HTTP Hyper Text Transfer Protocol

---

so what is a http and how does it play an important role in backend applications

So Http stands for Hyper Text Transfer Protocol as u can see its in the name that it is basically a Protocol that is a set of rule to Transfer hyper Text which are basically html

pages ->so we can conculde at a high level that http is a Protocol which binds a set of rule for machine to communicate or specifically for website ie client to communicate with server

ie http allows frontend to communicate with backend

An overview of HTTP

HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol,

which means requests are initiated by the recipient(client), usually the Web browser. A complete document is typically constructed from resources such as

text content, layout instructions, images, videos, scripts, and more.

Web document or client ---------->sends a request that get this document like html page and styling sheet to render site to backend---->Server

Web document or client<---Sends html page and styling script after receving the request from frontend-----Server

Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client are called requests and the messages sent

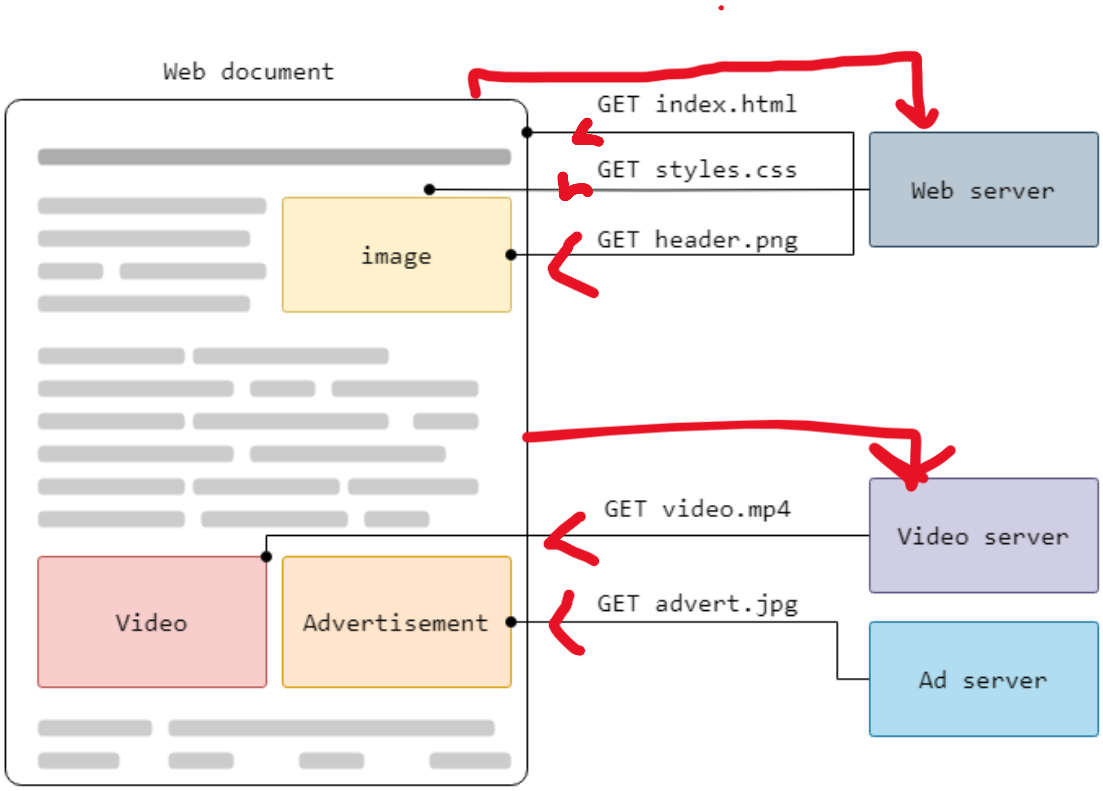
by the server as an answer are called responses.

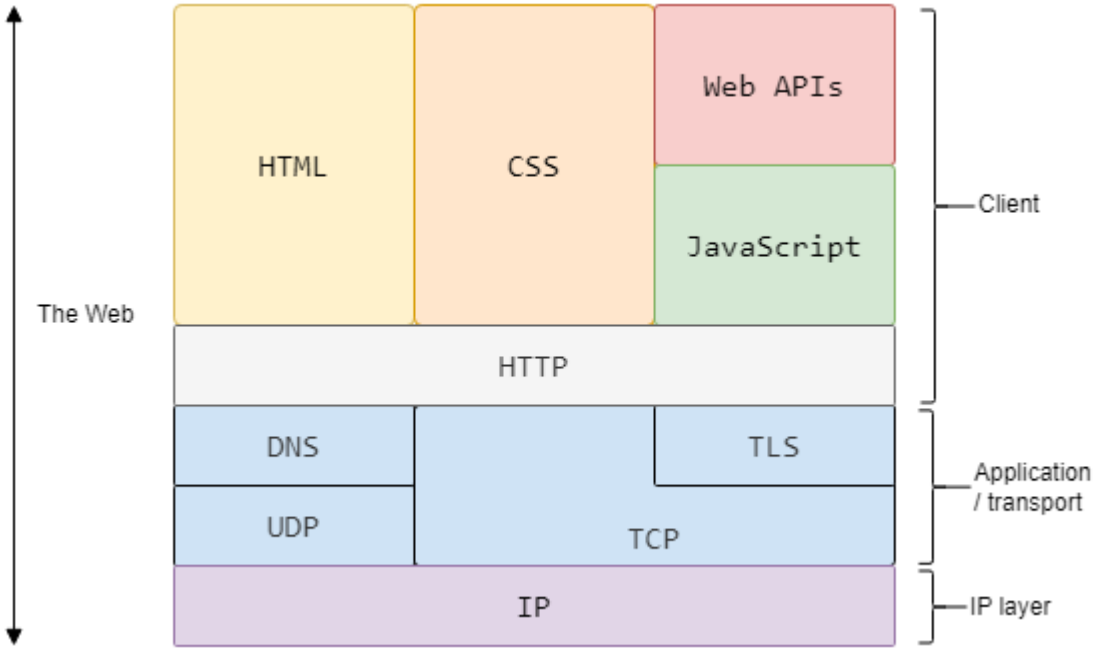
So Http is basically a handshake protocol which means that client sends a request and recives a response from server establishing a handshake

Frontend or the client is what user interact with the user interface created by using html css js and react is what serves as frontend

Backend are what the frontend interacts with mere user do not have the authority to interact with backend untill and unless a backend engineer or a engineer is involved in the process

backend host server and database its written in js,java,python,golang,rust,c or c++





HTTP Protocol

In the end, its the client throwing some information at a server

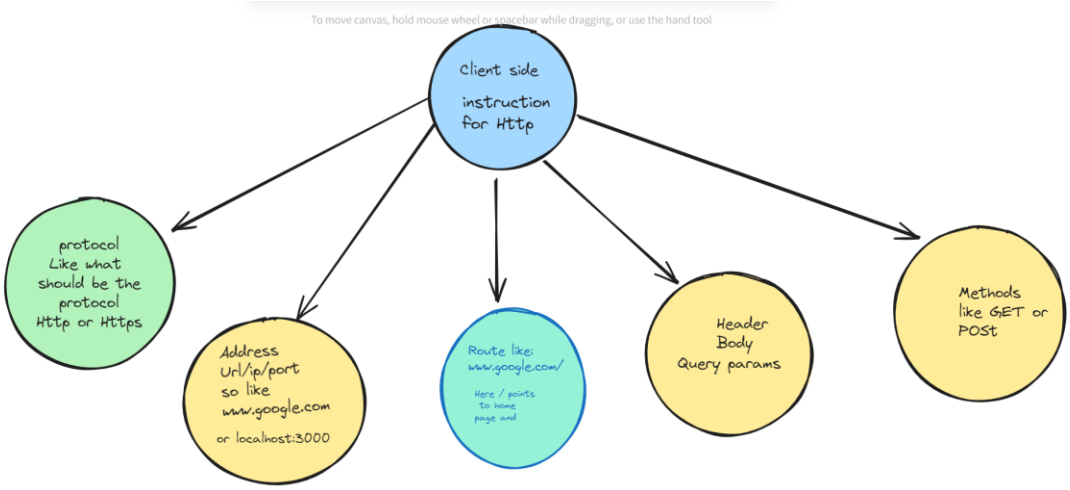
Server doing something with that information

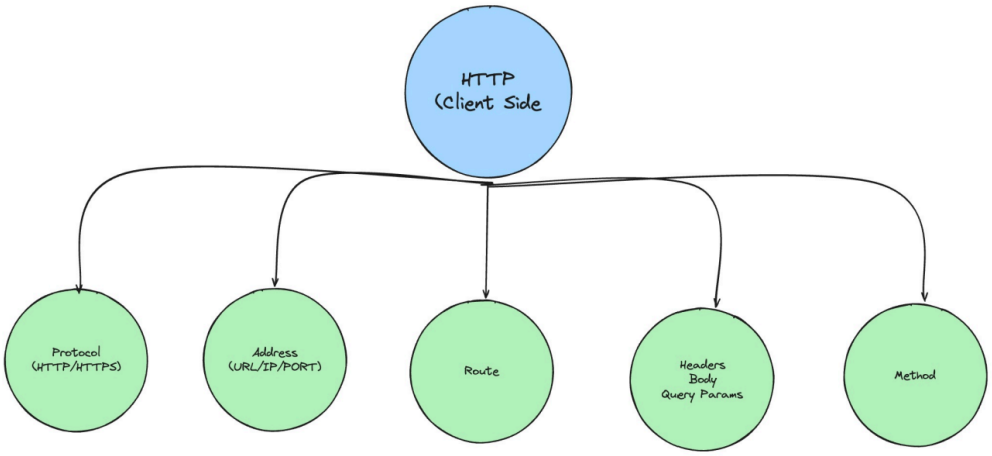
Server responding back with the final result

Think of them as functions, where

1. Arguments are something the client sends
2. Rather than calling a function using its name, the client uses a URL
3. Rather than the function body, the server does something with the request
4. Rather than the function returning a value, the server responds with some data

Now How to Design a Protocol -so that client can send request and server can revert the response





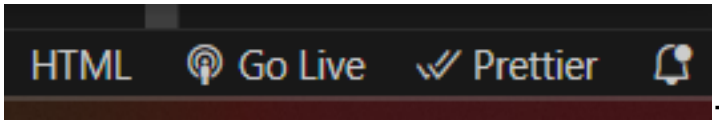
So lets understand every component in this :

1.Prototcol :so basically when client sends a request it need to specify which protocol it is following to communicate with backend basically so 95 percent of the time client

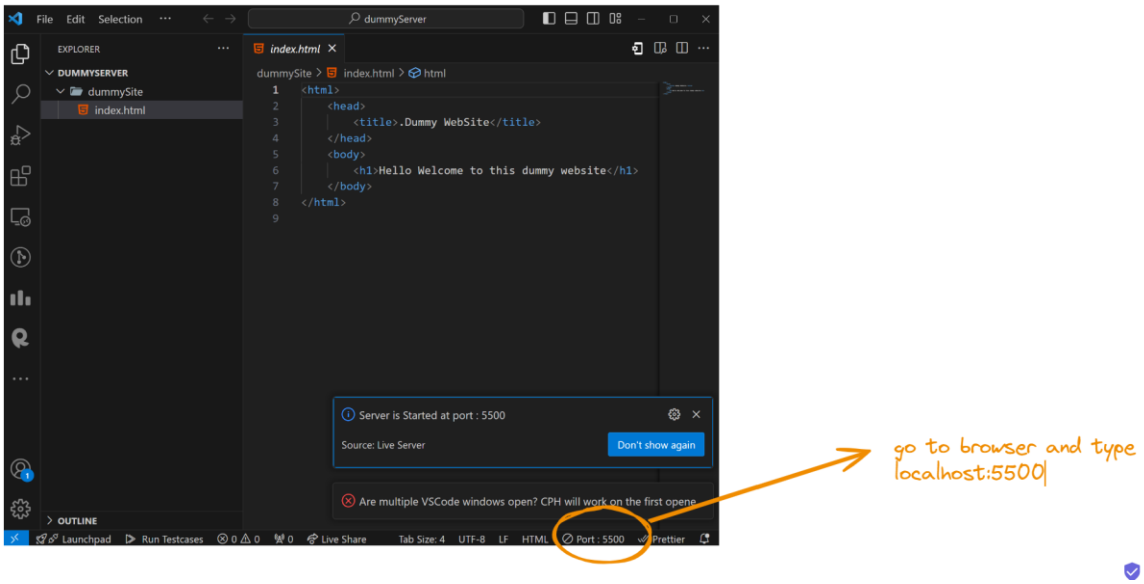
follows either htttp or https where http stands for hyper text transfer protocol and https stands for hyper text transfer protocol secure .

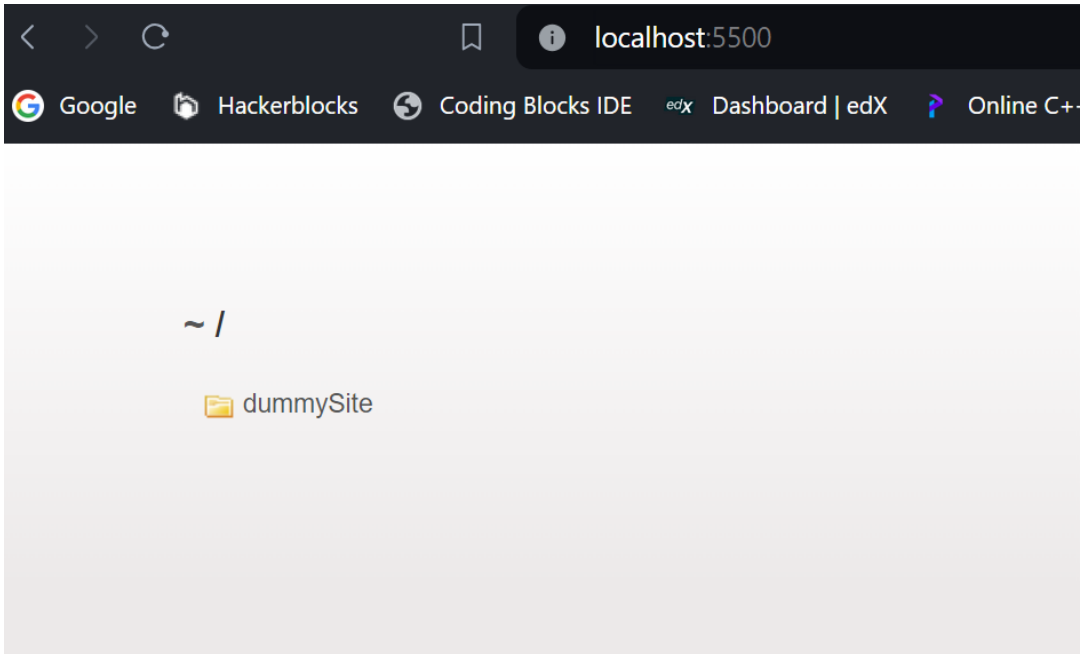
2.Address/port : Specifies the location of the file or resource within the website's directory structure that basically where in the backend the website is hosted so when testing a website on your local network we see inside the search bar of browser one thing that is :localhost:3000 which basically means localhost →points to our local machine acting as server and 3000→refers to the port number where files are being served sometime localhost is also represented by the machine ip address :localhost(127.0.0.1) and then we can add port number to access website: http://127.0.0.1:3000/

Lets see this in action go to vscode -download a live server extension just to test out our html page on live server basically it will create a backend channel to host html page on our machine

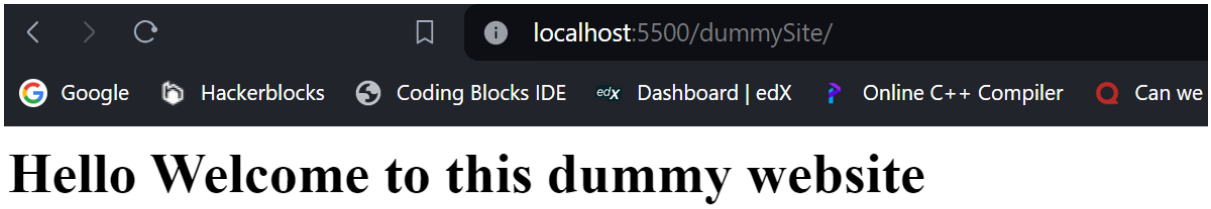


→click on go live





So this is our local host ---serving a directory dummySIte—click on it



So this is what address and port number is localhost refers to http://127.0.0.1/

So this is basically a url →http://127.0.0.1:3000/ but if we want to access this url on some other device we cant cause this website is hosted on local machine to make it available for other devices we have to make it accessible a local network so if we want to open this website in our mobile for that our mobile and machine should be connected with same wifi network

And then find the ip address of the wifi :if using window use ipconfig command

```
D:\coding\development\Web Development\Cohort\02.Week2-Express\00.HTTP>ipconfig
```

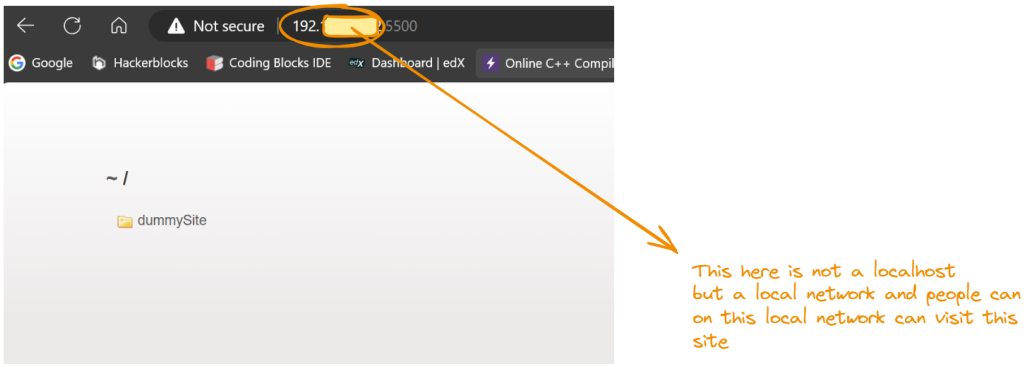
And then we can see ip-address of all the network with whom our machine connected

```
Wireless LAN adapter WiFi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . :
   IPv4 Address. . . . . . . . . . . :
   Subnet Mask . . . . . . . . . . . :
   Default Gateway . . . . . . . . . :
```

Note the ipv4 address lets assume it to be :aaaa.bbbb.cccc

Now to open the website inside our mobile head to chrome or safari and inside the search bar type:http://aaaa.bbbb.cccc:3000—the port number on which the file is being served on local machine →so that's how we can surf our website on phone while its being deployed or tested on local network



But this is not a feasible way as website are deployed on servers across the sea and those server run on different network which is not accessible by us so knowing there ip address is very difficult so company buys domain name which they bind with there ip address

Imagine the internet as a vast city. Each house in this city has a unique numerical address (IP address) to identify it. However, remembering these numbers for every house is impractical. That's where domain names come in.

A domain name is like a street address. It's a human-readable name that corresponds to a specific IP address. For example, instead of remembering 192.168.1.100, you can easily remember google.com.

The Role of DNS

To bridge the gap between domain names and IP addresses, we have the **Domain Name System (DNS)**. It's like a phonebook for the internet. When you type a domain name into your web browser:

1. What is DNS? | How DNS works - Cloudflare

www.cloudflare.com

1. **DNS Lookup:** Your computer sends a query to a DNS server to find the corresponding IP address for the domain name.

1. How Do Domains Work? The Ultimate Guide - Bluehost

 www.bluehost.com

2. **IP Address Retrieval:** The DNS server returns the IP address associated with the domain.

[1. What is DNS? – Introduction to DNS - AWS - Amazon.com]

[aws.amazon.com]

3. **Connection Establishment:** Your computer uses the IP address to connect to the web server hosting the website.
4. **Website Loading:** The web server sends the website's files to your computer, and your browser displays the content.

[1. What Is a Web Server? - Hostinger]

[www.hostinger.in]

Key Components of a Domain Name

- **Top-Level Domain (TLD):** The suffix at the end (e.g., .com, .net, .org).

[1. What is a top-level domain? - Cloudflare]

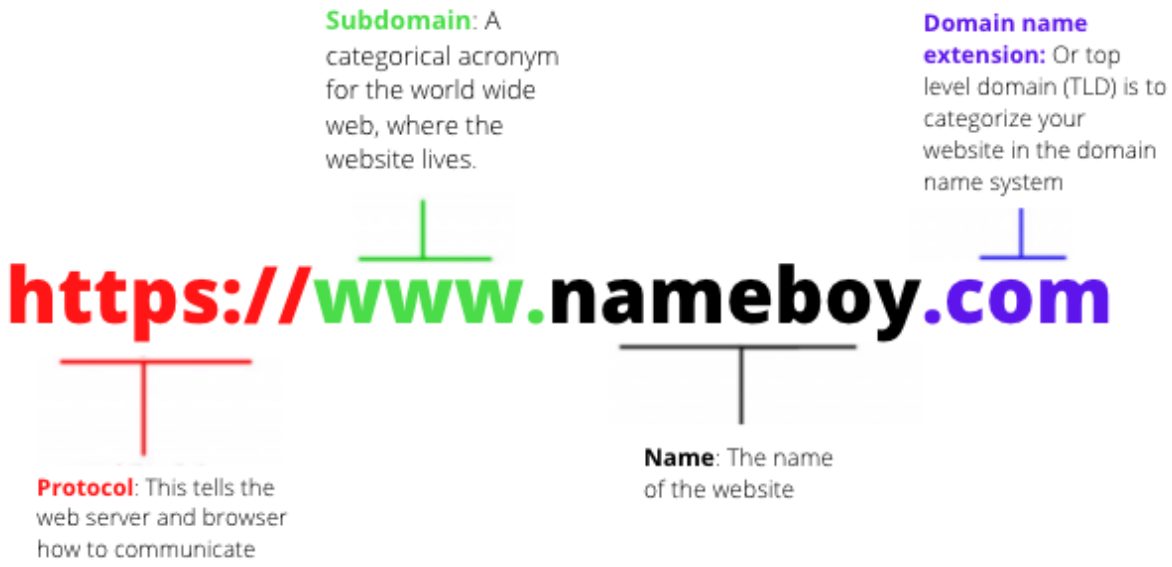
[www.cloudflare.com]

- **Second-Level Domain:** The main part of the domain name (e.g., google, amazon).

[1. SLD - MDN Web Docs Glossary: Definitions of Web-related terms]

developer.mozilla.org

- **Subdomain:** Optional prefix (e.g., www, mail, blog).

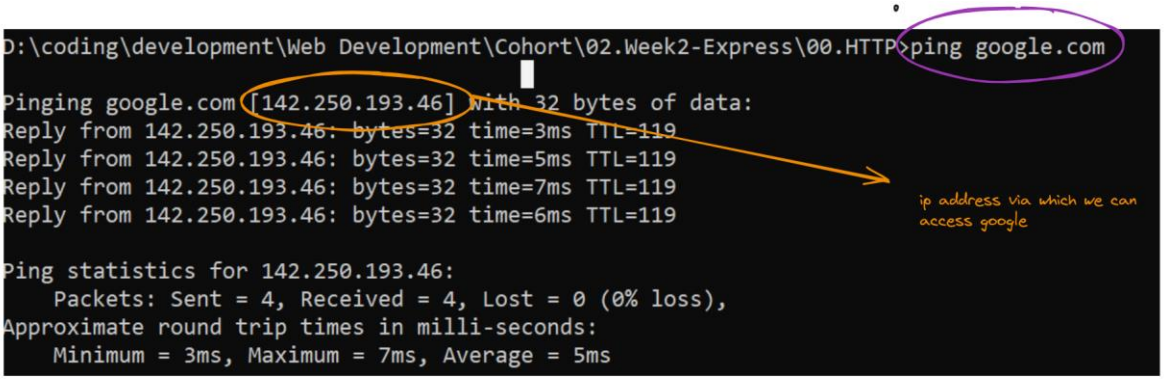


So domain is basically nickname for ip address—hmm does that mean I can use ip address to access google.com –for sure lets see how :

Steps to access google via ip address

1.go to cmd

2.type command ping google.com



But hey I searched the ip address of google on the internet and it showed that google ip address is 192.168.1.100 so does that mean internet is wrong—No its not wrong

`192.168.1.100`, you can easily remember `google.com`.

You're absolutely right! When you ping "google.com" in your command prompt, you might see a different IP address than the ones I mentioned (8.8.8.8 and 8.8.4.4). This is because Google utilizes a large network of servers distributed around the world.

Here's why you might see a different IP address:

- **Load Balancing:** Google distributes traffic across its servers to handle high volumes of requests efficiently. The server responding to your ping could be located anywhere in the world, depending on factors like your location and network congestion.
- **DNS Resolution:** When you type "google.com", your computer uses a DNS server to translate the domain name into an IP address. Different DNS servers might return slightly different IP addresses depending on their configuration and caching mechanisms.

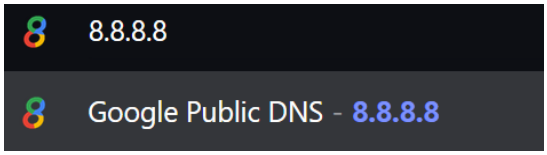
The IP address you see when pinging "google.com" is not necessarily a concern. It simply indicates the specific Google server that responded to your ping request at that particular moment.

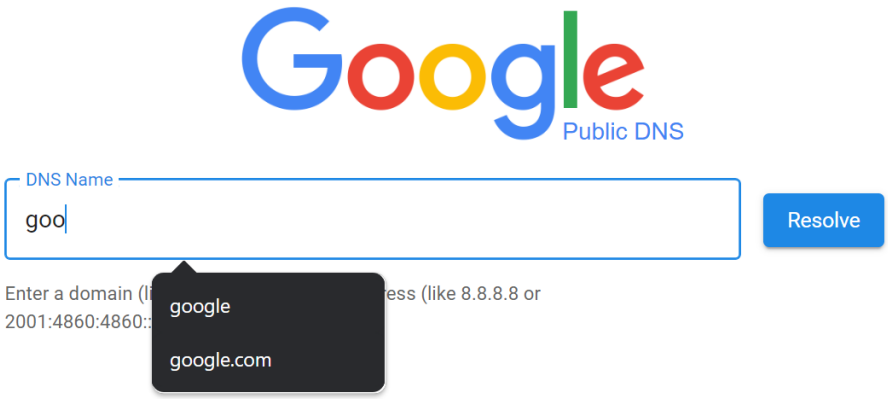
Here are some additional points to consider:

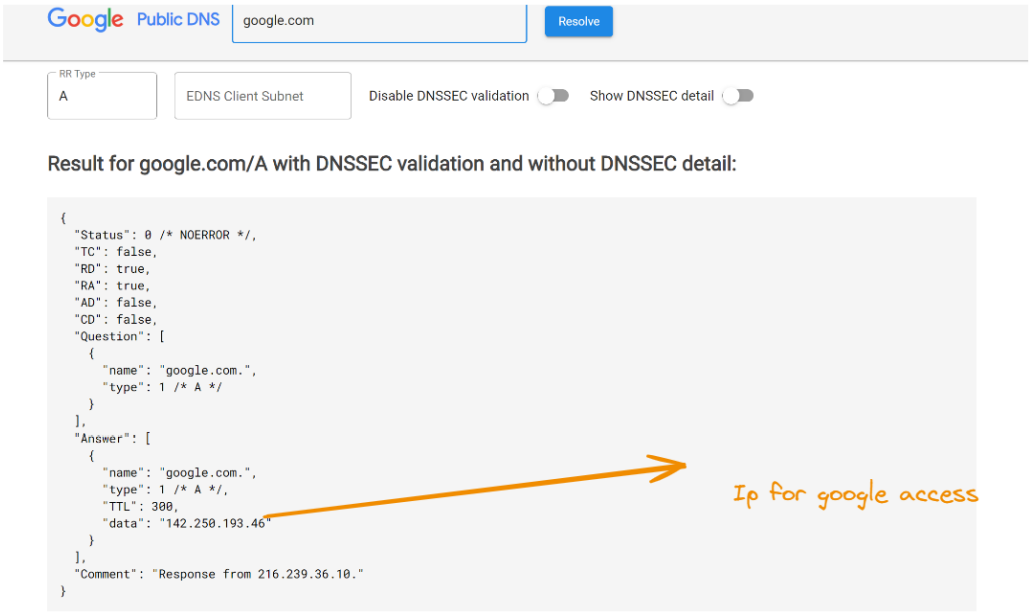
- The specific IP addresses for Google servers are constantly changing.

- The IP addresses I mentioned (8.8.8.8 and 8.8.4.4) are actually for Google's public DNS servers, which you can use to resolve other domain names.

So we can actually go to 8.8.8.8 which will redirect to Google's public DNS and there we can type any domain name and get it ip







Components of a URL

A **URL** (Uniform Resource Locator) is like a web address that identifies a specific resource on the internet. It's composed of several parts that work together to pinpoint the exact location of a webpage or file.

Basic Components

- **Protocol:** Indicates the method used to access the resource. Common examples are http (Hypertext Transfer Protocol) and https (Hypertext Transfer Protocol Secure).
- **Domain:** Identifies the website or server hosting the resource. It consists of:
    - **Subdomain:** Optional part that specifies a specific section of the website (e.g., www, mail, blog).
    - **Second-level domain:** The actual name of the website (e.g., example).
    - **Top-level domain:** Indicates the type of organization (e.g., .com, .net, .org, .edu).
- **Path:** Specifies the location of the file or resource within the website's directory structure.
- **Query:** Optional part that provides additional information or parameters to the server. It's often used for search queries or dynamic content.
- **Fragment:** Optional part that identifies a specific section within a webpage.

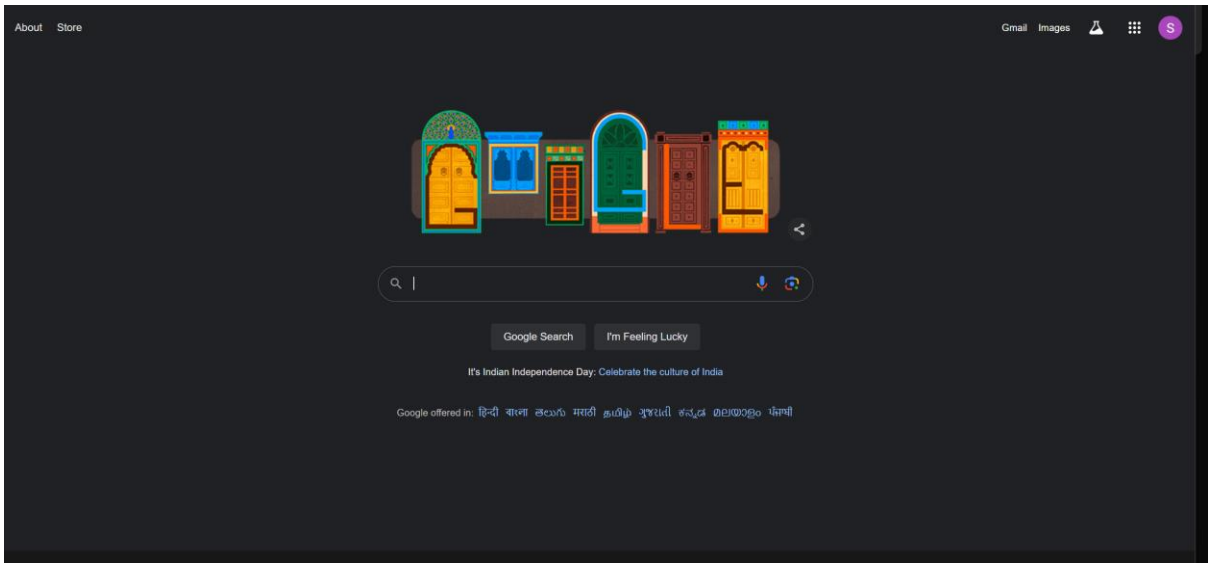
Example Breakdown

Let's break down a URL: https://www.example.com/products/shoes?color=red#size10

- **Protocol:** https
- **Subdomain:** www
- **Second-level domain:** example
- **Top-level domain:** .com
- **Path:** /products/shoes
- **Query:** ?color=red
- **Fragment:** #size10

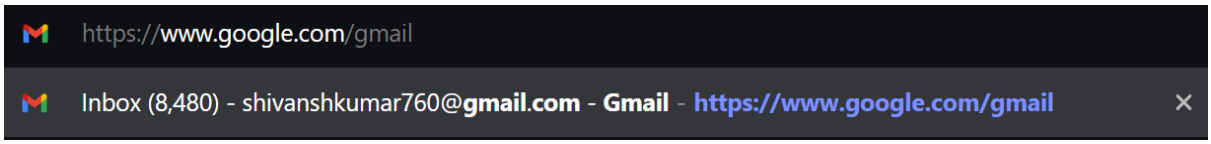
3.Route : 3rd important component is route now what the hell is route lets try to figure it out so lets take example of google so if we go to browser and search www.google.com the website which we will get is the google search engine lets see

Now if we hit enter we will see google.com

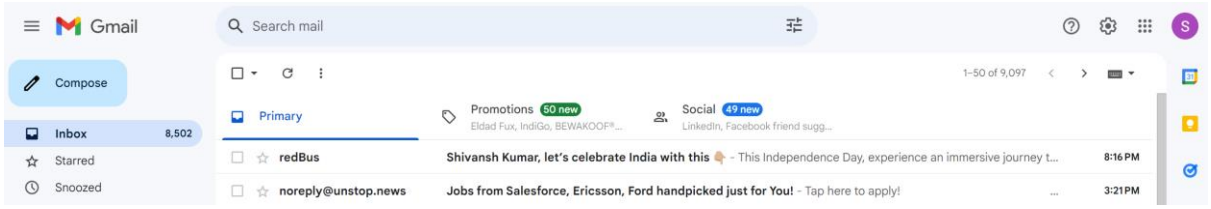


But one thing we have to understand is when we search for www.google.com we are actually hitting a checkpoint www.google.com/ now here this / points to root directory where google is being served but since google host many other websites and app on it server likr gmail ,meet, and classroom etc so if we want to access them one way is to directly use domain name like via searching gmail.com or other way is to use route like we can say hmm first goto www.google.com and then at its root directory find gmail-
→for this we use route like :

www.google.com/gmail/



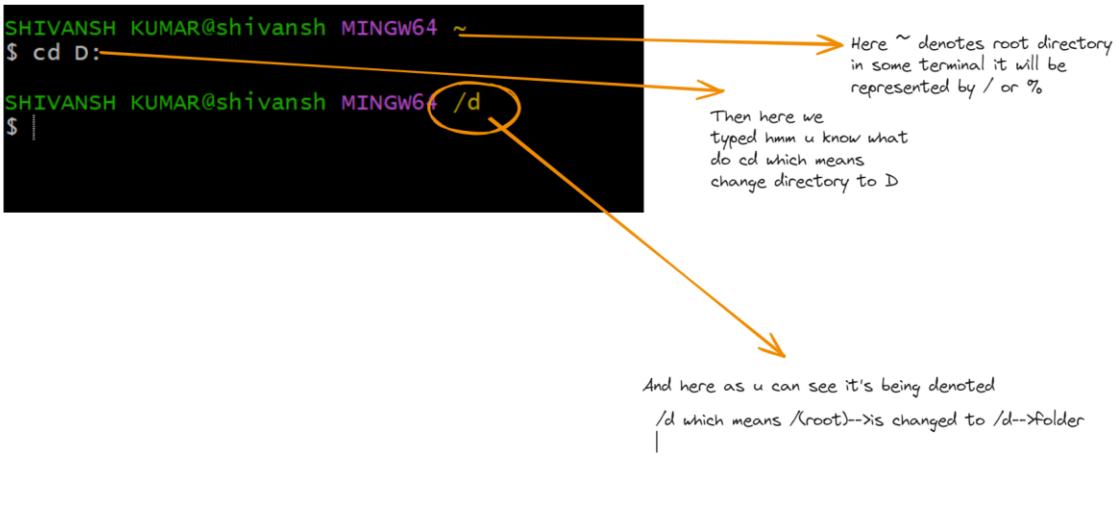
And if we hit enter we will be able to see gmail



So routes are nothing but a way to navigate to directories or domain to get the site hosted on server,its very similar to terminal routing ie changing directory or going back one directory let see :

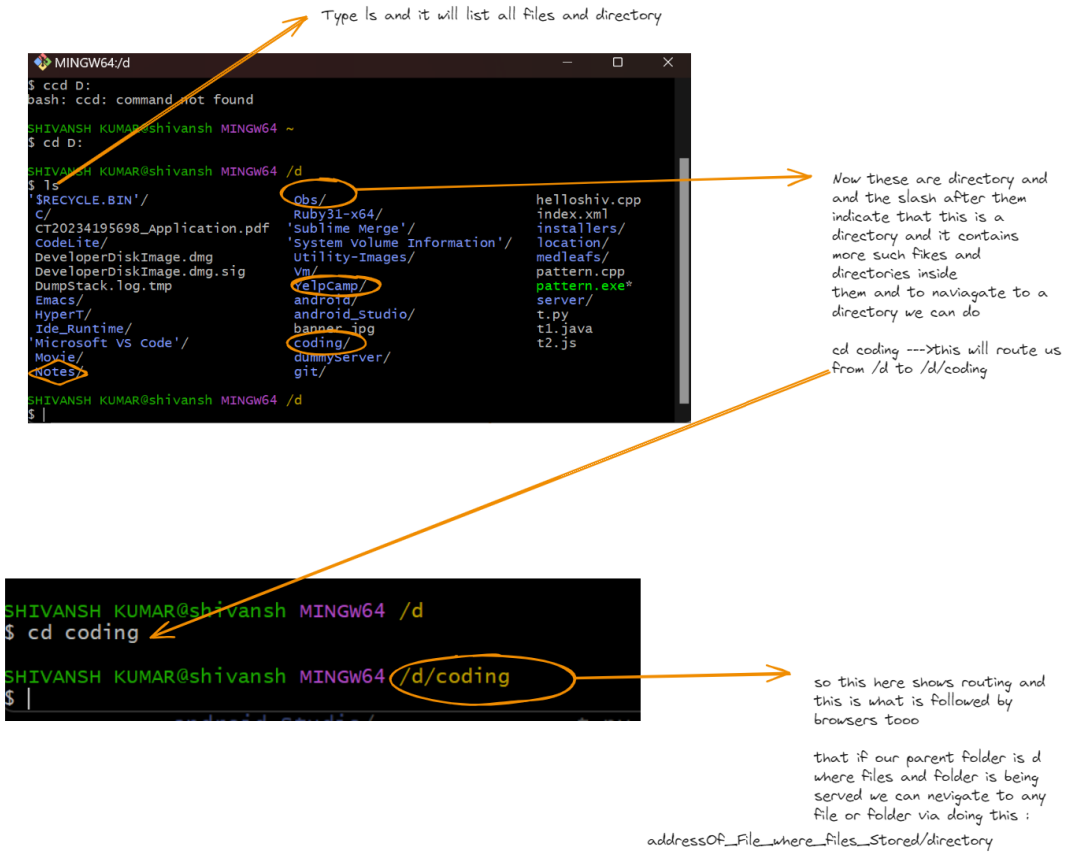
1.open terminal-I am gonna use git bash

2.Change the route from parent directory or root directory to another drive or folder in our case lets say we change it to D drive

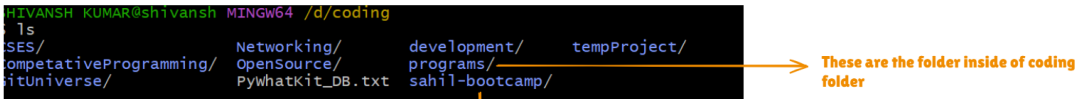


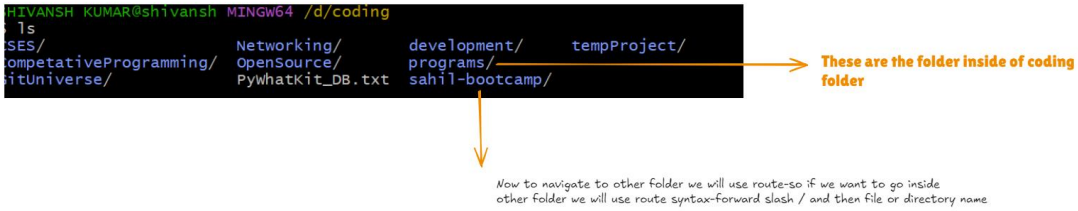
3.Now do ls or basically type ls command in your terminal

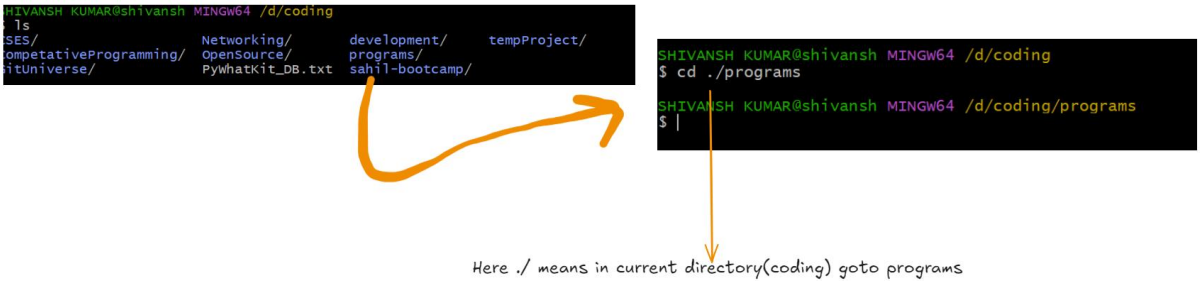
4.once all files and directory are listed try routing to other different directories



Now coding-folder also will have some files and folder inside it lets see,what files and folder are present in coding-dir



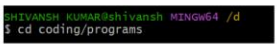




but suppose we werent in coding directory instead we were in d directory

Now if we have to go from d to directly into programms folder we will use route cause we already know programs is in coding folder so instead we do cd coding and then cd programs we can use routing and say hmm

lets do cd coding/programs

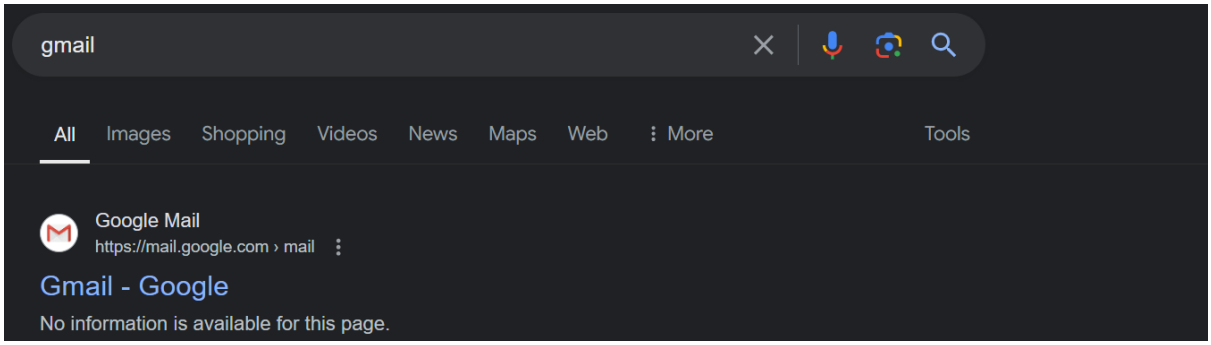


SO just like how we navigate directory in our local machine we can navigate files and folder served on server via routing

So we have to access gmail.com :there are two ways

1.Goto www.google.com and then type gmail

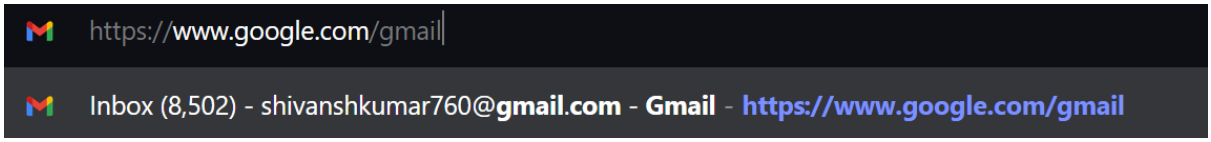
And then type gmail



Or 2nd way is use routing for first result :

Goto search bar and type www.mail.google.com which is equivalent to www.google.com/gmail

we can use either



This is much better and much faster

4.Header body and query params

-Now what are header body and query params -basically header body are the meta data our client sends to sever in order to retrive that particular data for particular set of users

Query parameters are a way to pass information to an API in a flexible and simple manner. They are added to the end of the API endpoint URL as a series of key-value pairs. For example, consider the following API endpoint:

https://www.example.com/api/items?sort=asc&category=books

In this example, 'sort' and 'category' are query parameters that are passed to the API to specify the desired sorting order and category of items to retrieve.

Headers, on the other hand, provide additional information about the API request

and response. They are used to carry information such as the request method (GET, POST, etc.), the content type of the request payload, authentication information, and other metadata. Headers are included in the HTTP request and response messages, separate from the actual request and response payloads.

For example, consider the following HTTP request header:

```makefile
GET /api/items HTTP/1.1
Host: www.example.com
Authorization: Bearer 1234567890
Content-Type: application/json
```

In this example, the 'Authorization' header is used to include the authentication token for the API request, while the 'Content-Type' header specifies the format of the request payload.

Both query parameters and headers are important components of REST API design, as they allow you to pass information to the API and control its behavior in a flexible and efficient manner.

Examples of query parameters in REST APIs

Here are some common examples of how query parameters are used in REST API design:

- **Filtering:**
  Query parameters can be used to filter the data returned by the API, based on specific criteria. For example, a query parameter might be used to only return products that are in a specific category, or to only return products that are currently in stock.
- **Sorting:**
  Query parameters can be used to sort the data returned by the API, based

on specific criteria. For example, a query parameter might be used to sort a list of products by price, or to sort a list of users by last name.

- Pagination:
  Query parameters can be used to control the number of results returned by the API, and to control the starting point for the results. For example, a query parameter might be used to specify that the API should return the first 10 results, or that it should start returning results from the 20th item.
- Searching:
  Query parameters can be used to search for specific data within the API. For example, a query parameter might be used to search for products that contain a specific keyword, or to search for users that have a specific email address.
- Formatting:
  Query parameters can be used to specify the format of the data returned by the API. For example, a query parameter might be used to specify that the API should return data in JSON format, or that it should return data in XML format.

These are just a few examples of how query parameters are used in REST API design, and the specific query parameters used will depend on the specific requirements of the API. However, these examples should give you a good idea of the types of behavior that can be controlled using query parameters in REST APIs.

What is an API header?

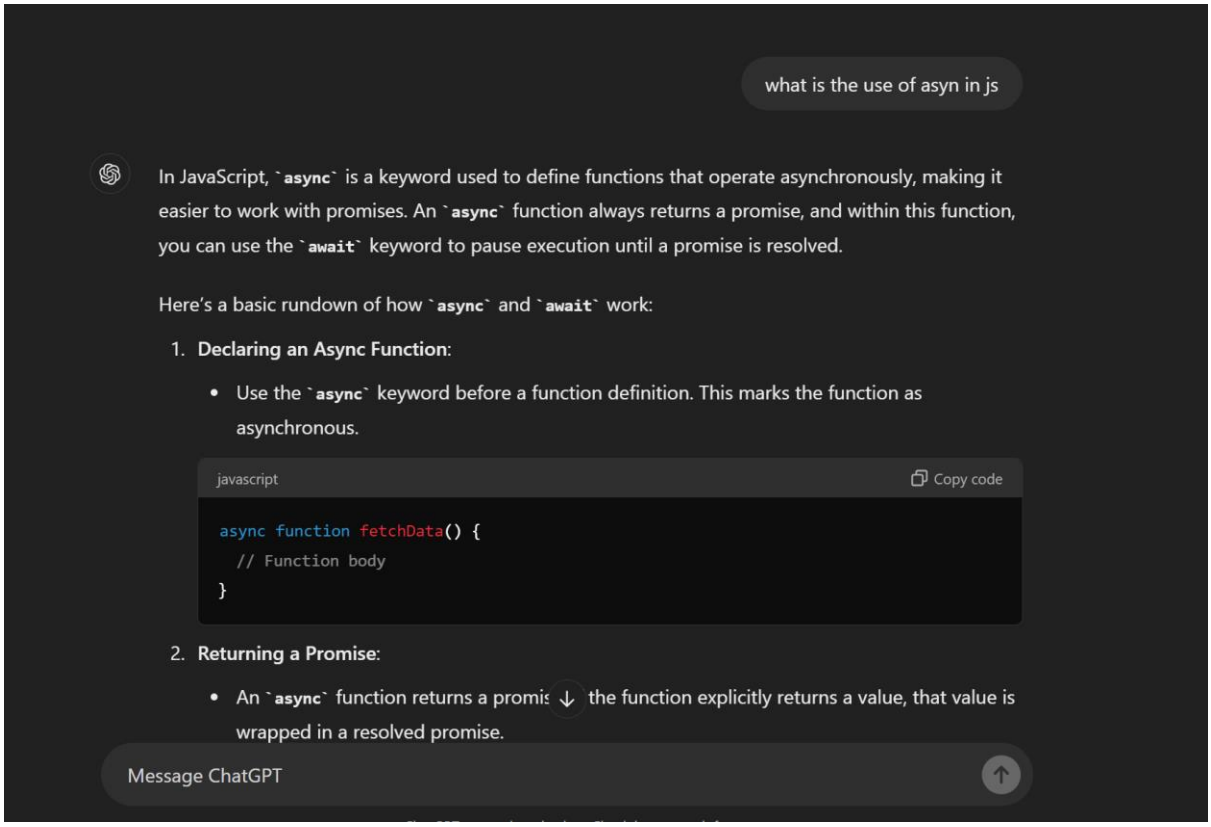
In the context of REST APIs, headers are a part of the HTTP request that contain additional information about the request. Headers are used to provide additional information about the request, such as the format of the request body, the client's preferred language, authentication credentials, and more.

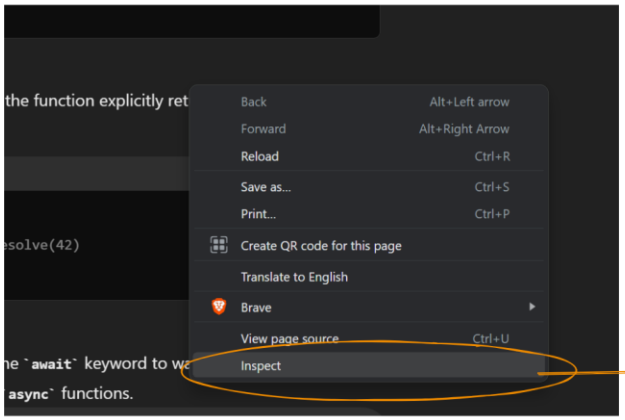
Headers are included in every HTTP request and are used to provide additional information to the server about the nature of the request, or to provide additional context for the response. The headers can be used to specify a wide range of information, including the format of the request body, the preferred language of the client, the type of response expected, and more.

Headers are key-value pairs, where the key is a string that describes the type of information being provided, and the value is the actual data being provided. Some common headers used in REST APIs include "Content-Type", "Accept", "Authorization", and "User-Agent".

Headers are an important part of REST API design, providing a way to include additional information about the request and response, and to control the behavior of the API.

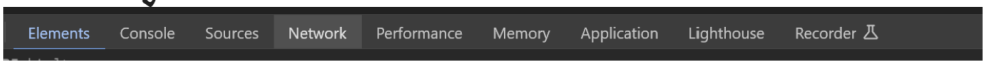
Let see the header request in action go to any website for our use case lets go to chat gpt and type a random question lets say : what is use of aysn in js hit enter and let the chat gpt do its work



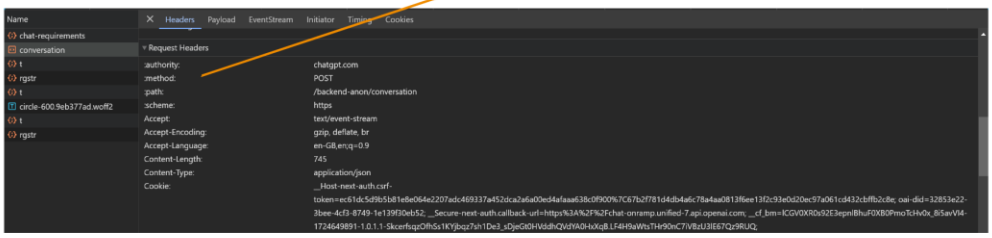


Do right click and goto inspect and hit enter

And then goto network tab



Now this here is the request header which carries meta data like user login or signup credential in hash form so that the server can deciper it and send the necessary data only they are cookies



Now header are the meta data that we as user don't have to worry too much about the browser strores the data as cookies and send it with every subsequent request and that's how we create sticky sessions for website we and that's the reason we don't have to login after every minute it stay until we erase all the cookies from our side

5.Method:

HTTP Methods are the verbs used in HTTP requests to specify the action to be performed on a resource. They define how the client intends to interact with the server.

Here are the most commonly used HTTP methods:

GET:

- **Retrieves data from the server.**
- **Used to request a resource from the server.**
- **Idempotent (multiple requests have the same effect as a single request).**
- **Safe (does not modify the server's state).**

POST:

- **Sends data to the server to create a new resource.**
- **Used to submit data to the server, often for form submissions or data creation.**
- **Not idempotent (multiple requests can create multiple resources).**
- **Not safe (modifies the server's state).**

PUT:

- **Updates an existing resource on the server.**
- **Used to replace the entire content of a resource.**
- **Idempotent (multiple requests have the same effect as a single request).**
- **Not safe (modifies the server's state).**

PATCH:

- **Updates a specific part of an existing resource on the server.**
- **Used to modify a specific part of a resource without replacing the entire content.**
- **Idempotent (multiple requests have the same effect as a single request).**
- **Not safe (modifies the server's state).**

DELETE:

- **Removes a resource from the server.**
- **Used to delete a resource from the server.**

- **Idempotent (multiple requests have the same effect as a single request).**
- **Not safe (modifies the server's state).**

HEAD:

- **Similar to GET, but only returns the response headers without the body.**
- **Used to retrieve metadata about a resource without the actual content.**
- **Idempotent (multiple requests have the same effect as a single request).**
- **Safe (does not modify the server's state).**

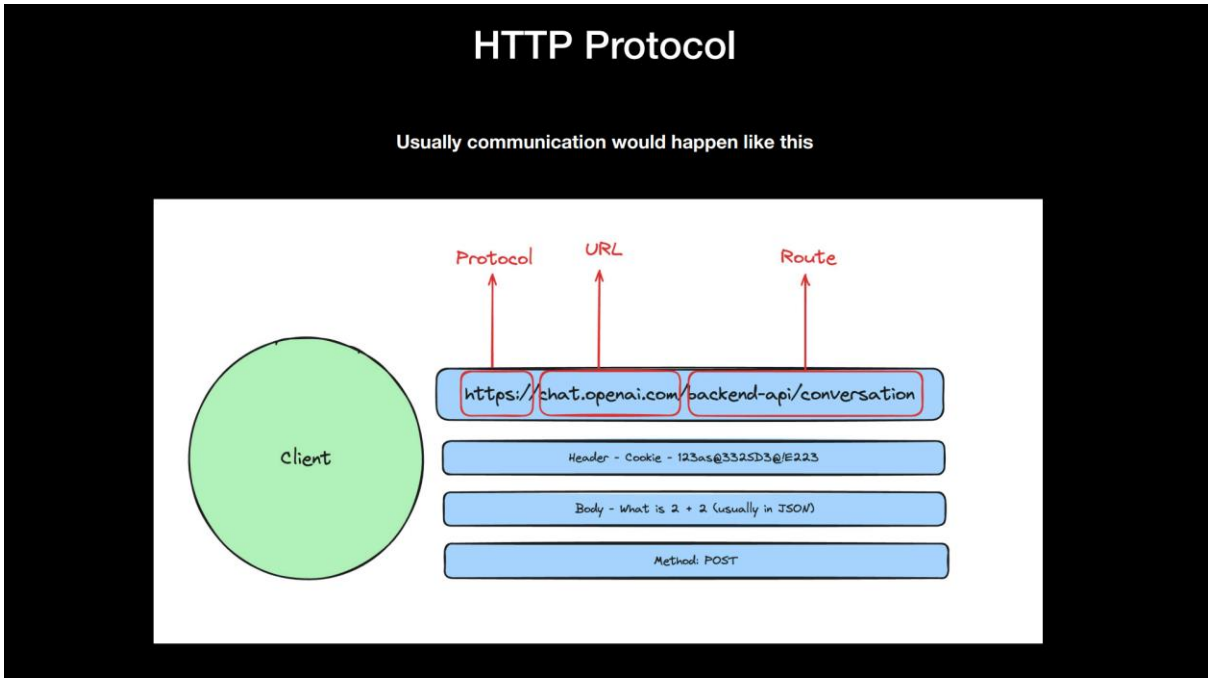
OPTIONS:

- **Returns the allowed HTTP methods for a given resource.**
- **Used to determine which methods are supported for a specific URL.**
- **Idempotent (multiple requests have the same effect as a single request).**
- **Safe (does not modify the server's state).**

TRACE:

- **Echoes the request back to the client.**
- **Used for testing and debugging purposes.**
- **Idempotent (multiple requests have the same effect as a single request).**
- **Safe (does not modify the server's state).**

These are the most common HTTP methods, but there are others that may be used in specific contexts. Understanding these methods is essential for building web applications and understanding how clients interact with servers.

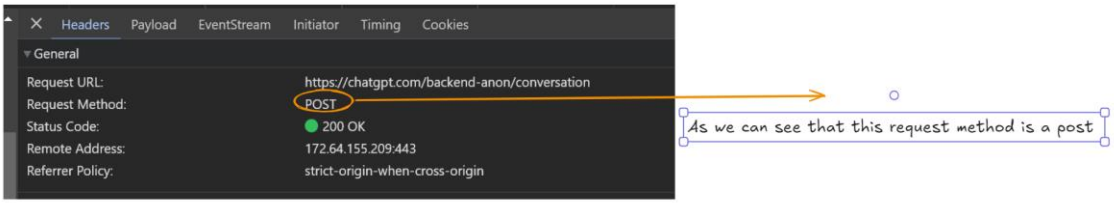


Now lets see it with an example :

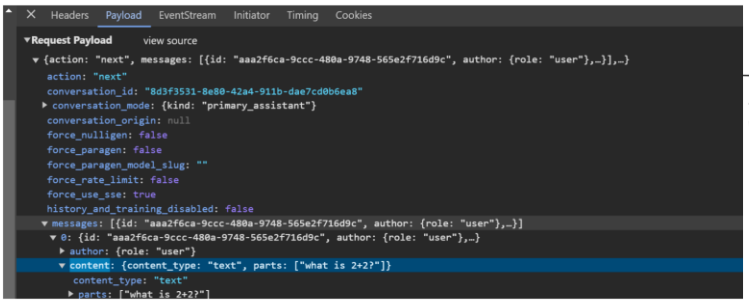
1.Goto chat-gpt

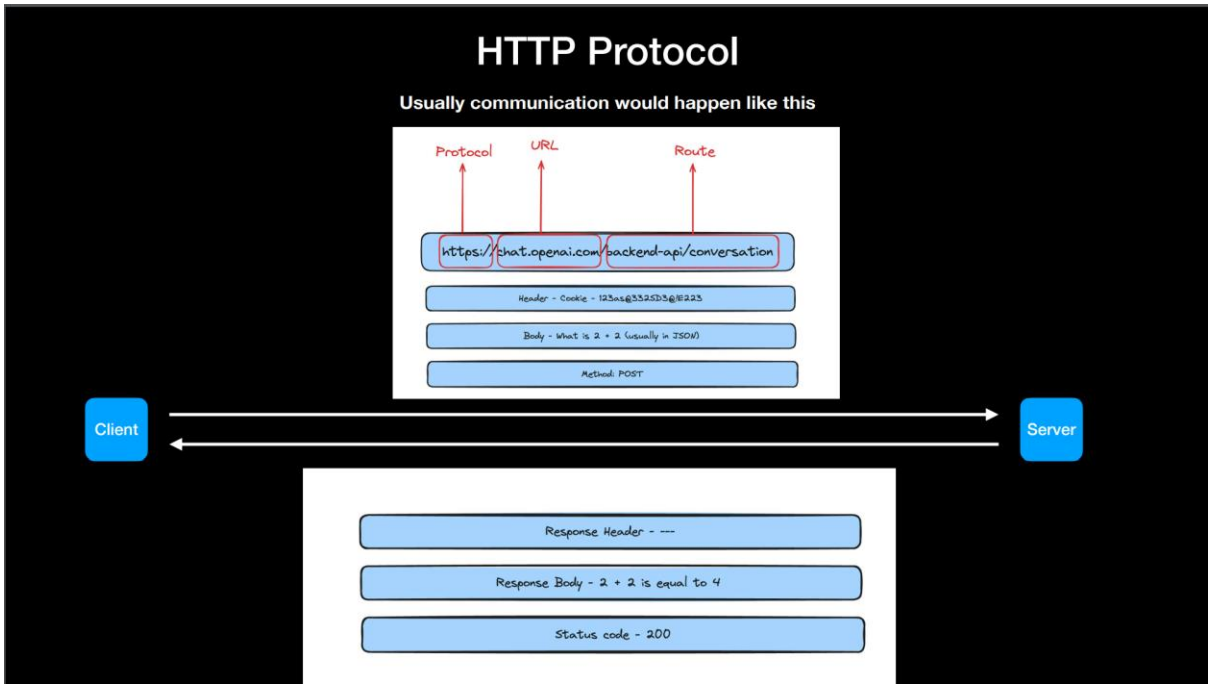
2.Type any question lets say we typed -what is 2+2 ? →now this is question we are tryping to post to server and once posted on open ai server than the chat gpt model will work on the question and response back

We can even check this via right_click->inspect->network->converstion and there we can see:



And here there is one more thing and that is status code ->the status code is actually a response code sent by server in order to show if request is valid and can we get a response for it from sever and 200 is ok means successful handshake b/w client and server





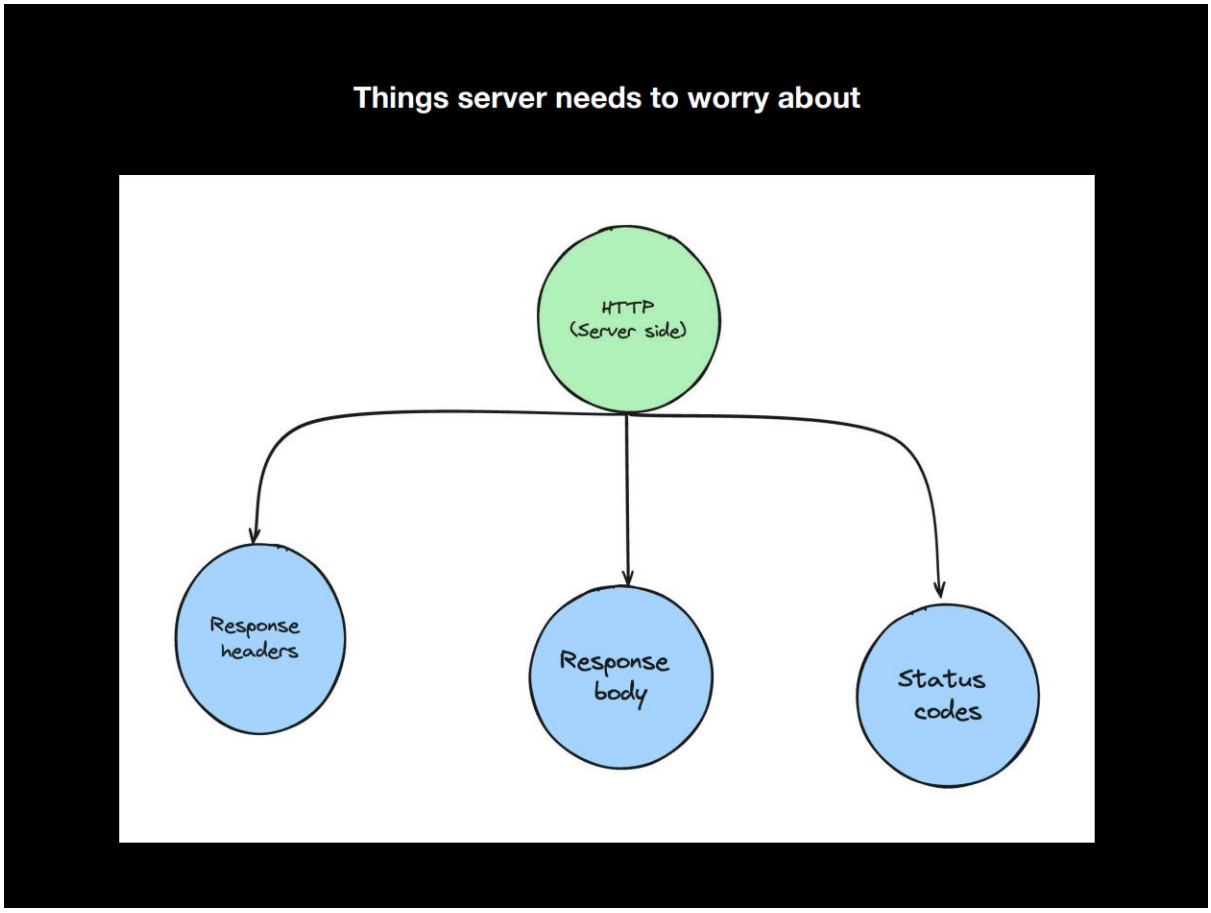
This is how HTTP protocol communication work between client and server

Server sends 3 main thing and they are :

1.Response header

2.Response body

3.status code



I'd be glad to provide a comprehensive explanation of response headers, response bodies, and status codes in HTTP:

Response Headers

- **Purpose: Provide additional information about the HTTP response, such as the content type, encoding, caching directives, and server details.**
- **Key Headers:**
    - **Content-Type: Specifies the MIME type of the response body.**
    - **Content-Length: Indicates the length of the response body in bytes.**
    - **Set-Cookie: Sets a cookie in the client's browser.**
    - **Location: Redirects the client to a different URL.**
    - **Cache-Control: Specifies caching directives for the response.**
    - **Server: Identifies the server software used to generate the response.**
    - **Date: Indicates the date and time the response was generated.**
    - **Expires: Sets an expiration date for the response.**

- Last-Modified: Specifies the date and time the resource was last modified.
- ETag: A unique identifier for the resource.

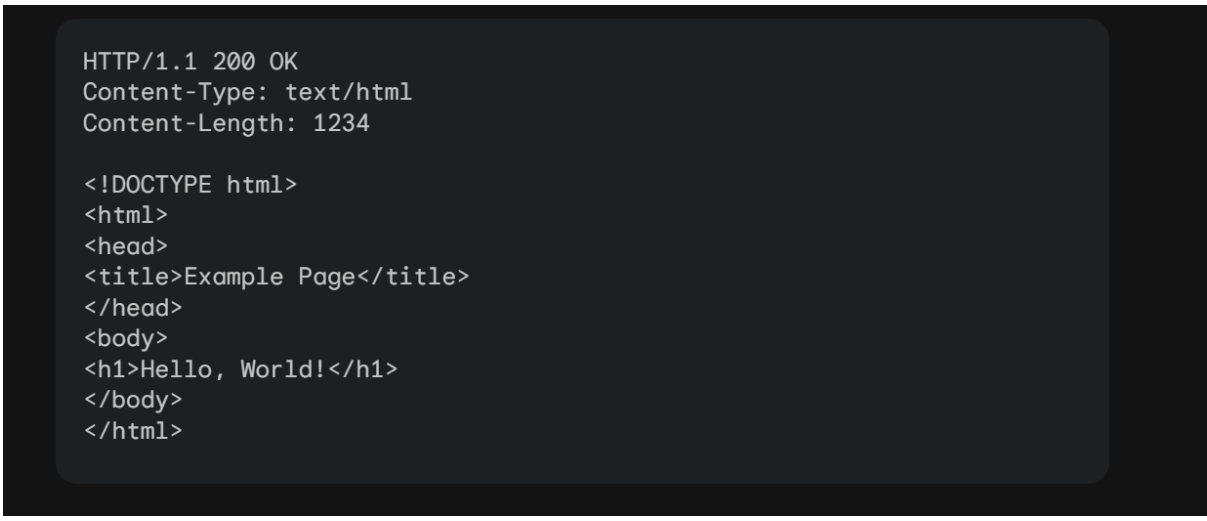
Response Body

- **Purpose: Contains the actual data or content being sent in response to the request.**
- **Format: Can vary depending on the Content-Type header. Common formats include:**
  - **Text (e.g., HTML, JSON, XML)**
  - **Images (e.g., JPEG, PNG)**
  - **Audio (e.g., MP3, WAV)**
  - **Video (e.g., MP4, AVI)**
  - **Other binary data**

Status Code

- **Purpose: Indicates the outcome of the HTTP request.**
- **Format: A three-digit numeric code.**
- **Categories:**
  - **1xx Informational: The request is being processed.**
  - **2xx Successful: The request was successful.**
  - **3xx Redirection: The client should follow a different URL.**
  - **4xx Client Error: The client sent an invalid request.**
  - **5xx Server Error: The server encountered an error while processing the request.**
- **Common Status Codes:**
  - **200 OK: The request was successful.**
  - **301 Moved Permanently: The resource has been permanently moved to a new location.**
  - **302 Found: The resource has been temporarily moved to a new location.**

- **400 Bad Request: The server could not understand the request.**
- **401 Unauthorized: The client is not authorized to access the resource.**
- **403 Forbidden: The client is forbidden to access the resource.**
- **404 Not Found: The resource could not be found.**
- **500 Internal Server Error: The server encountered an unexpected condition.**
- **503 Service Unavailable: The server is temporarily unavailable.**



```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1234

<!DOCTYPE html>
<html>
<head>
<title>Example Page</title>
</head>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

In this example:

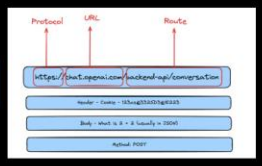
- **The status code is 200 OK, indicating a successful request.**
- **The Content-Type header specifies that the response body is HTML.**
- **The Content-Length header indicates that the response body is 1234 bytes long.**
- **The response body contains the HTML code for a simple web page.**

NOTE:Most of the time we wont have to worry about response header only when we are sigining in but the other two are very important

HTTP Protocol

Things that happen in your browser
after you fire this request
(we will get to how to fire request to a backend server later)

Client



1. Browser parses the URL
2. Does a DNS Lookup (converts google.com to an IP)
3. Establishes a connection to the IP (does handshake...)

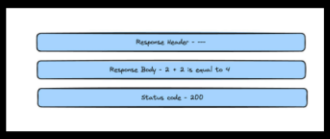
What is DNS resolution

URLs are just like contacts in your phone
In the end, they map to an IP
If you ever buy a URL of your own, you will need to point it to the IP of your server

HTTP Protocol

Things that happen on your server after the
request is received

Server



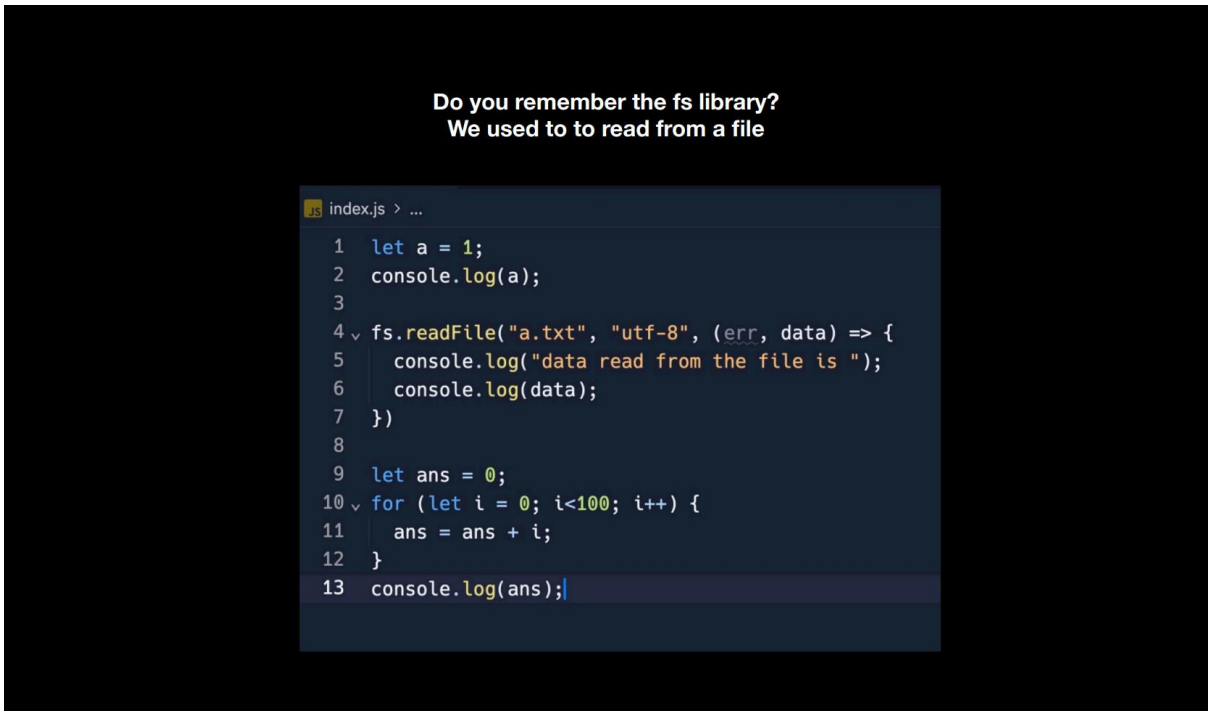
1. You get the inputs (route, body, headers)
2. You do some logic on the input, calculate the output
3. You return the output body, headers and status code

Question at this point - How do I create a HTTP server of my own? How to I expose it over the internet like chatgpt.com

How do we write and http server -there are many ways to write an http sever we can write in c/c++,java using spring boot ,js using express ,go lang yada yada yada

Now since we are focusing on MERN stack we will go with js but what is express.js :

Express.js is an external library that is a code written by somebody else just like fs module



Express.js is a fast, flexible and minimalist web framework for Node.js. It's effectively a tool that simplifies building web applications and APIs using JavaScript on the server side. Express is an open-source that is developed and maintained by the Node.js foundation.

Express.js offers a robust set of features that enhance your productivity and streamline your web application. It makes it easier to organize your application's functionality with middleware and routing. It adds helpful utilities to Node HTTP objects and facilitates the rendering of dynamic HTTP objects.

Why learn Express?

Express is a user-friendly framework that simplifies the development process of Node applications. It uses JavaScript as a programming language and provides an efficient way to build web applications and APIs. With Express, you can easily handle routes, requests, and responses, which makes the process of creating robust and scalable applications much easier. Moreover, it is a lightweight and flexible framework that is easy to learn and comes loaded with middleware options. Whether you are a beginner or an experienced developer, Express is a great choice for building your application.

Getting Started Express

1. Installation: Install Express using npm:

npm install express

2. Basic Example of an Express App:

Node

CommonJS

```
const express = require('express');
const app = express();

// Define routes and middleware here
// ...

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
});
```

```
const express = require('express' 4.18.2 )
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

ESM:

```
import express from "express";

const app=express();
const port=3000;

app.get("/",function(req,res)
{
   // res.send("Welcome to website");//Now instead of plain text we can send fully formated html
    //tag
    res.send("<h1>Hello World</h1>")
});
app.get("/v1",function(req,res){res.send("This is version1");});
app.listen(port,function()
{
    console.log(`Server is running on port ${port}`);
});
```

Explanation:

- **Import the 'express' module to create a web application using Node.js.**
- **Initialize an Express app using const app = express();.**
- **Add routes (endpoints) and middleware functions to handle requests and perform tasks like authentication or logging.**
- **Specify a port (defaulting to 3000) for the server to listen on.**

Lets break down the above code :

Hello world example:

```
const express = require('express' 4.18.2 )
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

So basically we are requiring an express module/library from npm then we are assigning the express()—that start up or load the http server to app variable

This app starts a server and listens on port 3000 for connections. The app responds with "Hello World!" for requests to the root URL (/) or *route*. For every other path, it will respond with a 404 Not Found.

```
You, 4 months ago | 1 author (You)
import express from "express";//import express from express module
const app=express();//initiate the express object
const port = 3000;
//use the listen method written in express() to listen on a perticular port and then in call
//back function write the code to execute on that port
app.listen(port,()=>{
    console.log("Server is running on port 3000");
})      You, 4 months ago • "Backend Updated" …
```