ECE 358 Lab 2

Part 1 - Web Server

Code Explanation

At a high level, the Web Server can handle http GET and HEAD requests, and return the requested file or an error if the file is not found.

The websocket is created as below. As HTTP occurs over TCP, a TCP socket was created and configured to be resusuable, and bound to the selected IP address and port.

```
# create and configure websocket as ipv4, tcp, resuable, and ip:port
webserver = socket.socket(family=socket.AF_INET, type=socket.SOCK_STREAM)
webserver.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
webserver.bind((HOME_IP_ADDRESS, PORT))
```

The decoded request's header lines are put into a Python Dictionary, which will be used in later logic.

```
# Process HTTP request and put headers in data structure
def HTTPHeaders(request):
    headers = {}
    for i, line in enumerate(request.splitlines()):
        if i == 0: # status line
            headers["METHOD"] = line.split(' ')[0]
            headers["PATH"] = line.split(' ')[1]
            headers["Version"] = line.split(' ')[2]
        else:
            headers[line.split(":")[0]] = line.split(':')[1]
    return headers
```

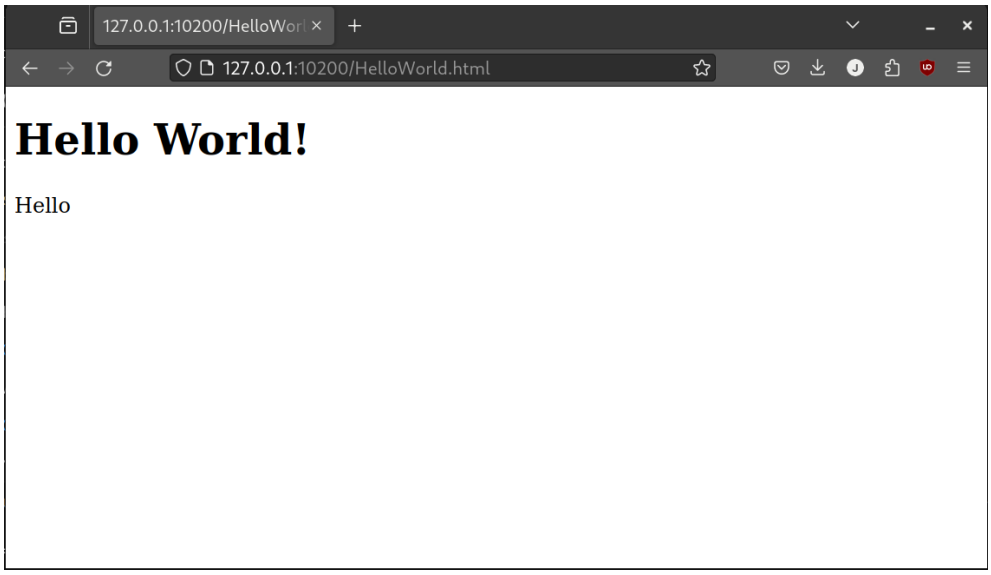
The server checks if the requested file is present, and creates the correct response header.

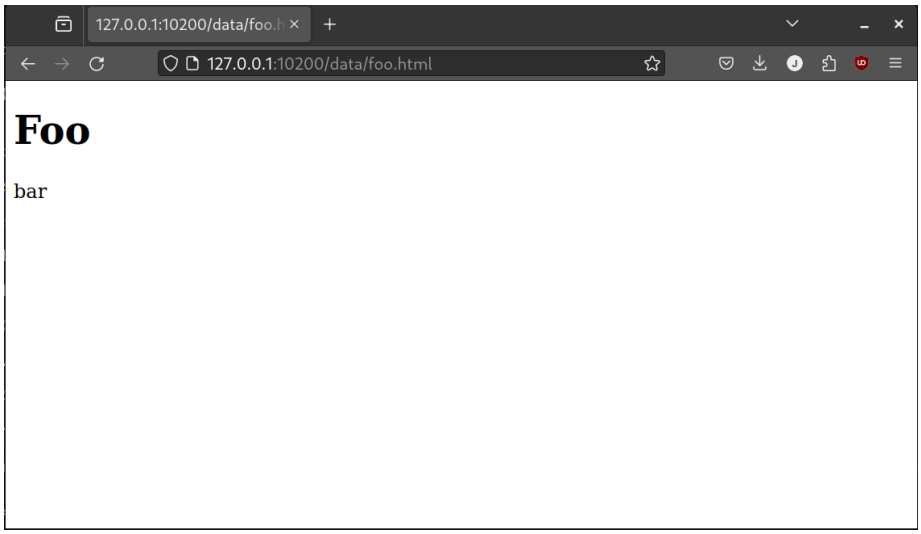
```
def CreateResponseHeader(connection="keep-alive", date="", server="Webserver", last_modified="", content_length="0",
content_type="text/html; charset=UTF-8\r\n"):
    date = time.asctime(time.localtime())
    return f"""
Connection: {connection}
Date: {date}
Server: {server}
Last-Modified: {last_modified}
Accept-Ranges: bytes
Content-Length: {content_length}
Content-Type: {content_type}
"""
```

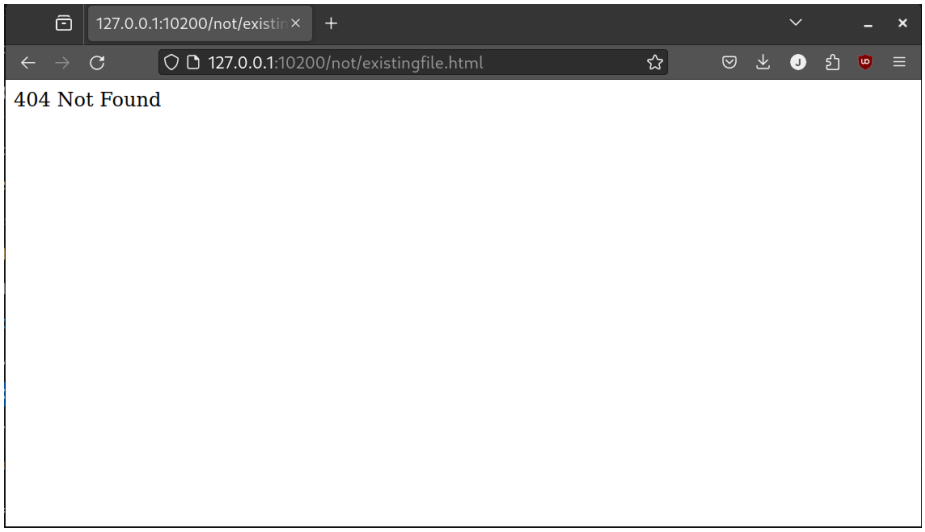
Finally, the response is returned to the client.

```
client_socket.send(response.encode("utf-8"))
```

Web Browser Screenshots

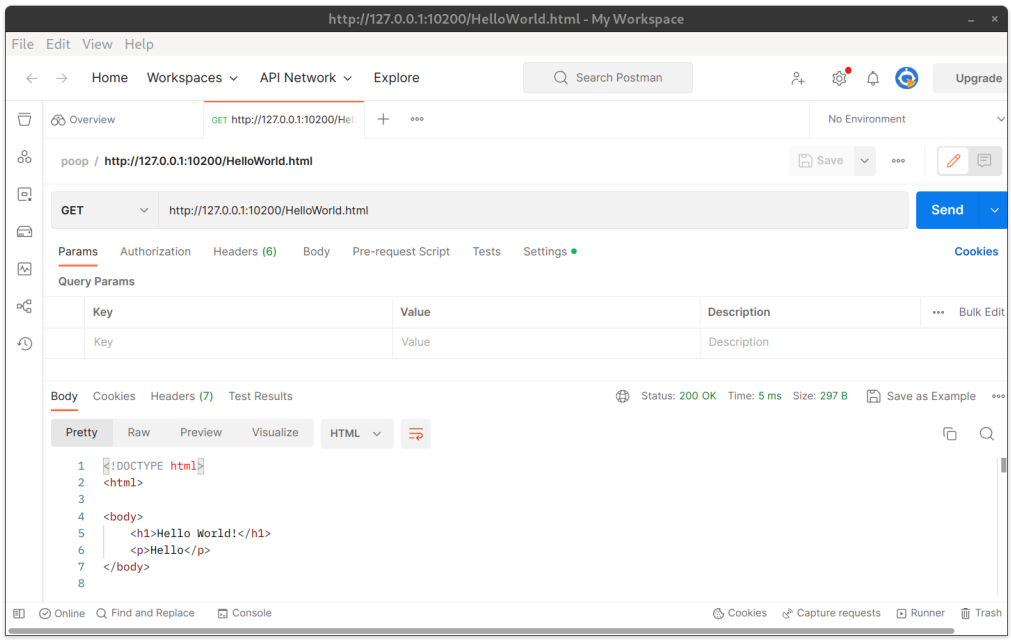


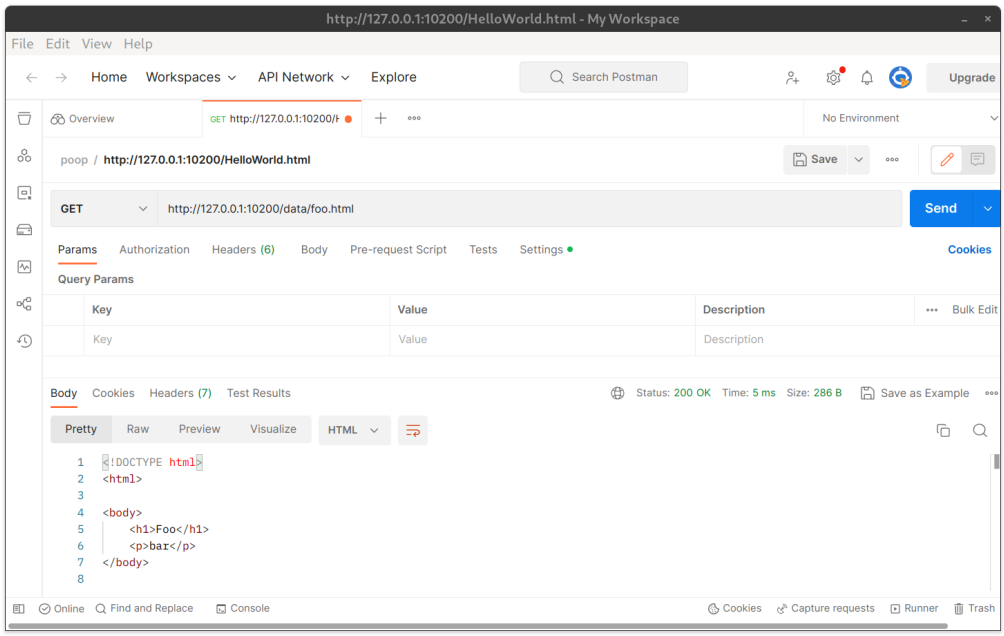


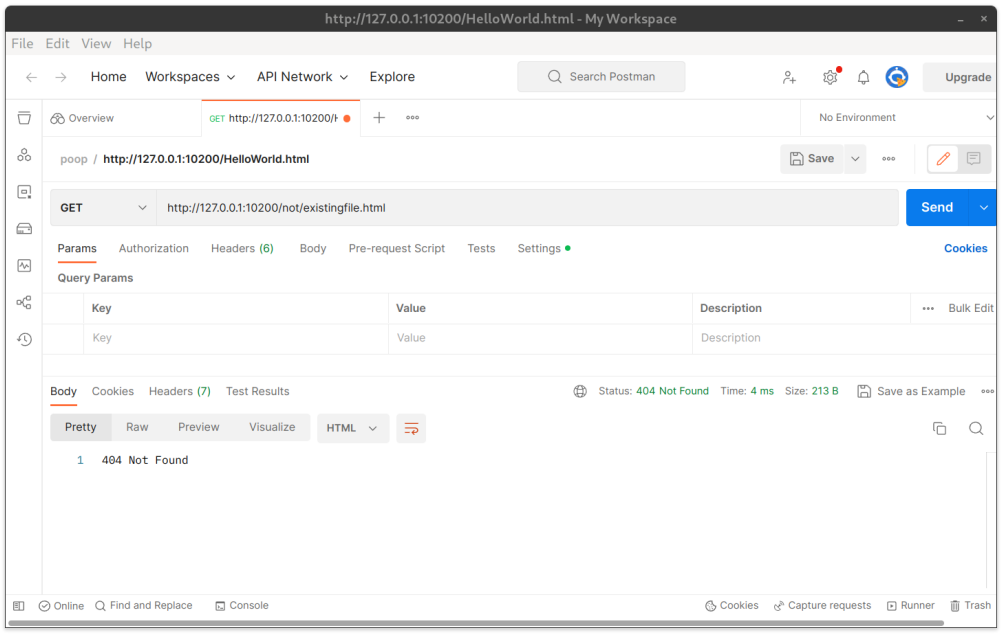


Postman Screenshots

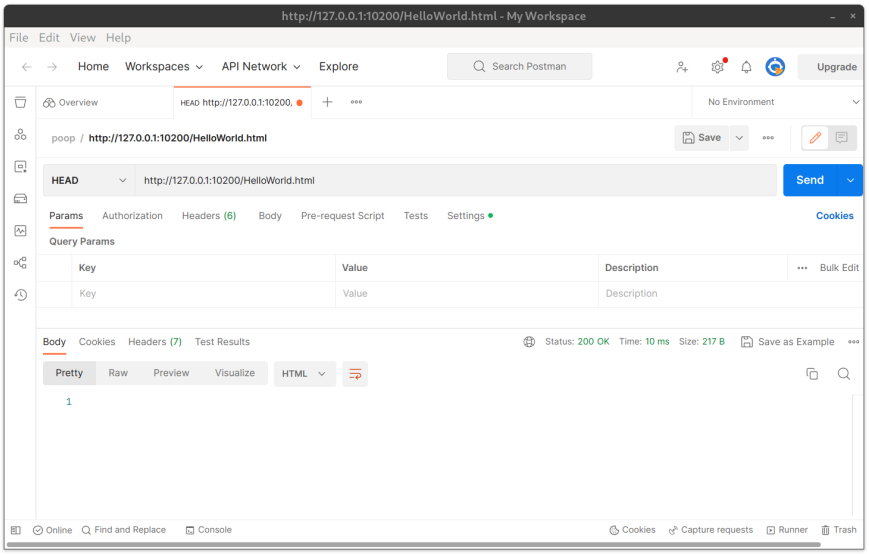
Postman GET Requests

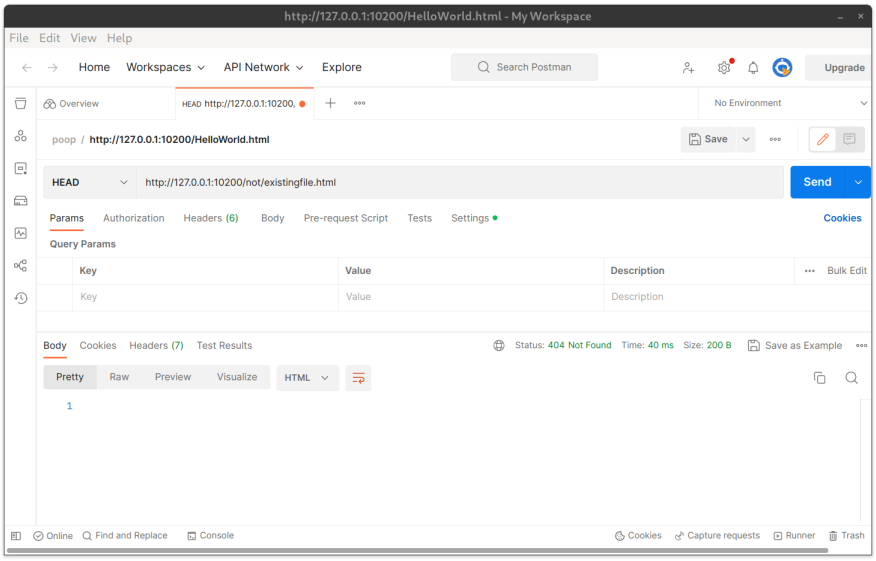



http://127.0.0.1:10200/HelloWorld.html - My Workspace
File Edit View Help
Home Workspaces API Network Explore
Search Postman
Upgrade
Overview
GET http://127.0.0.1:10200/h
No Environment
poop / http://127.0.0.1:10200/HelloWorld.html
Save
GET
http://127.0.0.1:10200/data/foo.html
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
Query Params
Key Value Description Bulk Edit
Key Value Description
Body Cookies Headers (7) Test Results
Status: 200 OK Time: 5 ms Size: 286 B
Save as Example
Pretty Raw Preview Visualize HTML
1 <!DOCTYPE html>
2 <html>
3
4 <body>
5 <h1>Foo</h1>
6 bar
7 </body>
8
Online Find and Replace Console
Cookies Capture requests Runner Trash


http://127.0.0.1:10200/HelloWorld.html - My Workspace
File Edit View Help
Home Workspaces API Network Explore
Search Postman
Upgrade
Overview
GET http://127.0.0.1:10200/h
No Environment
poop / http://127.0.0.1:10200/HelloWorld.html
Save
GET
http://127.0.0.1:10200/not/existingfile.html
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
Query Params
Key Value Description Bulk Edit
Key Value Description
Body Cookies Headers (7) Test Results
Status: 404 Not Found Time: 4 ms Size: 213 B
Save as Example
Pretty Raw Preview Visualize HTML
1 404 Not Found
Online Find and Replace Console
Cookies Capture requests Runner Trash

Postman Head requests





Part 2 - DNS queries

Code

The code for this problem was split into three main components. A DNSMessage class responsible for parsing and building requests, a client which is responsible for creating the queries and parsing responses, and a server class which is responsible for receiving the queries and creating the responses.

The DNSMessage has 3 main data structures which it maintains

```
        self.DNS_HDR = {
```

```
            "id": 0, "qr": 0, "opcode": 0,
            "aa": 1, "tc": 0, "rd": 0, "ra": 0,
            "z": 0, "rcode": 0, "qdcount": 1,
            "ancount": 0, "nscount": 0,
            "arcount": 0
        }
        self.QUERY = {
            "qname": [], # list of octets
            "qtype": int("0001", 16),
            "qclass": int("0001", 16),
        }
        self.ANSWER = {
            "name": int("C00C", 16),
            "type": int("0001", 16),
            "class": int("0001", 16),
            "ttl": 0,
            "rdlen": 0,
            "rddata": []
        }
        self.ANSWERS = []
```

These contain all of the information needed to build and store DNS requests and responses. The class provides from_bytes and to_bytes methods which allow the class to be serialized for transmission as well as initialized from a received byte stream.

The overall flow is:

- Initialize a DNSMessage of type Query, and configure question section based on passed url
- Serialize this class into a byte stream, and use the client to send it to the server
- At the server, initialize a class of type Query from the received byte stream
- Create a response based on the query
- Serialize this response and transmit it back to the client
- At the client, parse this byte stream into a response class and print the results

Output Images

```
mostler   Lab2 $ python3 client.py
Enter domain name: google.com
Response:
google.com: type A, class IN, TTL 260, addr (4) 192.165.1.1
google.com: type A, class IN, TTL 260, addr (4) 192.165.1.10
Enter domain name:
```

⌘K to generate a command

```
mostler   Lab2 $ python3 server.py
Server started, waiting for client to connect...
Client connected: ('127.0.0.1', 64831)
Request:
2b 6f 04 00 00 01 00 00 00 00 00 00 06 67 6f 6f
67 6c 65 03 63 6f 6d 00 00 01 00 01
Response:
2b 6f 84 00 00 01 00 02 00 00 00 00 06 67 6f 6f
67 6c 65 03 63 6f 6d 00 00 01 00 01 c0 0c 00 01
00 01 00 00 01 04 00 04 c0 a5 01 01 c0 0c 00 01
00 01 00 00 01 04 00 04 c0 a5 01 0a
```

```
mostler   Lab2 $ python3 client.py
Enter domain name: wikipedia.org
Response:
wikipedia.org: type A, class IN, TTL 160, addr (4) 192.165.1.4
Enter domain name:
```

⌘K to generate a comm

```
mostler   Lab2 $ python3 server.py
Server started, waiting for client to connect...
Client connected: ('127.0.0.1', 64835)
Request:
9c f3 04 00 00 01 00 00 00 00 00 00 09 77 69 6b
69 70 65 64 69 61 03 6f 72 67 00 00 01 00 01
Response:
9c f3 84 00 00 01 00 01 00 00 00 00 09 77 69 6b
69 70 65 64 69 61 03 6f 72 67 00 00 01 00 01 c0
0c 00 01 00 01 00 00 00 a0 00 04 c0 a5 01 04
```

Type	Key
DNS HEADER	ID
	FLAGS
	QDCOUNT
	ANCOUNT
	NSCOUNT
	ARCOUNT
QUERY	QNAME
	QTYPE
	QCLASS

Answer is Pink

Google.com
Request:
15 f4 04 00 00 01 00 00 00 00 00 00 06 67 6f 6f
67 6c 65 03 63 6f 6d 00 00 01 00 01
Response:
15 f4 04 00 00 01 00 00 00 00 00 00 06 67 6f 6f
67 6c 65 03 63 6f 6d 00 00 01 00 01 c0 0c 00 01
00 01 00 00 01 04 00 04 c0 a5 01 01 c0 0c 00 01
00 01 00 00 01 04 00 04 c0 a5 01 0a

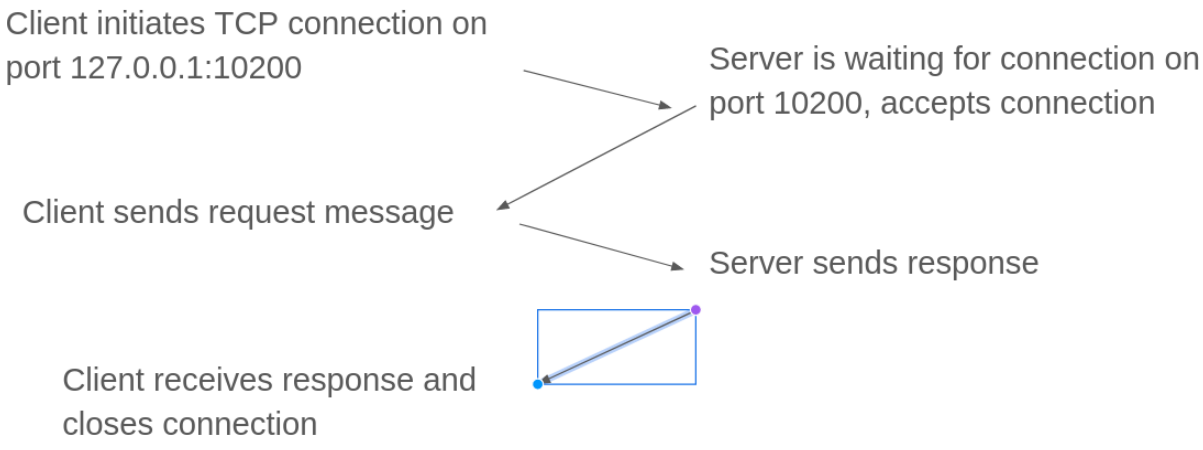
Wikipedia.org
Request:
9c f3 04 00 00 01 00 00 00 00 00 00 09 77 69 6b
69 70 65 64 69 61 03 6f 72 67 00 00 01 00 01
Response:
9c f3 04 00 00 01 00 00 00 00 00 00 09 77 69 6b
69 70 65 64 69 61 03 6f 72 67 00 00 01 00 01 c0
0c 00 01 00 01 00 00 00 a0 00 04 c0 a5 01 04

Part 3 - Written Answers

1. A socket is a software abstraction which allows two peers to communicate over a network. It involves a unique combination of two things: an IP address and a port number

2. Sockets work by first establishing a connection, and then allowing data to be sent and received through this connection using a protocol which is decided on during initialization. They are required because they allow communication without any context about what the two endpoints are or what the nature of the network is.
3. The types of internet sockets are:
    a. Datagram sockets - Used in UDP, connectionless, each packet is routed individually
    b. Stream Sockets - Generally implemented via TCP, connection oriented
    c. Raw sockets - Not wrapped in any protocol, just enable transmission of raw data
4. The above types are used for:
    a. Video/voice streaming
    b. File transfer
    c. Inter-process communication
5.

socket()	This function is used to create a new socket object.
bind()	Used to associate a socket with a specific network address and port number
connect()	establish a connection to a remote server
listen()	on a server-side socket, listen for incoming connections from clients
accept()	accept an incoming connection from a client
close()	used to close a socket and release its resources



6.

7. A socket is executed on the transport layer.
8.

Protocol	**Port #**	**Common Function**
HTTP	80	Serve websites by HTML
FTP	20/21	File transfer
IMAP4	993	Email
SMTP	25/465	Email
Telnet	23	Access remote computers
POP3	995	Email

9.
10. DNS is a system that translates domain names (human readable strings like www.google.com) into IP addresses that computers can understand. It works by using a hierarchical structure of servers to store and distribute domain name information. For a given domain name, the DNS system is queried to find the corresponding IP address, allowing the user's device to establish a connection with the correct server hosting the requested website.