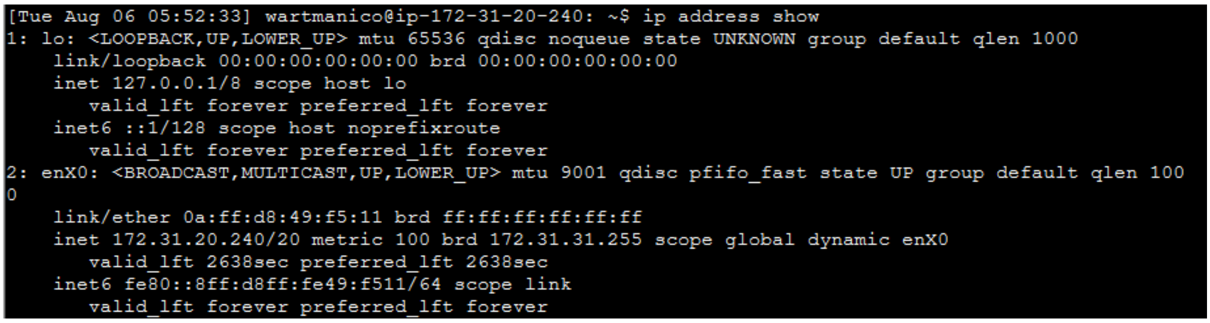
NWEN_243 Project 1

Name: Nico Wartmann

Student ID: 300671406

Date: 08.08.2024

Q1:



```
[Tue Aug 06 05:52:33] wartmanico@ip-172-31-20-240: ~$ ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enX0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc pfifo_fast state UP group default qlen 100
0
    link/ether 0a:ff:d8:49:f5:11 brd ff:ff:ff:ff:ff:ff
    inet 172.31.20.240/20 metric 100 brd 172.31.31.255 scope global dynamic enX0
       valid_lft 2638sec preferred_lft 2638sec
    inet6 fe80::8ff:d8ff:fe49:f511/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 1: ‘ip address show’ output

(a) enX0

(b) 0a:ff:d8:49:f5:11

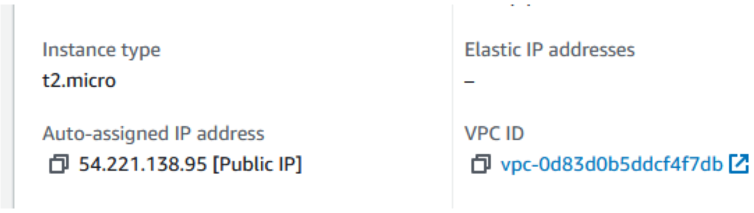
(c) 00001010:11111111:11011000:01001001:11110101:00010001

(d) 6 Segments * 8 Bits = 48 Bits

(e) ff:ff:ff:ff:ff:ff

Q2:

(a) Private IP – see "*Figure 1: 'ip address show' output*" → 172.31.20.240
Public IP → 54.221.138.95



Instance type
t2.micro

Auto-assigned IP address
54.221.138.95 [Public IP]

Elastic IP addresses
–

VPC ID
vpc-0d83d0b5ddcf4f7db

Figure 2: Public IP in AWS Portal

(b) Private IP – see "*Figure 1: 'ip address show' output*" → fe80::8ff:d8ff:fe49:f511

(c) 4 Segments * 8 Bits = 32 Bits

(d) 00110110.11011101.10001010.01011111

(e) 8 Segments * 16 Bits = 128 Bits

Q3:

(a) 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

(b) 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

(c) 32 – 20 = 12 → 12 Bits for Hosts → 2^12 = 4096 Addresses (4096 – 2 = 4094 Hosts)

Q4:

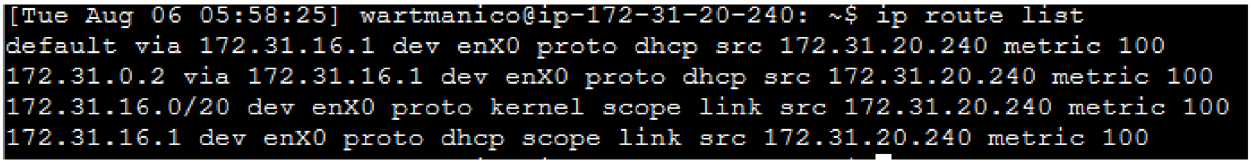
(a) The Netmask determines how many bits of an IP-Address are used for Network-, and Host-addressing (…/20)

(b) BC-Address (highest): 10101100.000011111.00011111.11111111 → 172.31.31.255

(c) 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:

(a)

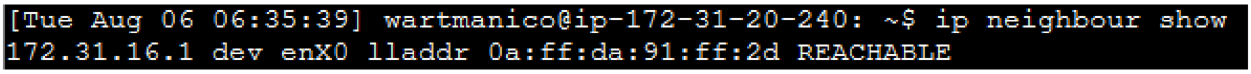


```
[Tue Aug 06 05:58:25] wartmanico@ip-172-31-20-240: ~$ ip route list
default via 172.31.16.1 dev enX0 proto dhcp src 172.31.20.240 metric 100
172.31.0.2 via 172.31.16.1 dev enX0 proto dhcp src 172.31.20.240 metric 100
172.31.16.0/20 dev enX0 proto kernel scope link src 172.31.20.240 metric 100
172.31.16.1 dev enX0 proto dhcp scope link src 172.31.20.240 metric 100
```

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*.

(b)



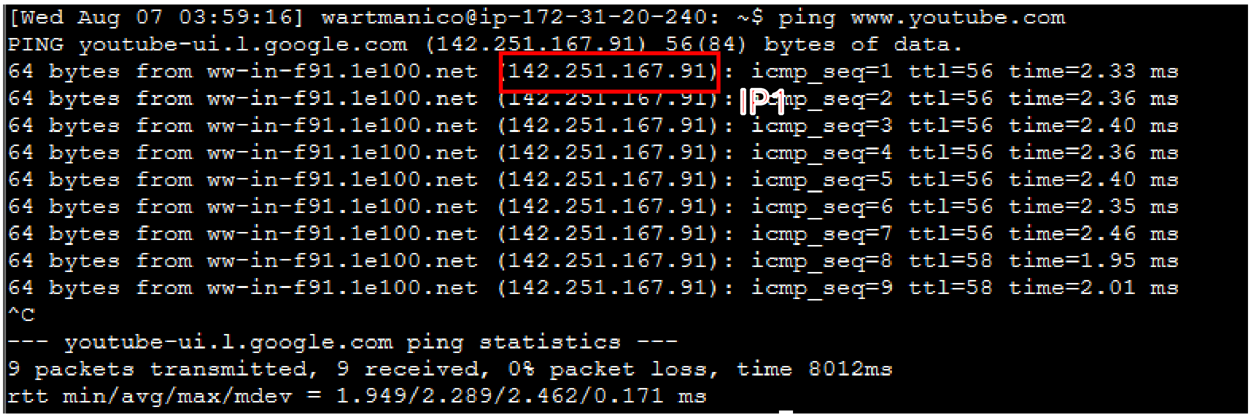
```
[Tue Aug 06 06:35:39] wartmanico@ip-172-31-20-240: ~$ ip neighbour show
172.31.16.1 dev enX0 lladdr 0a:ff:da:91:ff:2d REACHABLE
```

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:

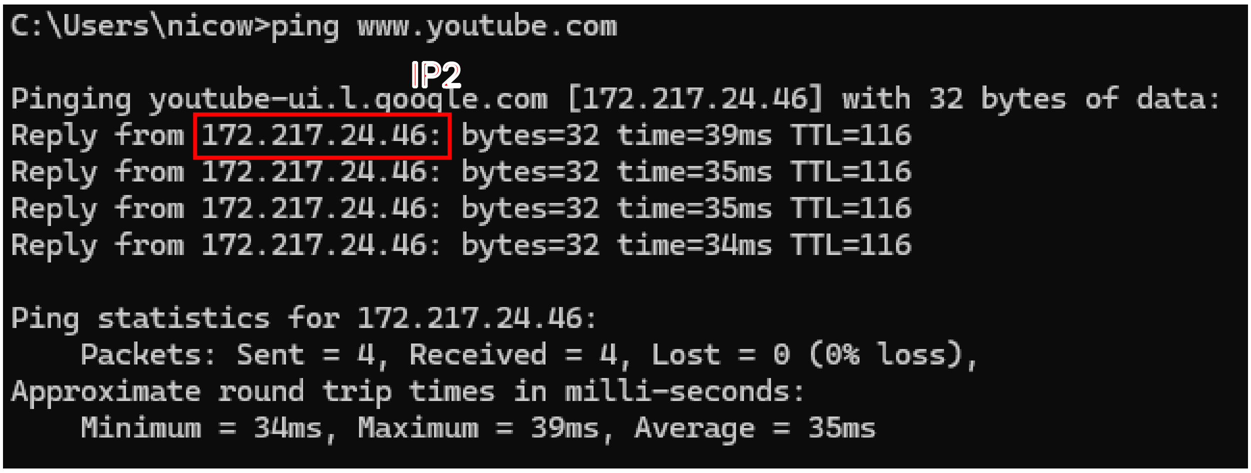
(a)



```
[Wed Aug 07 03:59:16] wartmanico@ip-172-31-20-240: ~$ ping www.youtube.com
PING youtube-ui.l.google.com (142.251.167.91) 56(84) bytes of data.
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=1 ttl=56 time=2.33 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=2 ttl=56 time=2.36 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=3 ttl=56 time=2.40 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=4 ttl=56 time=2.36 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=5 ttl=56 time=2.40 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=6 ttl=56 time=2.35 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=7 ttl=56 time=2.46 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=8 ttl=58 time=1.95 ms
64 bytes from ww-in-f91.1e100.net (142.251.167.91): icmp_seq=9 ttl=58 time=2.01 ms
^C
--- youtube-ui.l.google.com ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8012ms
rtt min/avg/max/mdev = 1.949/2.289/2.462/0.171 ms
```

Figure 5: Ping to YouTube from VM

(b)



```
C:\Users\nicow>ping www.youtube.com

Pinging youtube-ui.l.google.com [172.217.24.46] with 32 bytes of data:
Reply from 172.217.24.46: bytes=32 time=39ms TTL=116
Reply from 172.217.24.46: bytes=32 time=35ms TTL=116
Reply from 172.217.24.46: bytes=32 time=35ms TTL=116
Reply from 172.217.24.46: bytes=32 time=34ms TTL=116

Ping statistics for 172.217.24.46:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 34ms, Maximum = 39ms, Average = 35ms
```

Figure 6: Ping to YouTube from local machine

(c)	RTL_VM_IP1	→	Low	(> 100 ms)
	RTL_VM_IP2	→	High	(< 100 ms)
	RTL_VM_IP1	→	High	(< 100 ms)
	RTL_VM_IP2	→	Low	(> 100 ms)

(d) 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:

(a)

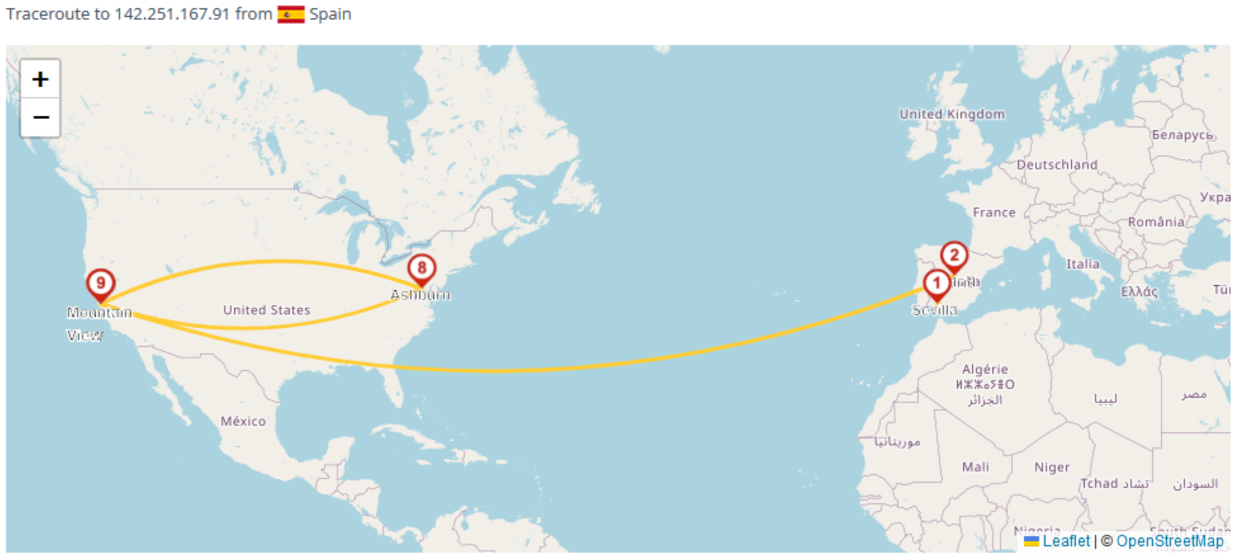

Traceroute to 142.251.167.91 from Spain

Figure 7: Traceroute to 142.251.167.91 from Spain

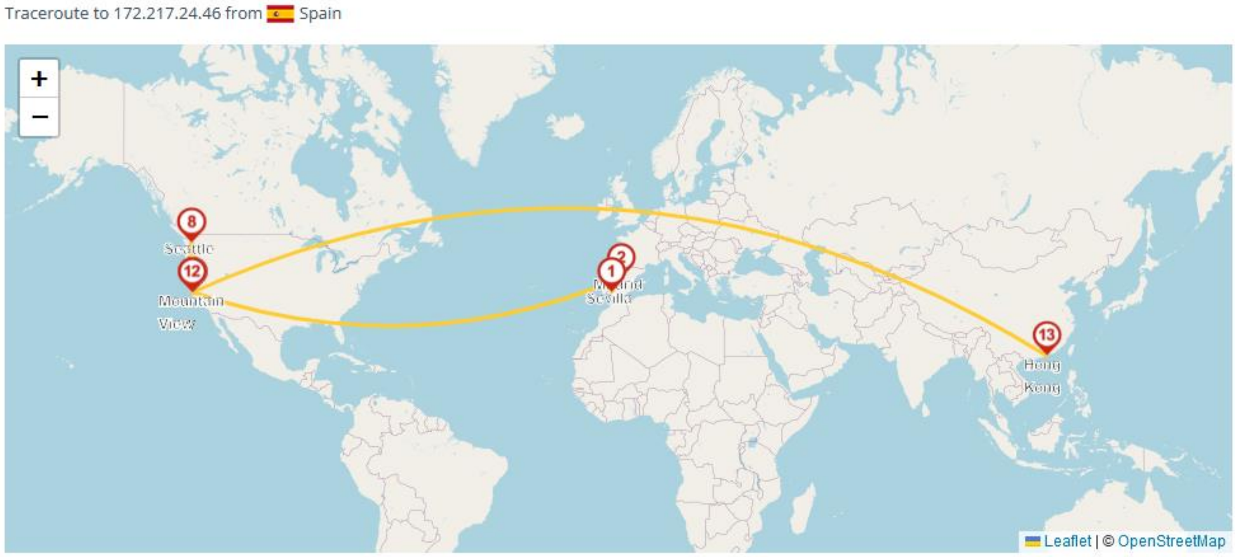
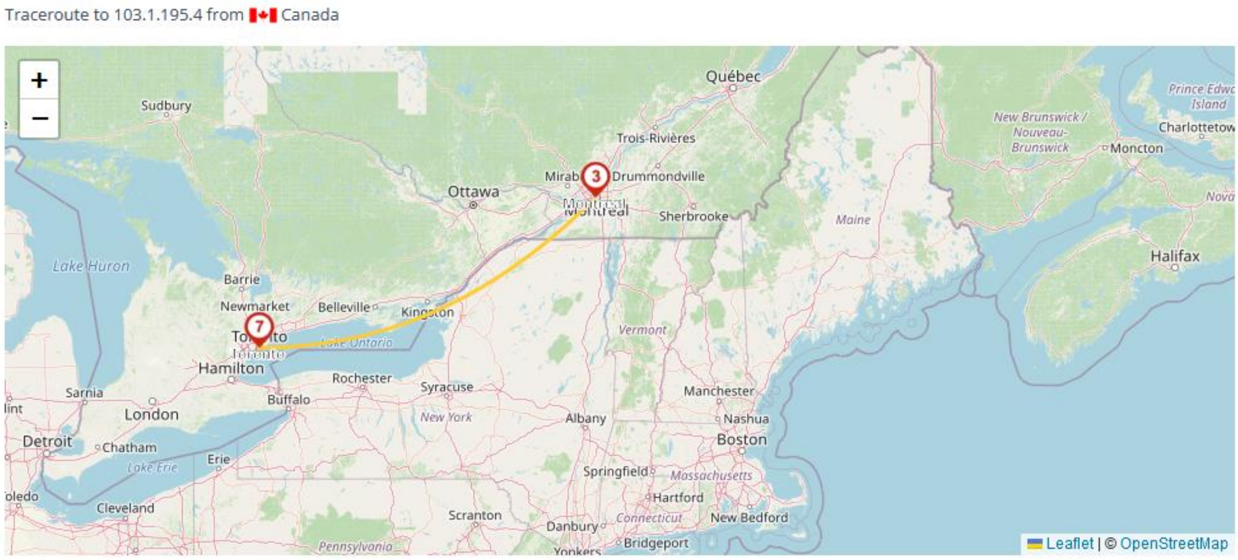

Traceroute to 172.217.24.46 from Spain

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).

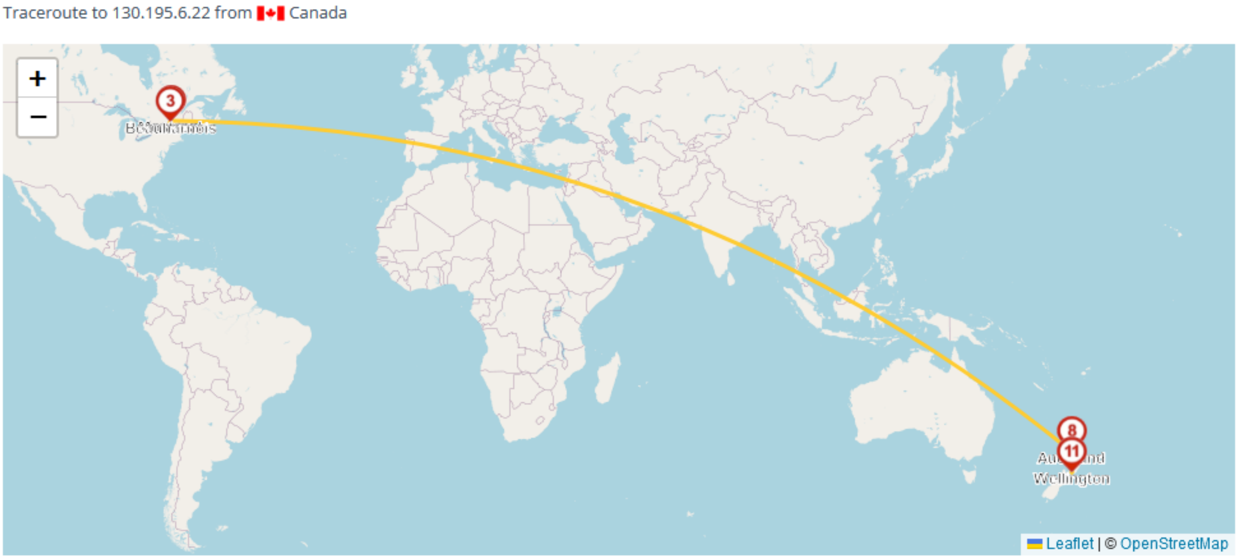
(b)



Traceroute to 103.1.195.4 from 🇨🇦 Canada

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.



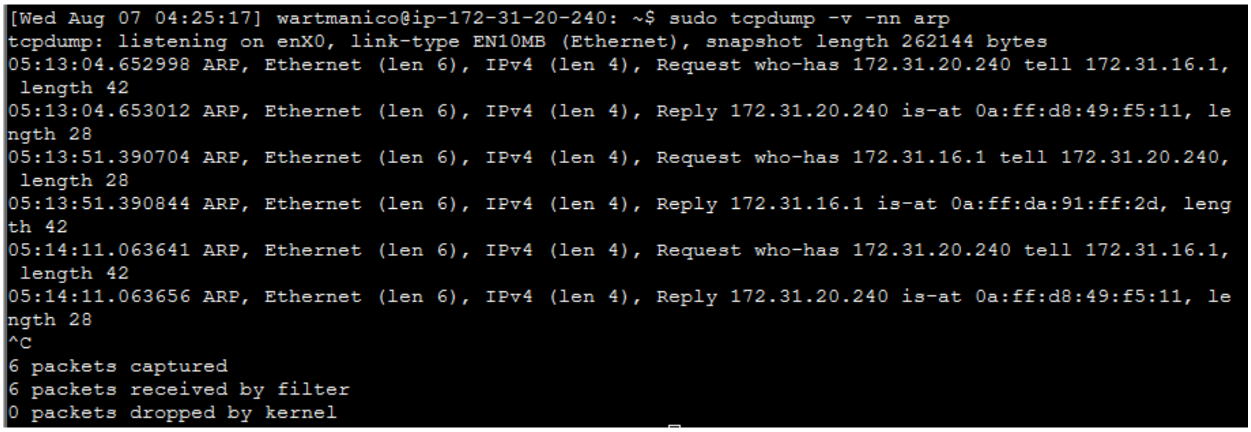
Traceroute to 130.195.6.22 from 🇨🇦 Canada

Figure 10: Traceroute to 130.195.6.22 from Canada

Here the last Hop is located in Wellington.

Q8:

(a)



```
[Wed Aug 07 04:25:17] wartmanico@ip-172-31-20-240: ~$ sudo tcpdump -v -nn arp
tcpdump: listening on enX0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
05:13:04.652998 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.31.20.240 tell 172.31.16.1,
 length 42
05:13:04.653012 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.31.20.240 is-at 0a:ff:d8:49:f5:11, le
ngth 28
05:13:51.390704 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.31.16.1 tell 172.31.20.240,
 length 28
05:13:51.390844 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.31.16.1 is-at 0a:ff:da:91:ff:2d, leng
th 42
05:14:11.063641 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 172.31.20.240 tell 172.31.16.1,
 length 42
05:14:11.063656 ARP, Ethernet (len 6), IPv4 (len 4), Reply 172.31.20.240 is-at 0a:ff:d8:49:f5:11, le
ngth 28
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

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).

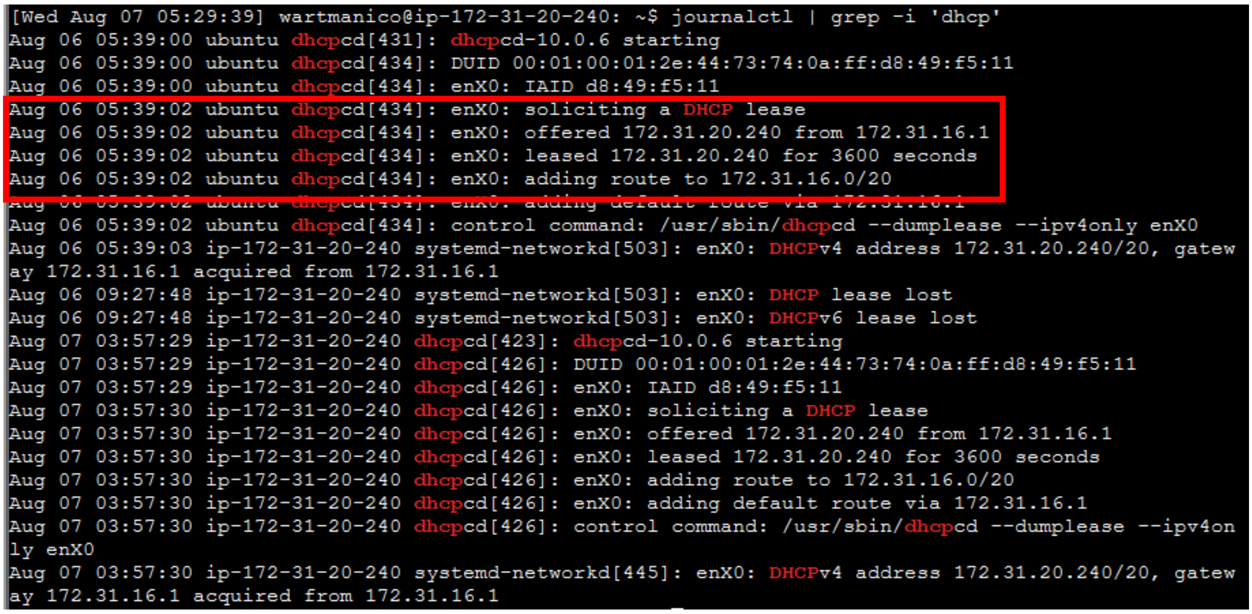
In the Response the VM replies with its MAC Address 0a:ff:d8:49:f5:11

(b) 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.

(c) About every 70 seconds.

Q9:

(a)

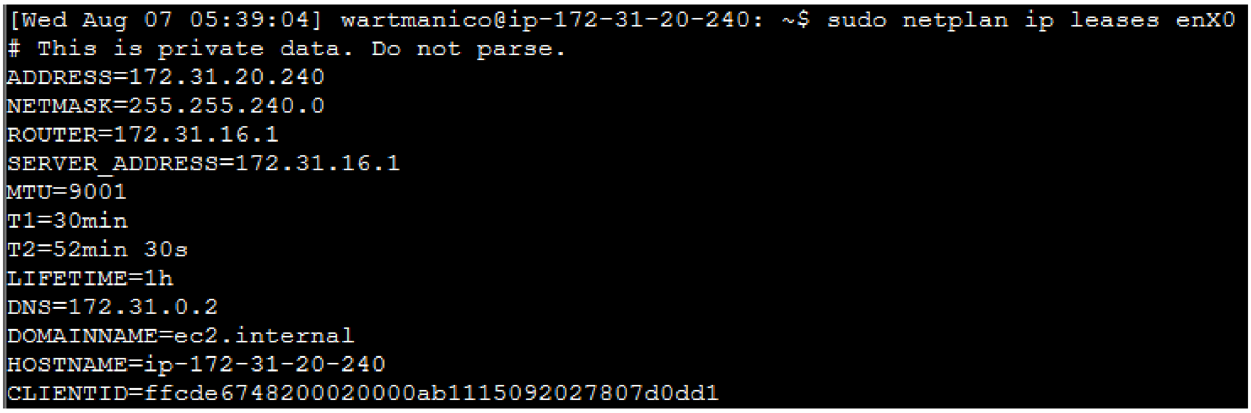


```
[Wed Aug 07 05:29:39] wartmanico@ip-172-31-20-240: ~$ journalctl | grep -i 'dhcp'
Aug 06 05:39:00 ubuntu dhcpcd[431]: dhcpcd-10.0.6 starting
Aug 06 05:39:00 ubuntu dhcpcd[434]: DUID 00:01:00:01:2e:44:73:74:0a:ff:d8:49:f5:11
Aug 06 05:39:00 ubuntu dhcpcd[434]: enX0: IAID d8:49:f5:11
Aug 06 05:39:02 ubuntu dhcpcd[434]: enX0: soliciting a DHCP lease
Aug 06 05:39:02 ubuntu dhcpcd[434]: enX0: offered 172.31.20.240 from 172.31.16.1
Aug 06 05:39:02 ubuntu dhcpcd[434]: enX0: leased 172.31.20.240 for 3600 seconds
Aug 06 05:39:02 ubuntu dhcpcd[434]: enX0: adding route to 172.31.16.0/20
Aug 06 05:39:02 ubuntu dhcpcd[434]: enX0: adding default route via 172.31.16.1
Aug 06 05:39:02 ubuntu dhcpcd[434]: control command: /usr/sbin/dhcpcd --dumplease --ipv4only enX0
Aug 06 05:39:03 ip-172-31-20-240 systemd-networkd[503]: enX0: DHCPv4 address 172.31.20.240/20, gatew
ay 172.31.16.1 acquired from 172.31.16.1
Aug 06 09:27:48 ip-172-31-20-240 systemd-networkd[503]: enX0: DHCP lease lost
Aug 06 09:27:48 ip-172-31-20-240 systemd-networkd[503]: enX0: DHCPv6 lease lost
Aug 07 03:57:29 ip-172-31-20-240 dhcpcd[423]: dhcpcd-10.0.6 starting
Aug 07 03:57:29 ip-172-31-20-240 dhcpcd[426]: DUID 00:01:00:01:2e:44:73:74:0a:ff:d8:49:f5:11
Aug 07 03:57:29 ip-172-31-20-240 dhcpcd[426]: enX0: IAID d8:49:f5:11
Aug 07 03:57:30 ip-172-31-20-240 dhcpcd[426]: enX0: soliciting a DHCP lease
Aug 07 03:57:30 ip-172-31-20-240 dhcpcd[426]: enX0: offered 172.31.20.240 from 172.31.16.1
Aug 07 03:57:30 ip-172-31-20-240 dhcpcd[426]: enX0: leased 172.31.20.240 for 3600 seconds
Aug 07 03:57:30 ip-172-31-20-240 dhcpcd[426]: enX0: adding route to 172.31.16.0/20
Aug 07 03:57:30 ip-172-31-20-240 dhcpcd[426]: enX0: adding default route via 172.31.16.1
Aug 07 03:57:30 ip-172-31-20-240 dhcpcd[426]: control command: /usr/sbin/dhcpcd --dumplease --ipv4on
ly enX0
Aug 07 03:57:30 ip-172-31-20-240 systemd-networkd[445]: enX0: DHCPv4 address 172.31.20.240/20, gatew
ay 172.31.16.1 acquired from 172.31.16.1
```

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

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

(b)



```
[Wed Aug 07 05:39:04] wartmanico@ip-172-31-20-240: ~$ sudo netplan ip leases enX0
# This is private data. Do not parse.
ADDRESS=172.31.20.240
NETMASK=255.255.240.0
ROUTER=172.31.16.1
SERVER_ADDRESS=172.31.16.1
MTU=9001
T1=30min
T2=52min 30s
LIFETIME=1h
DNS=172.31.0.2
DOMAINNAME=ec2.internal
HOSTNAME=ip-172-31-20-240
CLIENTID=ffcde6748200020000ab1115092027807d0dd1
```

Figure 13: 'sudo netplan ip leases enX0' output

Main Information:

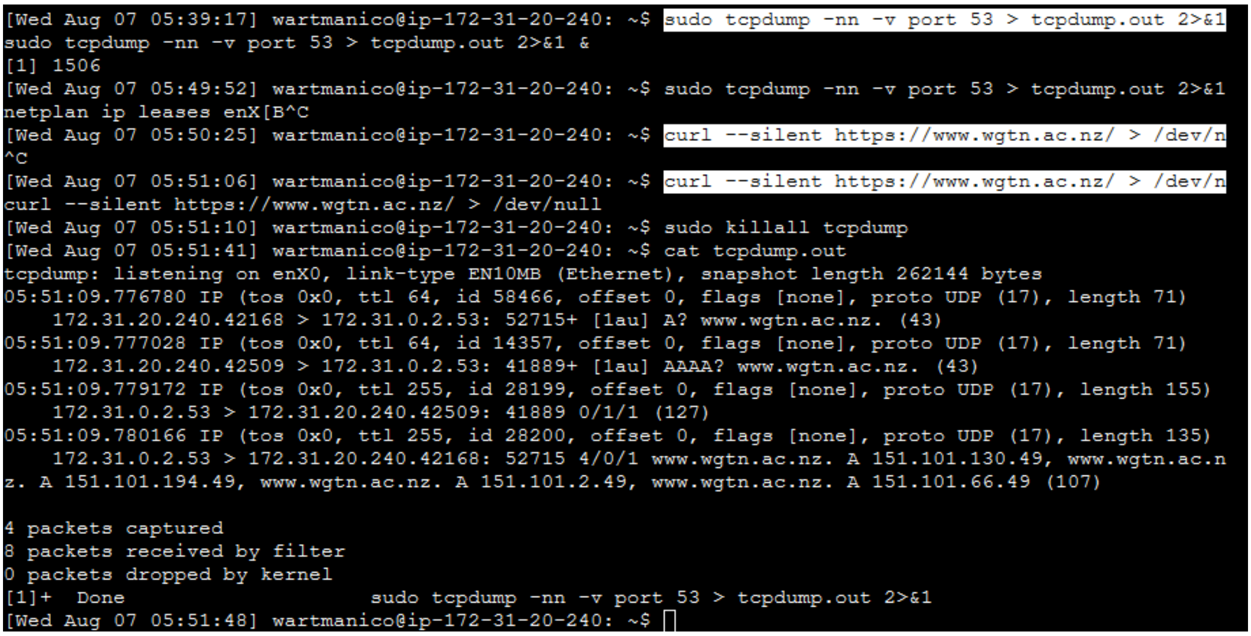
- ADDRESS:	IP-Adress leased
- NETMASK:	subnet mask
- ROUTER:	default gw
- SERVER_ADDRESS:	IP Address of DHCP Server
- LIFETIME:	1h (the 3600 seconds seen before)

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)



```
[Wed Aug 07 05:39:17] wartmanico@ip-172-31-20-240: ~$ sudo tcpdump -nn -v port 53 > tcpdump.out 2>&1
sudo tcpdump -nn -v port 53 > tcpdump.out 2>&1 &
[1] 1506
[Wed Aug 07 05:49:52] wartmanico@ip-172-31-20-240: ~$ sudo tcpdump -nn -v port 53 > tcpdump.out 2>&1
netplan ip leases enX[B^C
[Wed Aug 07 05:50:25] wartmanico@ip-172-31-20-240: ~$ curl --silent https://www.wgtn.ac.nz/ > /dev/n
^C
[Wed Aug 07 05:51:06] wartmanico@ip-172-31-20-240: ~$ curl --silent https://www.wgtn.ac.nz/ > /dev/n
curl --silent https://www.wgtn.ac.nz/ > /dev/null
[Wed Aug 07 05:51:10] wartmanico@ip-172-31-20-240: ~$ sudo killall tcpdump
[Wed Aug 07 05:51:41] wartmanico@ip-172-31-20-240: ~$ cat tcpdump.out
tcpdump: listening on enX0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
05:51:09.776780 IP (tos 0x0, ttl 64, id 58466, offset 0, flags [none], proto UDP (17), length 71)
    172.31.20.240.42168 > 172.31.0.2.53: 52715+ [1au] A? www.wgtn.ac.nz. (43)
05:51:09.777028 IP (tos 0x0, ttl 64, id 14357, offset 0, flags [none], proto UDP (17), length 71)
    172.31.20.240.42509 > 172.31.0.2.53: 41889+ [1au] AAAA? www.wgtn.ac.nz. (43)
05:51:09.779172 IP (tos 0x0, ttl 255, id 28199, offset 0, flags [none], proto UDP (17), length 155)
    172.31.0.2.53 > 172.31.20.240.42509: 41889 0/1/1 (127)
05:51:09.780166 IP (tos 0x0, ttl 255, id 28200, offset 0, flags [none], proto UDP (17), length 135)
    172.31.0.2.53 > 172.31.20.240.42168: 52715 4/0/1 www.wgtn.ac.nz. A 151.101.130.49, www.wgtn.ac.n
z. A 151.101.194.49, www.wgtn.ac.nz. A 151.101.2.49, www.wgtn.ac.nz. A 151.101.66.49 (107)

4 packets captured
8 packets received by filter
0 packets dropped by kernel
[1]+  Done                    sudo tcpdump -nn -v port 53 > tcpdump.out 2>&1
[Wed Aug 07 05:51:48] wartmanico@ip-172-31-20-240: ~$ 
```

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):

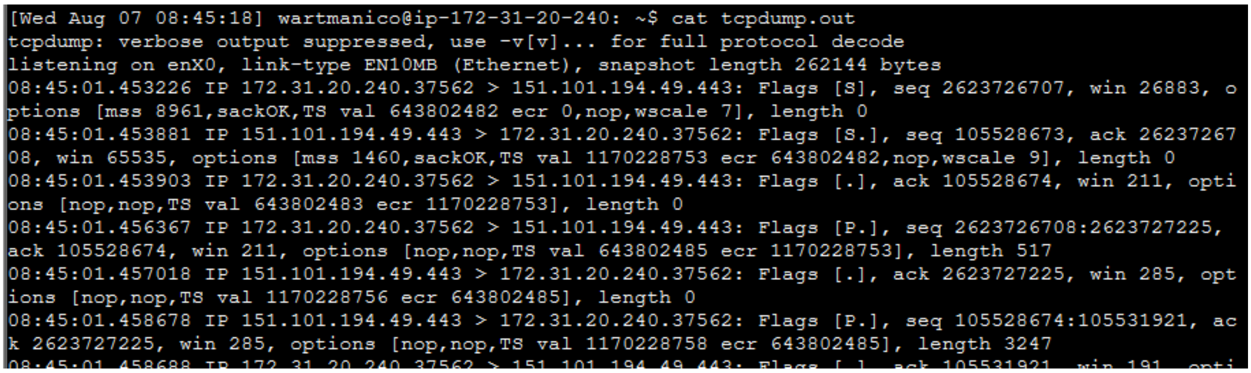
Source → 172.31.20.240 (VM)

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)

(b) 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:



```
[Wed Aug 07 08:45:18] wartmanico@ip-172-31-20-240: ~$ cat tcpdump.out
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enX0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
08:45:01.453226 IP 172.31.20.240.37562 > 151.101.194.49.443: Flags [S], seq 2623726707, win 26883, o
ptions [mss 8961,sackOK,TS val 643802482 ecr 0,nop,wscale 7], length 0
08:45:01.453881 IP 151.101.194.49.443 > 172.31.20.240.37562: Flags [S.], seq 105528673, ack 26237267
08, win 65535, options [mss 1460,sackOK,TS val 1170228753 ecr 643802482,nop,wscale 9], length 0
08:45:01.453903 IP 172.31.20.240.37562 > 151.101.194.49.443: Flags [.], ack 105528674, win 211, opti
ons [nop,nop,TS val 643802483 ecr 1170228753], length 0
08:45:01.456367 IP 172.31.20.240.37562 > 151.101.194.49.443: Flags [P.], seq 2623726708:2623727225,
ack 105528674, win 211, options [nop,nop,TS val 643802485 ecr 1170228753], length 517
08:45:01.457018 IP 151.101.194.49.443 > 172.31.20.240.37562: Flags [.], ack 2623727225, win 285, opt
ions [nop,nop,TS val 1170228756 ecr 643802485], length 0
08:45:01.458678 IP 151.101.194.49.443 > 172.31.20.240.37562: Flags [P.], seq 105528674:105531921, ac
k 2623727225, win 285, options [nop,nop,TS val 1170228758 ecr 643802485], length 3247
08:45:01.458688 IP 172.31.20.240.37562 > 151.101.194.49.443: Flags [.], ack 105531921, win 191, opti
```

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)

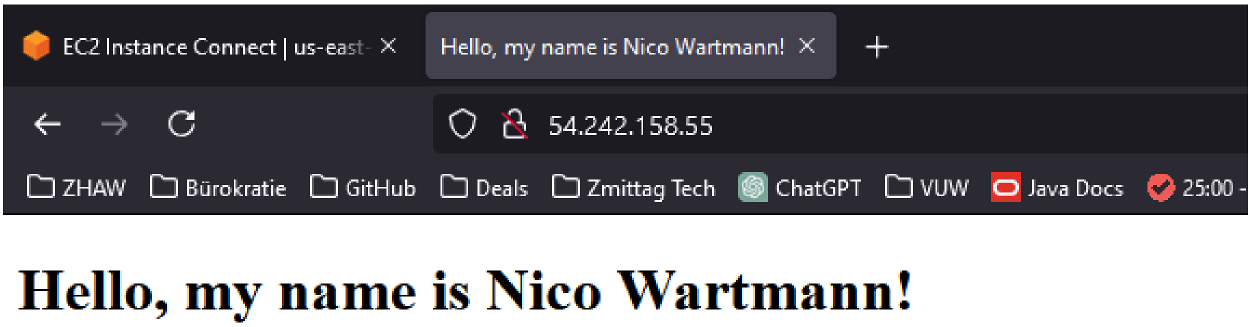
Acknowledgment Number: 2623727225 (same as last → client seq + 1 + 517)

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.

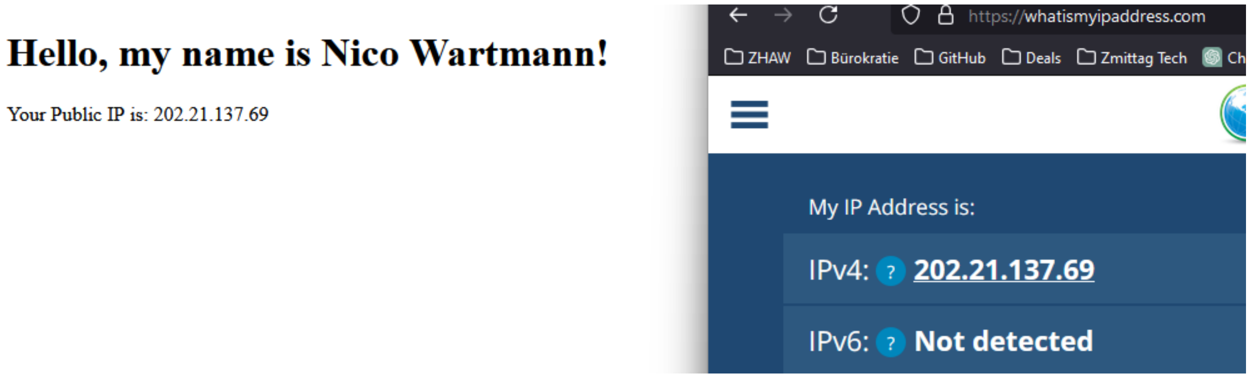
Q13:



Hello, my name is Nico Wartmann!

Figure 16: Simple website with Name

Q14:



Hello, my name is Nico Wartmann!

Your Public IP is: 202.21.137.69

https://whatismyipaddress.com

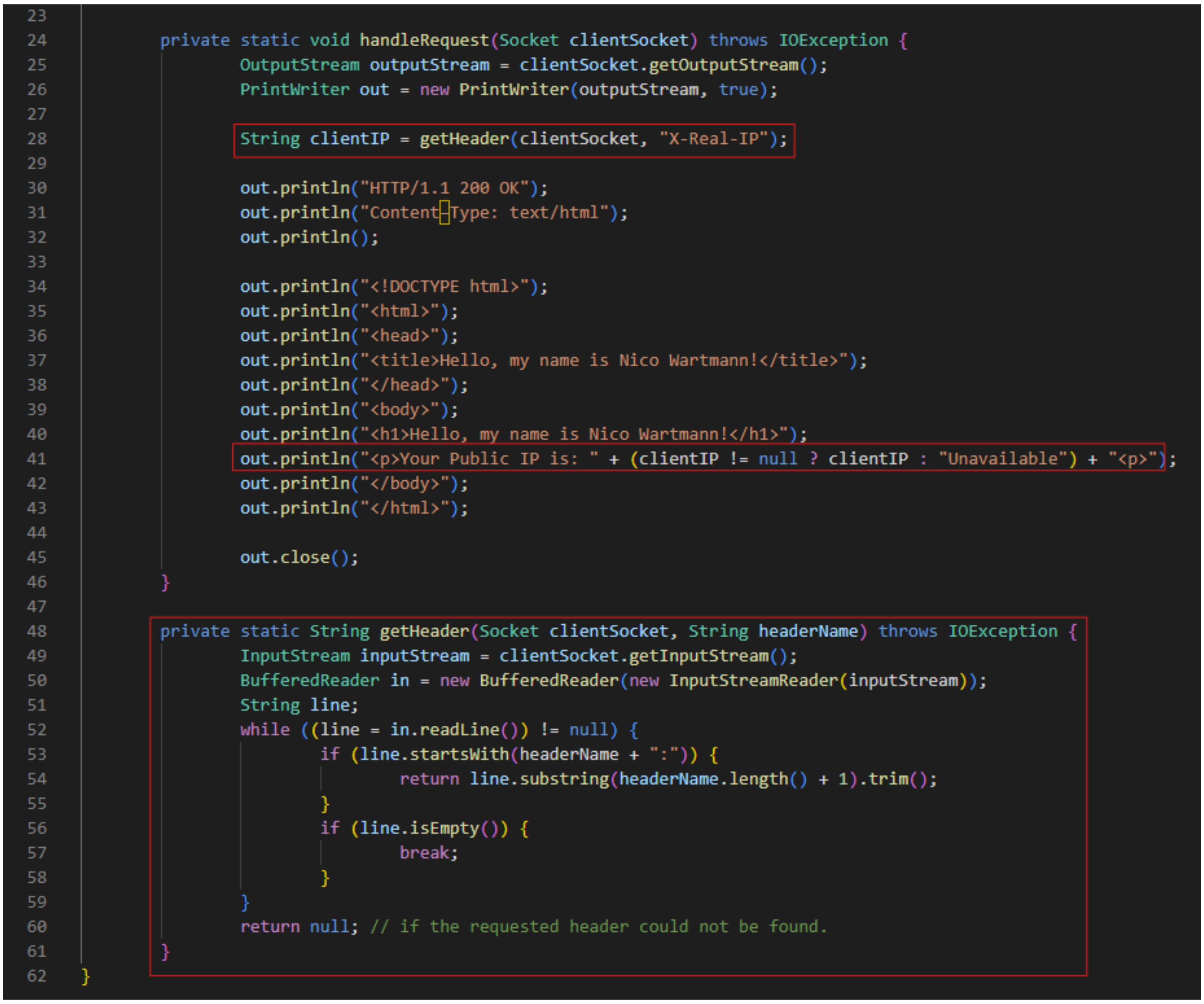
My IP Address is:

IPv4: ? 202.21.137.69

IPv6: ? Not detected

Figure 17: Simple website with client public IP

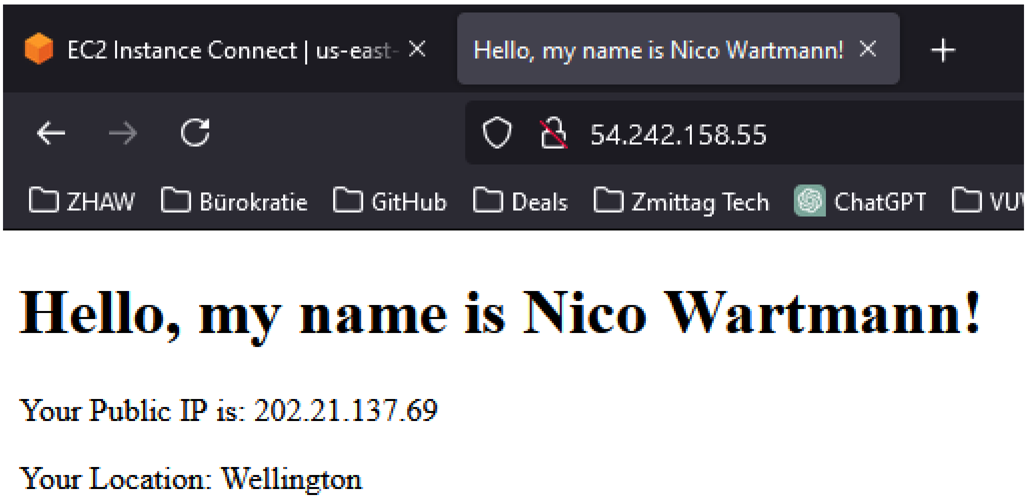
Added Code:



```java

    private static void handleRequest(Socket clientSocket) throws IOException {
        OutputStream outputStream = clientSocket.getOutputStream();
        PrintWriter out = new PrintWriter(outputStream, true);

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

        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: " + (clientIP != null ? clientIP : "Unavailable") + "<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; // if the requested header could not be found.
    }
}
```

Figure 18: Added code to get Client Public IP

Q15:


EC2 Instance Connect | us-east-
Hello, my name is Nico Wartmann!
54.242.158.55
ZHAW
Bürokratie
GitHub
Deals
Zmittag Tech
ChatGPT
VU

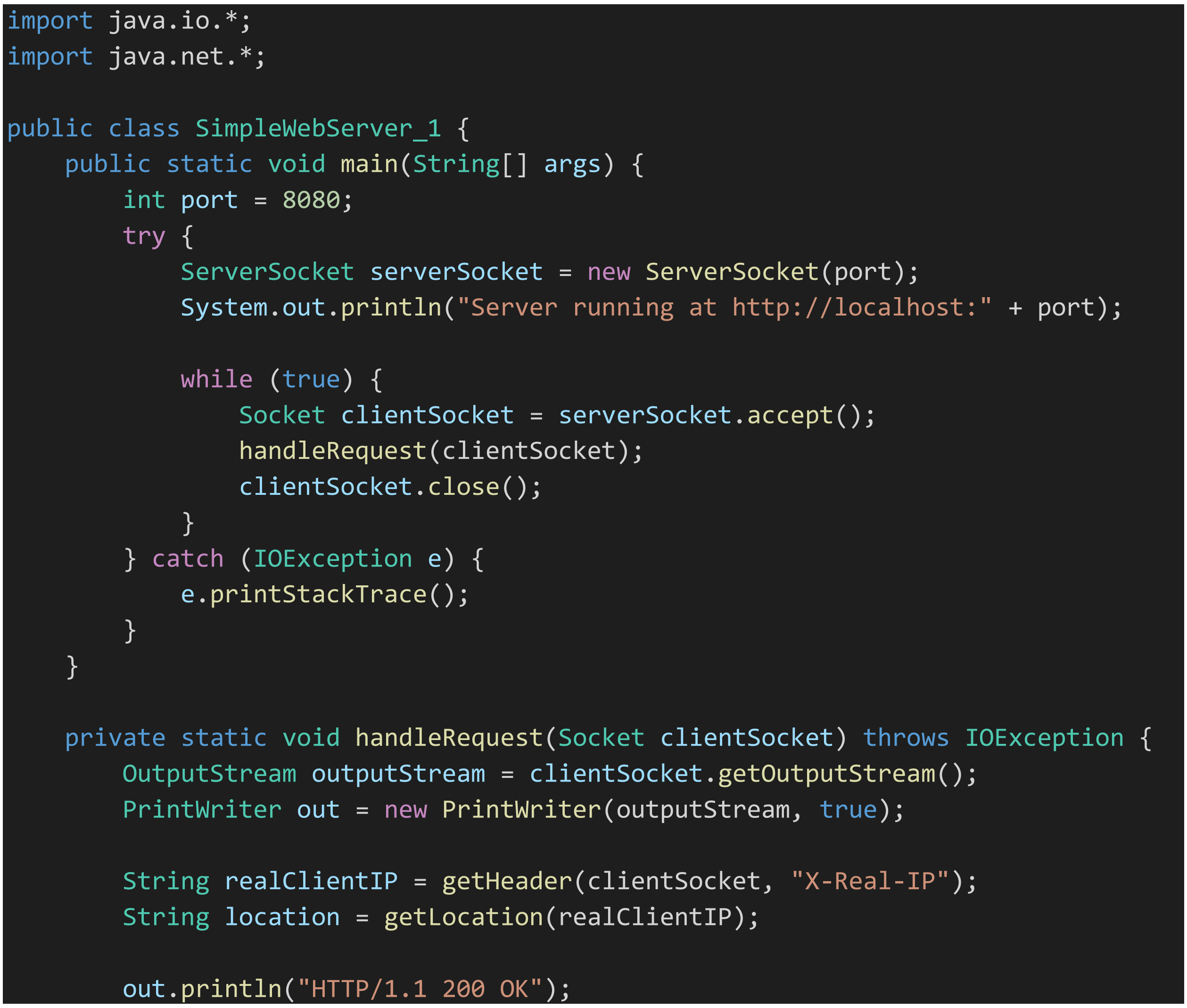
Hello, my name is Nico Wartmann!

Your Public IP is: 202.21.137.69

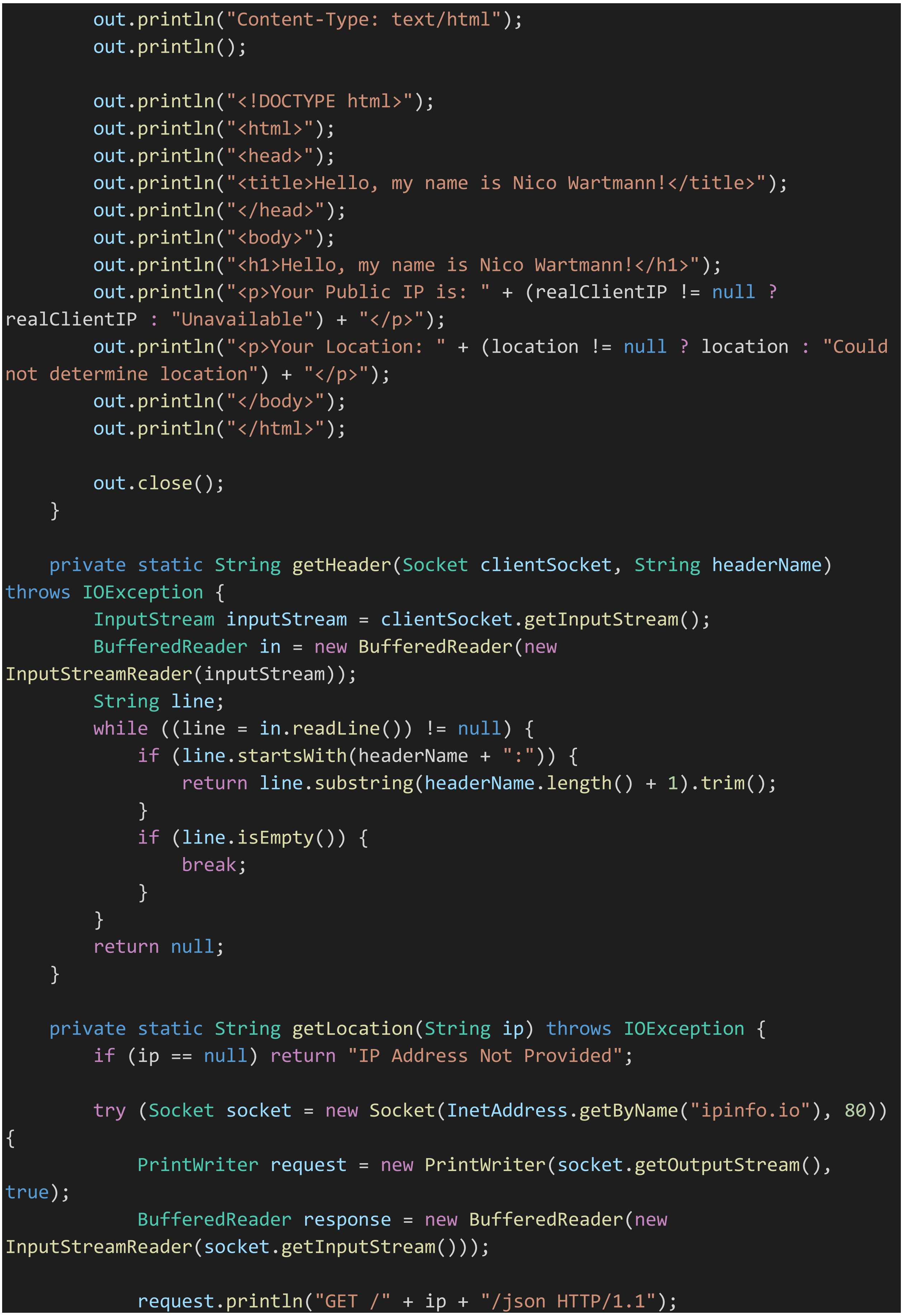
Your Location: Wellington

Figure 19: Simple website including Location

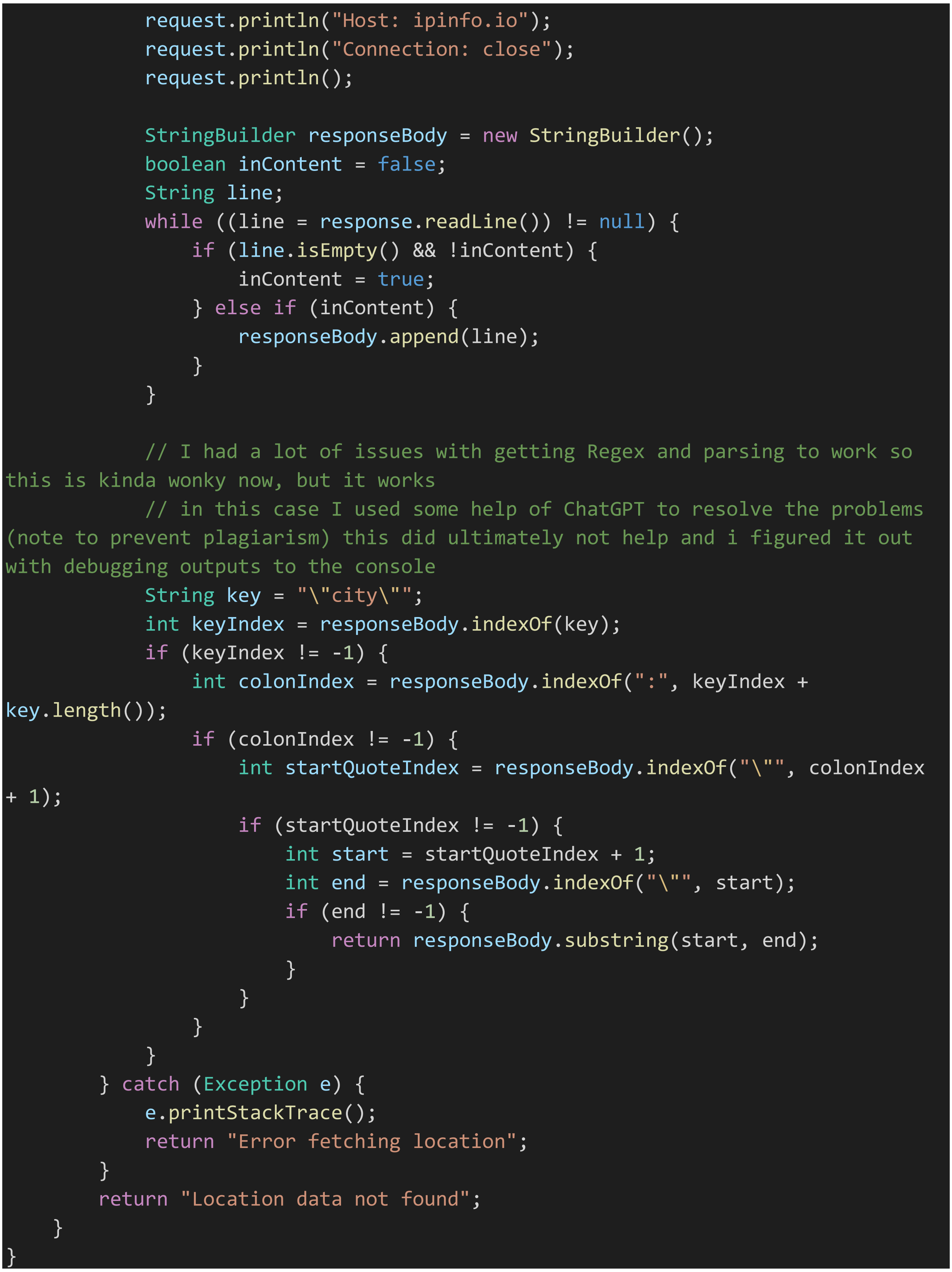
Code:



```
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