nslookup**: Queries the Domain Name System (DNS) to resolve domain names into IP addresses. It is useful for verifying the existence and accuracy of DNS records.
- **Wireshark**: Wireshark is a widely used, open source network analyzer that can capture and display real-time details of network traffic. It is particularly useful for troubleshooting network issues, analyzing network protocols and ensuring network security. [2]

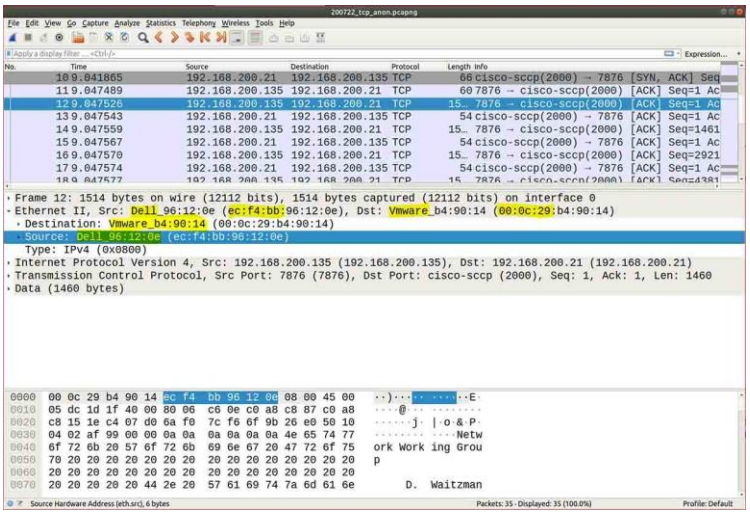


Figure 2: Wireshark Display [2]

All tools were run via the Windows Command Prompt (CMD), while Wireshark was launched separately with filters applied for DNS traffic.

1.1.2. Procedure

1) **Viewing IP Configuration**:

The task began by executing ipconfig /all in the Command Prompt to collect basic network configuration details. These included the system's IP address, subnet mask, default gateway, and DNS server — which are necessary to verify local network setup.

2) **Pinging a Local Device**:

The ping command was used to test connectivity to another device on the same Wi-Fi network. Successful replies confirmed that both devices were within the same local subnet and reachable over the LAN.

3) **Pinging an External Server**:

A ping was sent to **gaia.cs.umass.edu** to observe whether it responded and to measure the round-trip time across the internet. This also revealed the TTL (Time To Live) value, which indirectly indicates how many routers (hops) the packet passed through.

4) **Tracing the Route**:

The tracert command was used to list each intermediate router between the local device and gaia.cs.umass.edu. This confirmed the multi-hop nature of internet communication and showed where delays or packet drops could occur.

5) **Performing DNS Lookup**:

The nslookup command was run to retrieve the IP address associated with gaia.cs.umass.edu. This step confirmed that DNS resolution was functioning correctly and identified the authoritative name servers responsible for the domain.

6) **Testing Port 80 with Telnet**:

A Telnet connection was initiated to gaia.cs.umass.edu on port 80. After establishing the

connection, a manual HTTP request (**GET / HTTP/1.1**) was typed to prompt a raw HTTP response from the server. This verified that the server was listening on port 80 and processing basic web traffic.

7) **Capturing DNS Packets in Wireshark**:

Before starting the capture, the DNS cache was flushed using ipconfig /flushdns to ensure that a real query would be made. Wireshark was then launched with the display filter dns applied. A DNS request to gaia.cs.umass.edu was triggered, and both the **query** and **response** packets were successfully captured. The IP address resolved in the response matched previous results, confirming consistency.

1.2. Web Server Using Socket Programming

Task 2 required building a basic HTTP web server using TCP socket programming, which we chose to do using Python. This server follows the client-server model, where a server continuously listens for client connections and responds with the appropriate HTML content, images, or redirection instructions. The web browser acts as the client, sending HTTP requests, which the server receives and processes based on the request's structure and path.

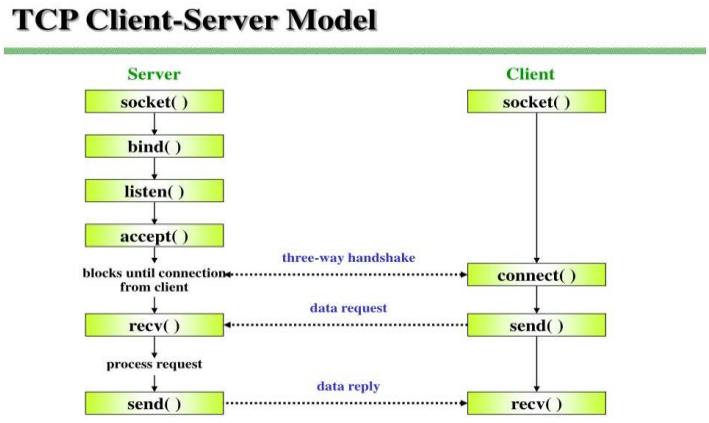


Figure 3: Client-Server Architecture TCP socket programming [3]

Web Server & HTTP:

A web server is software that listens for incoming **HTTP requests** on a specified port, processes them, and responds with content or an error message. It uses the **Hypertext Transfer Protocol (HTTP)** to handle this exchange. HTTP follows a request-response model: the browser (client) initiates a request by sending a line such as GET /index.html HTTP/1.1, and the server replies with a status line (e.g., HTTP/1.1 200 OK), headers like Content-Type, and the requested content.

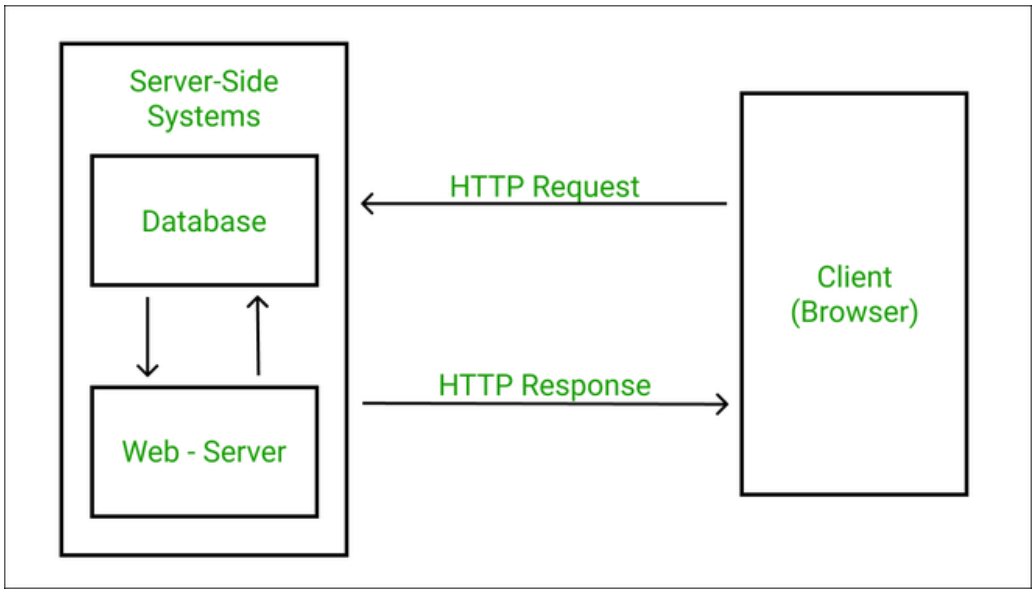


Figure 4: HTTP request response cycle [4]

This task included support for several types of HTTP responses:

- **200 OK:** The requested file was found and delivered.
- **307 Temporary Redirect:** The user was redirected to a Google search for images or videos if the requested file did not exist.
- **400 Bad Request:** Client-side error, occurs when the server can't process the request due to an incorrect or invalid request message.
- **500 Internal Server Error**: The server encountered an unexpected condition or issue that prevented it from fulfilling the request. [5]

TCP Socket Programming:

Sockets are endpoints for sending and receiving data between two machines. The server was built using TCP sockets, which provide reliable, ordered delivery of data. TCP ensures that all bytes sent are received in the correct order, making it ideal for transferring web pages and resources.

To start, the server created a listening socket using **socket(AF_INET, SOCK_STREAM)** and bound it to a port derived from the student ID. Once a client connected, the server read the HTTP request, located the requested file or performed redirection logic, and sent back a response.

Static File Serving:

The server supports multiple static HTML pages:

- **main_en.html**: Displays the main English page, styled with CSS, featuring team member bios, images, and a computer networking topic explanation.
- **mySite_1220031_en.html**: Contains a form for users to request media files(images or videos) from server's files or redirection to Google search.
- **main_ar.html** and **mySite_1220031_ar.html**: Arabic equivalents of the pages above.

CSS was used to improve the visual layout, while .jpg and .png images were embedded into the pages.

HTTP Redirection

In HTTP, redirection is triggered by a server sending a special redirect response to a request. Redirect responses have status codes that start with 3, and a Location header holding the URL to redirect to.[6] If a user entered a keyword in the form (e.g., "IP Spoofing") and selected a type (image or video), the server would respond with the (image or video) saved in it's folders related to the keyword but if the server couldn't find any, it would:

- Parse the request parameters from the URL.

- Generate a 307 Temporary Redirect pointing to a Google search for that media type.

This demonstrated dynamic request handling and URL parsing using only sockets and string operations.

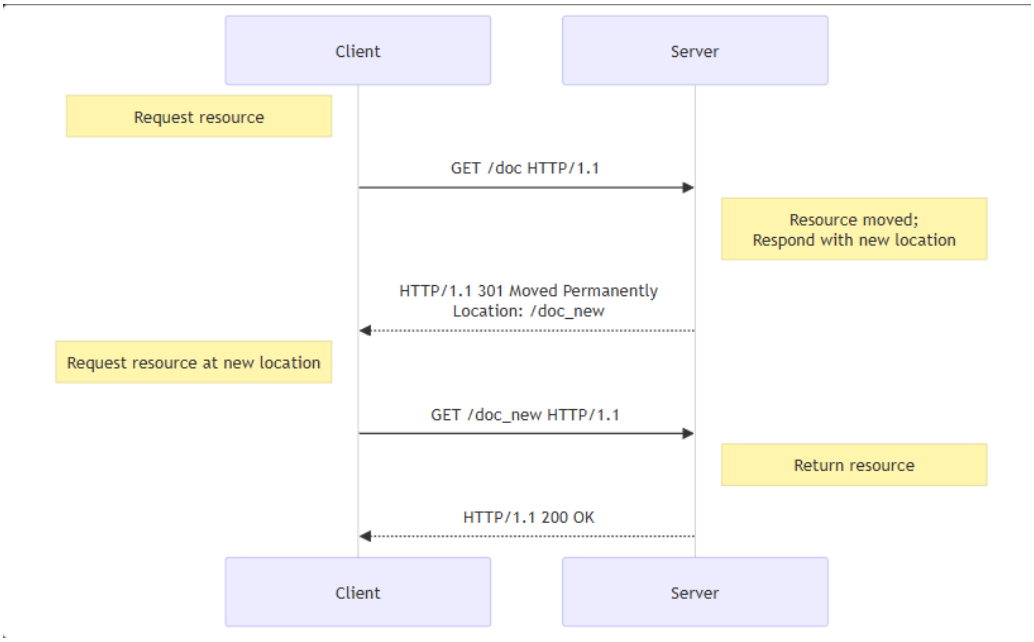


Figure 5: Redirections in HTTP

1.2.1. Components and Technologies Used

- **Python socket module:** Used to create and manage the server socket for handling client connections.
- **HTML files:** Include structured content and links, both in English and Arabic versions.
- **CSS:** Used to apply styles such as colors, font sizes, and layout boxes.
- **HTTP Protocol:** The core protocol used to handle requests and return content or redirect users.
- **Redirection logic:** Custom logic implemented in Python to analyze GET parameters and issue proper redirect responses.
- Content type handling: The server sends appropriate Content-Type headers based on file extensions like .html, .css, .jpg, .png.
- **Logging:** Every request, response, and error is printed to the terminal, including client and server port numbers.

1.2.2. Procedure

1) **Initializing the Server**

The server begins by creating a TCP socket and binding it to port **9901**. It starts listening for incoming connections using listen().

2) **Accepting Connections and Receiving Requests**

When a client connects, the server accepts the socket and reads the HTTP request using recv(). If the request is valid, it proceeds to parse the request line.

3) **Routing and Content Lookup**

a) Requests to /, /en, /index.html, or /main_en.html are routed to main_en.html.

b) Requests to /ar or /main_ar.html are routed to the Arabic version.

c) Other file types (e.g., .css, .jpg) are located and served from the file system.

d) If the file path contains file-name= and file-type=, the parameters are extracted and a **307 redirect** is issued to Google search for that keyword.

4) **Serving Files and Redirecting**

The server constructs the HTTP response by:

a) Sending a status line (200 OK, 307 Redirect, or 500 Internal Server Error)

b) Setting the Content-Type header based on the file type

c) Sending the file content or redirect location accordingly

5) **Handling Errors and Logging**

If a file is missing or invalid, a simple HTML page with error code **500** is returned. Each request, along with its method, requested path, client IP, and response code, is printed to the terminal.

1.3. TCP/UDP Hybrid Client-Server Game Using Socket Programming

This task involves creating a **hybrid networked guessing game** where both **TCP** and **UDP** protocols are utilized to achieve reliable setup and fast-paced gameplay. The game operates as a **client-server model** where the server handles game logic and communication with multiple players.

TCP for Reliable Setup and Control

Transmission Control Protocol (TCP) is a connection-oriented protocol that ensures reliable, ordered, and error-checked delivery of data between the client and server. TCP is suitable for:

- **Player registration**: Players connect to the server using TCP to establish a stable communication channel.
- **Game setup**: The server sends rules, start messages, and game results through TCP to ensure every player receives the complete message.
- **Results announcement**: After a round, the server informs all clients of the winner via TCP.

The **reliable nature of TCP** makes it ideal for control messages that must be received correctly, such as player names, start signals, and final scores.

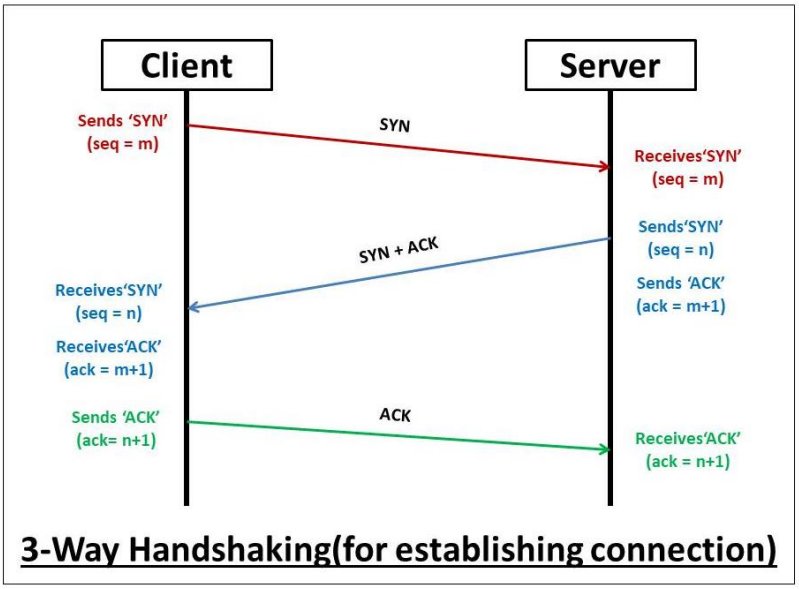


Figure 6 : TCP handshake for player registration [7]

UDP for Fast-Paced Gameplay

User Datagram Protocol (UDP) is a connectionless protocol known for its low latency. It is used for:

- **Guess submissions**: Players send their guesses via UDP to minimize delay.
- **Game feedback**: The server responds with hints ("Higher", "Lower", "Correct") using UDP for quick updates.

UDP is chosen because:

1. Real-time responses are crucial for gameplay.
2. Minimal delay improves user experience.
3. Occasional packet loss is acceptable, as minor misses in feedback do not critically affect gameplay.

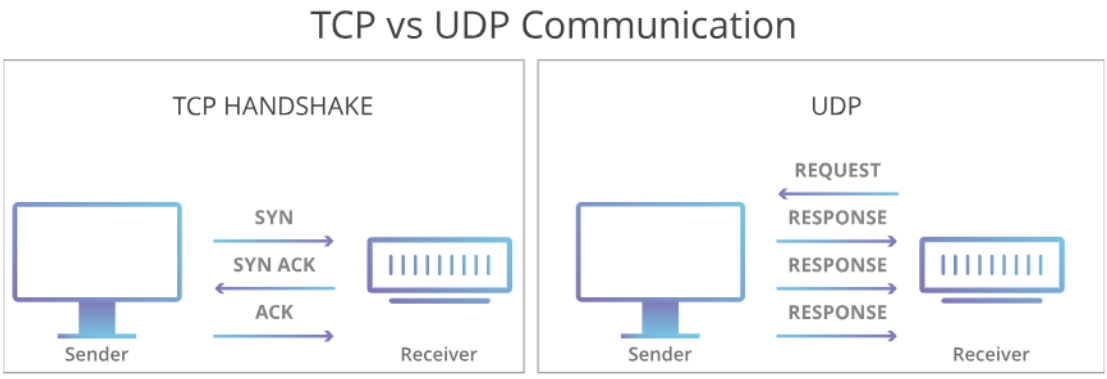


Figure 7: TCP vs UDP communication

1.3.1. Components and Technologies Used

- TCP Connection (Port 6000): Establishes initial connection for registration and game setup.
- UDP Connection (Port 6001): Handles fast-paced guessing during the game.
- Python socket library: Used to create and manage both TCP and UDP sockets.
- Multithreading: Allows multiple players to participate simultaneously.
- Random number generator: Generates the secret number to be guessed by players
- Player registration logic: Ensures that all players have unique names.
- Game round manager: Controls the flow of rounds and initiates new ones after a win.
- Client modules: Each player uses a client script to connect, register, and send guesses.

1.3.2. Procedure

1) **Server Initialization**:

a) The server initializes by creating a TCP socket for control and a UDP socket for gameplay.

b) It binds the TCP socket to port **6000** and the UDP socket to port **6001**.

c) The server starts listening for player connections.

2) **Player Registration (TCP Phase)**:

a) Clients connect to the server via TCP.

b) Players send a **JOIN** command with their username.

c) The server checks if the username is unique:

   i) If yes, it acknowledges the registration.

   ii) If no, it prompts the player to choose another name.

d) After sufficient players have joined, the server broadcasts the game start message.

3) **Game Start and Guessing Phase (UDP Phase)**:

a) The server generates a random secret number.

b) It sends the start message, including the guessing range (e.g., 1 to 100).

c) Players send guesses via UDP.

d) The server evaluates each guess:

   i) Sends **"Higher"** if the guess is too low.

   ii) Sends **"Lower"** if the guess is too high.

   iii) Sends **"Correct"** if the guess matches the secret number.

e) The first player to guess correctly is declared the **winner**.

4) **Victory and Game completeness**:

a) The server sends the winning message to all players via TCP, announcing the winner.

b) The game ends, connections closed.

5) **Handling Disconnections and Errors**:

a) The server monitors player connections.

b) If a player disconnects unexpectedly, the server updates the player list.

c) In case of connection issues, the server handles errors gracefully and continues running for other players.

2. Results & Discussion

2.1. Task 1 – Network Commands and Wireshark

A. Briefly explain each of the following commands in your own words, using no more than two sentences per command:

a. ipconfig:

Shows the computer's configuration such as its IP address, subnet mask, and default gateway IP address. It helps check if we're connected to a network correctly.

b. ping:

Sends a signal to another device to see if it replies. It sends small packets and tells us how long it takes for them to come back.

c. tracert/traceroute:

This command shows the exact path the data takes to reach a destination, hop by hop. It helps identify delays or issues at any point along the route.

d. telnet:

Connects to a website or server using a port number. It's used to see if that port is open and working.

e. nslookup:

This command looks up the IP address behind a domain name. It's useful for troubleshooting DNS issues.

B. Network Commands

a. Execute the ipconfig /all command on your computer and locate the IP address, subnet mask, default gateway, and DNS server addresses for your main network interface:

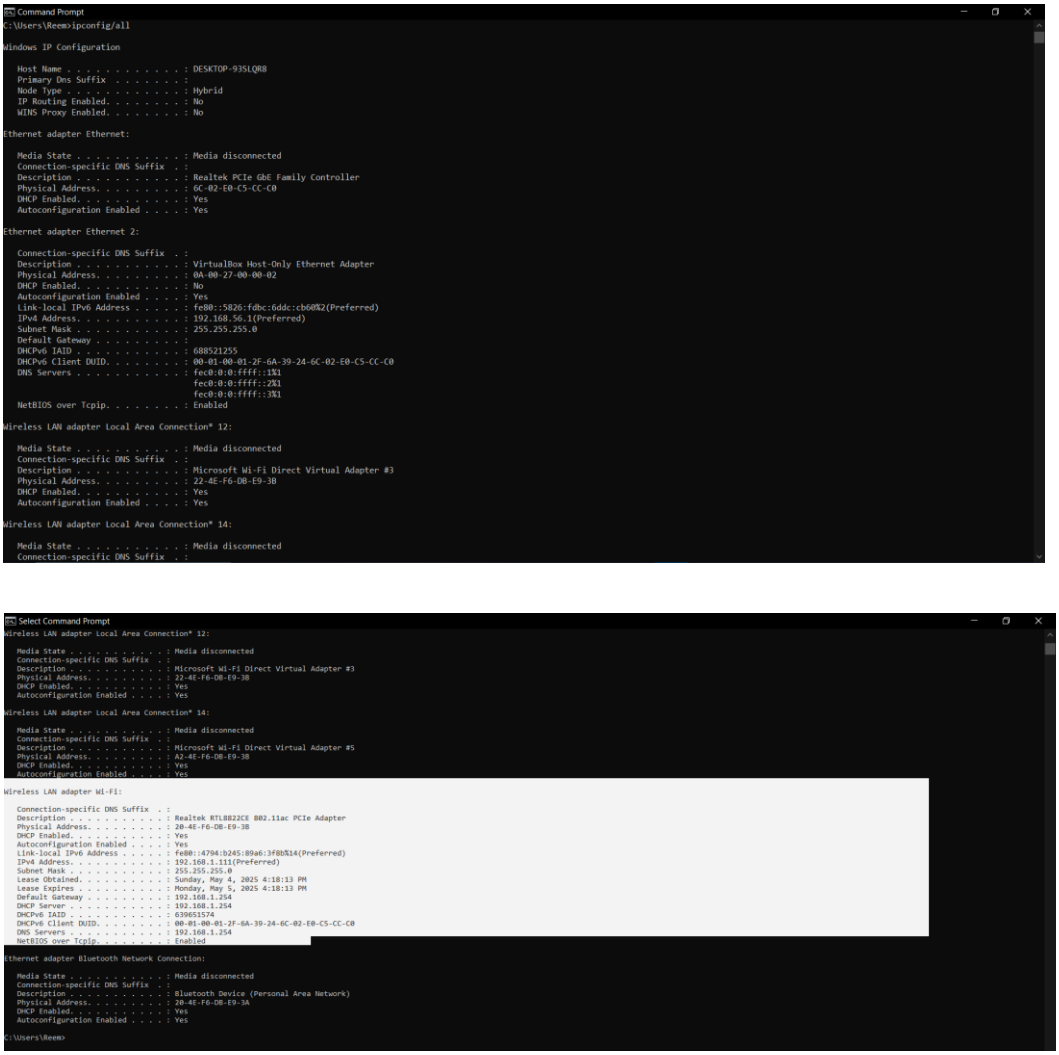


Figure 8: Task 1 ipconfig/all command

After typing **ipconfig/all**, we can see the following details under the active Wi-Fi adapter:

- **IPv4 Address**: 192.168.1.111
- **Subnet Mask**: 255.255.255.0
- **Default Gateway**: 192.168.1.254
- **DNS Server**: 192.168.1.254

These values show that the device is connected to a local Wi-Fi network with an IP in the private 192.168.x.x range. The default gateway and DNS server are both set to the router's IP address, meaning internet traffic and name resolution are both handled by the router.

b. Send a ping request to a device within your local network, such as from a laptop to a smartphone connected to the same wireless network:

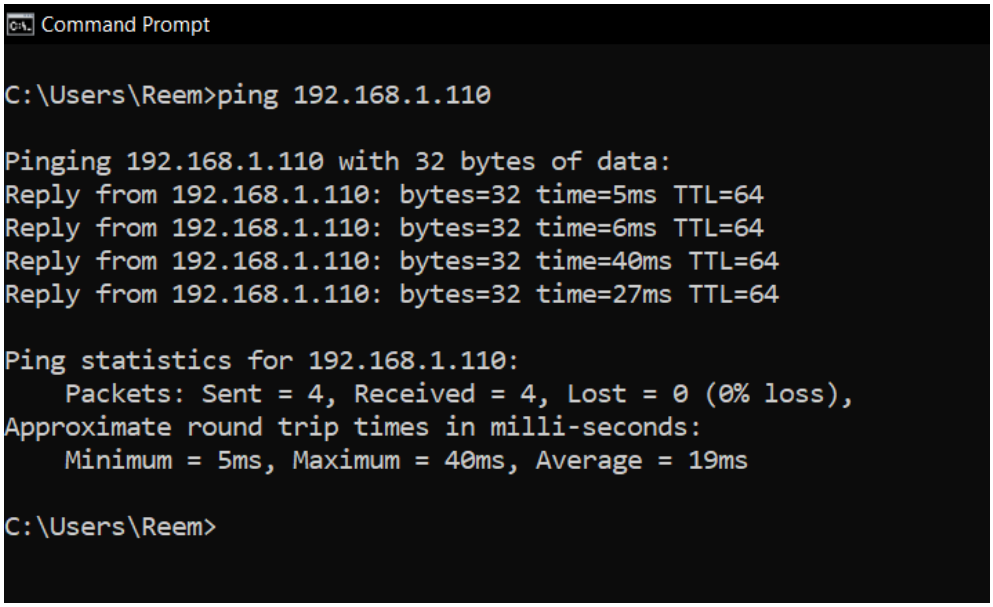


Figure 9: Task 1 ping command

The **ping** command was used to check if another device on the local Wi-Fi (a phone) was reachable. The IP address of the device was 192.168.1.110.

All four packets were successfully sent and received, with **0% packet loss**, confirming that the device is online and communicating properly over the local network. The response times were low, with an average of **19 ms**, which is expected for devices connected to the same router.

This shows that the two devices are on the same subnet and can exchange data without any network issues.

c. Ping gaia.cs.umass.edu, and using the results, provide a brief explanation of whether you believe the response originates from:

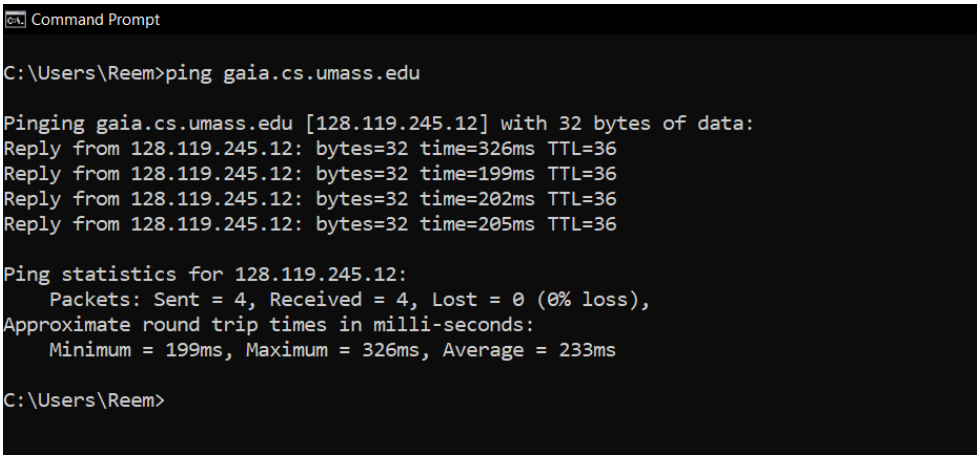


Figure 10: Task 1 ping gaia.cs.umass.edu command

The command successfully contacted **gaia.cs.umass.edu** at IP address 128.119.245.12. All 4 packets were received with 0% loss, and the average response time was around 233 ms. This delay is expected due to the long physical distance between my device and the server (which is located in the U.S.).

Since the replies came from the actual IP 128.119.245.12, it's very likely that the response is from the real server, not a cached or fake result. The consistent TTL value (36) also supports that the packets made it all the way to the destination and back.

By comparing the TTL values in the two ping results — one to a local device and one to a distant server — we can understand how many routers (hops) each packet traveled through. When I pinged my phone on the same local network, the TTL value was high at a value of 64, meaning it reached the device directly or with just 1 or 2 hops. In contrast, the ping to gaia.cs.umass.edu showed a TTL of 36, which suggests the packet passed through around **28 routers** (assuming a starting TTL of 64). This difference confirms that the server is geographically and network-wise much farther away, and TTL gives us a way to estimate that distance based on the number of network hops.

d. Run tracert/traceroute gaia.cs.umass.edu to see the route it takes:

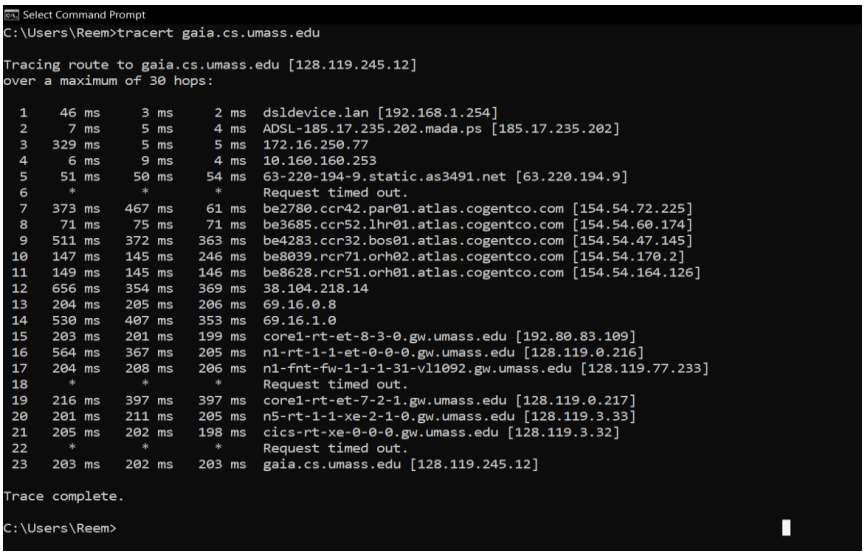


Figure 11: Task 1 tracert command

The **tracert** command revealed the complete path from the computer to the server gaia.cs.umass.edu, passing through **23 routers** (or hops). This aligns closely with the earlier ping TTL value of 36, which had suggested about 28 hops if the starting TTL was 64 — so the actual number of hops makes sense.

Each line in the trace shows a stop the packet made, including the hostname (when available), IP address, and the round-trip time (in milliseconds) for each of the 3 attempts.

Notable observations:

- The first few hops are inside my local network (192.168.1.254) and ISP infrastructure (185.17.235.202).
- Several hops in the middle are through Cogent Communications (e.g., .par01, .lhr01, .bos01), a major Tier 1 ISP.
- A few lines show Request timed out. which is normal — it usually means that the router is set to ignore trace packets for security or load reasons.
- The last few hops clearly show routers inside the University of Massachusetts network (e.g., gw.umass.edu, cics-rt-xe.).

- Final IP: 128.119.245.12 — the same IP we saw from the ping.

This trace confirms that the server is indeed reachable, follows a logical internet path through international networks, and ends within the university's own infrastructure.

e. Execute the nslookup command to retrieve the Domain Name System (DNS) information for gaia.cs.umass.edu:

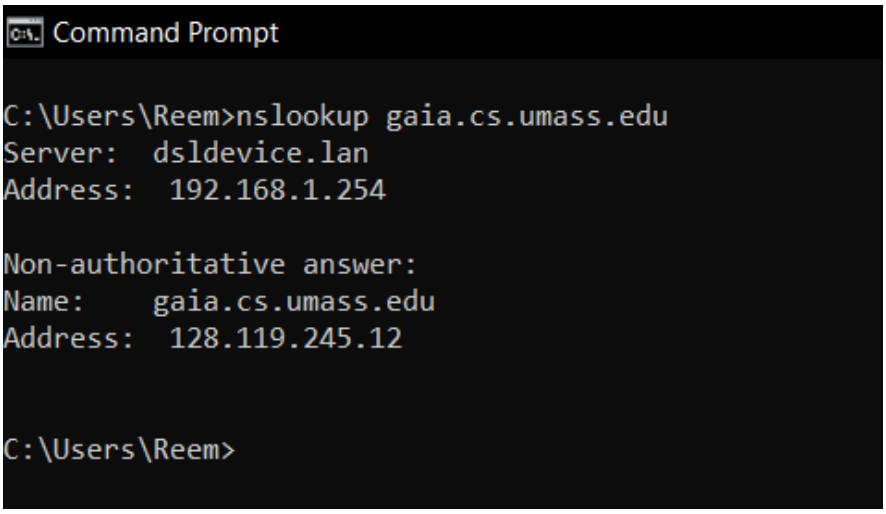


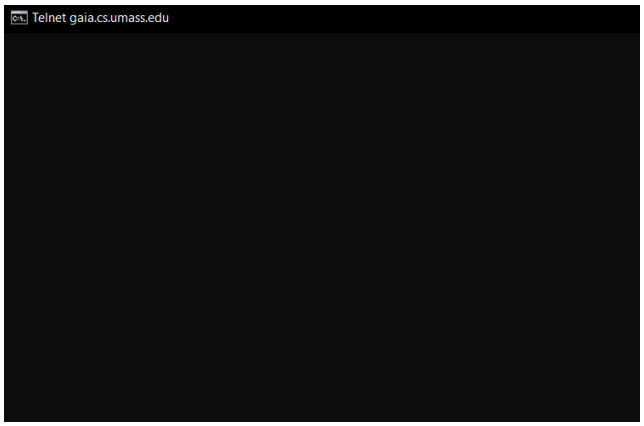
Figure 12: Task 1 nslookup command

The **nslookup** command was used to resolve the domain name gaia.cs.umass.edu into its corresponding IP address. The DNS query was handled by my local router (**192.168.1.254**), which is acting as the DNS server.

The result showed a **non-authoritative answer**, meaning the response came from a cached record or forwarded request — not directly from the original DNS zone hosting the domain. Still, the lookup successfully returned the correct IP address: **128.119.245.12**, which matches the one we observed in both ping and tracert.

This confirms that DNS resolution is working correctly, and the system is able to translate domain names into IP addresses using the local network's DNS setup.

f. Use telnet to try connecting to gaia.cs.umass.edu at port 80:



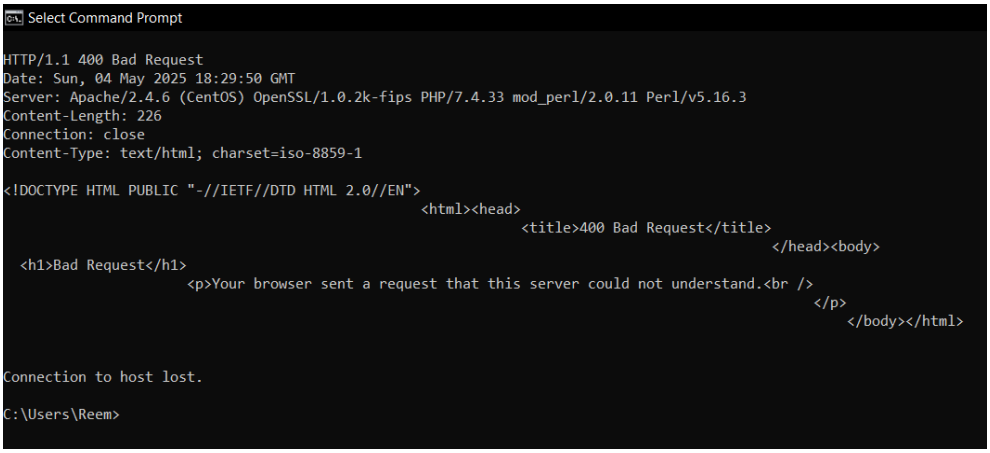


Figure 13: Task 1 telnet command

A **Telnet** connection was established to gaia.cs.umass.edu on port 80, and a manual HTTP request was submitted. The server responded with a **400 Bad Request**, which indicates that the server received the connection but could not understand the malformed or incomplete request.

This type of response is expected when no valid HTTP headers are provided in a request. Despite the error, the reply confirms that:

- The HTTP server at gaia.cs.umass.edu is actively running.
- Port 80 is open and reachable.

- The server correctly handles incoming requests using the Apache web server on a CentOS system.

The full HTML response header includes server details, content type, and connection status, demonstrating that HTTP communication with the server is functioning as intended.

g. Provide details about the autonomous system (AS) number, number of IP addresses, prefixes, peers, and the name of Tier 1 ISP(s) associated with gaia.cs.umass.edu:

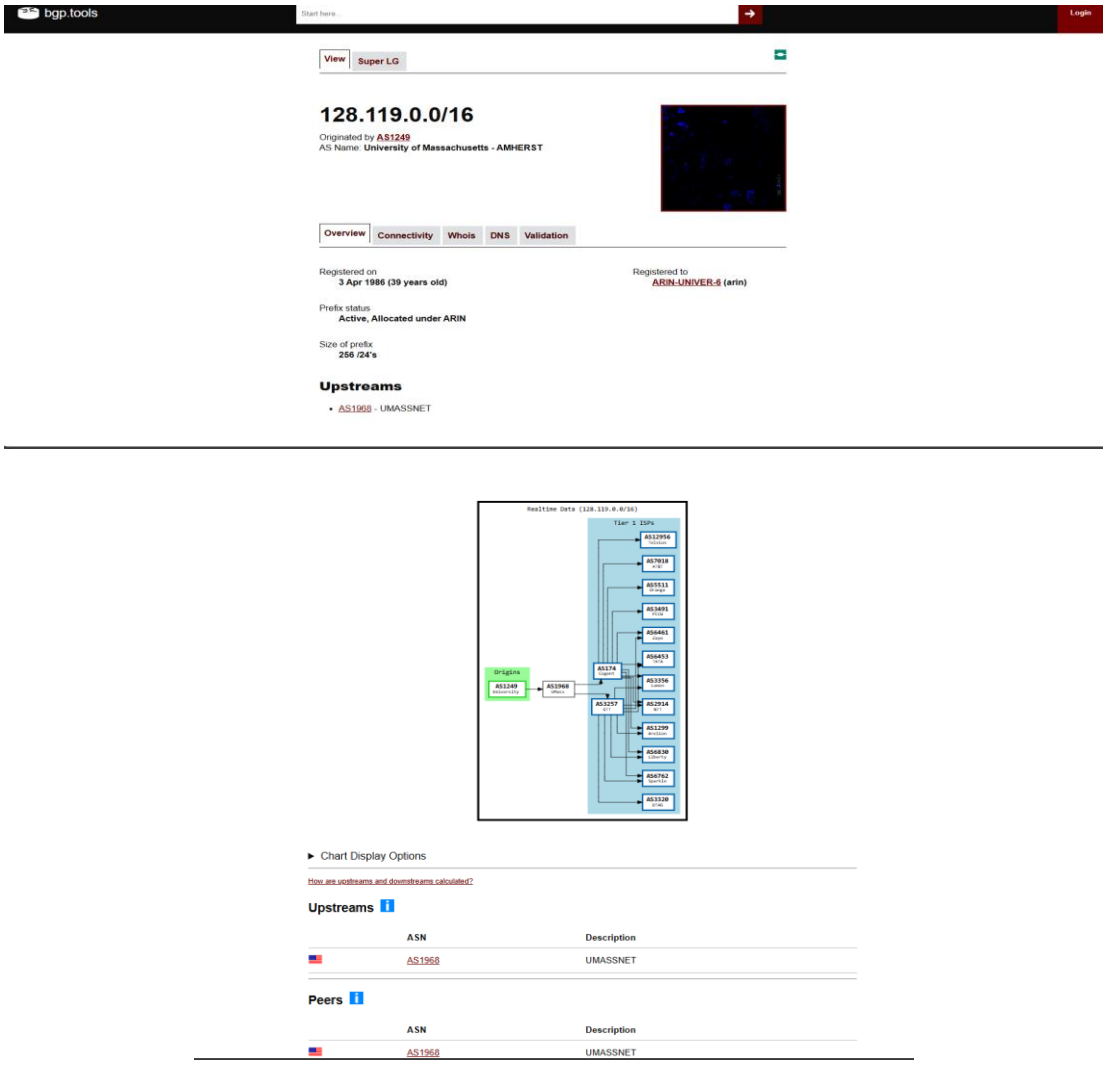


Figure 14: Task 1 AS information

Using the online tool bgp.tools, the following details were gathered about the IP address **128.119.245.12**, which is associated with gaia.cs.umass.edu.

- **Autonomous System (AS) Number**: **AS1249**

  This identifies the University of Massachusetts - Amherst as the network owner and operator of the IP address.
- **Number of IP Addresses**: The prefix is **128.119.0.0/16,** which includes **65,536 IP addresses**.

  This represents the total address space assigned to the AS.
- **Prefixes**: The AS originates **256 /24 prefixes**, meaning it announces 256 subnet blocks of 256 addresses each.
- **Peers**: The AS has one direct upstream peer: **AS1968 (UMASSNET**), which handles its external routing.
- **Tier 1 ISPs**: Through UMASSNET, the AS is connected to several Tier 1 ISPs, including Cogent (AS174), AT&T (AS7018), Zayo (AS6461), and Lumen (AS3356).

  These Tier 1 providers ensure global routing without relying on transit from other networks.

C. Wireshark

This information confirms the institutional ownership, routing capacity, and connectivity ofgaia.cs.umass.edu.

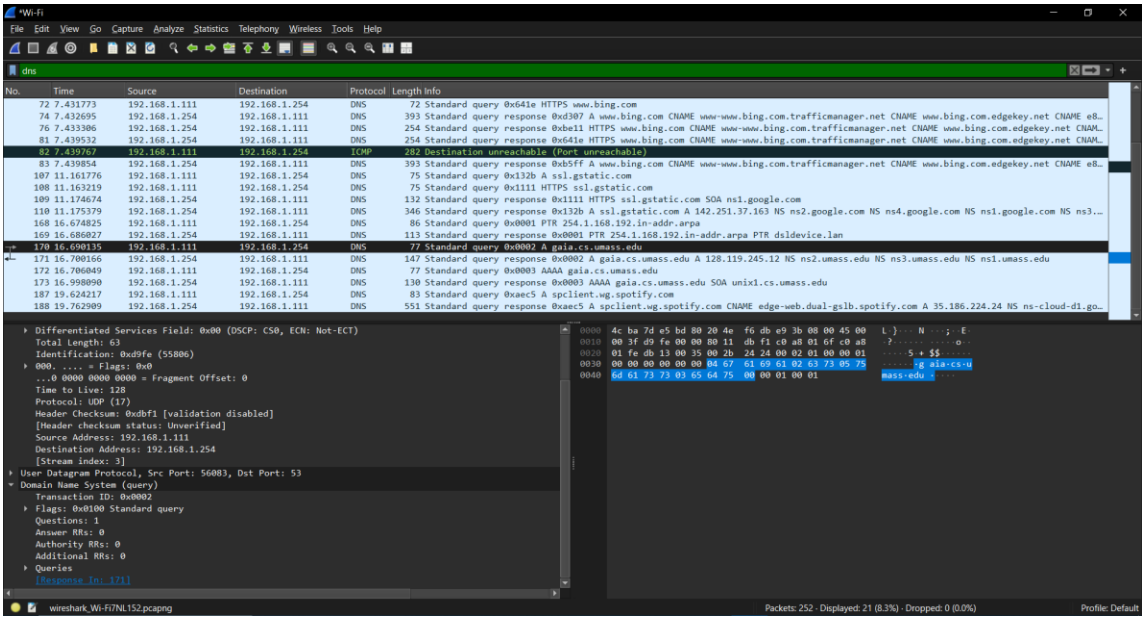


Figure 15: Task 1 Wireshark output

Wireshark was used to capture the DNS resolution process for the domain gaia.cs.umass.edu. Before capturing, the local DNS cache was flushed to ensure that a fresh query would be generated. The display filter dns was applied to isolate DNS-related packets during the capture session.

The screenshot shows a **standard DNS query** packet from the local machine (192.168.1.111) to the local DNS server (192.168.1.254) requesting the A record for gaia.cs.umass.edu. Immediately following that, a **standard DNS response** is visible, returning the IP address 128.119.245.12, along with the authoritative name servers for the domain (**ns1.umass.edu, ns2.umass.edu**, etc.).

This confirms that the DNS resolution process successfully completed and that Wireshark can be effectively used to observe the full query-response cycle for DNS lookups.

2.2. Task 2 – Web Server Using Socket Programming

In this task, we developed a comprehensive web server using Python socket programming. The server operates on port **9901** and follows the client-server model, where the server continuously listens for incoming client connections, handles HTTP requests, and provides the appropriate responses. The server is designed to support HTML pages, CSS styling, PNG and JPEG images, and redirection to external resources if the requested content is not available locally.

The web server was implemented to dynamically handle multiple types of HTTP requests and generate appropriate responses, including 200 OK, 307 Temporary Redirect, and 404 Not Found. Additionally, the server outputs detailed logs of the client requests, including headers, requested resources, and the response sent.

a) Main English Webpage (main_en.html)

The main English webpage is the primary interface of our web server. It is designed with a structured layout, incorporating key web development elements such as headers, paragraphs, tables, and images. The design focuses on both visual appeal and structured content, utilizing CSS for styling to enhance the user experience.

1. **Webpage Title and Header:**
   - The main heading at the top reads "Welcome to ENCS3320 - Computer Networks Webserver" in blue.
2. **Team Member Information:**
   - The page displays a table-like structure where each team member's name, ID, and photo (in .png format) are presented. The layout groups each member's details within separate styled boxes to maintain clarity.
     - Each member's description includes details about projects completed in various courses, skills, and hobbies, providing a comprehensive view of the team's capabilities.

3. **Topic Presentation from Textbook:**
   - A well-organized section explains a topic from Chapter 1 of the course textbook, focusing on Network Security and Attacks.
   - The section contains formatted text with headings and subheadings, ordered and unordered lists, and relevant images (in .jpg format).
   - Text formatting includes bold and italic styles to emphasize key points, while CSS ensures consistent font styles and color schemes.

4. **Navigation and Links:**
   - The page includes hyperlinks to external resources for additional information:
     i. The Kurose & Ross textbook site
     ii. Birzeit University website
     iii. A link to the local page: **mySite_1220031_en.html**

CSS Styling:

- The visual presentation is enhanced using CSS, which defines font colors, borders, table layouts, and background settings.
- The header and subheadings utilize larger fonts with contrasting colors to improve readability, while the image borders create a structured visual flow.

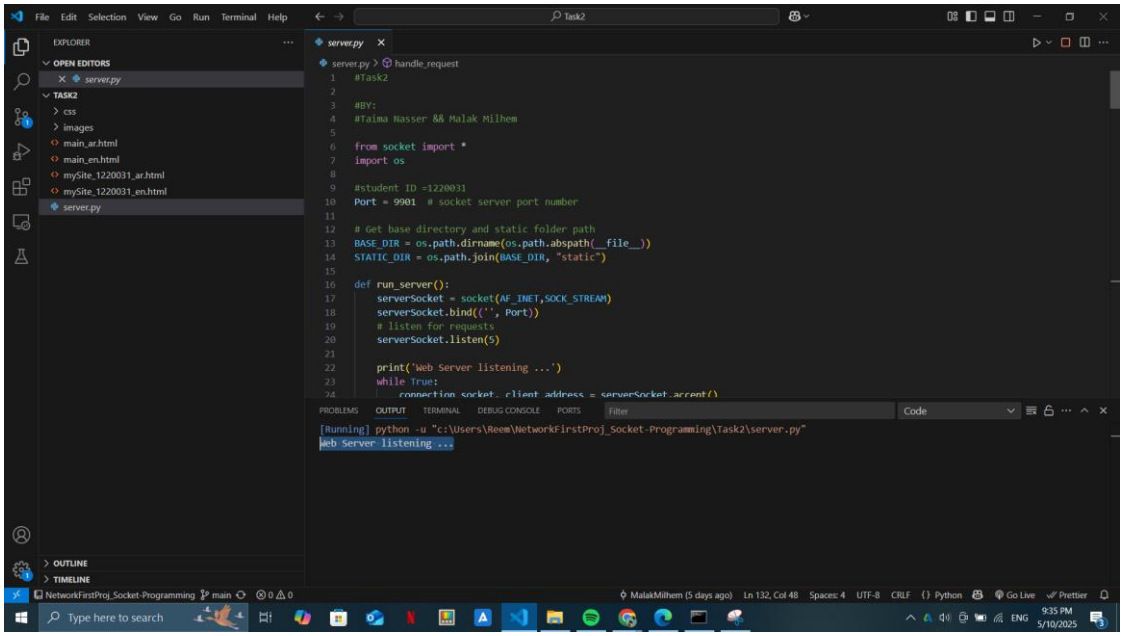


Figure 16: The server running waiting for a connection

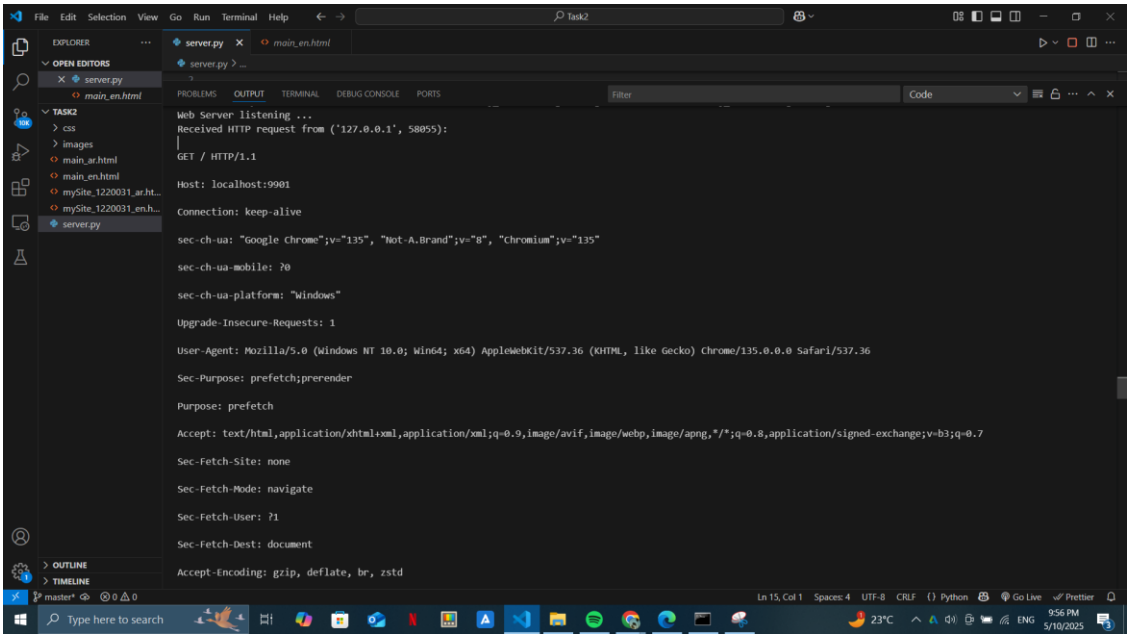
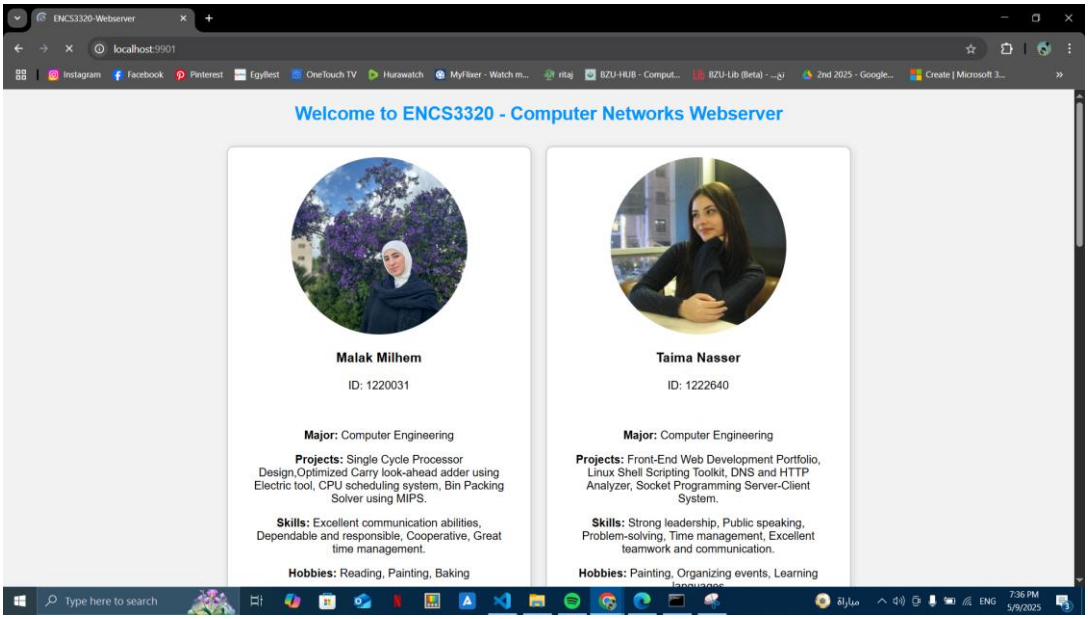
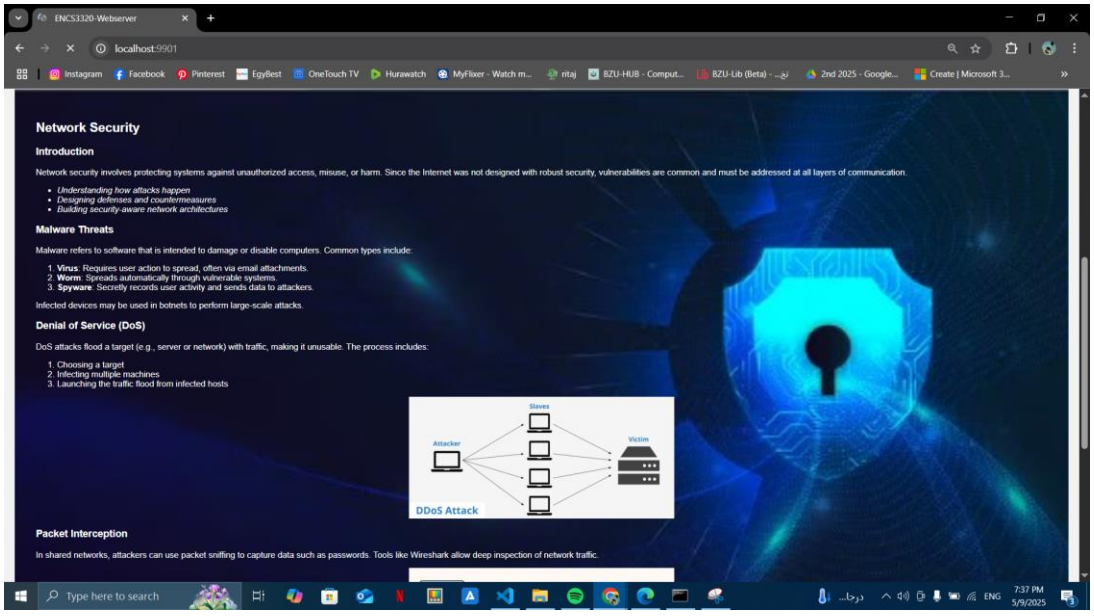


Figure 17: Client making an HTTP request to server by requesting (localhost:9901) and server receiving the request


ENCS3320-Webserver
localhost:9901
Welcome to ENCS3320 - Computer Networks Webserver
Malak Milhem
ID: 1220031
Major: Computer Engineering
Projects: Single Cycle Processor Design,Optimized Carry look-ahead adder using Electric tool, CPU scheduling system, Bin Packing Solver using MIPS.
Skills: Excellent communication abilities, Dependable and responsible, Cooperative, Great time management.
Hobbies: Reading, Painting, Baking
Taima Nasser
ID: 1222640
Major: Computer Engineering
Projects: Front-End Web Development Portfolio, Linux Shell Scripting Toolkit, DNS and HTTP Analyzer, Socket Programming Server-Client System.
Skills: Strong leadership, Public speaking, Problem-solving, Time management, Excellent teamwork and communication.
Hobbies: Painting, Organizing events, Learning


ENCS3320-Webserver
localhost:9901
Network Security
Introduction
Network security involves protecting systems against unauthorized access, misuse, or harm. Since the Internet was not designed with robust security, vulnerabilities are common and must be addressed at all layers of communication.
Understanding how attacks happen
Designing defenses and countermeasures
Building security-aware network architectures
Malware Threats
Malware refers to software that is intended to damage or disable computers. Common types include:
1. Virus: Requires user action to spread, often via email attachments.
2. Worm: Spreads automatically through vulnerable systems.
3. Spyware: Secretly records user activity and sends data to attackers.
Infected devices may be used in botnets to perform large-scale attacks.
Denial of Service (DoS)
DoS attacks flood a target (e.g., server or network) with traffic, making it unusable. The process includes:
1. Choosing a target
2. Infecting multiple machines
3. Launching the traffic flood from infected hosts
Slaves
Attacker
Victim
DDoS Attack
Packet Interception
In shared networks, attackers can use packet sniffing to capture data such as passwords. Tools like Wireshark allow deep inspection of network traffic.

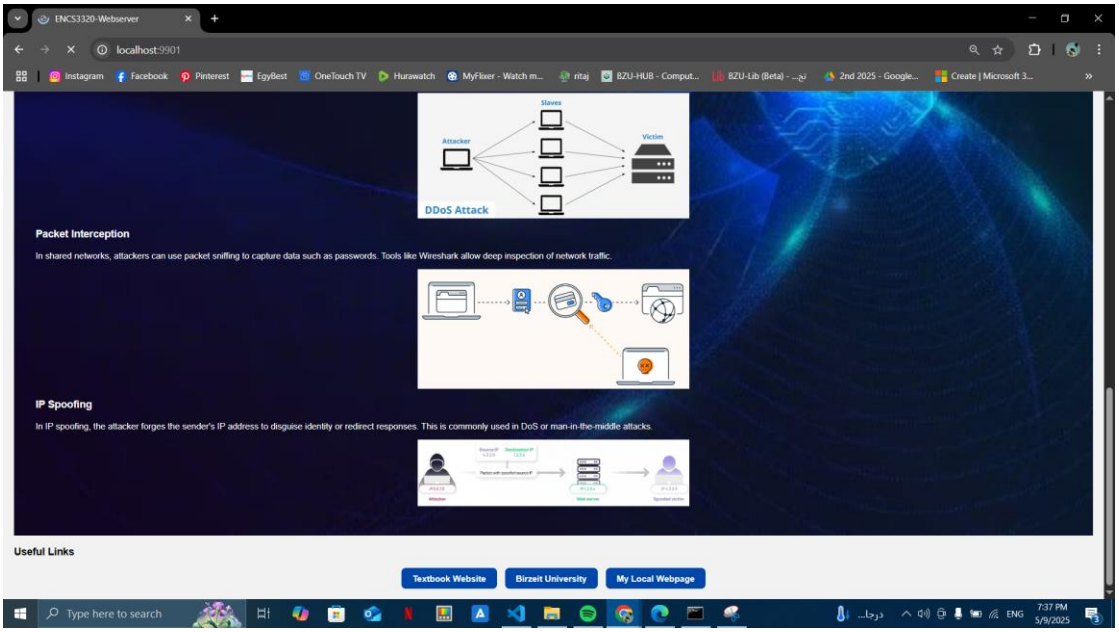


Figure 18: Server redirects the client to main web page successfully when using localhost:9901 .

And if the user request target was any of the following (localhost:9901/, localhost:9901/en, localhost:9901/index.html) the server will respond with the main_en.html.

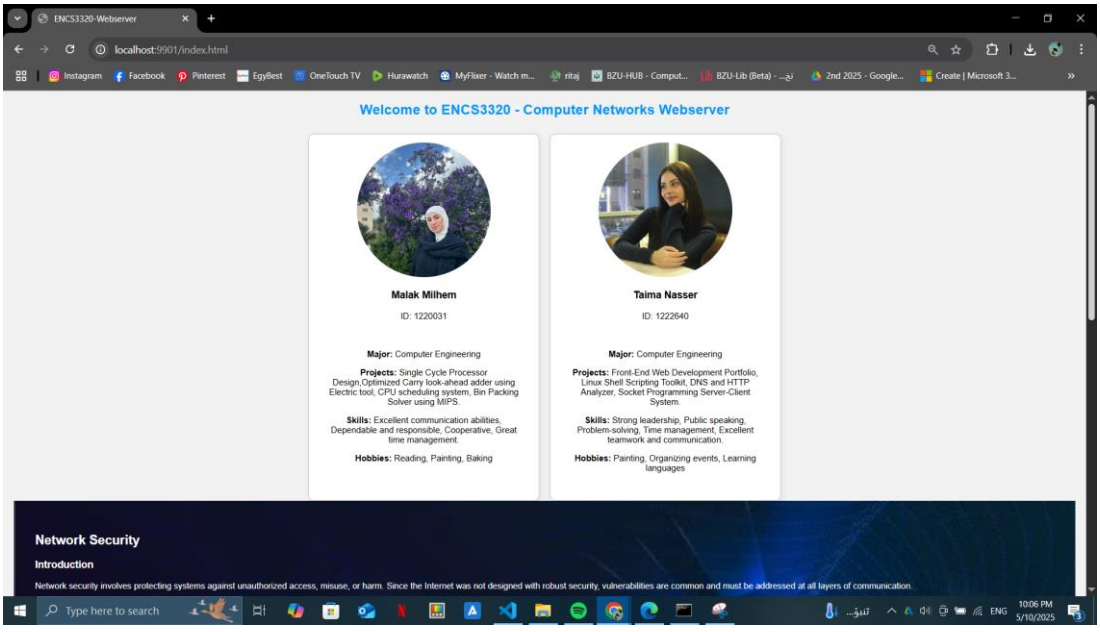


Figure 19: Server redirects the client to main web page successfully using localhost:9901/index.html.

b) Arabic Version (main_ar.html)

To accommodate users who prefer Arabic, we developed a localized version of the main webpage. The Arabic webpage (main_ar.html) mirrors the English page in structure and content but is fully translated into Arabic. And the local Arabic webpage (mySite_1220031_ar.html) functions similarly to its English counterpart, allowing users to search for images or videos related to networking topics.

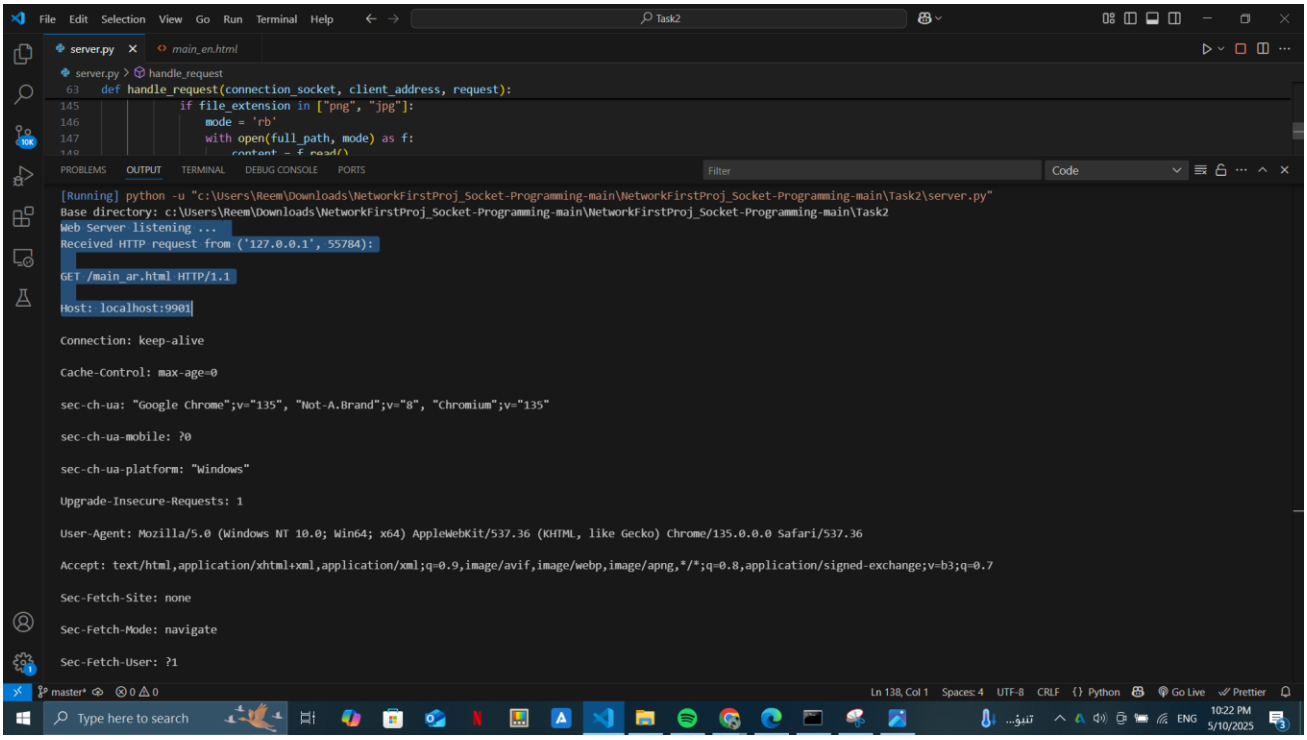
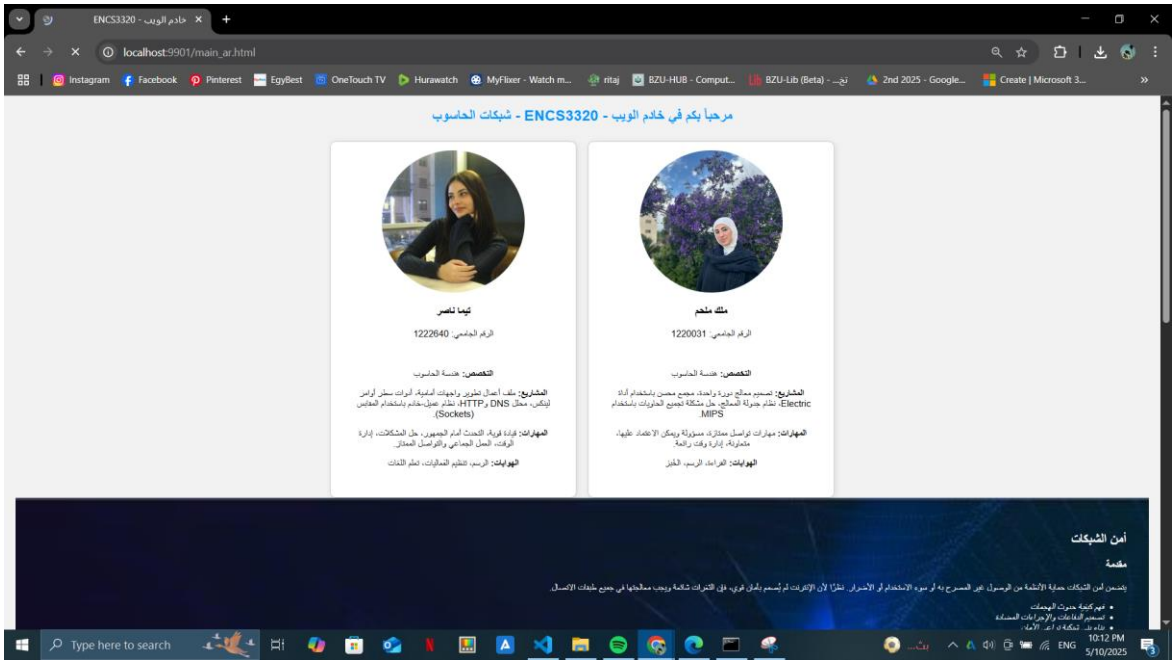
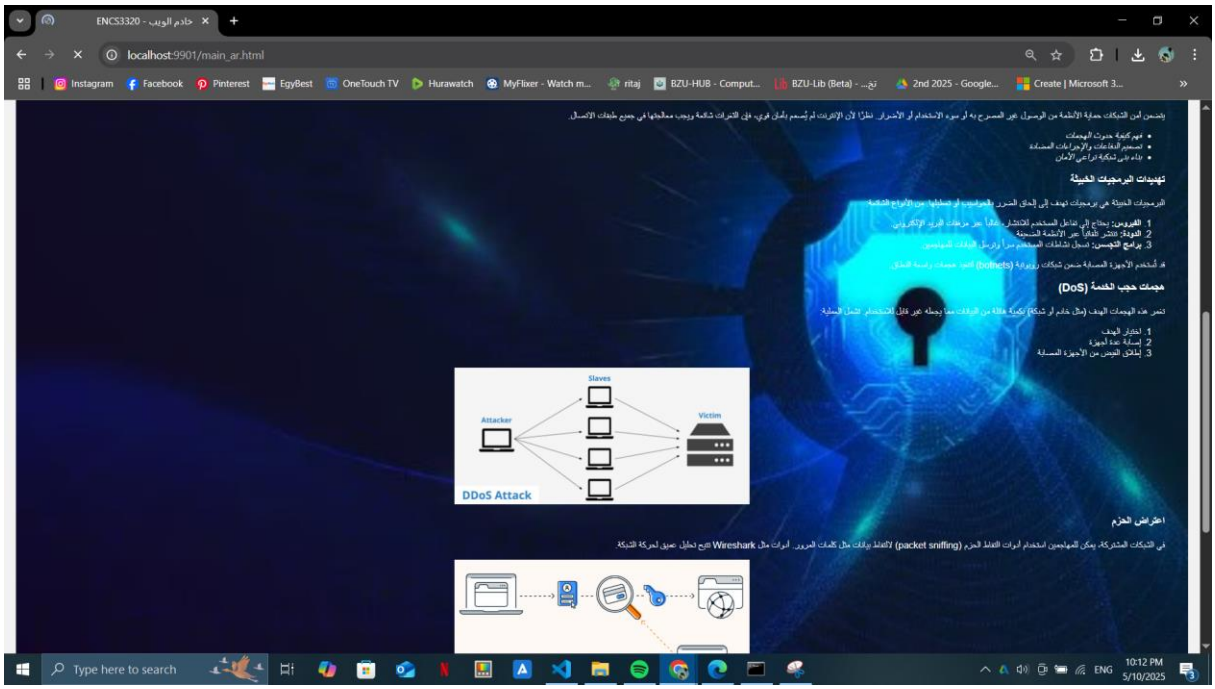


Figure 20: Server receiving (localhost:9901/main_ar.html) request from client


ENCS3320 - خادم الويب
localhost:9901/main_ar.html
مرحباً بكم في خادم الويب - ENCS3320 - شبكات الحاسوب
ملك ملحم
الرقم الجامعي: 1220031
التخصص: هندسة الحاسوب
المشاريع: تصميم معالج دورة واحدة، مجمع مصمن باستخدام أداة Electric، نظام جدولة للمعالج، حل مشكلة لجميع الحزوبات باستخدام MIPS.
المهارات: مهارات تواصل ممتازة، مسؤولة ويمكن الاعتماد عليها، ملعاونة، إدارة وقت رائعة.
الهوايات: القراءة، الرسم، الخبز
نيما ناصر
الرقم الجامعي: 1222640
التخصص: هندسة الحاسوب
المشاريع: ملف أعمال تطوير واجهات أمامية، أدوات سطر أوامر (بنكس، محلل HTTP و DNS، نظام تعويل-خادم باستخدام المقابس (Sockets.
المهارات: قيادة فريق، التحدث أمام الجمهور، حل المشكلات، إدارة الوقت، العمل الجماعي والتواصل الممتاز.
الهوايات: الرسم، تنظيم الفعاليات، تعلم اللغات
أمن الشبكات
مقدمة
يتضمن أمن الشبكات حماية الأنظمة من الوصول غير المصرح به أو سوء الاستخدام أو الأضرار. نظرًا لأن الإنترنت لم يُصمم بأمان قوي، فإن الثغرات شائعة ويجب معالجتها في جميع طبقات الاتصال.
فهم كيفية حدوث الهجمات
تصميم الدفاعات والإجراءات المضادة
بناء بنى تحتية تراعي الأمان
Type here to search
ENG
10:12 PM
5/10/2025


ENCS3320 - خادم الويب
localhost:9901/main_ar.html
يتضمن أمن الشبكات حماية الأنظمة من الوصول غير المصرح به أو سوء الاستخدام أو الأضرار. نظرًا لأن الإنترنت لم يُصمم بأمان قوي، فإن الثغرات شائعة ويجب معالجتها في جميع طبقات الاتصال.
فهم كيفية حدوث الهجمات
تصميم الدفاعات والإجراءات المضادة
بناء بنى تحتية تراعي الأمان
تهديدات البرمجيات الخبيثة
البرمجيات الخبيثة هي برمجيات تهدف إلى إلحاق الضرر بالحواسيب أو عملياتها. من أنواع الشائعة:
1. الفيروس: يحتاج إلى تفاعل المستخدم للانتشار، غالبًا عبر مرفقات البريد الإلكتروني.
2. الدودة: تنتشر تلقائيًا عبر الأنظمة المتصلة.
3. برامج التجسس: تسجل نشاطات المستخدم سرًا وترسل البيانات للمهاجمين.
قد تُستخدم الأجهزة المصابة ضمن شبكات روبوتية (botnets) لشن هجمات رقمية للحق.
هجمات حجب الخدمة (DoS)
تستهدف هذه الهجمات الهدف (مثل خادم أو شبكة) بكمية هائلة من البيانات مما يجعله غير قادر للمستخدم الشرعيين.
1. اختيار الهدف
2. إصابة عدة أجهزة
3. إطلاق الهجوم من الأجهزة المصابة
Slaves
Attacker
Victim
DDoS Attack
اعتراض الحزم
في الشبكات المشتركة، يمكن للمهاجمين استخدام أدوات التقاط الحزم (packet sniffing) لالتقاط بيانات مثل كلمات المرور. أدوات مثل Wireshark تتيح تحليل تصيق لحركة الشبكة
Type here to search
ENG
10:12 PM
5/10/2025

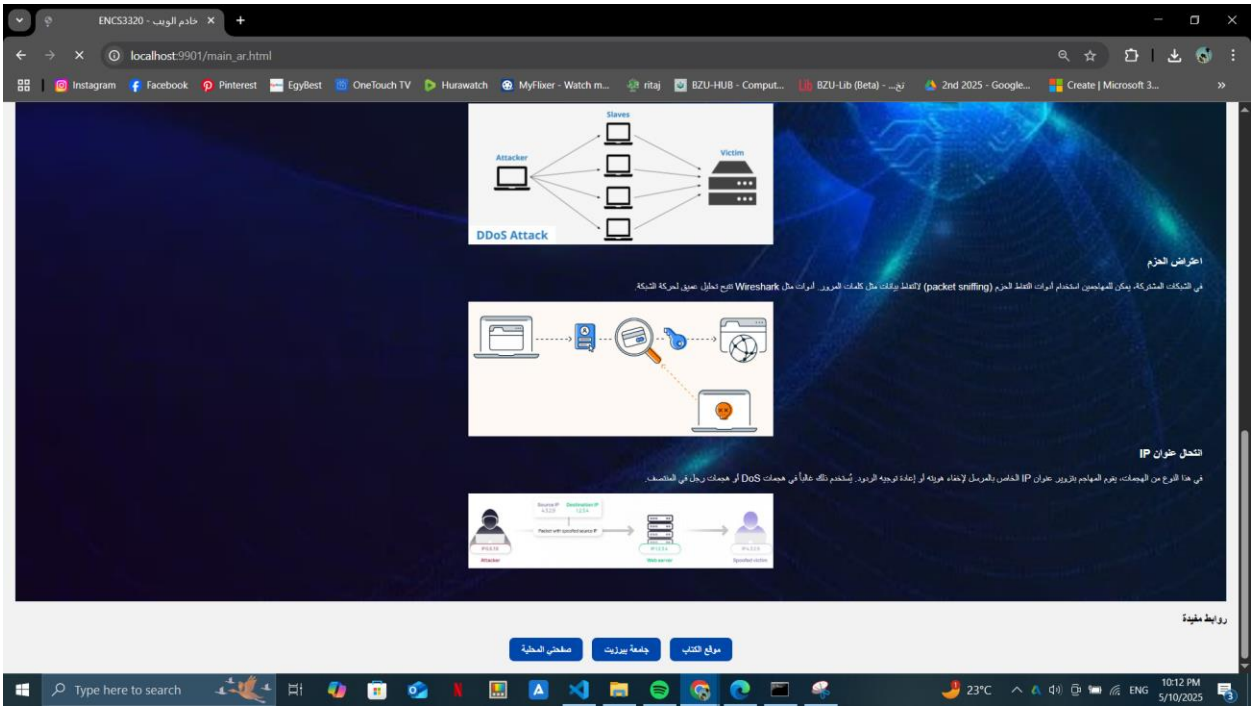


Figure 21 : Server redirects the client to Arabic web page successfully when using (localhost:9901/main_ar.html).

c) Local Webpage (mySite_1220031_en.html)

The local webpage serves as an interactive extension of the main page. Its primary feature is an HTML form that allows users to input a keyword and specify whether they are searching for an image or video related to the computer networking topic discussed on the main page.

Form Functionality:

- The form contains:
  - A text input field where users can enter a keyword (e.g., "IP spoofing").
  - Two radio buttons to specify whether the search is for images or videos.
  - A submit button that triggers the server to process the request.

Server Response:

- Upon submission, the server checks if the requested file exists in the server directory:

- If the file exists, it sends a **200 OK** response and displays the content.
- If the file does not exist, it responds with a 307 Temporary Redirect, leading the client to:
  - Google Images for image requests.
  - Google Videos for video requests.

This mechanism ensures the user always receives relevant results, even if the local server does not host the requested media.

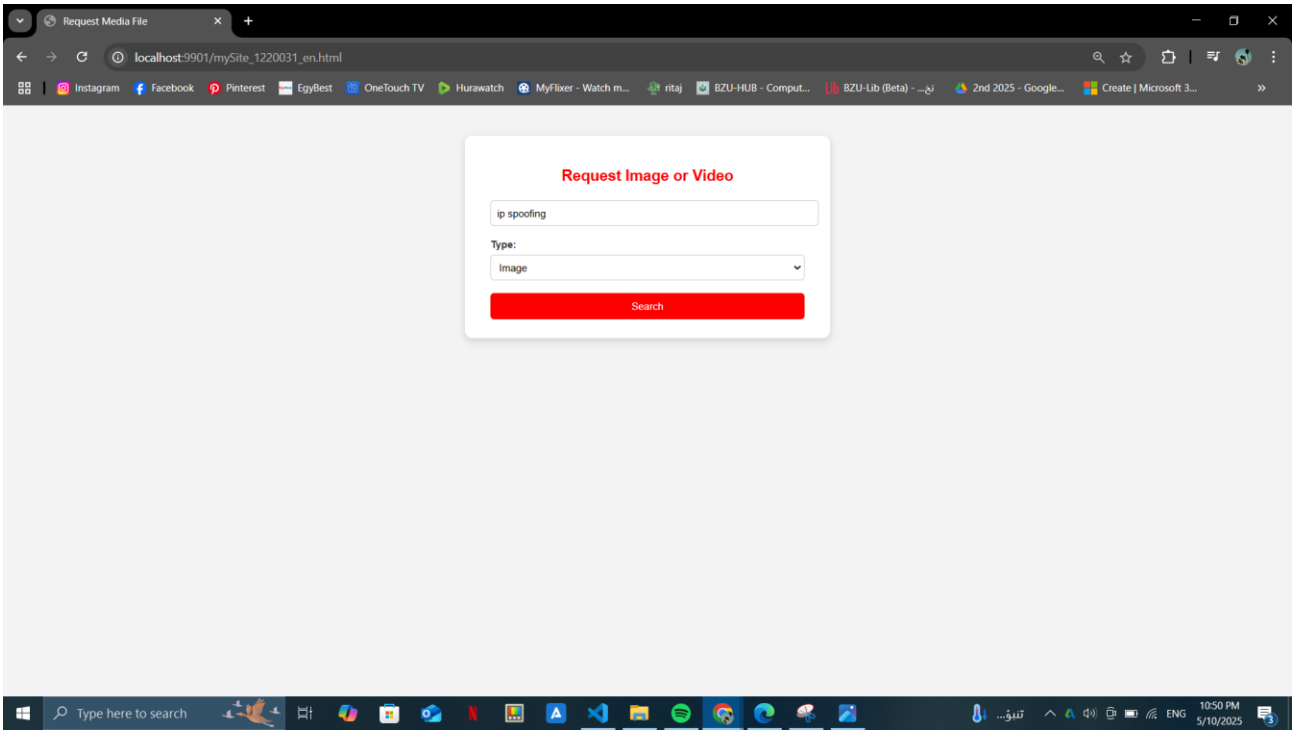


Figure 22 : Client entering a keyword to search for (ex. IP spoofing) requesting an image.

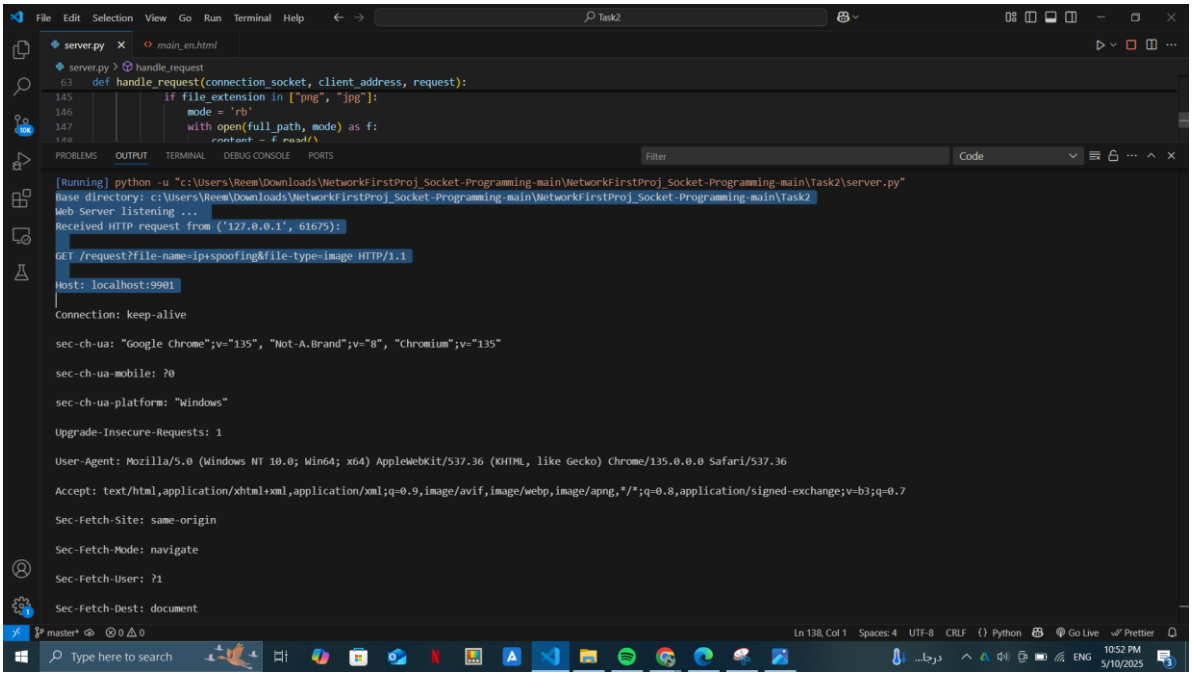


Figure 23: Server receives the request.

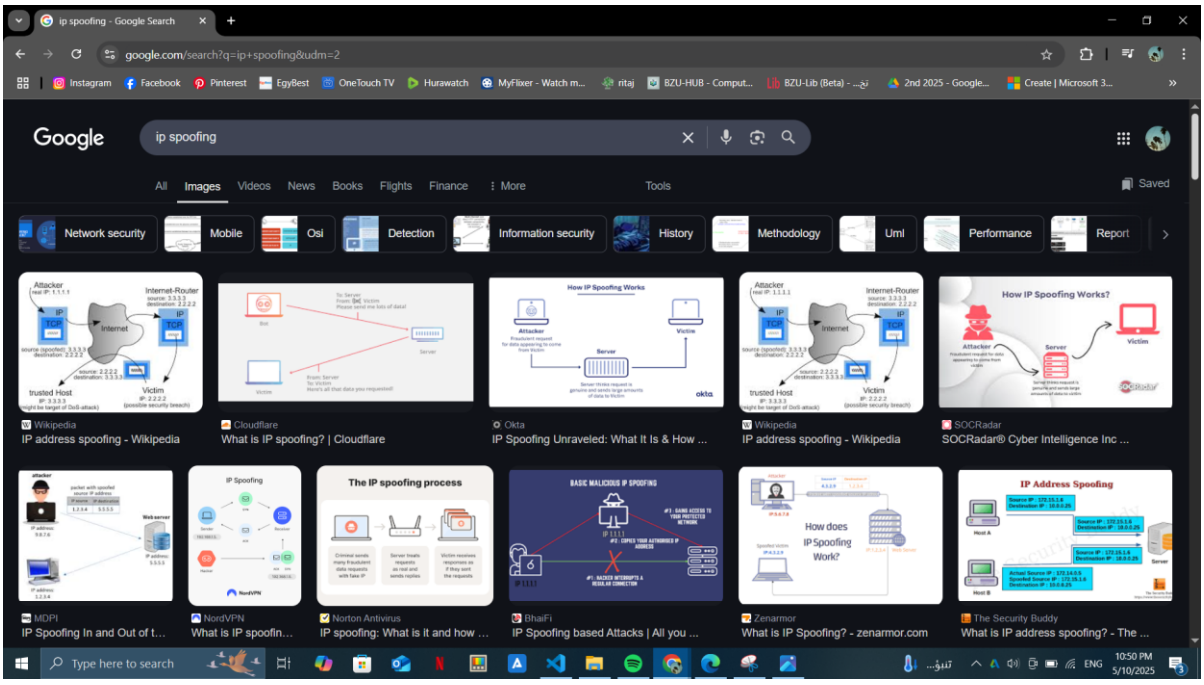


Figure 24: Client successfully redirected.

Now the client requested a video from our local webpage:

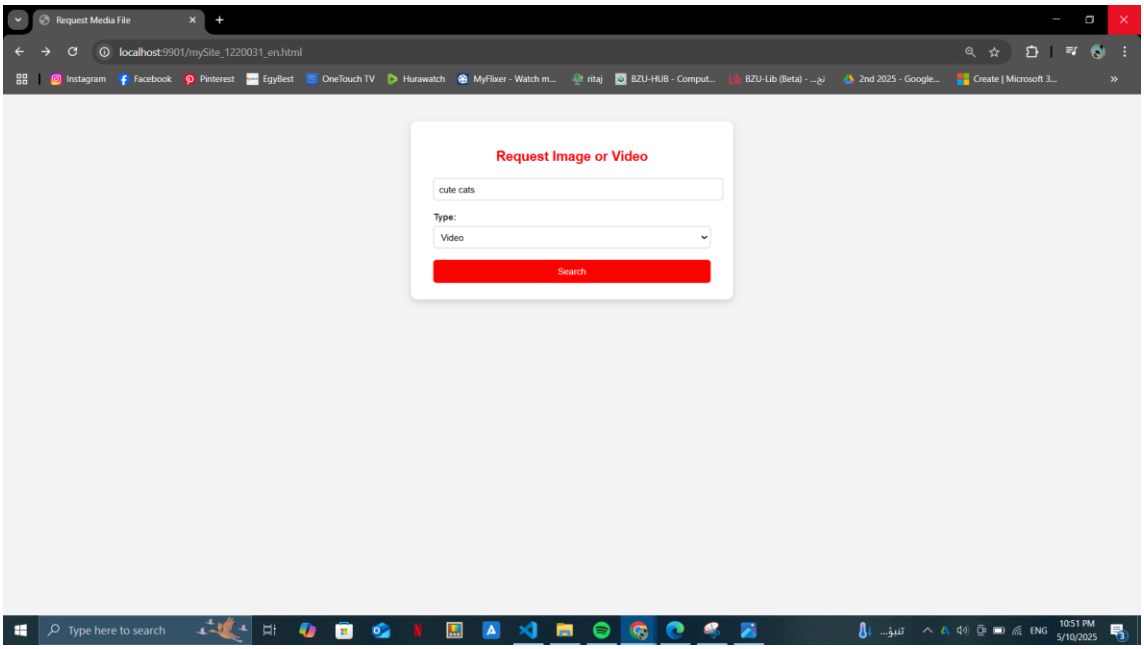


Figure 25: Client entering a keyword to search for (ex. cute cats) requesting a video.

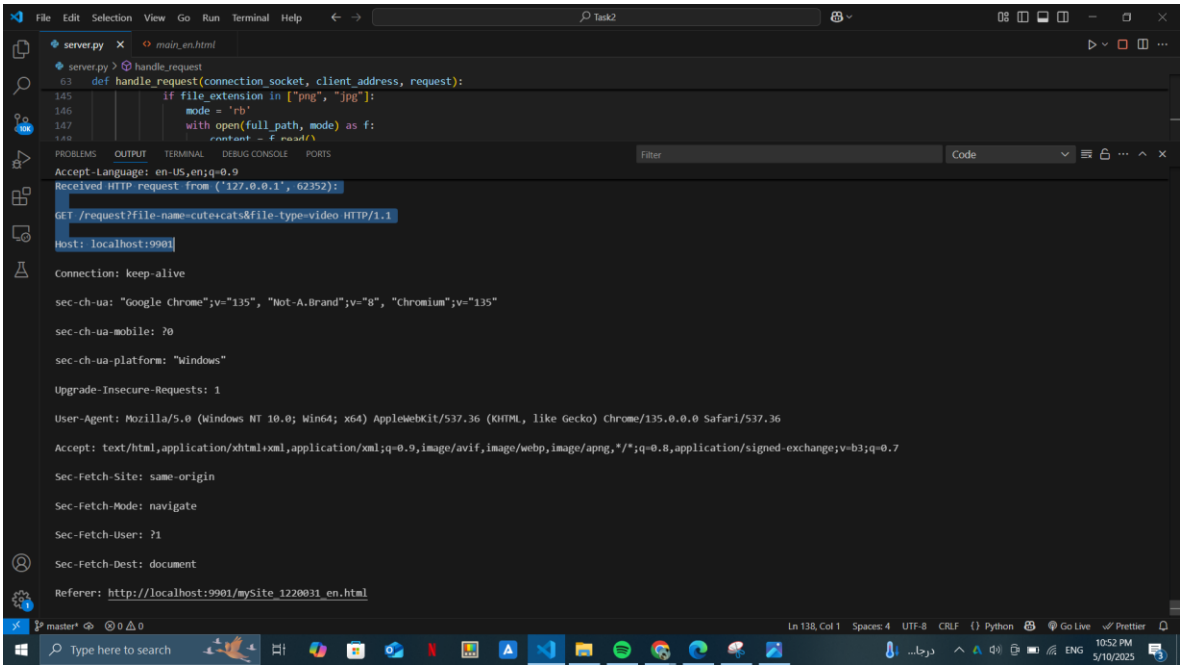


Figure 26: Server receives the response.

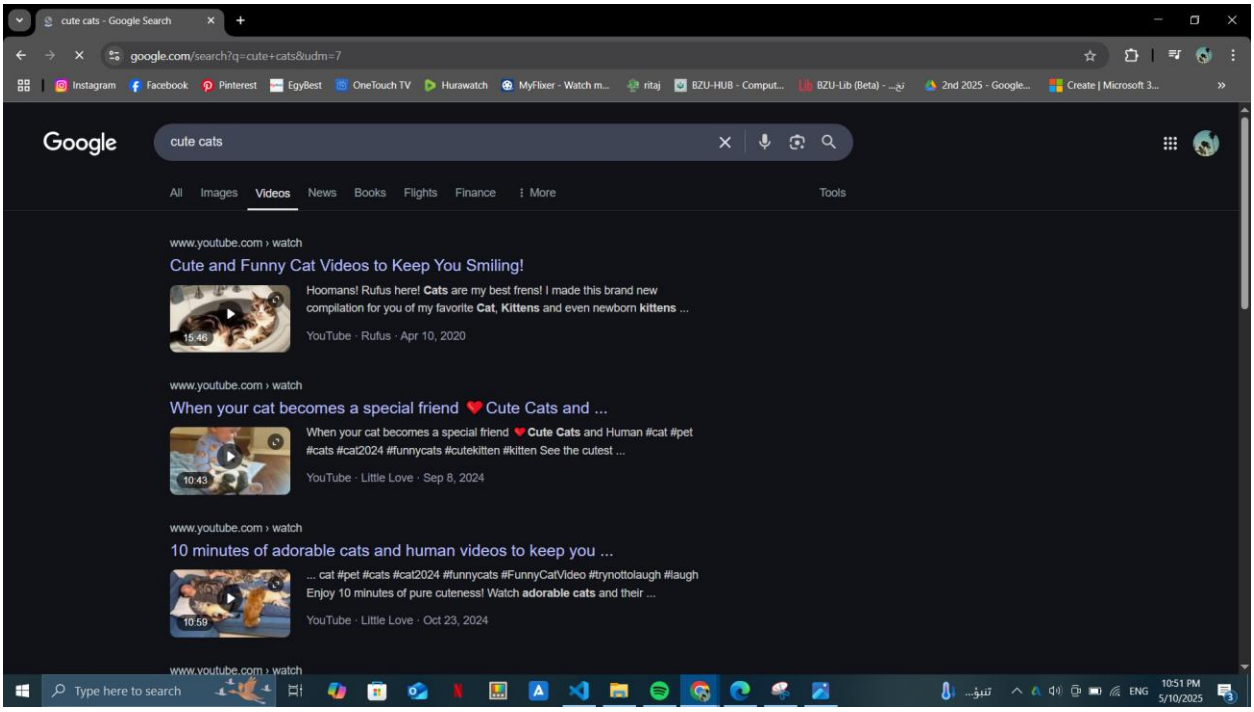


Figure 27: Client successfully redirected.

d) Arabic version (mySite_1220031_ar.html)

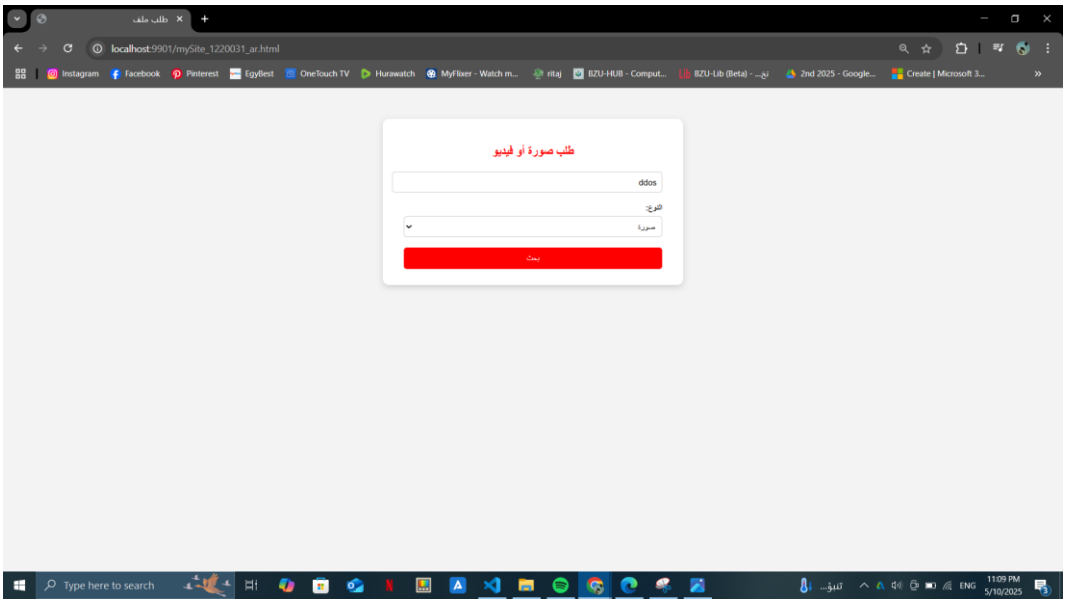


Figure 28: Client entering file request

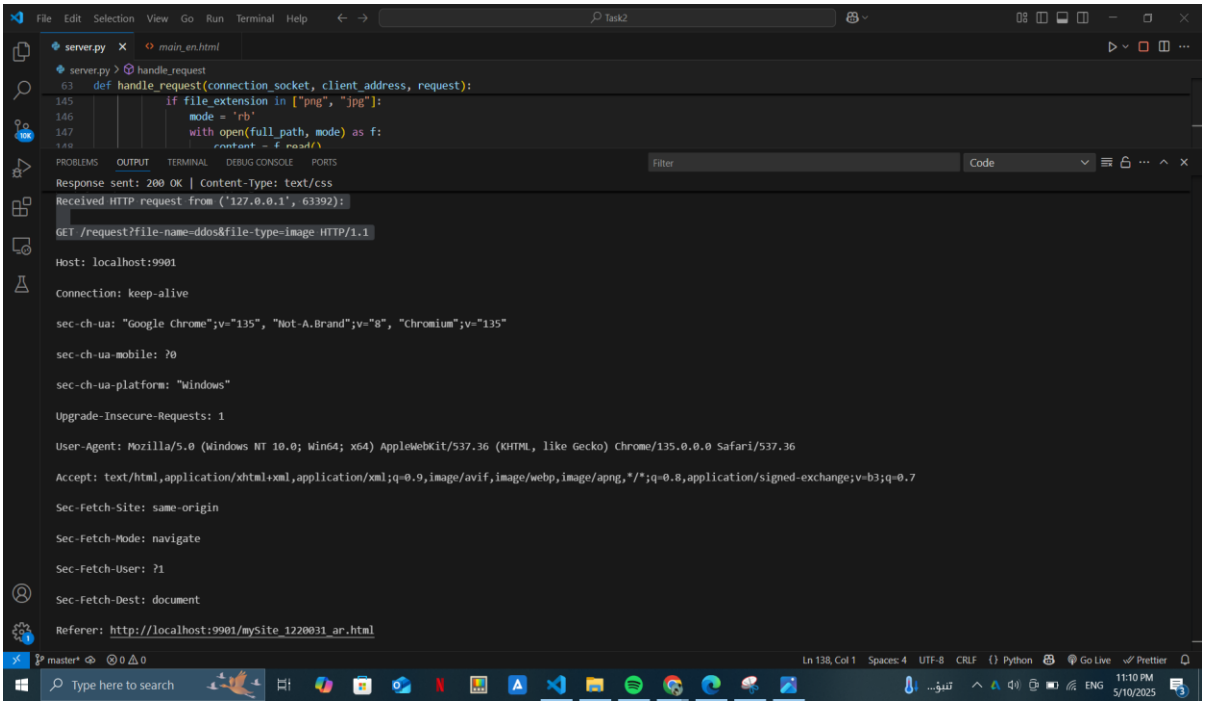


Figure 29: Server receiving the request.

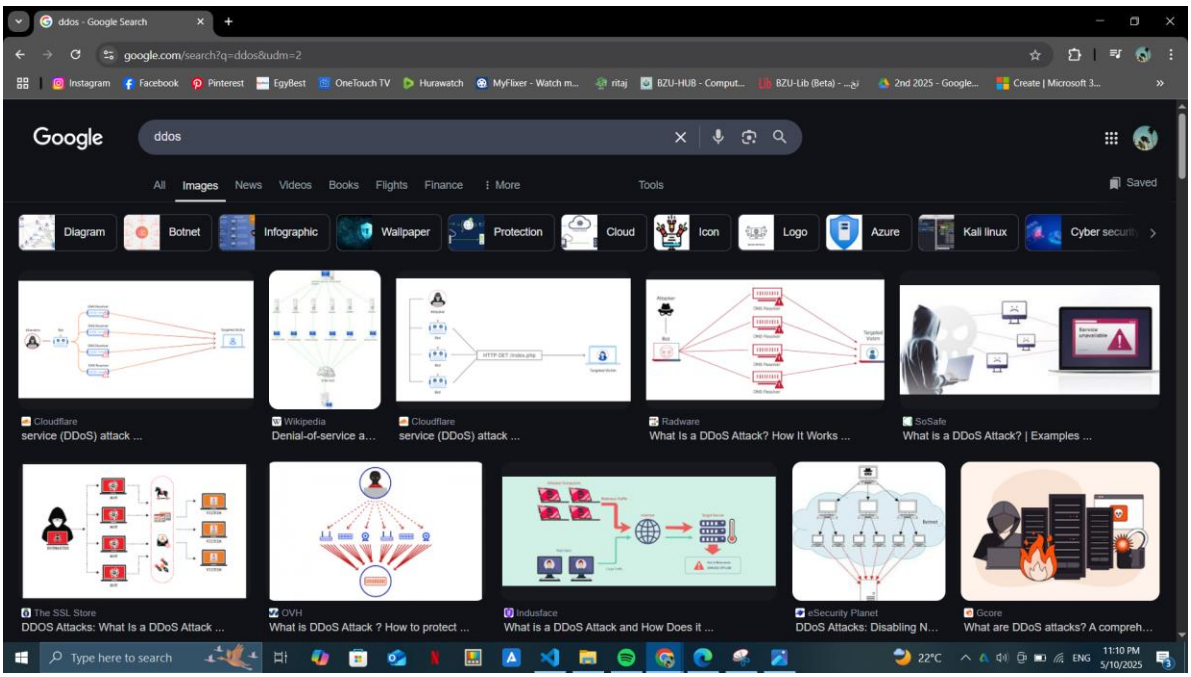


Figure 30: Client successfully redirected.

e) Default and Error Responses

1. Default Redirection:
   - Requests to /, /en, /index.html, or /main_en.html are redirected to the main English page (main_en.html).
   - Requests to /ar or /main_ar.html are redirected to the Arabic page (main_ar.html).
2. 404 Not Found Handling:
   - When a client requests an invalid or non-existent URL, the server generates a 404 Not Found response.
   - The error page displays:
     - The message "Error 404: The file is not found" in red text.
     - The client's IP address and port number for context.

This structured approach ensures that the web server remains robust, handling both valid and invalid requests efficiently while maintaining a user-friendly experience.

If a problem occurs with the server this error page is displayed

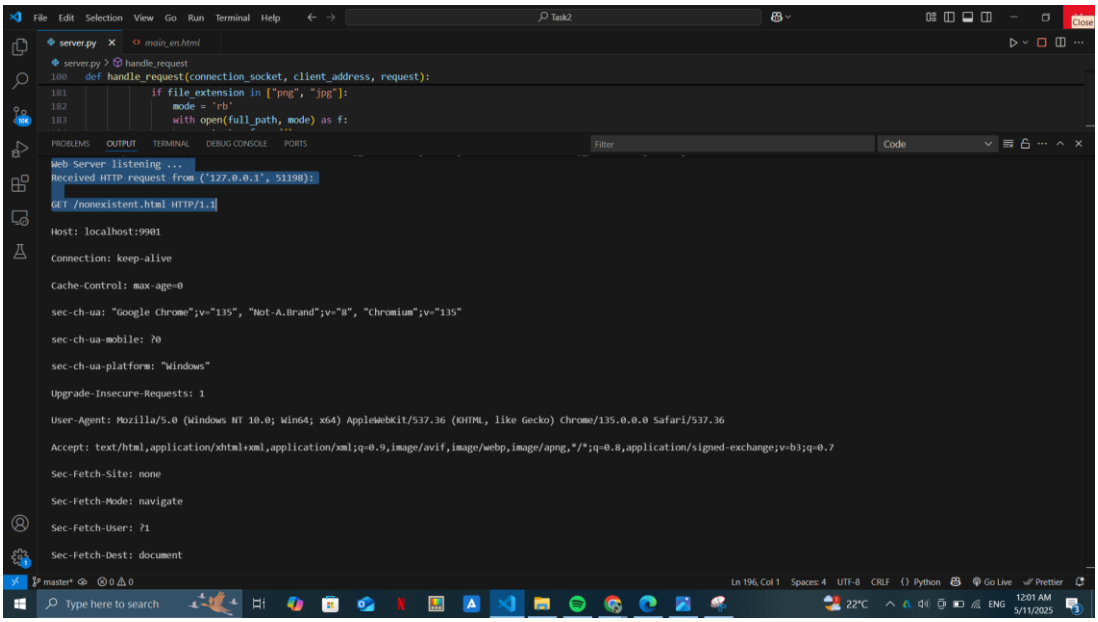


Figure 31: Server's terminal.

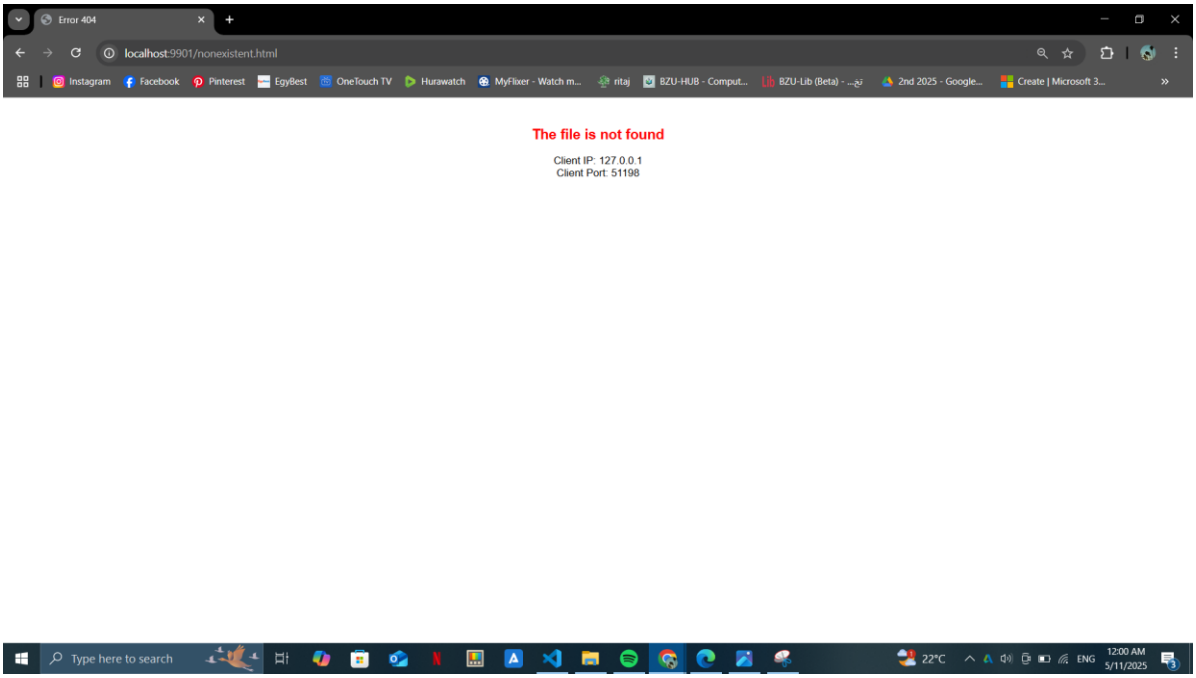


Figure 32: Error page displayed to client.

2.3. Task 3 – TCP/UDP Hybrid Client-Server Game Using Socket Programming

In this task, the focus was on building a multiplayer number-guessing game where players can submit guesses, and the server provides real-time feedback. The system utilized both **TCP** for connection management and **UDP** for fast, low-latency communication of guesses and feedback.

Key features successfully implemented:

- Multithreaded client and server architecture, enabling simultaneous handling of multiple connections and communication channels.
- Real-time feedback mechanism for player guesses via UDP.
- Handling of player disconnections, allowing the game to continue with fewer players

The client creates both TCP and UDP sockets for communication, and employed multithreading to handle both communications concurrently. The TCP socket connects to the server for player registration and game state management, while the UDP socket is used for sending and receiving guesses and feedback.

```
21
22      # Create TCP and UDP server sockets
23      tcp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
24      udp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25      tcp_server_socket.bind((serverIP, TCPPort))
26      udp_server_socket.bind((serverIP, UDPPort))
```

Figure 33: Socket creation.

This approach ensured that the client could receive and display updates without interrupting the player's input.and send player guesses without waiting for confirmation from previous guesses, ensuring a smooth and uninterrupted gameplay experience.

The client starts by registering a username with the server via a TCP connection. The client sends a "JOIN <username>" message, and the server responds by confirming the registration or indicating if the username is already taken.

```
23
24  def register_player(tcp_socket):
25      while True:
26          username = input("Enter your username: ").strip()
27          tcp_socket.send(f"JOIN {username}".encode())
28          response = tcp_socket.recv(1024).decode().strip()
29          if "Name already used" in response:
30              print(f"Server (TCP): {response}")
31              print("="*30)
32              print("Username already taken. Try again.\n")
33          else:
34              print_header(username)
35              return username
36
```

Figure 34: Code for registering players

During the game the client runs a multithreaded listener to handle incoming game feedback. One thread listens to TCP messages, while another thread listens for UDP feedback.

```
112  def main():
113      try:
114          tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
115          tcp_socket.connect((serverIP, TCPPort))
116
117          # Register player with username handling
118          username = register_player(tcp_socket)
119
120          # Start TCP and UDP feedback listeners
121          threading.Thread(target=listen_for_feedback, args=(tcp_socket,), daemon=True).start()
122          threading.Thread(target=listen_for_udp_feedback, daemon=True).start()
123
```

Figure 35: Code for clients listening

To help the client manage the game state we used two flags, game_active that indicates whether the game is currently active.and game_over indicates if the game has ended. These flags ensure

that the client only sends guesses during active game rounds and halts interaction after the game ends.

```
11      game_active = False
12      game_over = False
13
```

Figure 36: Game flags

The server accepts incoming TCP connections. Each connection is assigned to a separate thread for handling communication. The server listens for the "JOIN <username>" command from clients.

```
58              broadcast(f"Waiting Room: {', '.join(active_clients.values())}\n")
59
60              # Start the game automatically after WAIT_TIME if at least MIN_PLAYERS are in
61              if len(active_clients) == MIN_PLAYERS:
62                  threading.Thread(target=start_after_delay).start()
63              elif len(active_clients) == MAX_PLAYERS:
64                  start_round()
```

Figure 37: Code for server waiting for client's connection

Once the minimum number of players join, the game automatically starts. The server generates a random secret number and sends a broadcast message to all players, providing the range and time limit for guesses.

```
20      secret_number = random.randint(GUESS_RANGE[0], GUESS_RANGE[1])
```

We fully did Error handling in our task. The server handled unexpected disconnections gracefully by checking the remaining players and notifying them. The client managed input validation and feedback errors (e.g., invalid guesses or loss of UDP packets) without crashing, ensuring a seamless user experience.

Test Cases and Expected Output:

1. Two players register, game starts, one wins:

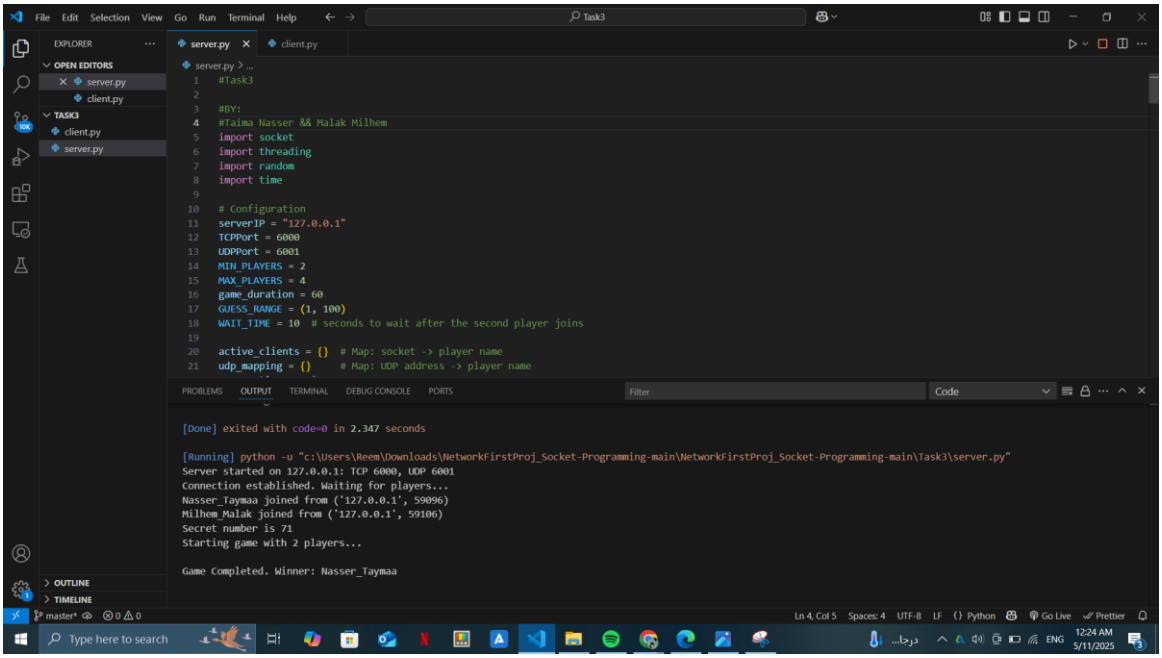


Figure 38: Server Terminal

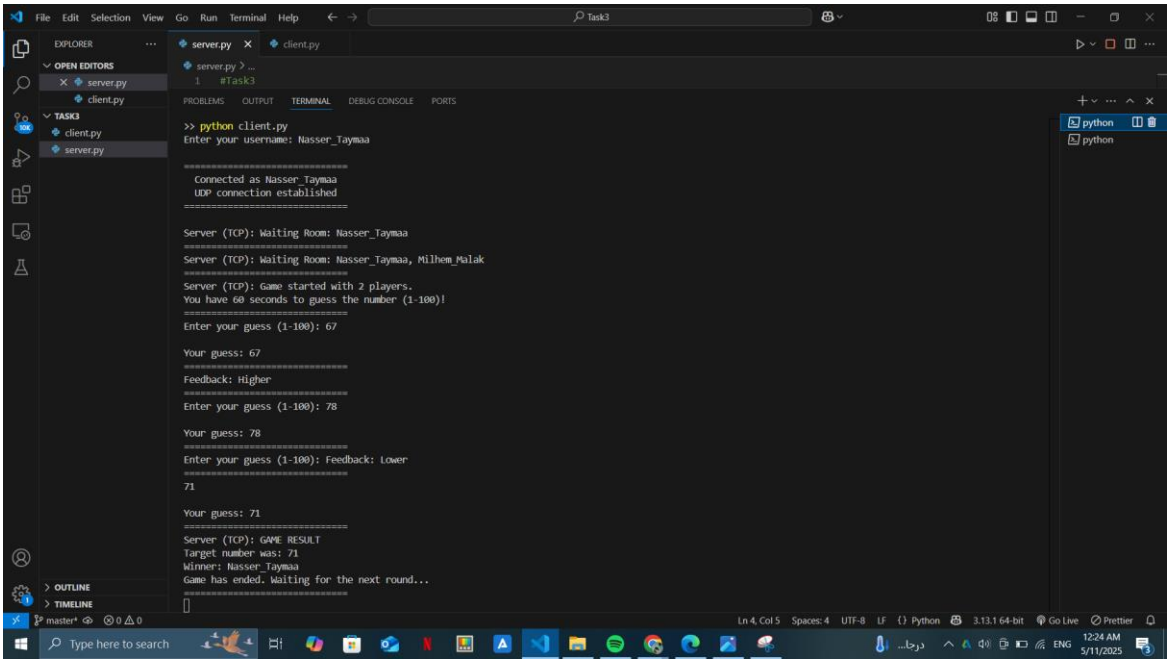


Figure 39: Player 1 terminal

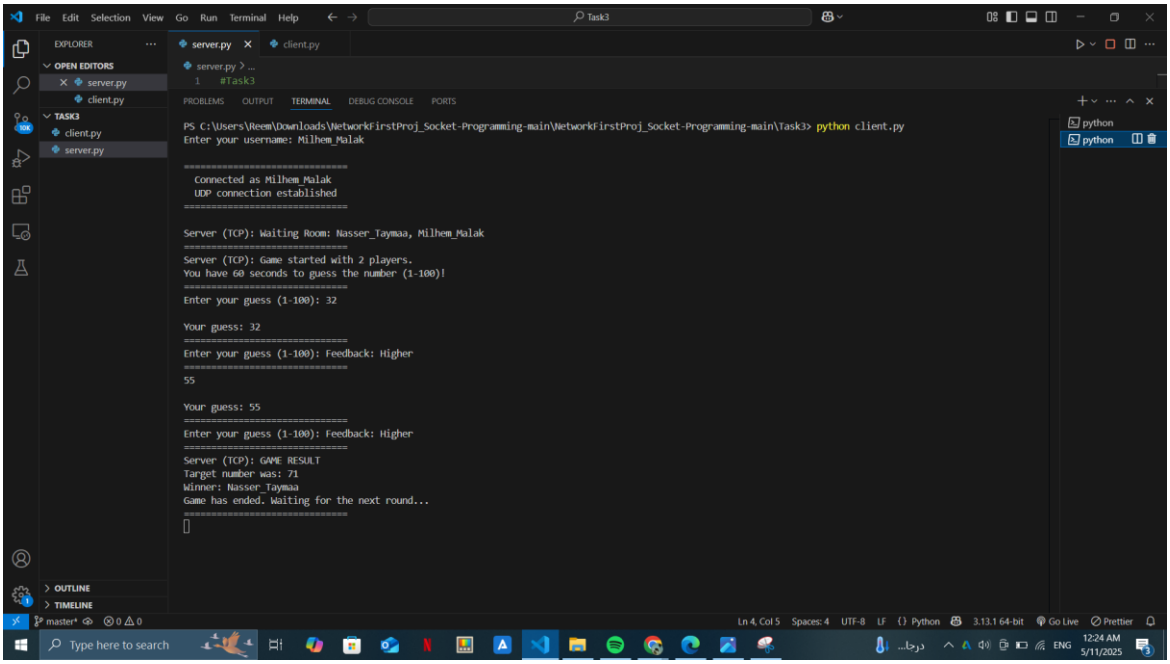


Figure 40: Player 2 terminal

2. Players guess out of bounds → get warning.

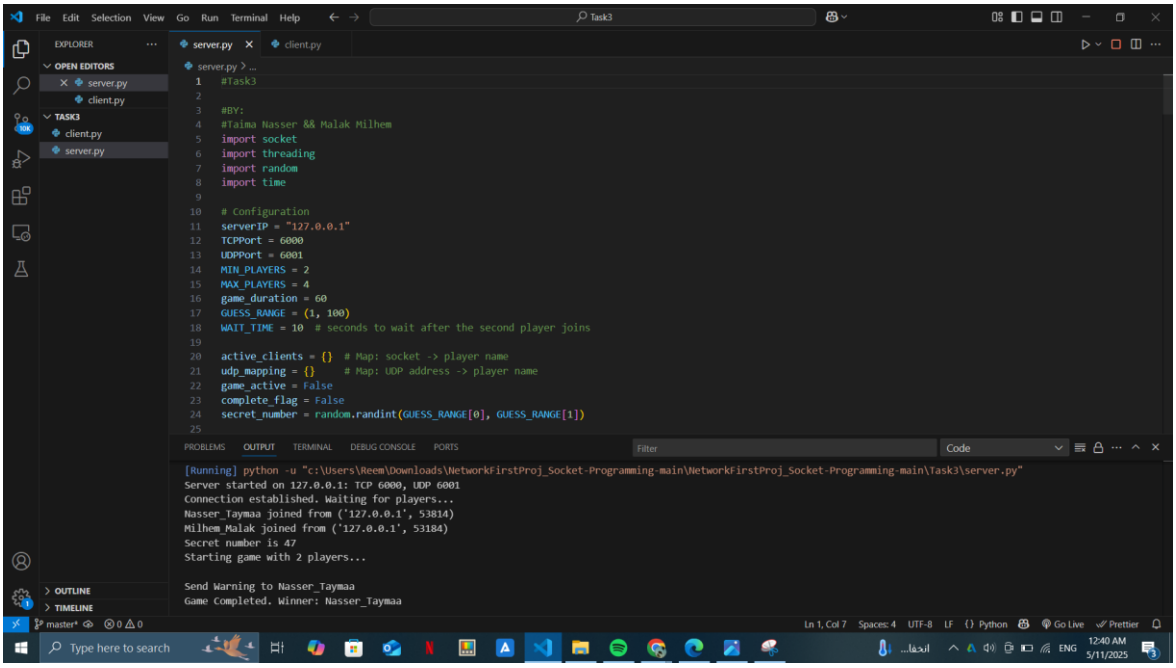


Figure 41: Server terminal

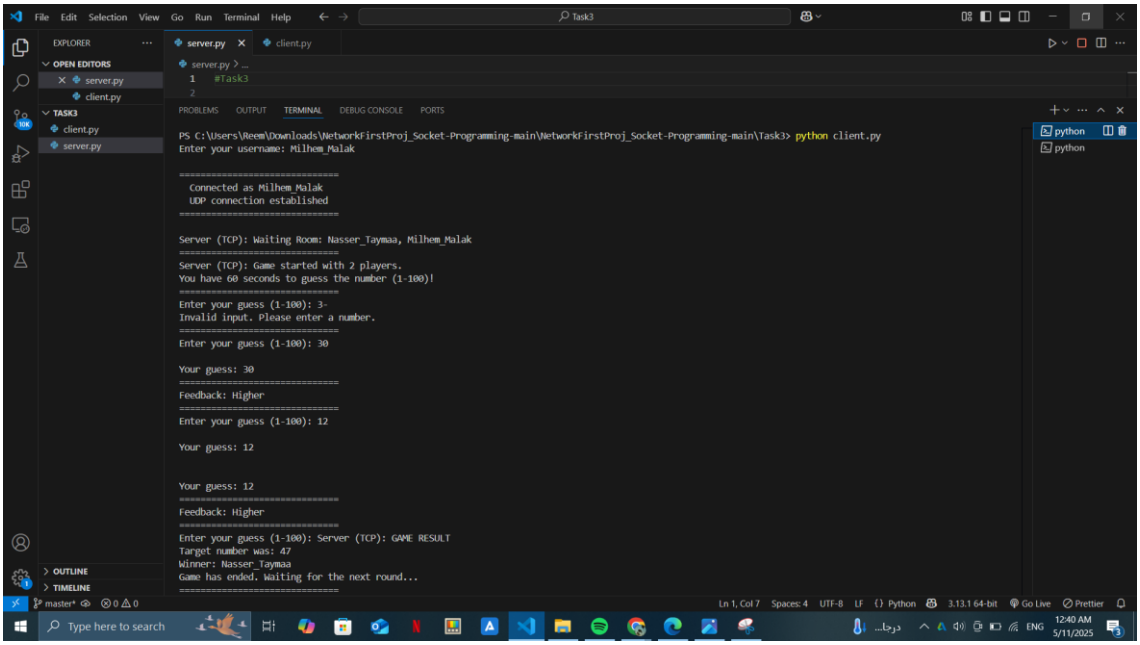


Figure 42: Player 2 terminal

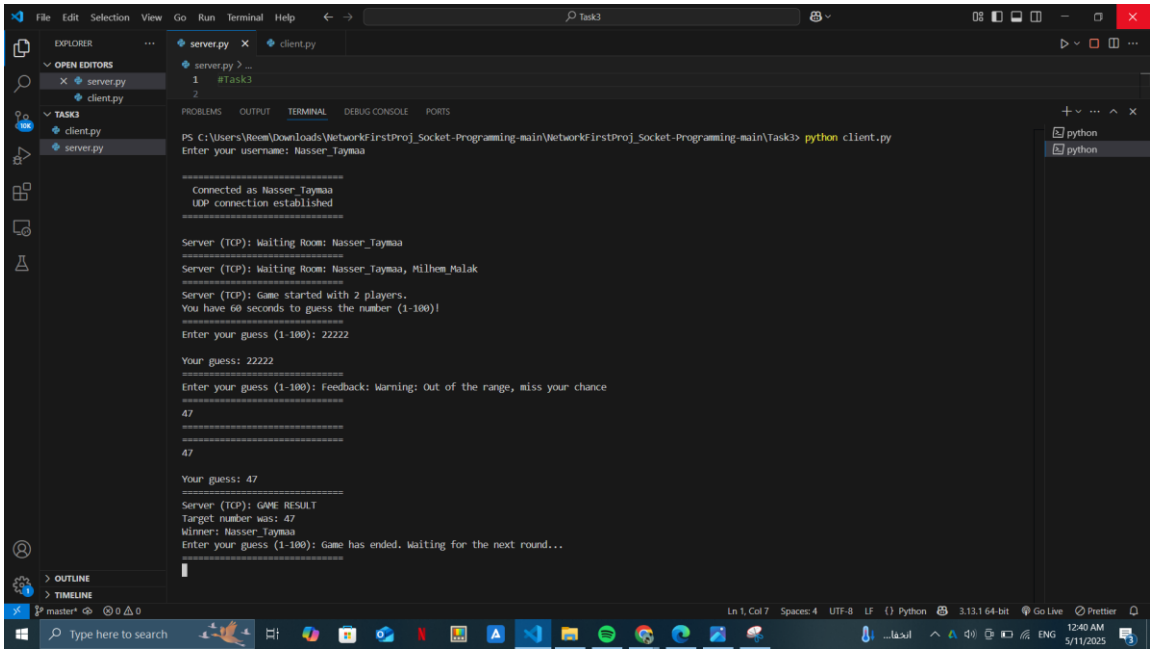


Figure 43: Player 1 terminal

3. One player disconnects mid-game → game continues or ends.

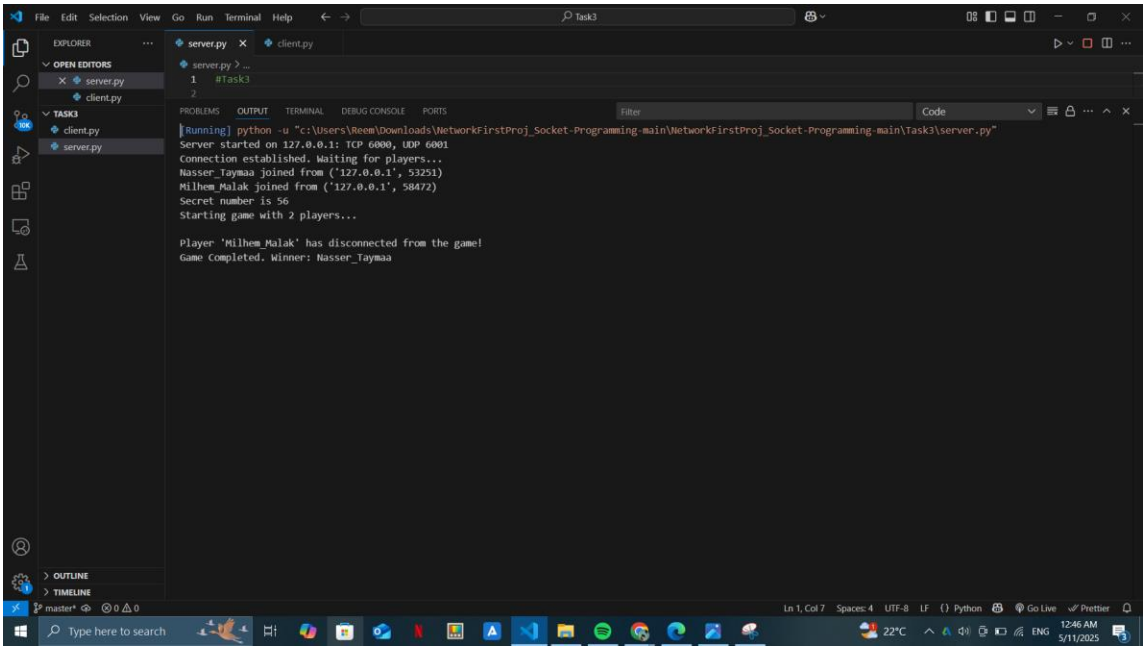


Figure 44: Server terminal after player 2 disconnected

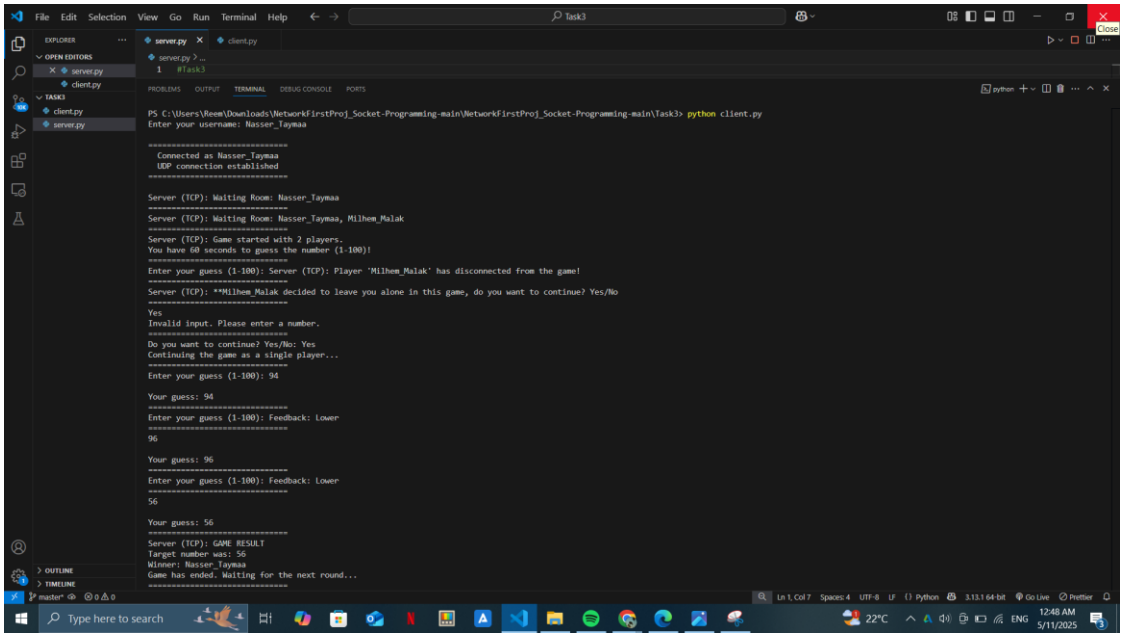


Figure 45: Player 1 terminal after player 2 disconnected

Conclusion

This project successfully demonstrated the core principles of network programming through three practical tasks that emphasize client-server communication and the use of both TCP and UDP protocols. Task 1 provided a foundational understanding of network configurations and analysis through various network commands, such as ipconfig, ping, tracert, and nslookup, along with the use of Wireshark to monitor DNS queries.

In Task 2, a web server was developed using socket programming, which enhanced our understanding of HTTP communication, client-server interactions, and web resource handling. The server was capable of responding to a variety of requests, including static web pages, redirects, and custom error handling, while also accommodating both English and Arabic content. This part of the project highlighted the complexities of creating and managing dynamic responses based on user requests.

Task 3 explored the integration of both TCP and UDP protocols in a multiplayer guessing game, where TCP ensured reliable player registration and game state management, while UDP was used for fast-paced game interactions. The implementation of multithreading and real-time feedback mechanisms showcased the efficiency of hybrid protocols in supporting seamless user experiences. This task also emphasized the importance of robust error handling to ensure uninterrupted gameplay, even in the case of unexpected disconnections or input errors.

Overall, this project provided a comprehensive understanding of socket programming, real-time communication, and the integration of multiple protocols in networked applications. It not only strengthened the theoretical knowledge of network communication but also provided practical experience in building and testing networked systems in real-world scenarios.

Teamwork:

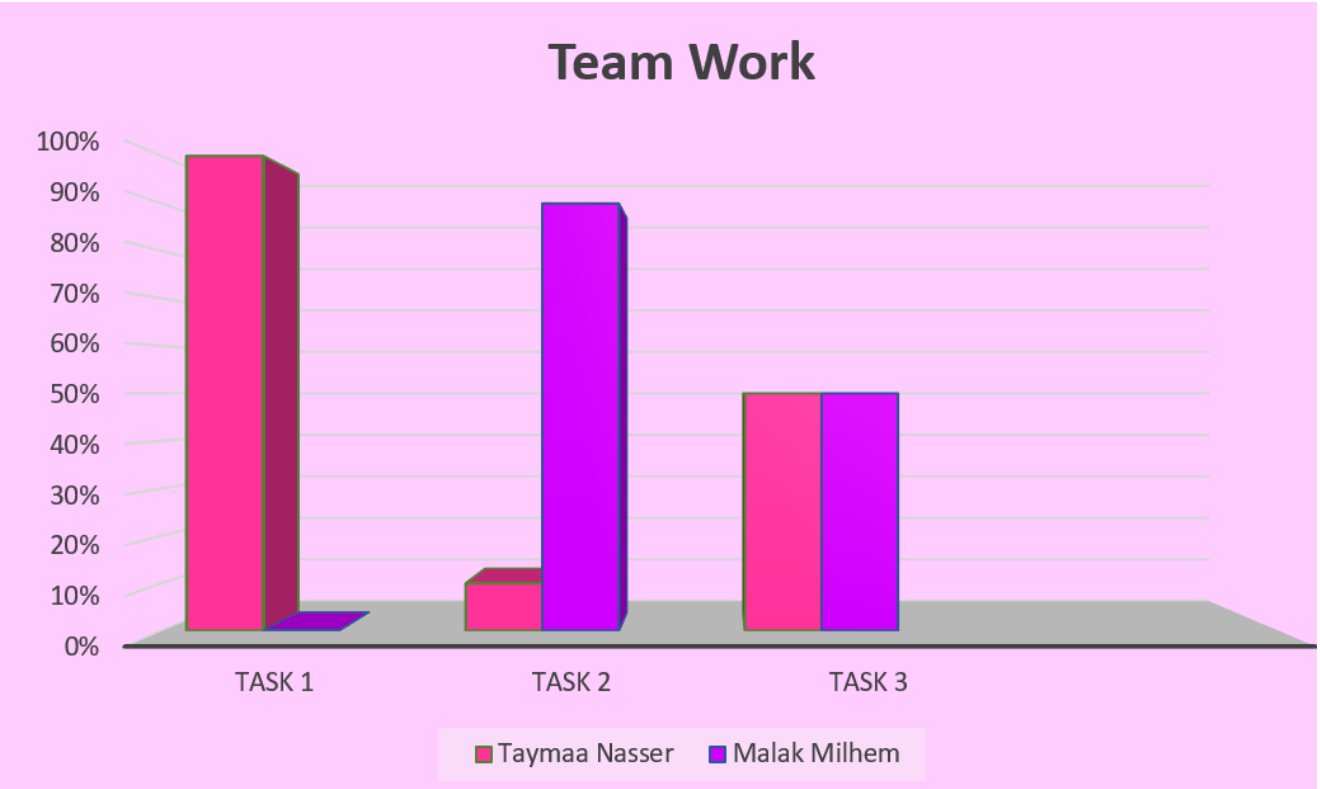


Figure 46: Teamwork distribution

References

[1] https://www.ringcentral.com/gb/en/blog/definitions/packet/ [Access Time: 05-05-2025]

[2] https://www.techtarget.com/whatis/definition/Wireshark [Access Time: 05-05-2025]

[3] https://www.slideserve.com/zephania-lynn/tcp-client-server-model [Access Time: 05-05-2025]

[4] https://www.geeksforgeeks.org/state-the-core-components-of-an-http-response/ [Access Time: 05-05-2025]

[5] https://www.infidigit.com/blog/http-status-codes/ [Access Time: 05-05-2025]

[6] https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Redirections [Access Time: 05-05-2025]

[7] https://afteracademy.com/blog/what-is-a-tcp-3-way-handshake-process/ [Access Time: 09-05-2025]

[8] https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/ [Access Time: 09-05-2025]