NWEN\_243 Project 1

**Name:** Nico Wartmann

**Student ID:** 300671406

**Date:** 08.08.2024

Q1:

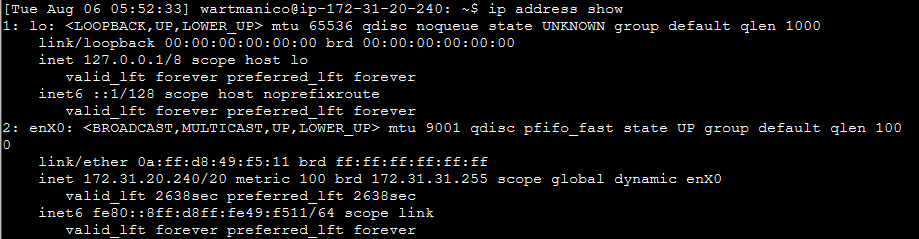


Figure 1: ‘ip address show’ output

1. enX0
2. 0a:ff:d8:49:f5:11
3. 00001010:11111111:11011000:01001001:11110101:00010001
4. 6 Segments \* 8 Bits = 48 Bits
5. ff:ff:ff:ff:ff:ff

Q2:

1. Private IP – see “*Figure 1: ‘ip address show’ output*” 🡪 172.31.20.240

Public IP 🡪 54.221.138.95

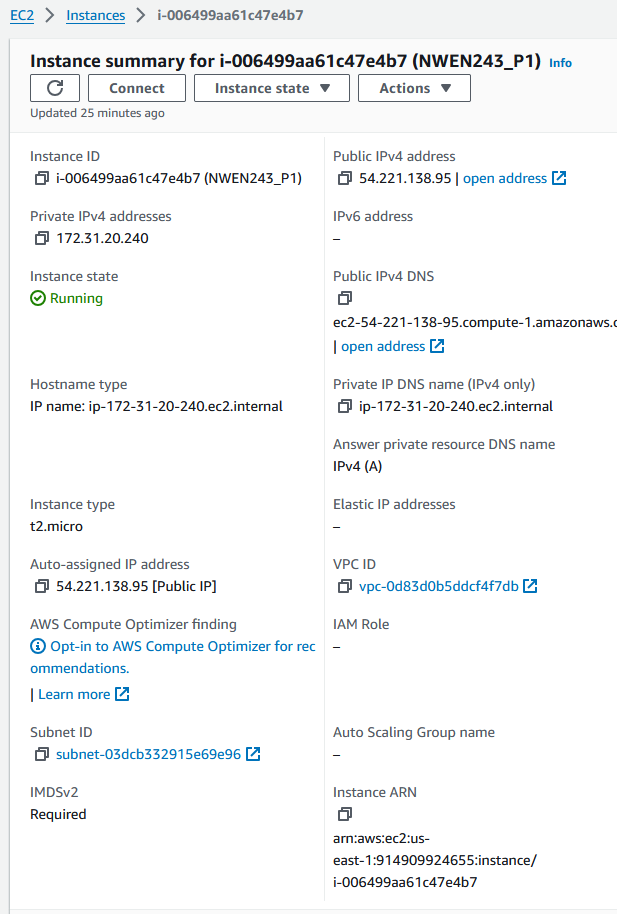


Figure 2: Public IP in AWS Portal

1. Private IP – see “*Figure 1: ‘ip address show’ output*” 🡪 fe80::8ff:d8ff:fe49:f511
2. 4 Segments \* 8 Bits = 32 Bits
3. 00110110.11011101.10001010.01011111
4. 8 Segments \* 16 Bits = 128 Bits

Q3:

1. Private IP – see “*Figure 1: ‘ip address show’ output*” 🡪 172.31.20.240/20

First 20 bits are NW Portion, last 12 Bits are host Portion

1. NW-Address (lowest): 10101100.000011111.00010000.00000000 🡪 172.31.16.0

BC-Address (highest): 10101100.000011111.00011111.11111111 🡪 172.31.31.255

Range: 172.31.16.0 - 172.31.31.255

1. 32 – 20 = 12 🡪 12 Bits for Hosts 🡪 2^12 = 4096 Addresses (4096 – 2 = 4094 Hosts)

Q4:

1. The Netmask determines how many bits of an IP-Address are used for Network-, and Host-addressing (…/20)
2. BC-Address (highest): 10101100.000011111.00011111.11111111 🡪 172.31.31.255
3. The highest Address in the Network is always the broadcast Address. Setting all the bits of the host portion, the Broadcast Address can be obtained.

Q5:

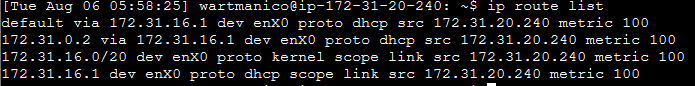


Figure 3: 'ip route list' output

Here are all the routes listed, each entry tells where the next “hop” i.e. IP-Address to contact is, to get to a certain network. For all he networks that are not explicitly listed here, the default route (*default* keyword) is used. The IP Address provided in this Line refers to the *default gateway*.

1. 

Figure 4: 'ip neighbour show' output

This command is used to check the ARP-Table (Address Resolution Protocol) that this machine holds. Only IP – MAC pair known to this Machine is the one of the default gateway. The knowledge of this mapping is used by the Machine to wrap an IP Request in a MAC Package and address it accordingly.

Q6:

IP1

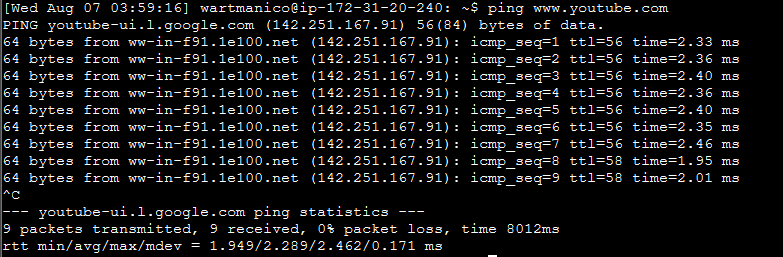
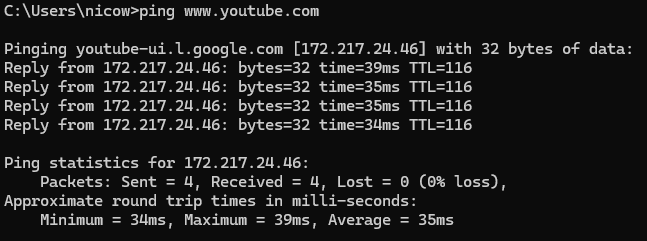


Figure 5: Ping to YouTube from VM



IP2

Figure 6: Ping to YouTube from local machine

1. RTL\_VM\_IP1 🡪 Low ( > 100 ms )

RTL\_VM\_IP2 🡪 High ( < 100 ms )

RTL\_VM\_IP1 🡪 High ( < 100 ms )

RTL\_VM\_IP2 🡪 Low ( > 100 ms )

1. Those Ips belong to YouTube servers in different geolocations. Depending in the machine sending the Request to YouTube, the DNS servers deliver a different IP-Address to reduce traffic on the international network Infrastructure and latency.

Q7:

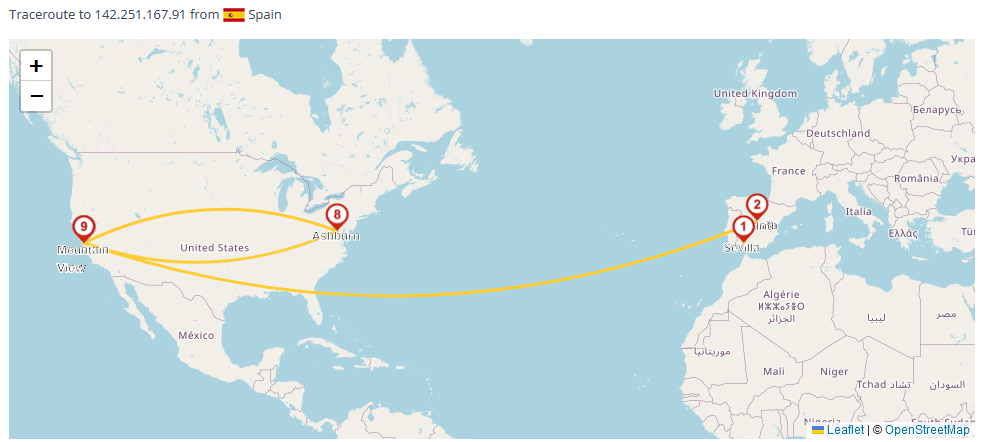


Figure 7: Traceroute to 142.251.167.91 from Spain

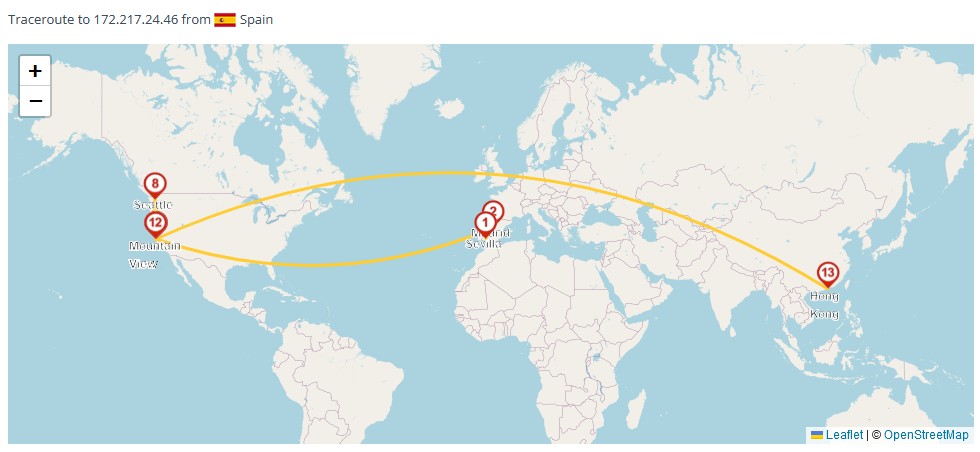


Figure 8: Traceroute to 172.217.24.46 from Spain

Since the VM (AWS) is located in the USA it makes sense for IP1 Being in the USA too. My local machine is currently in wellington and has no VPN activated. Therefore it makes sense, that IP2 is closer to New Zealand (Hong Kong in this case).

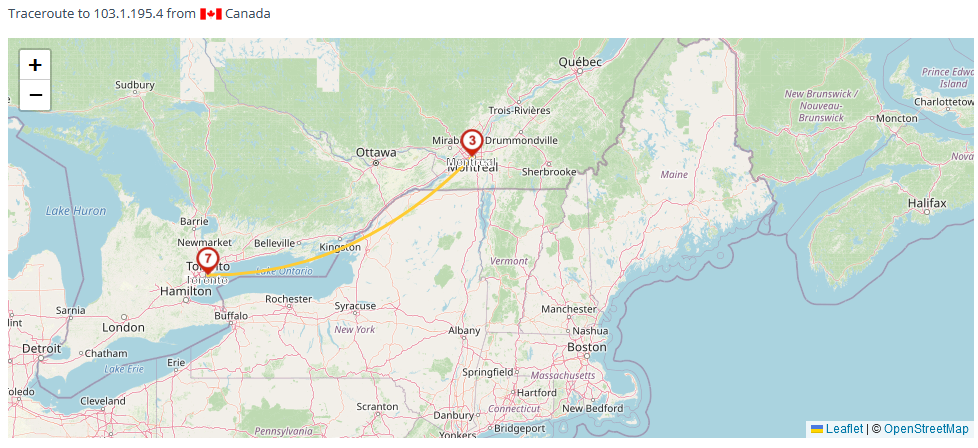


Figure 9: Traceroute to 103.1.195.4 from Canada

No matter which server I select, the last hop to destination IP (103.1.195.4) of wgtn.ac.nz is always located in the same country.

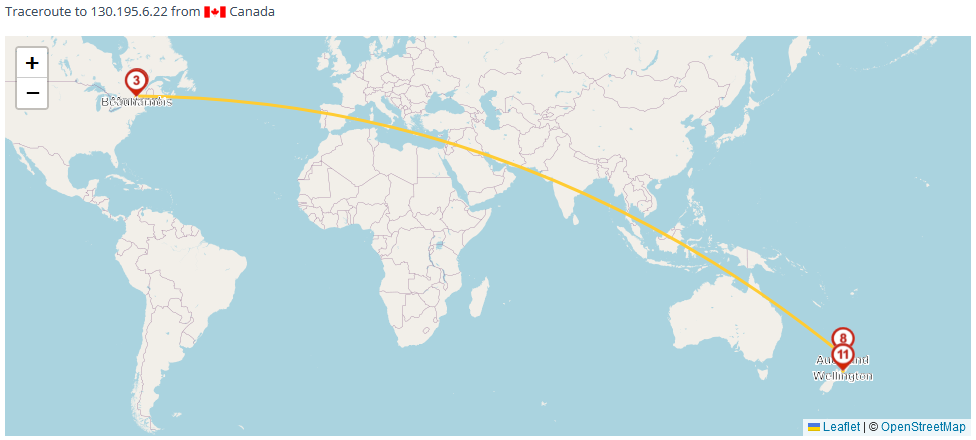


Figure 10: Traceroute to 130.195.6.22 from Canada

Here the last Hop is located in Wellington.

Q8:

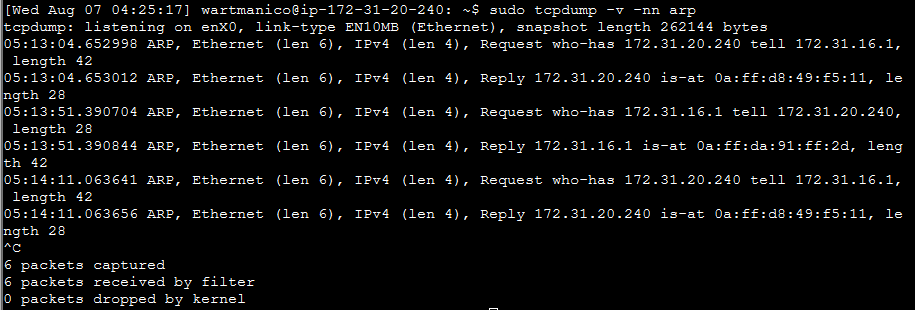


Figure 11: 'sudo tcpdump -v -nn arp' output

Looking at the first two entries we have a request reply pair. In the Request (first ernty) The Host at 172.31.16.1 (default gw) asks which MAC Address corresponds to the IP 172.31.20.240 (VM IP).

In the Response the VM replies with its MAC Address 0a:ff:d8:49:f5:11

1. ARP entries have a TTL. After this is expired the request is sent again. In this way it is made sure that a package meant for a certain IP isn’t delivered to the wrong host if the Ips are reallocated.
2. About every 70 seconds.

Q9:

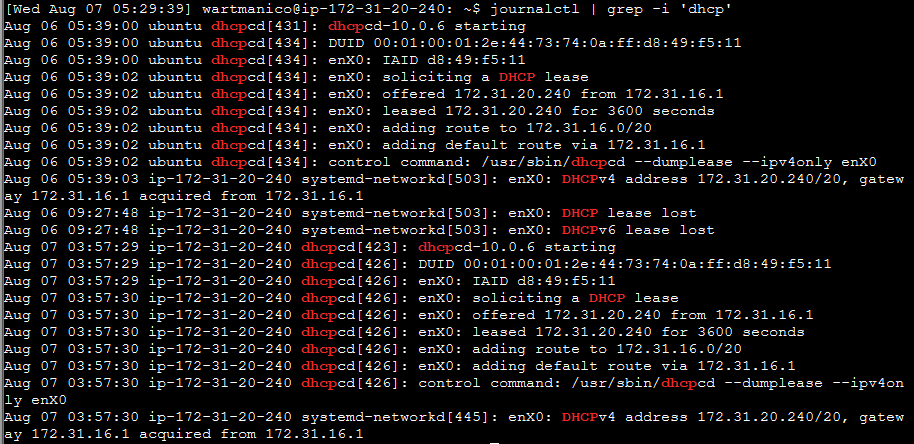


Figure 12: 'journalctl | grep -i 'dhcp'' output

* DHCP Server IP Address: 172.31.16.1 (default gw)
* Lease Duration: 3600 seconds (1 hour)

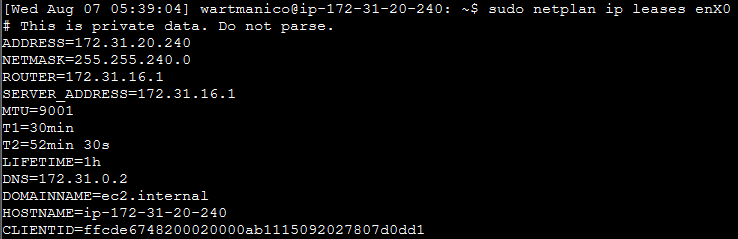


Figure 13: 'sudo netplan ip leases enX0' output

Main Information:

* ADDRESS: IP-Adress leased
* NETMASK: subnet mask
* ROUTER: default gw
* SERVER\_ADDRESS: IP Address of DHCP Server
* LIFETIME: 1h (the 3600 seconds seen before)

Optional Information

* MTU: Maximum Transmission Unit
* T1: Time after which client tries to renew lease
* T2: Time after which client looks for new DHCP server if there is no response after T+ expiry
* DNS: IP Address of DNS Server
* DOMAINNAME: Name of the domain
* HOSTNAME: hostname assigned to this client
* CLIENTID Unique ID assigned to client for DHCP to recognize returning clients

Q10

A screen shot of a computer program

Description automatically generated

Figure 14: Q10 instructions executed

**Packet 1** (Time: 05:51:09.776780):  
Source 🡪 172.31.20.240 (VM)  
Destination 🡪 172.31.0.2.53 (DNS Server)  
Purpose 🡪 DNS query: asking the DNS server to provide the IPv6 Address of the FQDN ‘www.wgtn.ac.nz.’

**Packet 2** (Time: 05:51:09.777028):  
Source 🡪 172.31.20.240 (VM)  
Destination 🡪 172.31.0.2.53 (DNS Server)  
Purpose 🡪 DNS query: asking the DNS server to provide the IPv4 Address of the FQDN ‘www.wgtn.ac.nz.’

**Packet 3** (Time: 05:51:09.779172):  
Source 🡪 172.31.0.2.53 (VM)  
Destination 🡪 172.31.0.2.53 (DNS Server)  
Purpose 🡪 DNS response: indicating no IPv6 entries for ‘www.wgtn.ac.nz.’.

**Packet 4** (Time: 05:51:09.780166):  
Source 🡪 172.31.20.240 (VM)  
Destination 🡪 172.31.0.2.53 (DNS Server)  
Purpose 🡪 DNS response: indicating multiple IPv4 entries for ‘www.wgtn.ac.nz.’: (151.101.130.49, 151.101.194.49, 151.101.2.49, 151.101.66.49)

1. UDP is used instead of TCP, because it makes no sense to establish a connection for DNS requests and create more traffic by doing so (which tcp would), because if the DNS request stays unanswered, another one will be sent shortly anyway.

Q11:

A computer screen shot of a black screen

Description automatically generated

Figure 15: 'cat tcpdump.out' output

**Packet 1** (Time: 08:45:01.453226):

(a) Source IP 🡪 172.31.20.240 (VM)  
(a) Destination IP 🡪 151.101.194.49 (Web Server - WUV)  
(b) Source Port 🡪 37562  
(b) Destination Port 🡪 443 (WKP HTTPS)  
(c) TCP Flags 🡪 [S] 🡪 SYN  
(d) Purpose 🡪 1st Package TCP Handshake client initiates connection to server

**Packet 2** (Time: 08:45:01.453881):  
(a) Source IP 🡪 151.101.194.49 (Web Server - WUV)  
(a) Destination IP 🡪 172.31.0.2.53 (DNS Server)  
(b) Source Port 🡪 443 (WKP HTTPS)   
(b) Destination Port 🡪 37562  
(c) TCP Flags 🡪 [S.] 🡪 SYN + ACK  
(d) Purpose 🡪 Server responds with SYN-ACK confirmation for reception of firs package and indicating readiness for connection

**Packet 3** (Time: 08:45:01.453903):  
(a) Source IP 🡪 172.31.20.240 (VM)  
(a) Destination IP 🡪 151.101.194.49 (Web Server - WUV)  
(b) Source Port 🡪 37562  
(b) Destination Port 🡪 443 (WKP HTTPS)  
(c) TCP Flags 🡪 [.] 🡪 ACK  
(d) Purpose 🡪 Client responds with ACK confirming reception of second package

Q12:

Screenshot see *Figure 15: 'cat tcpdump.out' output*.  
**Packet 1 (SYN):**   
 Sequence Number: 2623726707 (random generated “client seq”)  
 Acknowledgement Number: none  
 Data: 0 Bytes  
**Packet 2 (SYN-ACK):**   
 Sequence Number: 105528673 (random generated “server seq”)  
 Acknowledgement Number: 2623726708 (client seq + 1)  
 Data: 0 Bytes **Packet 3 (ACK):** Sequence Number: 2623726708 (previous Acknowledgement Number)  
 Acknowledgment Number: 105528674 (server seq + 1)Data: 0 Bytes **Packet 4 (PSH-ACK):** Sequence Number: 2623726708 (same as last 🡪 client seq + 1)  
 Acknowledgment Number: 105528674 (same as last 🡪 server seq + 1)  
 Data: 517 Bytes **Packet 5 (ACK):**  
 Sequence Number: 105528674 (previous Acknowledgement Number)  
 Acknowledgment Number: 2623727225 (same as last 🡪 client seq + 1 + 517)  
 Data: 0 Bytes

**Packet 6 (PSH-ACK):**   
 Sequence Number: 105528674 (previous Acknowledgement Number)  
 Acknowledgment Number: 2623727225 (same as last 🡪 client seq + 1 + 517)  
 Data: 3247 Bytes

Sequence Numbers are used to keep track of where in the stream of data the current packet fits. Each byte of data in TCP is sequentially numbered, so sequence numbers help ensure data is received and reassembled correctly.

Acknowledgment Numbers are the next expected sequence number the sender of the ack is expecting. It confirms receipt of all bytes up to that number minus one.

Q13:

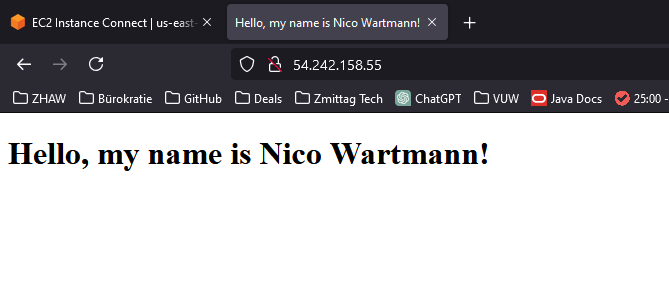


Figure 16: Simple website with Name

Q14:

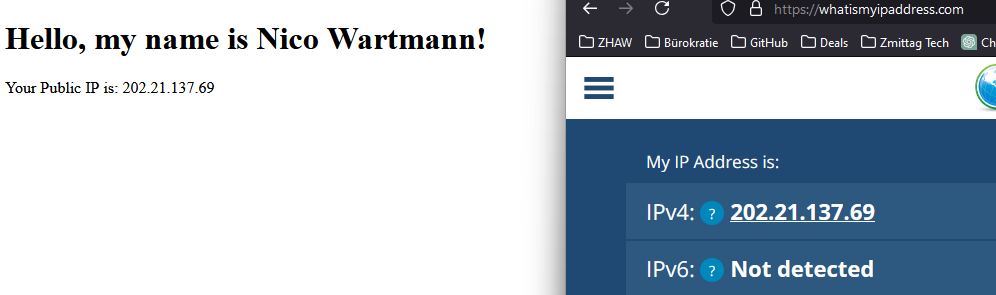


Figure 17: Simple website with client public IP

Added Code:

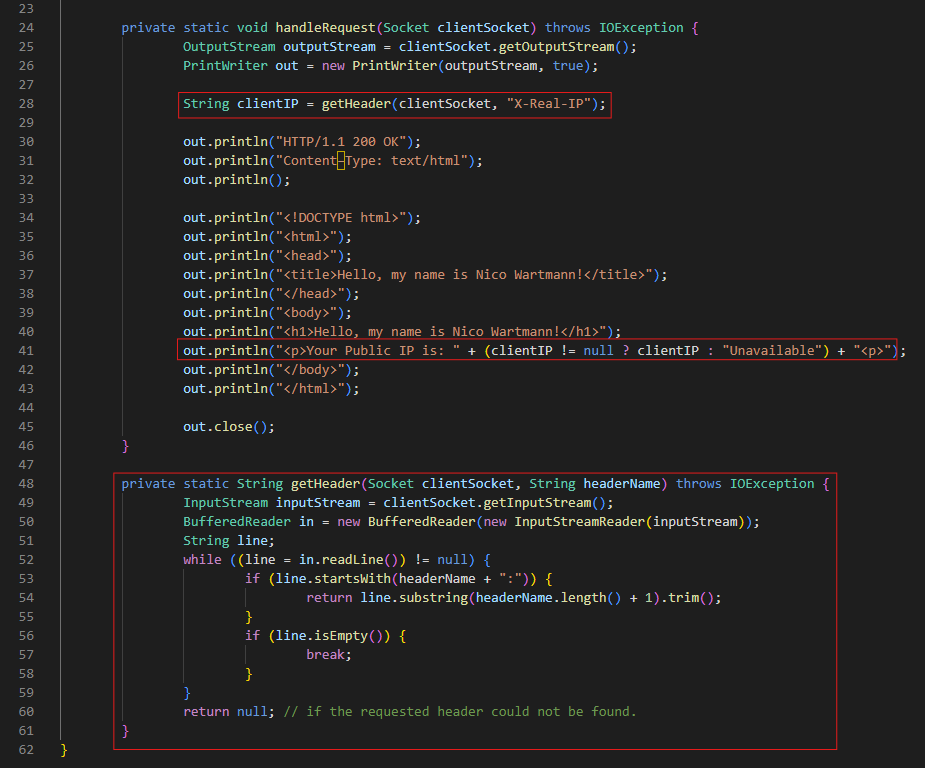


Figure 18: Added code to get Client Public IP

Q15:



Figure 19: Simple website including Location

Code:

import java.io.\*;

import java.net.\*;

public class SimpleWebServer\_1 {

    public static void main(String[] args) {

        int port = 8080;

        try {

            ServerSocket serverSocket = new ServerSocket(port);

            System.out.println("Server running at http://localhost:" + port);

            while (true) {

                Socket clientSocket = serverSocket.accept();

                handleRequest(clientSocket);

                clientSocket.close();

            }

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private static void handleRequest(Socket clientSocket) throws IOException {

        OutputStream outputStream = clientSocket.getOutputStream();

        PrintWriter out = new PrintWriter(outputStream, true);

        String realClientIP = getHeader(clientSocket, "X-Real-IP");

        String location = getLocation(realClientIP);

        out.println("HTTP/1.1 200 OK");

        out.println("Content-Type: text/html");

        out.println();

        out.println("<!DOCTYPE html>");

        out.println("<html>");

        out.println("<head>");

        out.println("<title>Hello, my name is Nico Wartmann!</title>");

        out.println("</head>");

        out.println("<body>");

        out.println("<h1>Hello, my name is Nico Wartmann!</h1>");

        out.println("<p>Your Public IP is: " + (realClientIP != null ? realClientIP : "Unavailable") + "</p>");

        out.println("<p>Your Location: " + (location != null ? location : "Could not determine location") + "</p>");

        out.println("</body>");

        out.println("</html>");

        out.close();

    }

    private static String getHeader(Socket clientSocket, String headerName) throws IOException {

        InputStream inputStream = clientSocket.getInputStream();

        BufferedReader in = new BufferedReader(new InputStreamReader(inputStream));

        String line;

        while ((line = in.readLine()) != null) {

            if (line.startsWith(headerName + ":")) {

                return line.substring(headerName.length() + 1).trim();

            }

            if (line.isEmpty()) {

                break;

            }

        }

        return null;

    }

    private static String getLocation(String ip) throws IOException {

        if (ip == null) return "IP Address Not Provided";

        try (Socket socket = new Socket(InetAddress.getByName("ipinfo.io"), 80)) {

            PrintWriter request = new PrintWriter(socket.getOutputStream(), true);

            BufferedReader response = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            request.println("GET /" + ip + "/json HTTP/1.1");

            request.println("Host: ipinfo.io");

            request.println("Connection: close");

            request.println();

            StringBuilder responseBody = new StringBuilder();

            boolean inContent = false;

            String line;

            while ((line = response.readLine()) != null) {

                if (line.isEmpty() && !inContent) {

                    inContent = true;

                } else if (inContent) {

                    responseBody.append(line);

                }

            }

            // I had a lot of issues with getting Regex and parsing to work so this is kinda wonky now, but it works

            // in this case I used some help of ChatGPT to resolve the problems (note to prevent plagiarism) this did ultimately not help and i figured it out with debugging outputs to the console

            String key = "\"city\"";

            int keyIndex = responseBody.indexOf(key);

            if (keyIndex != -1) {

                int colonIndex = responseBody.indexOf(":", keyIndex + key.length());

                if (colonIndex != -1) {

                    int startQuoteIndex = responseBody.indexOf("\"", colonIndex + 1);

                    if (startQuoteIndex != -1) {

                        int start = startQuoteIndex + 1;

                        int end = responseBody.indexOf("\"", start);

                        if (end != -1) {

                            return responseBody.substring(start, end);

                        }

                    }

                }

            }

        } catch (Exception e) {

            e.printStackTrace();

            return "Error fetching location";

        }

        return "Location data not found";

    }

}

Plagiarism Disclamer:

I only really needed some help on Q15:

Tried to use ChatGPT to help with figuring out some Regex and Parsing problems, did not help, no code included (marked in Code by comment)

I used this Website for help on HTTP Requests:  
https://www.baeldung.com/java-http-request