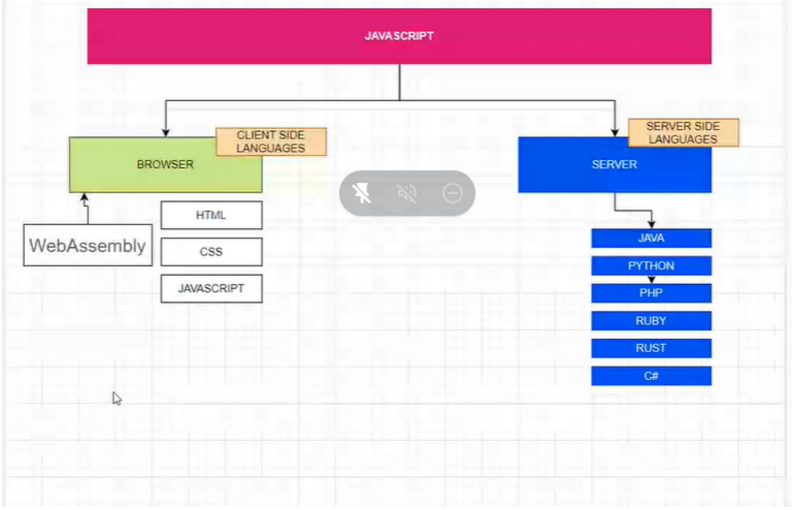
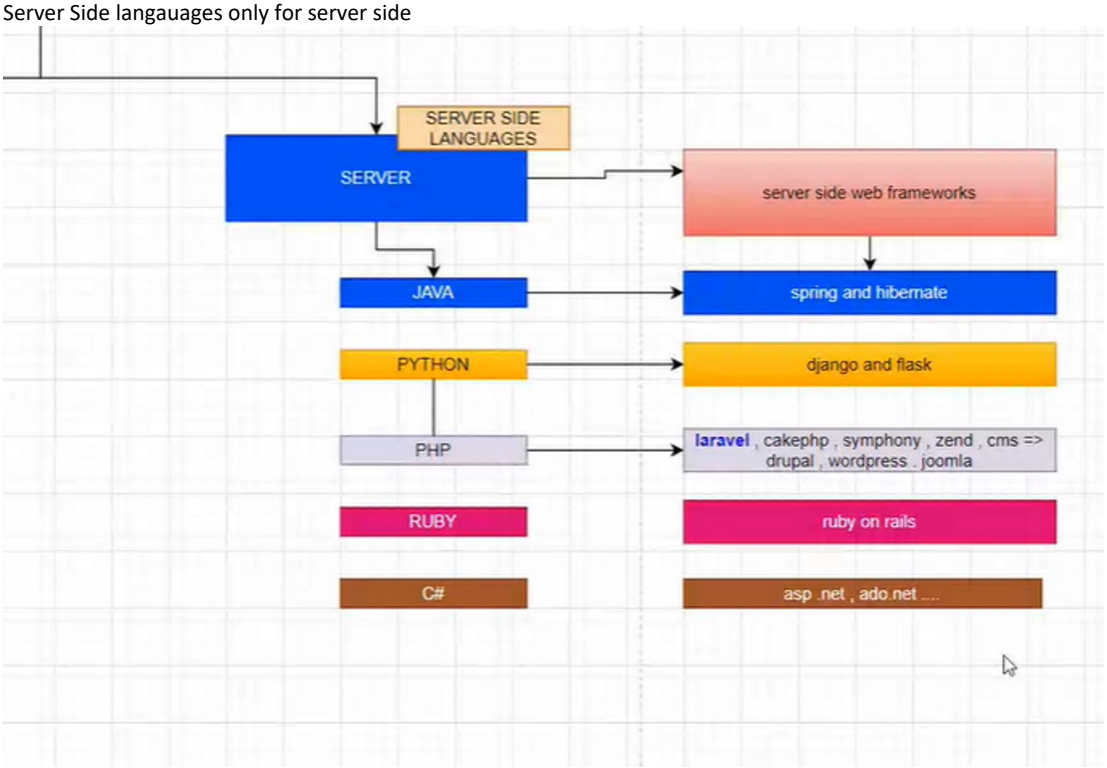
Introduction

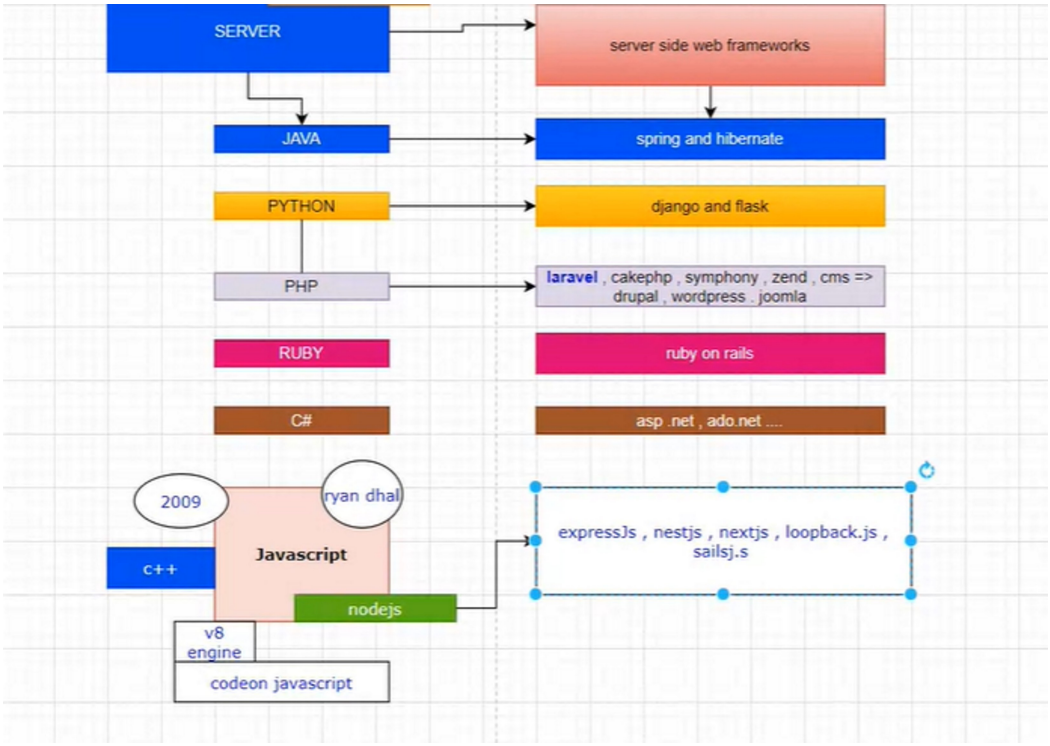
Wednesday, January 19, 2022 10:06 AM



HTML is mandatory
CSS is optional just for beautification.
For today's standard JS is mandatory, it is the most used programming langauage.
Web app=== JS

Server Side langauages only for server side



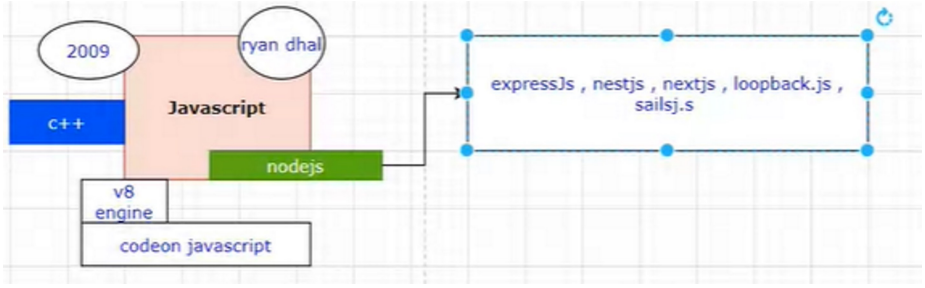


2009 onwards

Ryan dhal natively used c++ and created a run time environment wrapper
Nodejs is not a framework, it is not a programming language
It is a server side runtime environment.
Code on JS only, behind the scene it is C++.
JS is not a server side language but it is a browser side langauge.

Ryan dhal
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

From <https://nodejs.org/en/>



Dependency for node
- C++
- V8 engine

Frameworks of nodejs
- expressJs
- Nestjs
- Nextjs
- Loopbackjs
- sailsJs
- Socket.io

Node runtime enivronment frameworks
https://expressjs.com/
https://nestjs.com/
https://sailsjs.com/
https://loopback.io/
https://nextjs.org/

https://socket.io/

java web frameworks
https://spring.io/
https://hibernate.org/
https://spring.io/projects/spring-boot

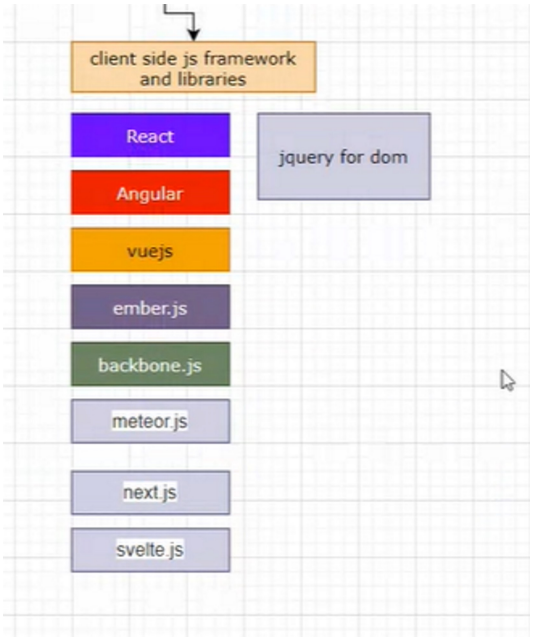
python web frameworks
https://www.djangoproject.com/
https://flask.palletsprojects.com/en/2.0.x/

ruby web frameworks
https://rubyonrails.org/

Php
https://laravel.com/
https://codeigniter.com/
https://cakephp.org/
https://symfony.com/
https://wordpress.com/
https://www.drupal.org/

C#
https://dotnet.microsoft.com/en-us/apps/aspnet

Client side JS frameworks and libraries



meteor and nextjs both support front end and back end.

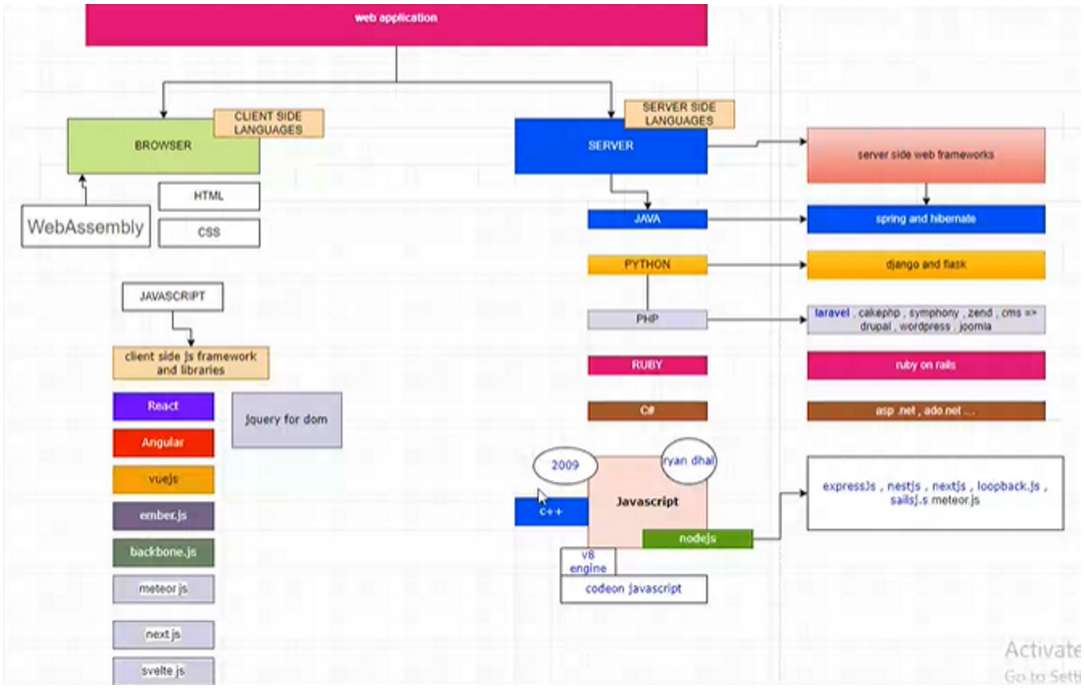
https://reactjs.org/
https://angular.io/
https://vuejs.org/
https://emberjs.com/
https://backbonejs.org/

https://www.meteor.com/
https://nextjs.org/
https://nuxtjs.org/

https://www.gatsbyjs.com/
https://svelte.dev/

For DOM manipulation
https://jquery.com/

For web applications



Mobile applications

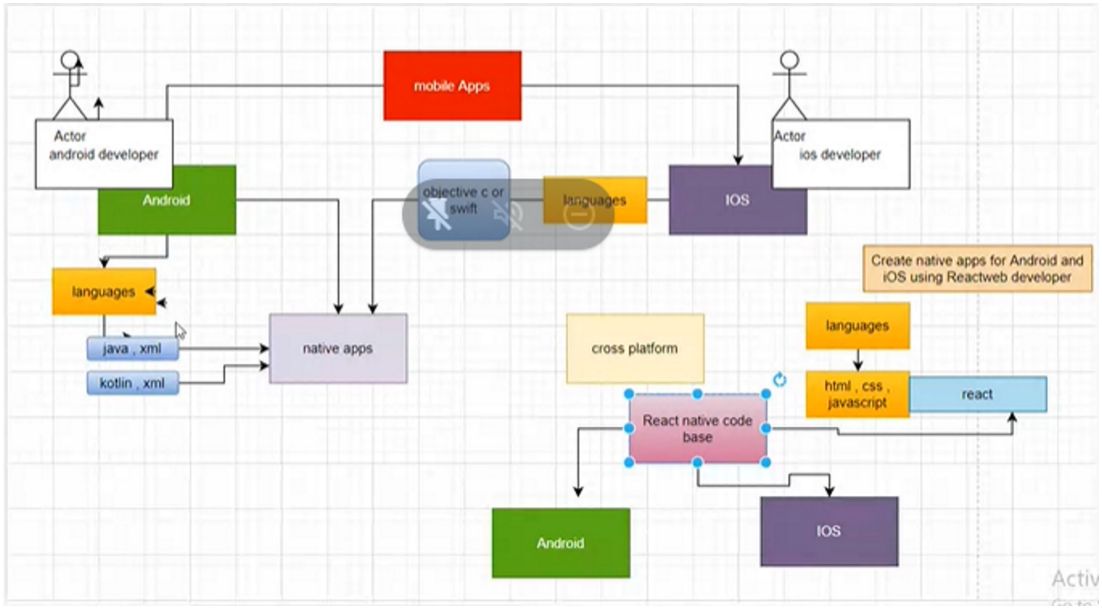
Android native applications

- Java,xml
- Kotlin, xml

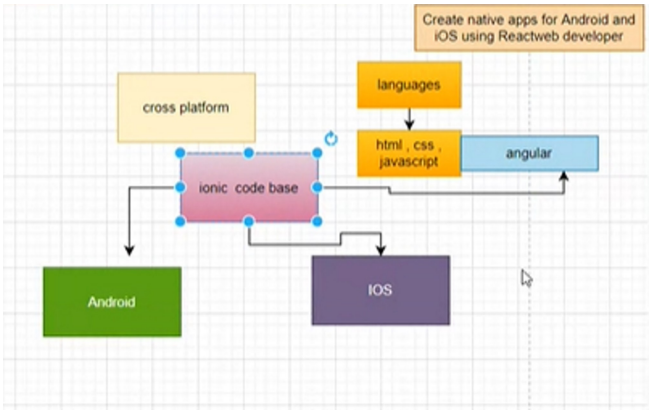
IOS native application

- Objective C
- Swift

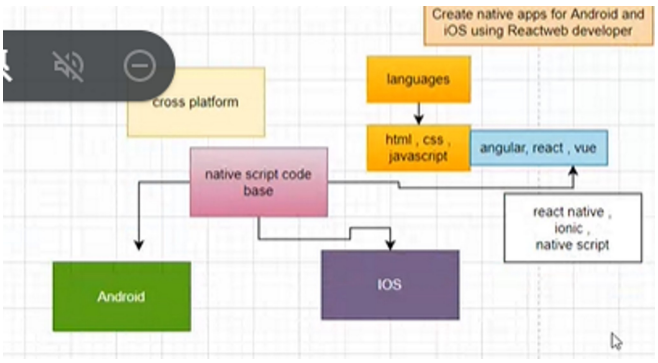
Codebase- you can use the same code for android and IOS at the same time



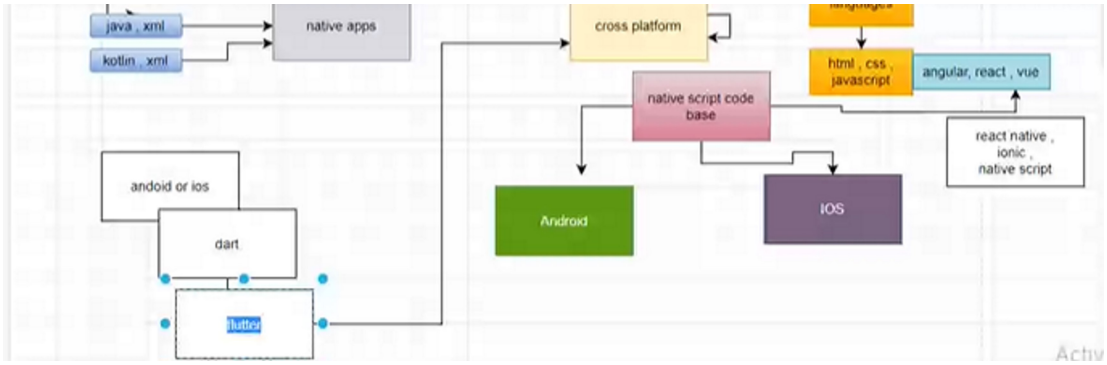
For angular devlopers



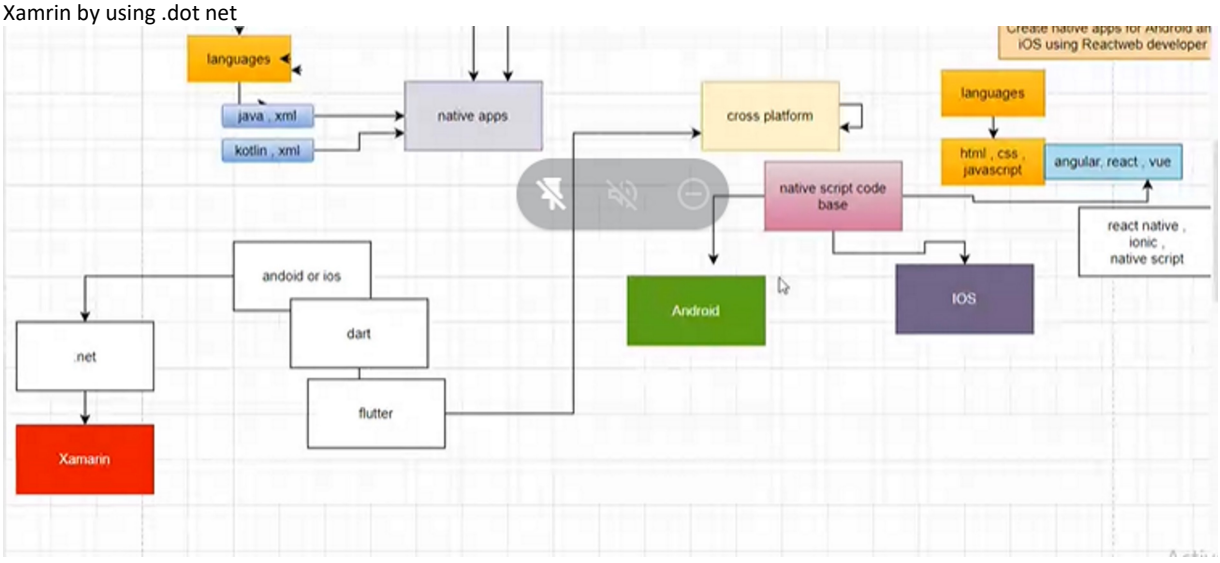
Nativescript framework is a code base allows you to use react native, angular ,etc



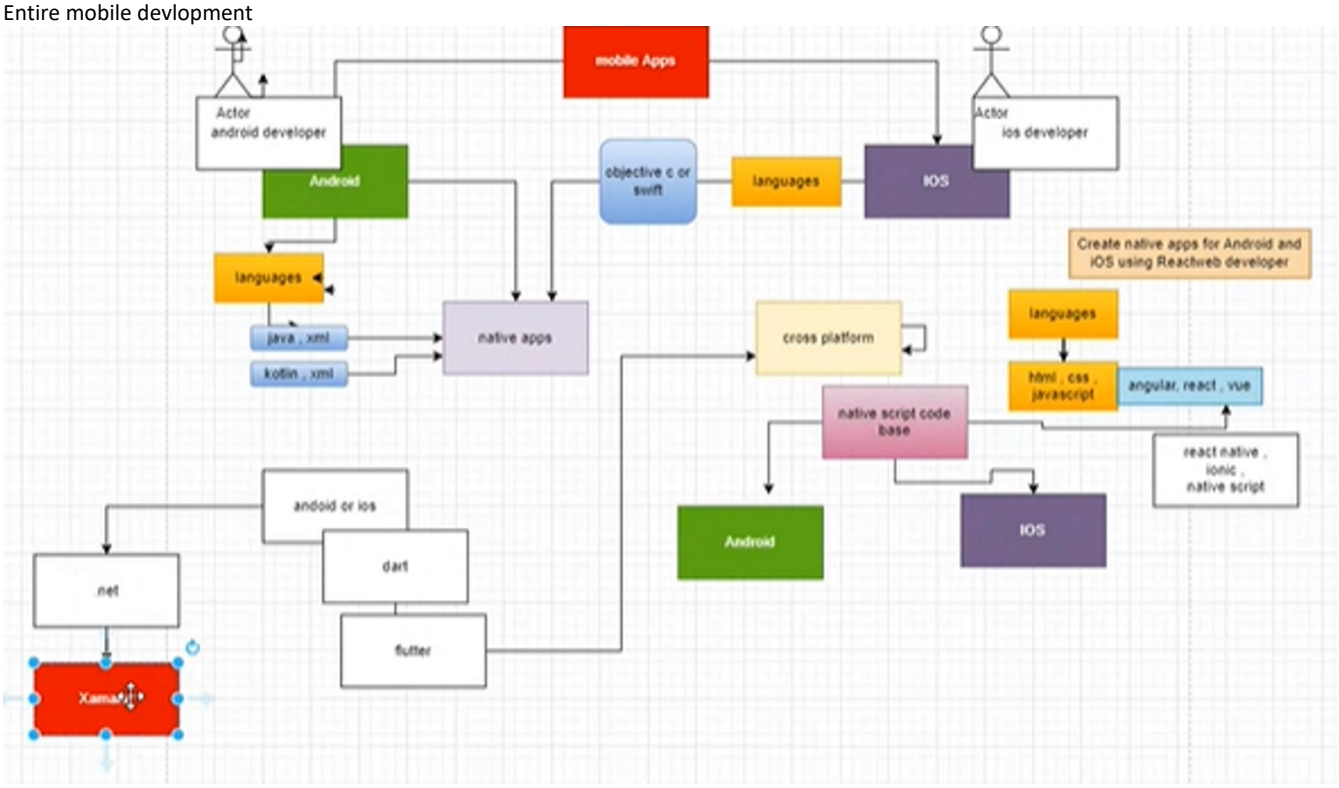
Dart programming language developed by Google
Flutter framework



Xamrin by using .dot net



Entire mobile devlopment



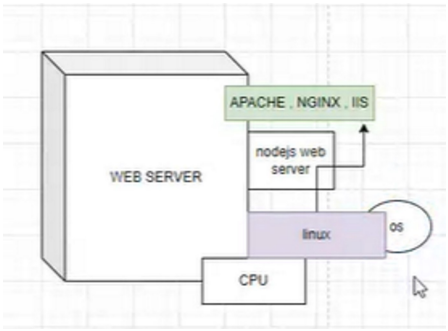
After short break

Web servers

1. Apache web server
2. IIS web server(microsoft product)
3. NGINX server(cross platform and proxy support)

Without above three we can create a local server with nodesjs server.

Webserver needs an OS
Most of them run on LINUX OS



Creating and example for node js server

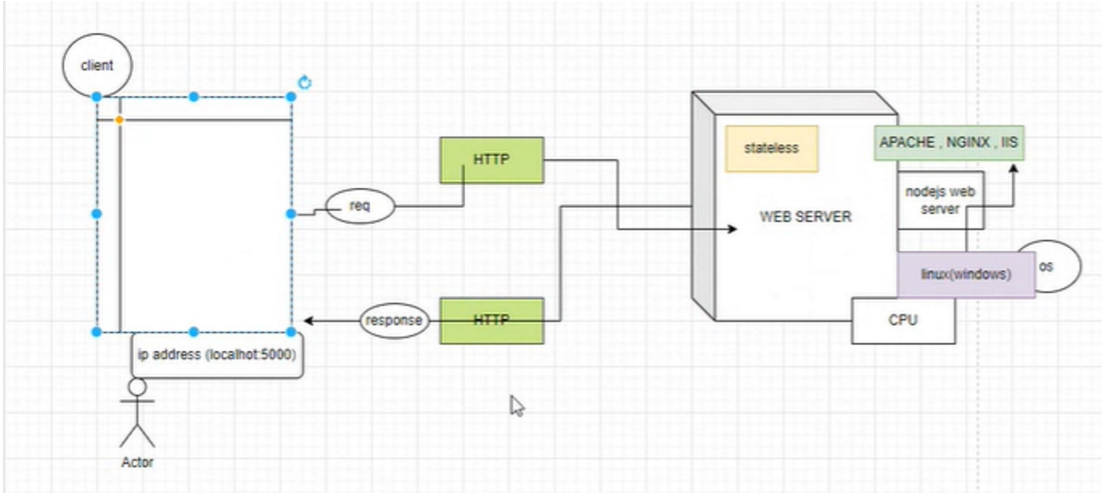
```
create web server
const http = require("http");

const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/html" });
  res.end("<h1>hello nodejs wen server</h1>");
});

server.listen(5000, err => {
  if (err) throw err;
  console.log("server is running on port number 5000");
});
```

Always server is stateless, means data is stored temprovary
When the state is activated, only when request is made

Server and client connect using a protocol.
We use http protocol for web applications.
Http keeps monitoring whether client and server are avilable or not

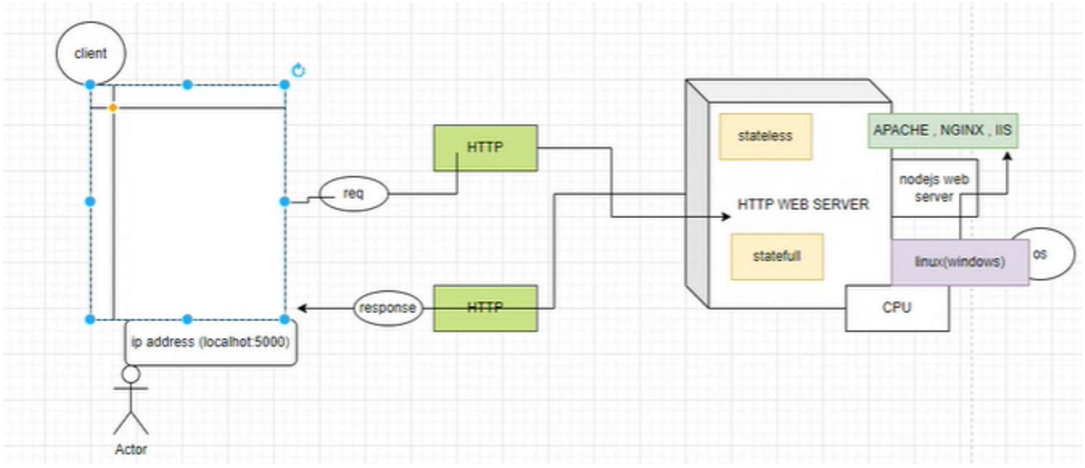
Http is a bridge, keeps checking whether client has given valid url or not.

Http Is the heart of web applications.
Client gives request object.



When client accepts request the server becomes stateful. (just that duration)
HTTP:hyper text transfer protocol

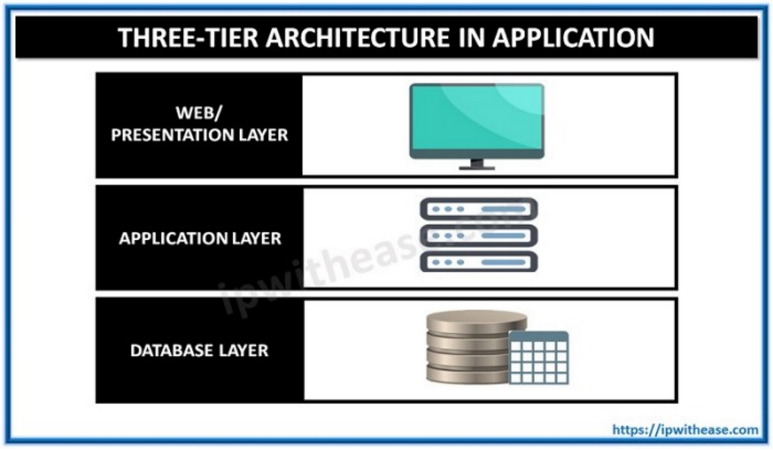
Job of web server
Accepting request
Proccessing request
And returns response



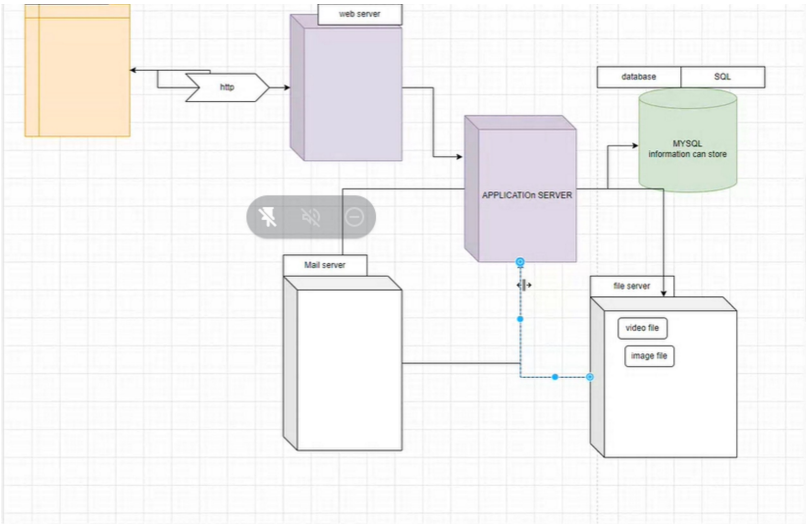

```
//Lan and wan
web app local lan  //no internet
web app global wan  //internet
```

Day 2==========

Three tier architecture
Onlt browser-> web server-. Database
Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is



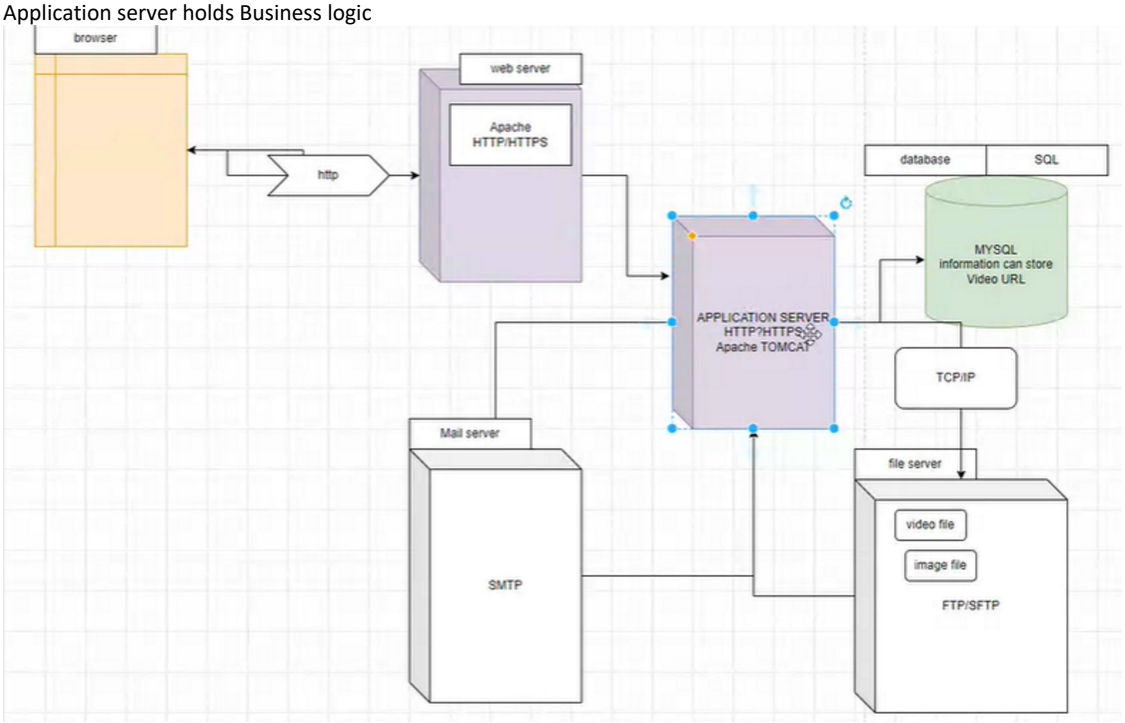
For complex data we use Distributed System----------
Web applications requires database.
Any SQL database can store only store text data or text content, it does not hold video, audio , etc.
FPP is used to store audio, video in the form of buckets which is linked to SQL DB.



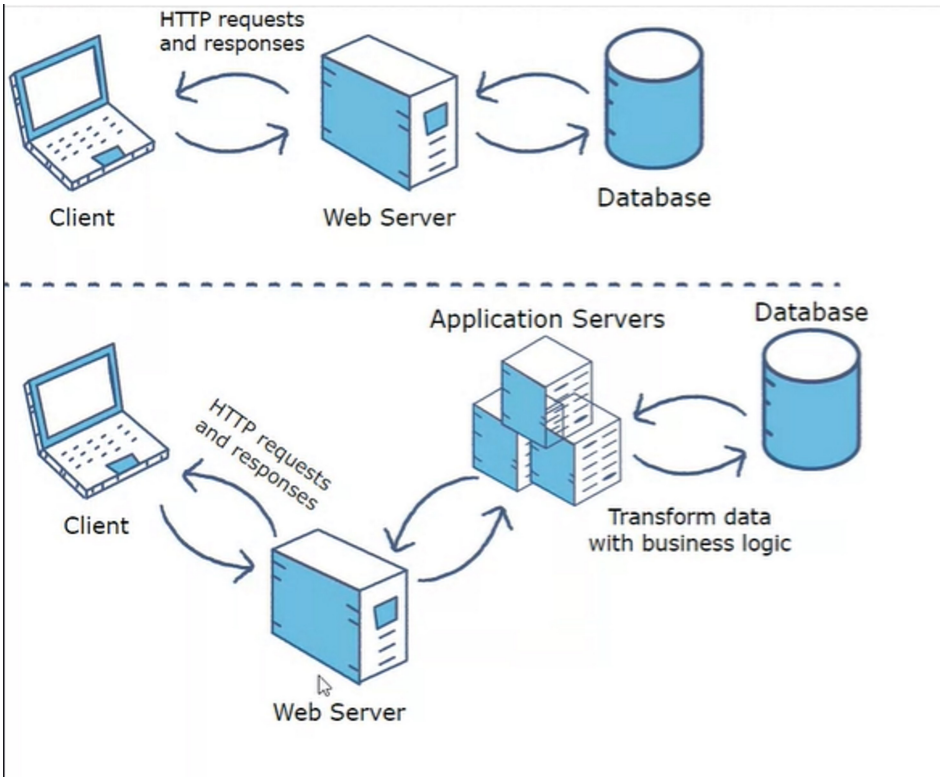
Distributed System
Complex application above figure

SMTP - simple mail transfer protocol
FTP file transfer protocol

Application server holds Business logic



Simple web server and distributed server.



Web Server

- Deliver static content.
- Content is delivered using the HTTP protocol only.
- Serves only web-based applications.
- No support for multi-threading.
- Facilitates web traffic that is not very resource intensive.

Application Server

- Delivers dynamic content.
- Provides business logic to application programs using several protocols (including HTTP).
- Can serve web and enterprise-based applications.
- Uses multi-threading to support multiple requests in parallel.
- Facilitates longer running processes that are very resource-intensive.

Databases

Highlighted most popular

SQl

- mysql
- plSql
- postgreSql
- msSql
- SqlLite
- oracle 12c
- mariadb
- microsft sql server
- ibm sql

NoSQL (Not only SQL)
JSON or BJSON huge data can store in the form of key and value pairs

NoSQL Not only SQL

- JSON or BJSON huge data can store in the form of key and value pair
- Mongodb
- cassandra
- Hbase (BIG DATA HADOOP)
- Amazon dynamodb (AWS)
- Elasticsearch

Day2 node

Thursday, January 20, 2022 12:27 PM

Browser uses web API
Node uses Node API

What is nodejs?
JS runtime environment that execute on server side platform with node api.
It is open source platform

Ryan dhal natively used c++ and created a run time environment wrapper
Nodejs is not a framework, it is not a programming language
It is a server side runtime environment.
Code on JS only, behind the scene it is C++.

- ***JS is not a server side language but it is a browser side langauge.***

Why we need node js?
Frontend and backend both are Javascript (fullstack).
Node js is used for network based application like chat, streaming
Node js is asynchronous in nature.
Easy to learn

Node.js Advantages: Why use Node.js for developing web apps

- High-performance for Real-time Applications
- Easy Scalability for Modern Applications
- Cost-effective with Fullstack JS
- Community Support to Simplify Development
- Easy to Learn and Quick to Adapt
- Helps in building Cross-functional Teams
- Improves App Response Time and Boosts Performance
- Reduces Time-to-Market of your applications
- Extensibility to Meet Customized Requirements
- Reduces Loading Time by Quick Caching
- Helps in Building Cross-Platform Applications

Node.js Disadvantages: What is Node.js not good for

- Reduces performance when handling Heavy Computing Tasks
- Node.js invites a lot of code changes due to Unstable API
- Node.js Asynchronous Programming Model makes it difficult to maintain code
- Choose Wisely – Lack of Library Support can Endanger your Code
- High demand with a few Experienced Node.js Developers

questions

Friday, January 21, 2022 6:02 PM

Domain Name System (DNS)
MVC architecture

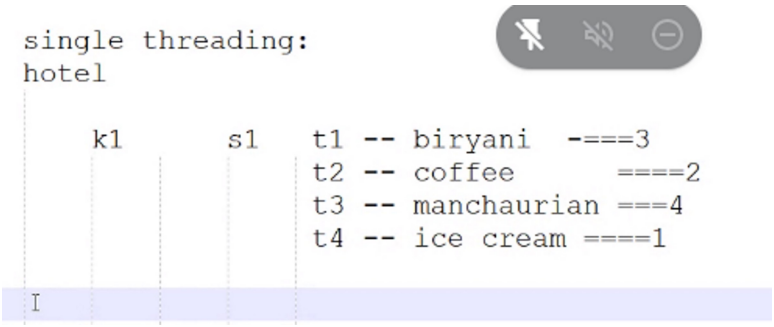
Server
1. Can hold business logic.
2. Write code to interact to database

What are the major features of node js?
1. It is single threaded
2. Supports Asynchronous
3. Supports non-blocking I/O operation.

Why node js is preferred?

Ex applications (real time)
Chat application:
Real time application:
Streaming application:

Paypal java to node js:
33% less lines of code
40% less files
2 times higher request per second
35% faster response



It doesn't wait until a single request gets completed it automatically takes the request and shares the response.

[21/01/2022 6:28 PM] Priyanka K M: Node JS:
A runtime Environment for executing JS code.i.e. to run the JS code on Server.

->Node JS is used for Backend services.(API)

Used to build:
-> Highly Scalable
-> Data-intensive
-> Real time apps

History of Node JS:

One key factor that led to the rise of Node.js was the timing.

2009 --> Node.js is born
The first form of npm is created
2010 --> Express is born
Socket.io is born
2011 --> npm hits version 1.0
Larger companies start adopting Node.js: LinkedIn, Uber, etc.
2015 --> The Node.js Foundation is born
npm introduces private modules

Node.js 4 (versions 1, 2 and 3 never previously released)
2016 --> The leftpad incident
Yarn is born
Node.js 6
2017 --> npm focuses more on security
Node.js 8
V8 introduces Node.js in its testing suite, officially making Node.js a target for the JS engine, in addition to Chrome
2018 --> Node.js 10
Node.js 11
2019 --> Node.js 12 & Node.js 13
2020 --> Node.js 14 & Node.js 15
2021 --> Node.js 16

==========

Why Node js..?
-> great for prototyping and agile development
-> Super fast & highly scalable
-> JavaScript Everywhere
-> Cleaner and more consistent codebase
-> Large eco system of open source libraries.(npm has 50,000 bundles for developers).

==========

PayPal App:
(previously built using java------> node js)
Built twice as fast with fewer people
33% fewer lines of code
40% fewer files
2x request/sec
35% faster response

Node Architecture:

The JS Engine:
JS Code ---> JS Engine ---> Machine code

Browser provides a runtime environment to execute the JS Code.

until 2009, JS was used in only browsers.
After,2009 ---> Ryan Dahl (Introduced the Node)

Chrome ------ + ----- C++ Program =====> Node JS
(v8 Engine) (embedded)

Note:
-> Node doesn't have dom objects,instead it as modules.

"NODE is NOT a Programming Language"
"NODE is NOT a FRAMEWORK"

i.e its a runtime environment for executing JS Code.

But There are few frameworks which can be used with Node Js.

AdonisJS, Egg.js, Express, Fastify, Next js,Gatsby etc...

==========

Day 3 node

Saturday, January 22, 2022 9:37 AM

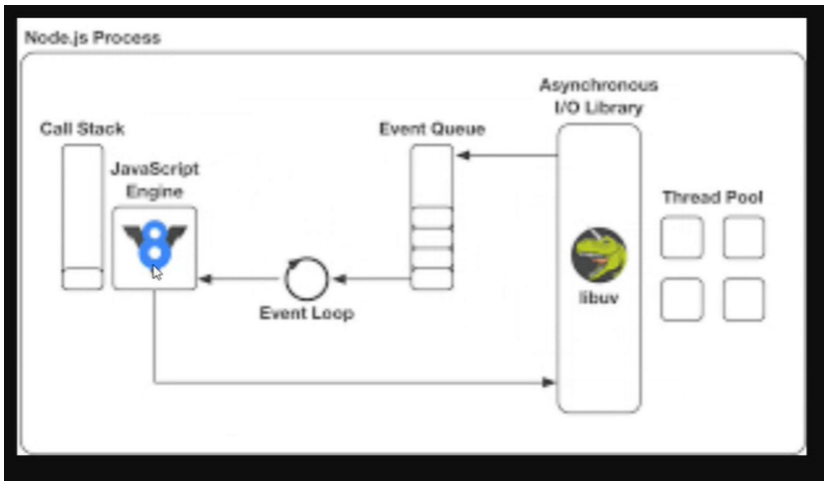
Node js is event driven asynchronous architecture.C:\Users\bhuvan\Desktop\MERN\node\app.js

You must use call back function to make it asynchronous .

```
let promise = new Promise((resolve, reject) => {
    let isavai = true;
    if (isavai) {
        resolve("yes avialble");
    } else {
        reject("nooo");
    }
});
promise.then(data => {
    console.log(data);
})
promise.catch(err => {
    console.log(err);
})
```

Whenever we call promises it goes to microtask queue

LIBUV is a library in C langaugae
Primarily created for node js
This supports asynchronous operation in node js



Libuv is a multi platform c library that provides support for asynchronous I/O based event loops

In the background all threads are running but main thread is not affected.
Nodesjs is a non blocking IO operation because of lib uv library.

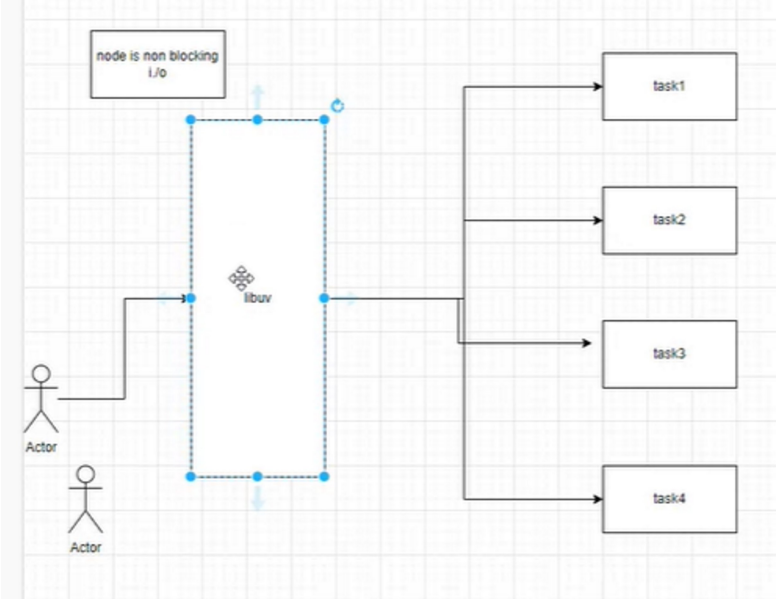
LIBUV maintains thread pool
The LIBUV library maintains a poll of threads that is ***used by nodejs to perform long running operations in the background, without blocking its main thread.***

So using LIBUV we can perform multithreading like operations.
We have only 4 threads
DNS
File system API
Cypto
Zip compression

Here mulithreading is achieved using

Why node js is faster?
Because of event loop and LIBUV

What type of applications we can create using nodejs?
Network based applications

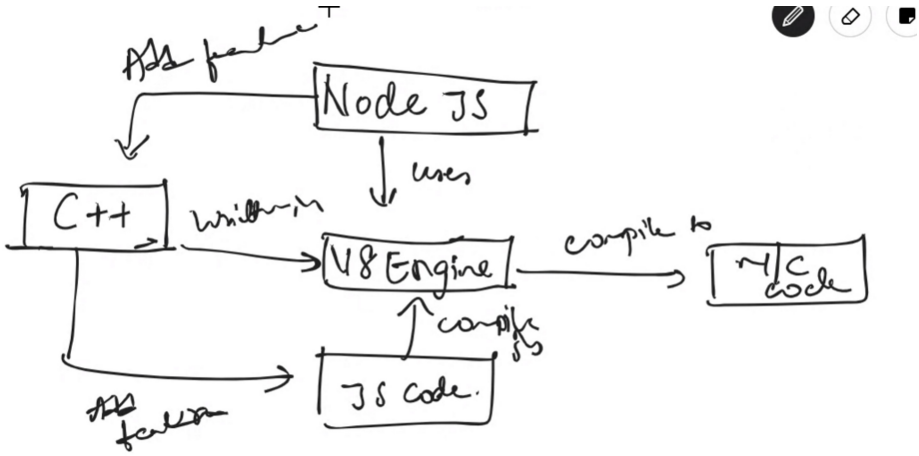


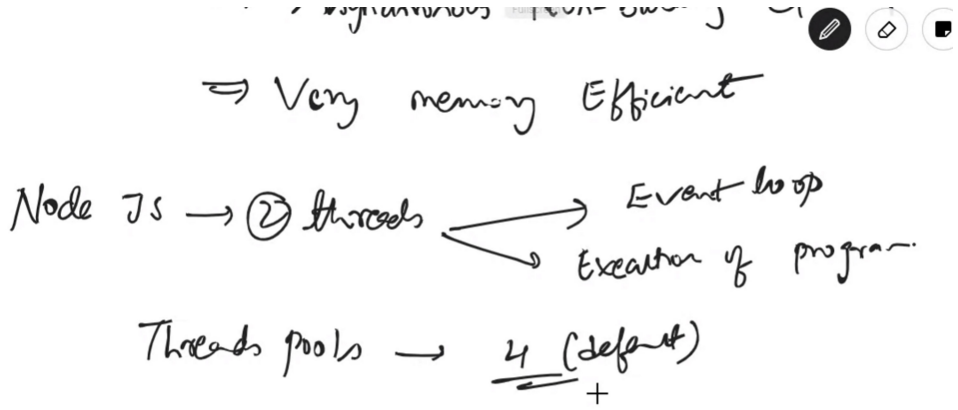
Main thread is never blocked.
Main door is always open, we can have N no of concurrent users requests.

Non Blocking I/O operation allow single process to serve multiple requests at the same time. Instead of the process being blocked and waiting for I/O operations to complete, the I/o operations are delegated to the system, so that

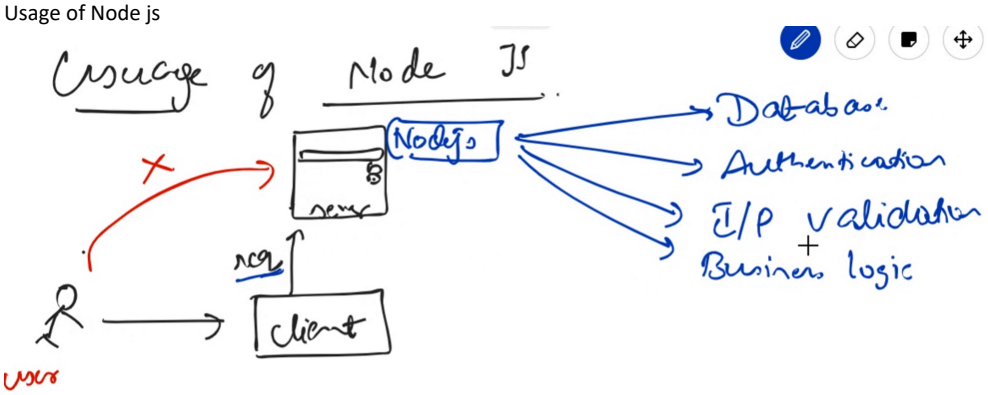
Terminlogies

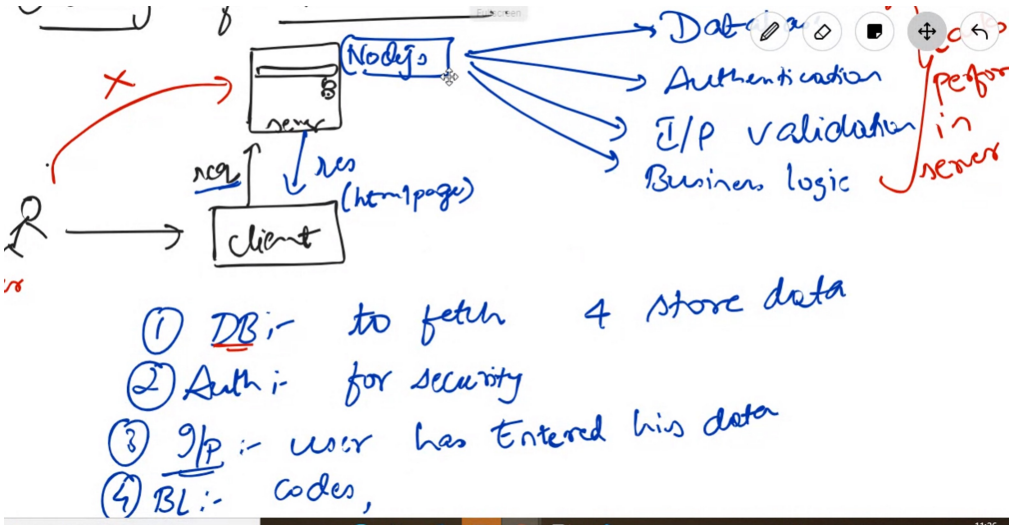
1. Process
2. What are threads?
3. Multiprocessing
4. Multithreading





Usage of Node js





nOTE:\

1. ALL THE js CODE RUNS ON server not on browser using node js.
2. In directly, it allows the user to work with server in request and response pattern.
3. Nodejs helps to write code on server and returns data to the client or user to work

Node JS Role wrt Web Development

1. Run Server → Create Server, Listen to Incoming Request
2. Business Logic → Handle Request, validate I/P, Connect to Database
3. Responses → Return Responses (HTML, json..)

We can execute code in node js even without file extension.
REPL,
R- read user input
E- Evaluatte user input
P- Print output(result)
L- wait for new input

R → Read user Input

E → Evaluate user Input

P → Print output(Result)

L → wait for new Input

Two ways of Running Node.js code

Execute files

- used for Real apps
- Sequence of steps is predictable

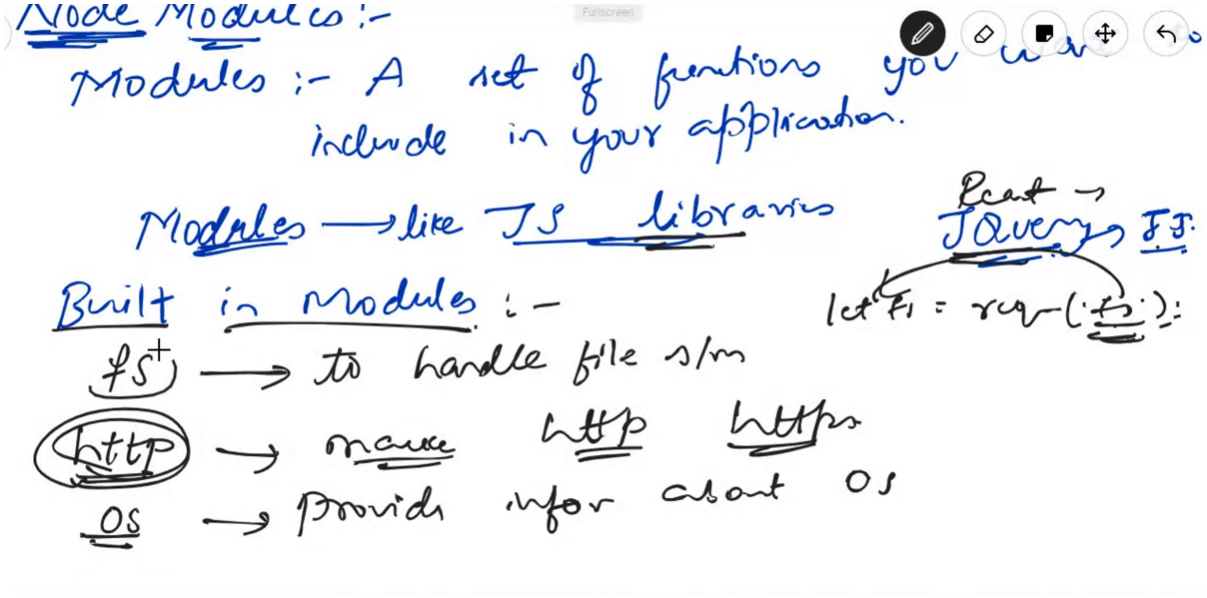
Ex. node app.js

use REPL

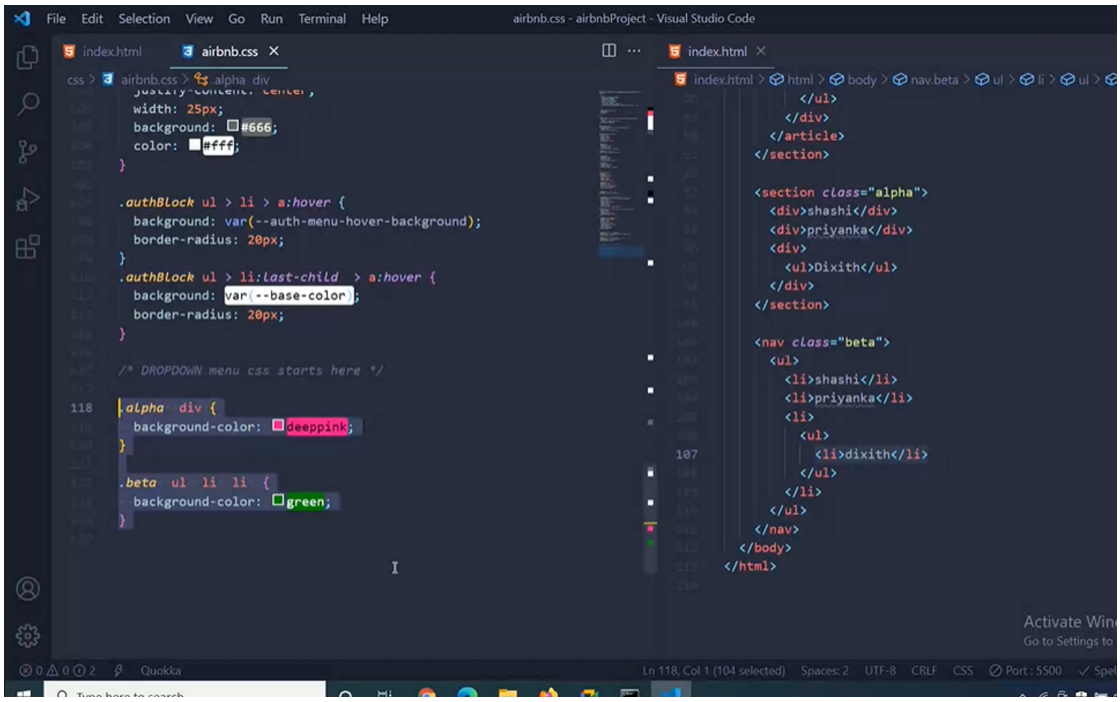
- Great playground
- Execute code as you write

Ex. node terminal

```
> 2+2
4
>
```




```
built in modules:
    1. fs
    2. os
    3. path
    4. http
    5. event
    6. https
```



Modules are the blocks of encapsulated code that communicate with an external application on the basis of their related functionalities.
Modules can be a single file or a collection of multiple file or folders.

Types

1. Common JS
2. ES6 modules

Common JS modules : it works only with nodejs (default with node js). It is not for the browser.
Allows both single and multiple export

ES6 module: it can be used anywhere in nodejs, reactjs but it requires Babbel.

Allows only one export per file.

When we use type=module we are using module as a flag.

Day 4 and 5

Tuesday, March 8, 2022 3:31 PM

FS module

12 February 2022 12:28

Synchronous

Each statement is processed line by line, one after another.

READ FILE IN Synchronous way

```
fs.readFileSync()
```

It takes 2 arguments

1. Path to the file from which we are reading, should be enclosed in double quotes.
2. Character encoding, should be enclosed in double quotes.

```
const fs = require("fs");
//synchronous way
let text = fs.readFileSync("./text/input.txt", "utf-8");
console.log(text);

In this example first file system module is required ,
Then the file is read
And then we log result to console
Each line waits for the result of previous line(synchronous)
This result in slow operation
So synchronous code is also called blocking code because a certain operation only
 be executed after the before has finished

If we will not write it will return buffer
Utf-8 does character encoding
```

WRITE FILE IN Synchronous way

```
fs.readFileSync()
```

It takes 2 arguments

1. Specify the path and name of new file in it , should be in string
2. Specify what we want to write in the file

```
const fs = require("fs");
//synchronous way reading file
let readText = fs.readFileSync("./text/input.txt", "utf-8");
console.log(readText);
let textOut = `Hello devloper : ${readText}.\n on ${Date.now()}`
console.log(textOut);
```

```
//writing file in synchronous way
fs.writeFileSync("./text/output.txt", textOut)
```

Here output.txt is new file created using write file

Another example1

```
// create file using writeFileSync
let readmeText = fs.readFileSync("./readme.txt", "utf-8");
let writeMeText = fs.writeFileSync("writeMeText.txt", readmeText);
```

Another example2

```
const fs = require("fs");//no need to install

let readmeText = fs.readFileSync("./readme.txt", "utf-8");
let writeMeText = fs.writeFileSync("writeMeText.txt", readmeText);
// we can create file using writeFileSync
//2nd parameter is text , we can read from any file or we can add direct text to new file by
writing content instead of filename
console.log(writeMeText);
```

To create directory

```
const fs = require("fs");//no need to install


fs.mkdirSync("jspider"); //folder created
```

To create a folder and inside that we need to create 2 files

```
const fs = require("fs"); // no need to install

fs.mkdirSync("qspider");

fs.writeFileSync("./qspider/fronend.js","html,css , js")
fs.writeFileSync("./qspider/backend.js","node js, express");
```

How to delete file synchronously

```
const fs = require("fs");
fs.unlinkSync("./qspider/backend.js");
//backend.js deleted
```

```
fs.unlinkSync("./qspider/fronend.js");
```
Frontend..js deleted

To delete folder

```
fs.rmdirSync("./qspider");
```

```
rename (sync)

fs1.renameSync("readnew","readme.txt")
```

To rename file

```
fs1.renameSync("karthik.txt","readme.txt"
```

Asynchronous/non- blocking

To overcome synchronous , we have to use asynchronous/non-blocking code

In asynchronous we upload heavy code to basically be worked on background.
And once the work is done the callback function which we register before is called to handle the results. And during that time the rest of the code can still be executing without being blocked by heavy task which is running in background to make code non-blocking

READ FILE IN Asynchronous way

```
fs.readFile()
```

It takes 3 arguments

1. Name of the file/ path
2. need to specify utf-8 encoding in readFile()
3. third parameter is callback function

```
fs.readFile("./text/input.txt")
```

So the node will start reading ./text/input.txt in the background and soon as it ready it will start callback function

We call callback function with 2 arguments
First one is err and second actual data.

```
//Non-blocking ,asynchronous readFile

const fs = require("fs");
fs.readFile("./text/input.txt", "utf-8", (err, data) => {
  if (err) throw err;
  console.log("successfully read data");
  console.log(data);
});
```

As soon as the readFile function runs, it will start reading file ./text/input.txt in the background without blocking the rest of the code execution

To demonstrate this

```
const fs = require("fs");
fs.readFile("./text/input.txt", "utf-8", (err, data) => {
  if (err) throw err;
  console.log("successfully read data");
  console.log(data);
});
console.log("helloWorld");
```

So here first we see helloworld executed and than successfully read data ,
So here nodejs will start reading file in background without blocking rest of code and immediately move next line of code.

```
const fs = require("fs");
fs.readFile("./text/start.txt", "utf-8", (err, data1) => {
    fs.readFile(`./text/${data1}.txt`, "utf-8", (err, data2) => {
      console.log(data2);
    });
});
console.log("helloWorld");
```

Here first helloworld and after that it will print data inside the file which se read using readFile

Write FILE IN Asynchronous way

```
fs.writeFile()
```

It takes 4 arguments

1. Name of the new file if needed with path

2. data/text we want to add in new file
3. Character encoding
4. Fourth parameter is callback function with err and no need to pass data as we just directly add the file/data in second parameter.

```
//writing file asynchronous
const fs = require("fs");
fs.readFile("./text/start.txt", "utf-8", (err, data1) => {
    fs.readFile(`./text/${data1}.txt`, "utf-8", (err, data2) => {
        console.log(data2);
         fs.readFile(`./text/output.txt`, "utf-8", (err, data3) => {
             console.log(data3);
             fs.writeFile("./text/final.txt",`${data2}\n ${data3}`,"utf-8" ,err => {

             });
         });
    });

});
console.log("helloWorld");
```

```
Node implements asynchronous by calling callback operations which we are doing
```

Asynchronously reading a file

Fs.readFile(path, encoding,callback)

```
//3argument: first path, second encoding, third callback

let readIndex = fs.readFile("./index.js", "utf8", (err, chunk) => {
    if (err) throw err;
    console.log(chunk);
});
```

Reading file through promises

```
const fs = require("fs/promises");
fs.readFile("./readme.txt", "utf-8").then(chunk => {
    console.log(chunk);
}).catch(err => {
    console.log(err);
})
```

doing using ES6

```
 import { readFile, writeFile } from "fs";

readFile("./mernstack.js", "utf-8", (err, data) => {
  if (err) throw err;


      write file

  writeFile("./meanstack.js", "utf-8", (err) => {
     if (err) throw err;
     console.log("successfully created");
     console.log(data);
   });
 });
```

Asynchronously deleting a file(ES6)

Delete

```
import { readFile, writeFile } from "fs";
readFile("./mernstack.js", "utf-8", (err, data) => {
  if (err) throw err;
  // write file
  writeFile("./meanstack.js", data, err => {
    if (err) throw err;
    console.log("successfully meanstack created ");
    console.log(data);
  });
});
fs.unlink("./meanstack.js", err => {
  if (err) throw err;
  console.log("successfully meanstack deleted");
});
```

Folder structure example

```
import { mkdir, writeFile } from "fs";
mkdir("src", (err) => {
  if (err) throw err;
  console.log("successfully folder created")
  writeFile("./src/index.js", "index data", (err) => {
    if (err) throw err;
    console.log("successfully index file created");
  });
  writeFile("./src/app.js", "app data", err => {
    if (err) throw err;
    console.log("successfully app file created");
  });
```

```
  mkdir("./src/components", err => {
    if (err) throw err;
    console.log("component folder created");
    mkdir("./src/components/navBar", err => {
      if (err) throw err;
      console.log("navbar folder created");
      writeFile("./src/components/navBar/Navbar.jsx", "done", (err) => {
        if (err) throw err;
        console.log("successfully navbar file created");
      });
    });
  });
});
```

doing folder structure using promise

```
const fs = require("fs").promises; //or
const fs = require("fs/promises");

fs.mkdir("src").then(_ => {
    console.log("src created using promise");
    fs.writeFile("./src/app.js", "data").then(_ => {
        console.log("app.js created inside src");
    }).catch(err => {
        console.log(err);
    })
})
```

Using async and await

```
const fs = require("fs/promises");
let ReadDataByUsingAW = async () => {
    try {
        let data = await fs.readFile("./readme.txt", "utf-8")
        console.log(data);
    } catch (error) {
        console.log(error);
    }
}
ReadDataByUsingAW();
```

Using immediate invoking

```
const fs = require("fs/promises");
(async () => {
    try {
        let data = await fs.readFile("./readme.txt", "utf-8")
        console.log(data);
    } catch (error) {
        console.log(error);
    }
```

```
})()
```

Async append data to a file

```
fs.appendFile("../write.txt","where r u karthik").then(_=>console.log("append
data is read me")).catch(err=>console.log(err))
```

and

```
fs.appendFile('./readMe.txt', "attending class..")
    .then(_ => {
        console.log("appnded data to reamme file");
    }).catch(err => console.log(err));
fs.readFileSync("../readMe.txt","utf-8",)
```

```
rename (async)
fs.rename("../readme.txt", "readnew.txt").then(_ => {
 console.log("renamed");
}).catch(err => {
 console.log("error");
})
```

```
delete (async)
fs.unlink("./meanstack.js", err => {
  if (err) throw err;
  console.log("successfully meanstack deleted");
});
```

Readable Stream

```
const fs = require("fs");
```

create stream

```
let readableStream = fs.createReadStream("./DOM.mp4", "utf-8");

which file we want to read,data is passed in chunks,
if we remove utf-8 we will get as buffer , utf-8 use for encoding to actual
format we want to read

 nodejs events
readableStream.on("data", chunk=> {
```

Please wait while OneNote loads this Printout...

Please wait while OneNote loads this Printout...

Please wait while OneNote loads this Printout...

Please wait while OneNote loads this Printout...

Day 6 Buffer and streams

Wednesday, January 26, 2022 9:10 AM

Buffer

It is temprovary raw chunk of data.
Buffers are class in node js designed to handle raw binary data.
Each buffer corresponds to raw memory allocated outside v8 engine.
Buffer is not an array but it acts like an array.
Stream is a flow of data

```
let x = Buffer.from("shashi");
console.log(x);
console.log(x.toString());
```

```
C:\Users\bhuvan\Desktop\MERN\node\Day6>node app.js
<Buffer 73 68 61 73 68 69>
shashi
```

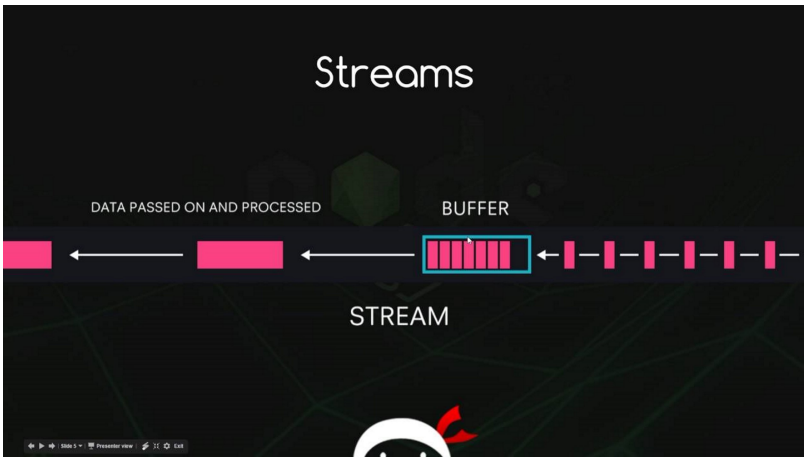
Difference between stream and buffer

Buffer	Stream
Raw data stored in one place.	Data travelling from one resource to another.

Streams are objects that allows developers to read/write data to and from a source in a continous manner.

4 types of streams

1. Readable - a stream which is used for read operation
2. Writeable - stream which is used for write operation
3. Duplexstream - stream which can be used for both read and write operation
4. Tranformstream - a type of duplex stream where output is computed based on input.



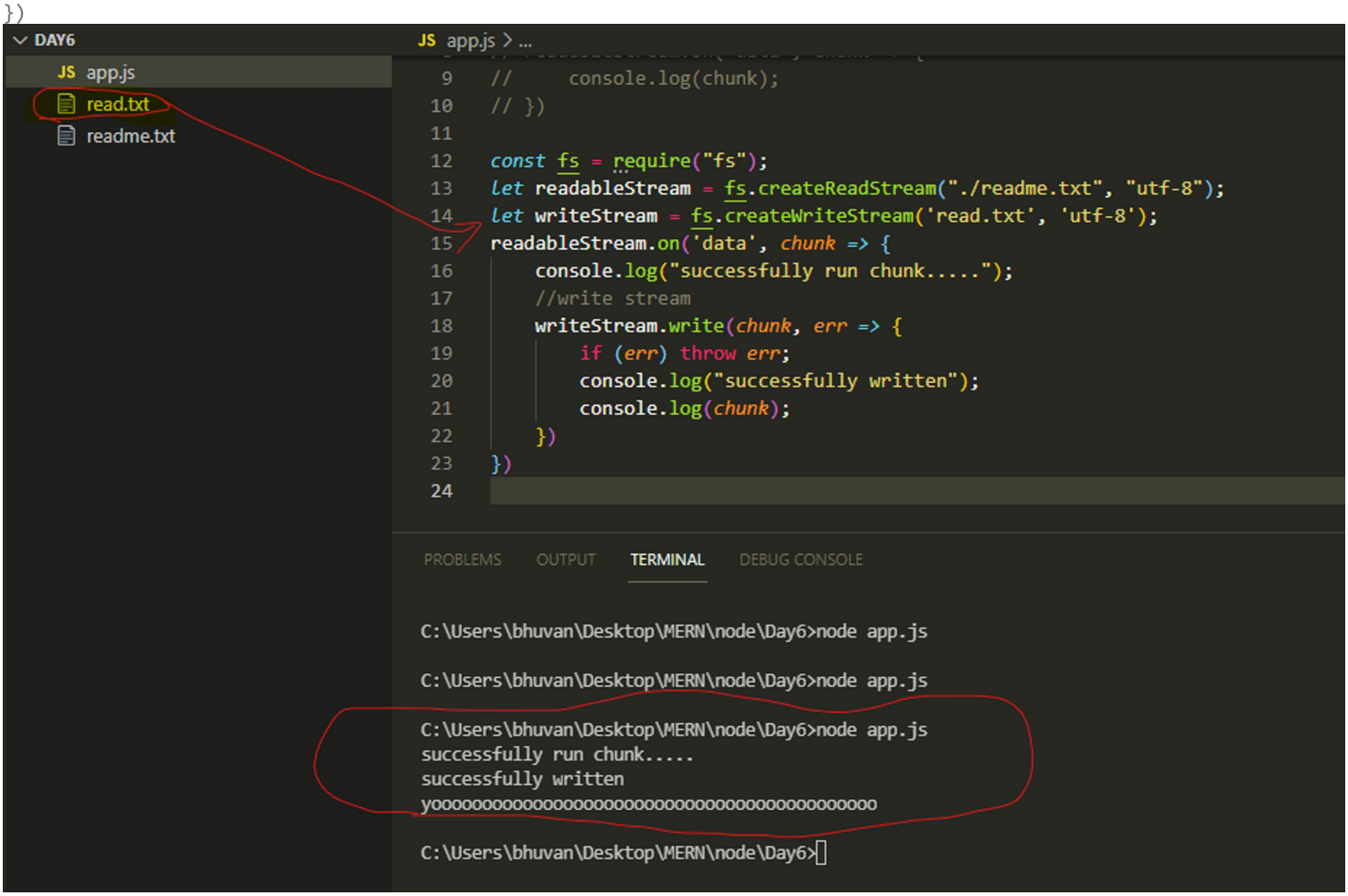
Readable stream

```
const fs = require("fs");
let readableStream = fs.createReadStream("./readme.txt", "utf-8");
//add events to stream
readableStream.on('data', chunk => {
    console.log(chunk);
})
```

We need to add events and we achieve that using on method

Writeablestream

```
const fs = require("fs");
let readableStream = fs.createReadStream("./readme.txt", "utf-8");
let writeStream = fs.createWriteStream('read.txt', 'utf-8');
readableStream.on('data', chunk => {
    console.log("successfully run chunk.....");
    //write stream
    writeStream.write(chunk, err => {
        if (err) throw err;
        console.log("successfully written");
        console.log(chunk);
    })
})
```



For redable and writeable stream do not use promises it has to be written using fs module only.

Another way to read and write we can use pipe()
Internally It pipes all streams and buffers

```
//using pipes
const fs = require("fs");
let readableStream = fs.createReadStream("./readme.txt", "utf-8");
let writeStream = fs.createWriteStream("read.txt", "utf-8");
readableStream.pipe(writeStream);
```

==========

Npm init
Hit enter
Go to package.json

```
"type": "module"
```

After license
==========

Http module

Wednesday, January 26, 2022 10:21 AM

Https
Ssl - software secure layer

How to create web server in node js
.createServer() takes a call back function with two parameters request and response
First ask request then reesponse.

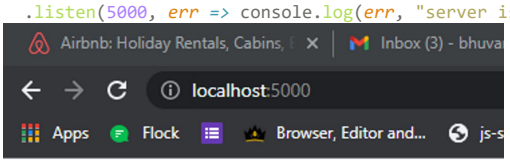
80 Is the default port number but do not use it

```
//how to create web server in nodesjs
const http = require('http');
const server = http.createServer((request, response) => {
    console.log(response);
    // response.end("ok");
})
//assign port number
let port = 4000;
server.listen(port, (err) => {
    if (err) throw err;
    console.log("server is running on port number", port);
})
```

```
C:\Users\bhuvan\Desktop\MERN\node\Day6>node app.js
server is running on port number 4000
^C
```

Short and easy way to make a server

```
const http = require("http");
http
  .createServer((req, res) => {
    console.log(res);
    res.end("hey client be happy i am giving response");
  })
  .listen(5000, err => console.log(err, "server is running on 5000"));
```




```
hey client be happy i am giving response
```

HTTP methods

1. Get - read data or fetching data from server
2. Post -
3. Put
4. Delete

Get method requests a respresntation of the of the specified resource. Requests using GET should only be used to request data (they shouldn't have payload in it or data in it).

Get request table

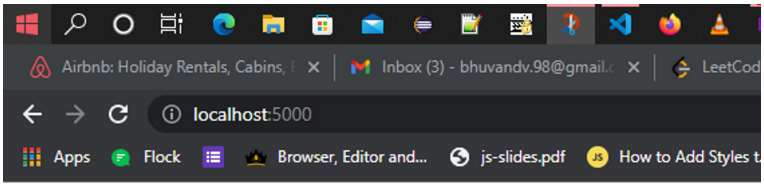
Request has body	No
Successful response has body	Yes
safe	Yes
Idempotent or identical	Yes
Cacheable	Yes
Allowed in HTML forms	Yes

Paste picture here

- It is safe as it is only read only does not modify data on server
- Used in html forms
- It is not a dynamic request it is a identical request

Get method

```
const http = require("http");
const fs = require("fs");
http
  .createServer((req, res) => {
    //set header
    res.setHeader("Content-type", "text/html");
    fs.readFile("./index.html", "utf-8", (err, data) => {
      if (err) throw err;
      res.end(data); //ending req /res cycle
    });
  })
  .listen(5000, err => {
    if (err) throw err;
    console.log("server is listing on port number 5000");
  });
```



welcome to node environment

Lorem ipsum dolor sit amet consectetur, adipisicing elit. Voluptas dicta ab dolore quod! quasi commodi ea aliquid ratione enim eum veniam eligendi ut molestias reiciendis. Po consequatur dolorum, officia fugit amet a impedit consectetur error facere porro archite aut nihil dolor, esse quidem libero? Soluta nisi minima corrupti dicta rem eius labore cc

To attach css from server

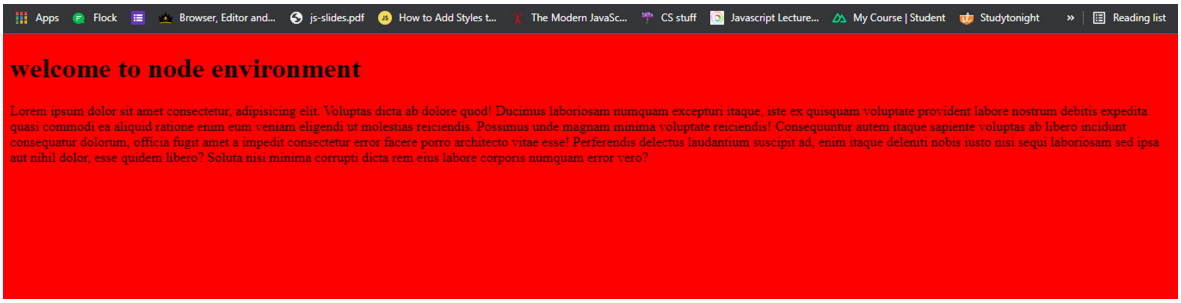
```
//to attach html and css
const http = require("http");
const fs = require("fs");
//create web server
const server = http.createServer((req, res) => {
  if (req.url === "" || req.url === "/") {//home page checking
    //set header
    res.setHeader("Content-Type", "text/html");
    //response body
    fs.createReadStream("./index.html", "utf-8").pipe(res);
  } else if (req.url === "/style.css") {
    res.setHeader("Content-Type", "text/css");
    //response body
    fs.createReadStream("./style.css", "utf-8").pipe(res);
  } else {
    res.end(`<h1 style="color:red">Page not found</h1> `);
  }
});
let port = 8000;
server.listen(port, err => {
  if (err) throw err;
  console.log("server is listing on port number", port);
});
```

```
body {
  background-color: red;
}
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="/style.css" />
  </head>
  <body>
    <h1>welcome to node environment</h1>
    <p>
      Lorem ipsum dolor sit amet consectetur, adipisicing elit. Voluptas dicta
      ab dolore quod! Ducimus laboriosam numquam excepturi itaque, iste ex
      quisquam voluptate provident labore nostrum debitis expedita quasi commodi
      ea aliquid ratione enim eum veniam eligendi ut molestias reiciendis.
```

```
      Possimus unde magnam minima voluptate reiciendis! Consequuntur autem
      itaque sapiente voluptas ab libero incidunt consequatur dolorum, officia
      fugit amet a impedit consectetur error facere porro architecto vitae esse!
      Perferendis delectus laudantium suscipit ad, enim itaque deleniti nobis
      iusto nisi sequi laboriosam sed ipsa aut nihil dolor, esse quidem libero?
      Soluta nisi minima corrupti dicta rem eius labore corporis numquam error
      vero?
    </p>
  </body>
</html>
```

Output



How to serve api

```
// using ES6 method
import { createServer } from "http";
import { createReadStream } from "fs";
createServer((req, res) => {
  let employes = [
    {
      emp_name: "manu",
      emp_id: "ty1",
      emp_salary: 20000,
      emp_designation: "node developer",
    },
    {
      emp_name: "shashi",
      emp_id: "ty2",
      emp_salary: 2000000,
      emp_designation: "html developer",
    },
  ];
  //set header
  res.setHeader("Content-type", "application/json");
  //send response to client
  res.end(JSON.stringify(employes));
}).listen(5000, err => {
  if (err) throw err;
  console.log("server running successfully in 5000");
});
```

To render audio or video

```
// for video and audio file
const http = require("http");
const fs = require("fs");
const server = http
  .createServer((req, res) => {
    //set header
    res.setHeader("content-type", "video/mp4");
    let video = fs.readFileSync("video.mp4");//The file should be in same folder with name video.mp4
    res.end(video);
  })
  .listen(5000, err => {
    if (err) throw err;
    console.log("app is runing");
  });
```

To create a full fleged basic web application

```
//basic web application
const http = require("http");
const fs = require("fs");
const server = http.createServer((req, res) => {
  res.setHeader("content-type", "text/html");
  let path = "";
  switch (req.url) {
    case "/":
      path += "./index.html";
      break;
    case "/style.css":
      path += "./style.css";
      res.setHeader("content-type", "text/css");
      break;
    case "/api":
      path += "./data.json";
      res.setHeader("content-type", "application/json");
      break;
    case "/video":
      path += "./video.mp4";
      res.setHeader("content-type", "video/mp4");
      break;
    case "/audio":
      path += "./Jack.mp3";
      res.setHeader("content-type", "audio/mp3");
      break;
    default:
      path += "./pagenotfound.html";
  }
  fs.createReadStream(path).pipe(res);
});
server.listen(5000, err => {
  if (err) throw err;
  console.log("app is running");
});
```

POST

- This is more secure than GET
- Everytime the request is not identical
- It is available in HTML forms

POST is an HTTP method designed to send data to the sever from an HTTP client.

Request has body	Yes
Succesful response has body	Yes
Safe	No
idempotent	No
cacheable	Only if freshness information is included
Allowed in HTML forms	yes

Get	Post
Get I used for reading or fetching data from the server	Post is creating data in the server
Requst has no payload	Request has payload
safe	Not safe

Content type - application / x-www-form-urlencoded

For post request partial snippet
Step1

```
const server = http.createServer((req, res) => {
  if (req.method === "POST") {
    let URL_ENCODED = "application/x-www-form-urlencoded";
    //====================for urlencoded
    if (req.headers["content-type"] === URL_ENCODED) {
    } else {
      res.end(null);
    }
    //====================for urlencoded
  }
```

Step 2:

```
const server = http.createServer((req, res) => {
  if (req.method === "POST") {
    let URL_ENCODED = "application/x-www-form-urlencoded";
    //====================for urlencoded
    //setPOST FORM Header
    if (req.headers["content-type"] === URL_ENCODED) {
      //send body to server call events
      let body = "";
      //node event captured using on method
      req.on("data", chunk => {
        body += chunk;
      });
      //ending the event
      req.on("end", _ => {
        console.log(body);
        res.end(`successfully logged in ${body}`);
      });
    } else {
      res.end(null);
    }
    //====================for urlencoded
```

Whole code for post request

```
const http = require("http");
const fs = require("fs");
const server = http.createServer((req, res) => {
  if (req.method === "POST") {
    let URL_ENCODED = "application/x-www-form-urlencoded";
    //====================for urlencoded
    //setPOST FORM Header
    if (req.headers["content-type"] === URL_ENCODED) {
      //send body to server call events
      let body = "";
      //node event captured using on method
      req.on("data", chunk => {
        body += chunk;
      });
      //ending the event
      req.on("end", _ => {
        console.log(body);
        res.end(`successfully logged in ${body}`);
      });
    } else {
      res.end(null);
    }
    //====================for urlencoded
  } else {
```

```
    if (req.url === "" || req.url === "/") {
      res.setHeader("content-type", "text/html");
      fs.createReadStream("./index.html", "utf-8").pipe(res);
    } else if (req.url === "./style.css") {
      res.setHeader("content-type", "text/css");
      fs.createReadStream("./style.css", "utf-8").pipe(res);
    } else {
      res.end("<h1 style='color:red'>page not found</h1>");
    }
  }
});
server.listen(5000, err => {
  if (err) throw err;
  console.log("server is online");
});
```

MDN status quotes

HTTP response codes indicates whether a specific HTTP request has been successfully completed.

Status quotes

100 - 199	Informational messages
200 - 299	Success messages
300 - 399	Redirection
400 - 499	Client side errors
500 - 599	Server side errors

200	Ok
201	Created
301	Moved permanently
302	Found
304	Not modified
400	Bad request
401	Unauthorized
402	Payment required
403	forbidden
404	Not found
500	Internal server error
501	Not implemented
502	bad gateway
503	Service unavailable
504	Gateway timeout
505	HTTP version not supported

Using status codes

```
import http from "http";
import fs from "fs";
const server = http.createServer((req, res) => {
  if (req.url === "" || req.url === "/") {
    res.statusCode = 200;
    res.statusMessage = "ok";
    res.setHeader("content-type", "text/html");
    fs.createReadStream("./index.html", "utf-8").pipe(res);
    //homepage
  } else if (req.url === "/style.css") {
    res.writeHead(200, "ok", { "content-type": "text/css" });
    fs.createReadStream("./style.css", "utf-8").pipe(res);
  } else {
    //not found
    res.writeHead(404, "page not found", { "content-type": "text/html" });
    res.end("<h1>Page not found</h1>");
  }
});
server.listen(5000, err => {
  if (err) throw err;
  console.log("server is running");
});
```

We can use two methods for setting status codes

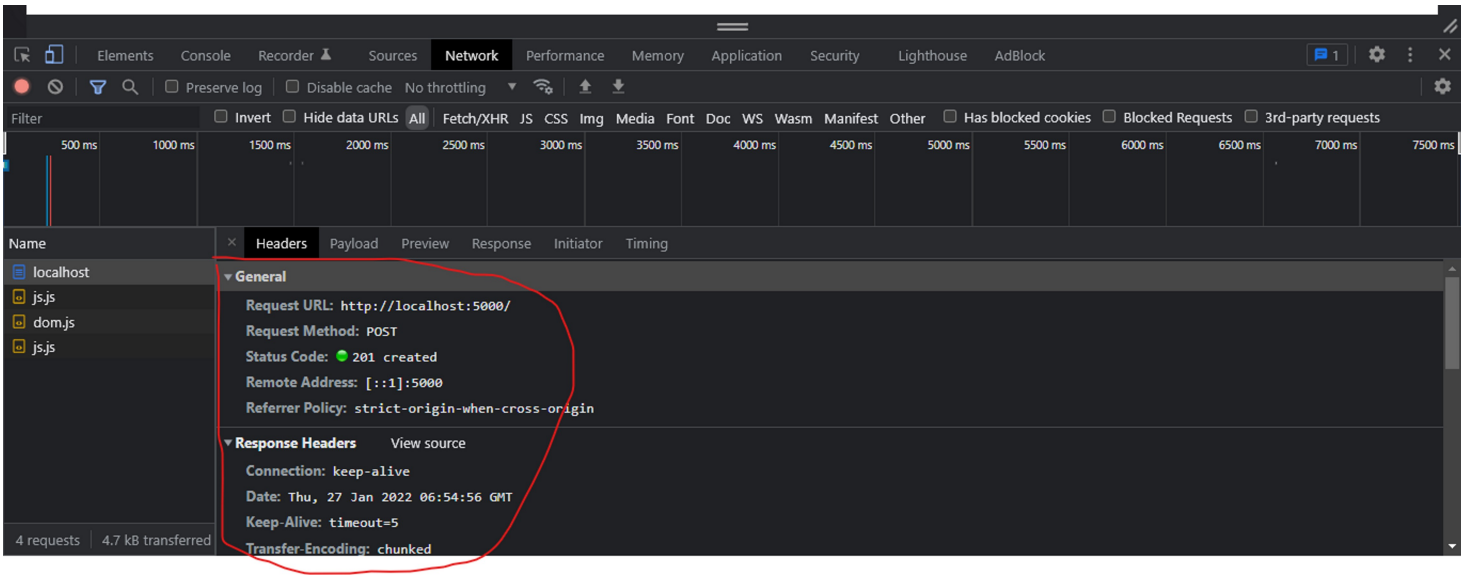
`res.statusCode = 200;`
`res.statusMessage = "ok";`
`res.setHeader("content-type", "text/html");`
	`res.writeHead(404, "page not found", { "content-type":"text/html" });`

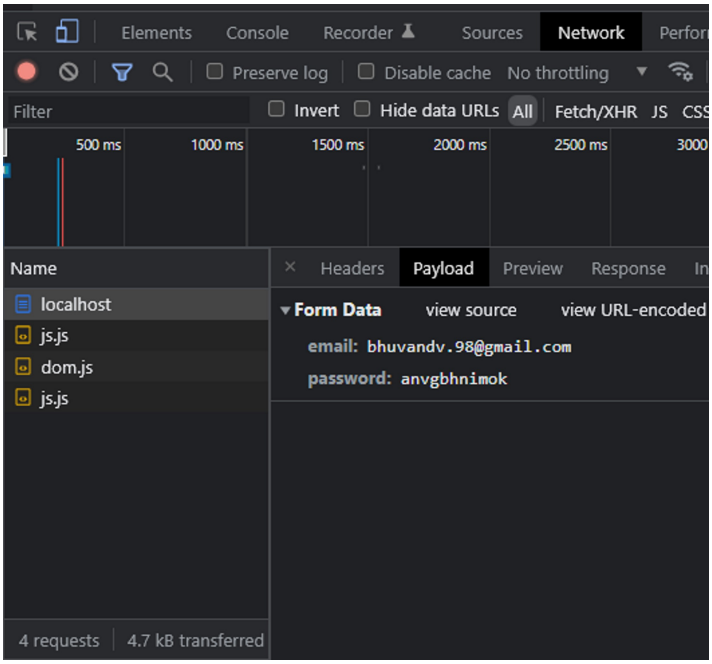
WHAT is querystring module?
The querystring module provides utilites for parsing and formatting URL query strings.

Import module and just use parse method
pass the value to be parsed inside parse()

```
import http from "http";
import fs from "fs";
import { parse } from "querystring";
const server = http.createServer((req, res) => {
  //post request with payload
  if (req.method === "POST") {
    //set header
    let URLENCODED = "application/x-www-form-urlencoded";
    if (req.headers["content-type"] === URLENCODED) {
      //event
      let body = "";
      req.on("data", chunk => {
        body += chunk;
      });
      //end event
      req.on("end", _ => {
        res.writeHead(201, "created");
        res.end("successfully logged in");
        let finalParseBody = parse(body);
        console.log(finalParseBody);
      });
    } else {
      res.end(null);
    }
  } else {
    if (req.url === "" || req.url === "/") {
      res.statusCode = 200;
      res.statusMessage = "ok";
      res.setHeader("content-type", "text/html");
      fs.createReadStream("./index.html", "utf-8").pipe(res);
      //homepage
    } else if (req.url === "/style.css") {
      res.writeHead(200, "ok", { "content-type": "text/css" });
      fs.createReadStream("./style.css", "utf-8").pipe(res);
    } else {
      //not found
      res.writeHead(404, "page not found", { "content-type": "text/html" });
      res.end("<h1>Page not found</h1>");
    }
  }
});
server.listen(5000, err => {
  if (err) throw err;
  console.log("server is running");
});
```

outputs






```
C:\Users\bhuvan\Desktop\MERN\node\http module>node index.js
server is running
[Object: null prototype] {
  email: 'bhuvandv.98@gmail.com',
  password: '123456'
}
^C
C:\Users\bhuvan\Desktop\MERN\node\http module>node index.js
server is running
[Object: null prototype] {
  email: 'bhuvandv.98@gmail.com',
  password: '123456'
}
[Object: null prototype] {
  email: 'bhuvandv.98@gmail.com',
  password: 'anvgbhnimok'
}
[Object: null prototype] {
  email: 'bhuvandv.98@gmail.com',
  password: 'anvgbhnimok'
}
^C
C:\Users\bhuvan\Desktop\MERN\node\http module>
```

Install node mailer module

```
C:\Users\bhuvan\Desktop\MERN\node\http module>npm install nodemailer
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN http-module@1.0.0 No description
npm WARN http-module@1.0.0 No repository field.

+ nodemailer@6.7.2
added 1 package from 1 contributor and audited 1 package in 1.857s
found 0 vulnerabilities


C:\Users\bhuvan\Desktop\MERN\node\http module>
```

SMTP simple mail transfer protocol

```
import nodemailer from "nodemailer";
const transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: "hypertasker98@gmail.com", //user and from should be same
    pass: "",
  },
});
const options = {
  from: "hypertasker98@gmail.com",
  to: "bhuvandv.98@gmail.com",
  subject: "testing🤣",
  html: `
        <h1>welcome to jspiders</h1>
        <p>we are learning nodejs and expressjs </p>`,
};
transporter.sendMail(options, err => {
```

```
  if (err) throw err;
  console.log("mail sent successfully");
});
```

Subscribe to newletter code

```
const fs = require("fs");
const http = require("http");
const nodemailer = require("nodemailer");
const { parse } = require("querystring");
http
  .createServer((req, res) => {
    if (req.method === "POST") {
      let URLENCODED = "application/x-www-form-urlencoded";
      if (req.headers["content-type"] === URLENCODED) {
        //event
        let body = "";
        req.on("data", chunk => {
          body += chunk;
        });
        req.on("end", _ => {
          //start nodemailer
          let { email } = parse(body);
          let transporter = nodemailer.createTransport({
            service: "gmail",
            auth: {
              user: "hypertasker98@gmail.com", //user and from should be same
              pass: "",
            },
          });
          //end transported
          let options = {
            from: "hypertasker98@gmail.com",
            to: `${email}`,
            subject: "hello from jspiders",
            html: `<h1>welcome to jspiders</h1>
            <p>we are learning nodejs and expressjs </p>
            <p>
            Lorem ipsum dolor sit amet consectetur adipisicing elit. Officiis
            molestiae quidem veritatis pariatur, optio magni omnis nesciunt ex
            eaque sequi sit animi, architecto culpa atque tempore natus
            temporibus asperiores maiores.
            </p>`,
          };
          //send mail
          transporter.sendMail(options, err => {
            if (err) throw err;
            console.log("mail has been sent!");
          });
        });
      } else {
        res.end(null);
      }
    } else {
      if (req.url === "" || req.url === "/") {
        res.writeHead(200, "ok", { "content-type": "text/html" });
        fs.createReadStream("./subscription.html", "utf-8").pipe(res);
      } else if (req.url === "/style.css") {
        res.writeHead(200, "ok", { "content-type": "text/css" });
        fs.createReadStream("./style.css", "utf-8").pipe(res);
      } else {
        //not found
        res.writeHead(404, "page not found", { "content-type": "text/html" });
        res.end("<h1>Page not found</h1>");
      }
    }
  })
  .listen(5000, err => {
    if (err) throw err;
    console.log("server is running");
  });
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <section>
      <article>
        <form action="/" method="post">
          <h2>email subscription</h2>
```

```
          <p>
            Lorem ipsum dolor sit amet consectetur adipisicing elit. Officiis
            molestiae quidem veritatis pariatur, optio magni omnis nesciunt ex
            eaque sequi sit animi, architecto culpa atque tempore natus
            temporibus asperiores maiores.
          </p>
          <div class="form-control">
            <label for="email">email</label>
            <input
              type="text"
              name="email"
              id="email"
              placeholder="enter email"
              required
            />
            <!-- <br />
            <label for="password">Password</label>
            <input
              type="password"
              name="password"
              id="password"
              placeholder="enter Password"
              required
            /> -->
          </div>
          <div id="formcontrol">
            <button type="submit">Send newsletters</button>
          </div>
        </form>
      </article>
    </section>
  </body>
</html>
```

```
* {
  box-sizing: border-box;
  padding: 0;
  margin: 0;
}
body {
  background-color: #efefef;
  font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
  font-size: 14px;
  color: #111;
}
article {
  display: flex;
  justify-content: center;
  align-items: center;
  width: 100%;
  height: 100vh;
}
form {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  height: 400px;
  width: 400px;
  background-color: #efefef;
}
.form-control {
  width: 100%;
  padding: 10px 14px;
}
input {
  width: 100%;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 5px;
  display: block;
}
label {
  width: 100%;
  display: block;
  padding: 5px 2px;
  font-size: 16px;
  font-weight: bold;
  color: rgb(20, 50, 220);
  text-transform: capitalize;
}
button {
  width: 100%;
  display: block;
  padding: 10px;
  background-color: royalblue;
  border: 1px solid rgb(39, 78, 194);
}
p{
  padding:20px;
```

```
}
```

Global objects & REPL

Friday, January 28, 2022 9:53 AM

```
console.log(__dirname);
console.log(__filename);
setTimeout(() => {
    console.log("execute later");
}, 1000);
console.log(process.stdin);
console.log(process);
```

Few node js global objects are listed below (no need to import directly use)

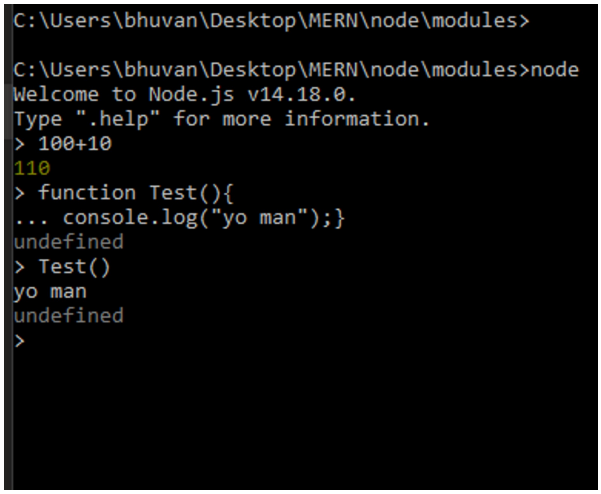
Console.log()
__dirname
__filename
Events
Module
process
Global
require()
URL
setTimeoout()
setInterval
exports

Global properties which need not require importing to use them

NodejS global objects are ***the objects that are available in all modules***. Global objects are built in objects that are part of the JS and can be used directly in the application without importing any particular module.

REPL

NodeJS READ-EVAL-PRINT-LOOP (REPL) is an interactive shell that processes Node. The shell reads JS code the user enters, evaluates the result of interpreting the line of code, prints the result to the user, and loops until the user signals to quit.



crypto

Friday, January 28, 2022 10:15 AM

Crypto is a built in module no need to install
It provides cryptographic functionality that includes a set of wrappers for OpenSSL's (software secure layer).

```
const crypto = require("crypto");
let password = "shashi123";
let final = crypto.createHmac("Sha256", password).update("i am legend").digest("hex");
console.log(password);
console.log(final);
```

```
C:\Users\bhuvan\Desktop\MERN\node\modules>node crypto.js
shashi123
03fad5d97c81805fa2c985483cca628fb721e42be5ba6b6f745a7f947f20d637

C:\Users\bhuvan\Desktop\MERN\node\modules>
```

Filename: passhash.js
Realtime implementation of hashing

```
const http = require("http");
const fs = require("fs");
const { parse } = require("querystring");
const crypto = require("crypto");
const server = http.createServer((req, res) => {
  //post request with payload
  if (req.method === "POST") {
    //set header
    let URLENCODED = "application/x-www-form-urlencoded";
    if (req.headers["content-type"] === URLENCODED) {
      //event
      let body = "";
      req.on("data", chunk => {
        body += chunk;
      });
      //end event
      req.on("end", _ => {
        res.writeHead(201, "created");
        res.end("successfully logged in");
        let finalParseBody = parse(body);//look here for sample
        let passHash = finalParseBody.password;
        let hashed = crypto
          .createHmac("Sha256", passHash)
          .update("password updated")
          .digest("hex");
        console.log(passHash);
        console.log(hashed);
      });//till here
    } else {
      res.end(null);
```

```
    }
  } else {
    if (req.url === "" || req.url === "/") {
      res.statusCode = 200;
      res.statusMessage = "ok";
      res.setHeader("content-type", "text/html");
      fs.createReadStream("./index.html", "utf-8").pipe(res);
      //homepage
    } else if (req.url === "/style.css") {
      res.writeHead(200, "ok", { "content-type": "text/css" });
      fs.createReadStream("./style.css", "utf-8").pipe(res);
    } else {
      //not found
      res.writeHead(404, "page not found", { "content-type": "text/html" });
      res.end("<h1>Page not found</h1>");
    }
  }
});
server.listen(5000, err => {
  if (err) throw err;
  console.log("server is running");
});
```

```
C:\Users\bhuvan\Desktop\MERN\node\modules>node passhash.js
server is running
qwertyuiop
46b3c3727f114ef8c3bde583f0c5f146d9348c1eeb1cbe6f93f2a3e77a643205
```

createHash
createHmac

DNS

Friday, January 28, 2022 10:53 AM

DNS
DNS servers translates request for names into IP address controlling which server an end user will reach when they type a domain name into web browser.

DNS module enables name resolution. For example, use it to look up IP addresses of host names.
Although named for the Domain Name System, it does not always use the DNS protocol for lookups. dns.lookup() uses the operating system facilities to perform name resolution.

```
const dns = require("dns");
dns.lookup("google.com", (address, family) => {
  console.log("google", address, family);
});
dns.lookup("flipkart.com", (err, address, family) => {
  if (err) throw err;
  console.log("flipkart", address, family);
});
```

```
google null 142.250.183.46
flipkart 163.53.76.86 4

C:\Users\bhuvan\Desktop\MERN\node\modules>node dns.j
google null 142.250.183.46
flipkart 163.53.76.86 4

C:\Users\bhuvan\Desktop\MERN\node\modules>
```

Also checkout resolve and

path

Friday, January 28, 2022 11:11 AM

Path

Filename path.js in modules folder

It is a built in module no need to import.

```
const path = require("path");
//?Join all arguments together and normalize the resulting path. Arguments must be strings. In v0.8, non-string arguments were silently ignored. In v0.10 and up, an exception is thrown.
console.log(path.join(__dirname + '/shashi' + '/priyanka'));
```

path.join() method
Join all arguments together and normalize the resulting path. Arguments must be string.

```
const path = require("path");
console.log(path.join(__dirname + '/shashi' + '/priyanka'));
console.log(path.join(__filename, 'qspider', 'jspiders'));
console.log(path.join(__filename, "qspider", "jspiders","..","..",".."));
```

```
C:\Users\bhuvan\Desktop\MERN\node\modules>node path.js
C:\Users\bhuvan\Desktop\MERN\node\modules\shashi\priyanka

C:\Users\bhuvan\Desktop\MERN\node\modules>node path.js
C:\Users\bhuvan\Desktop\MERN\node\modules\shashi\priyanka
C:\Users\bhuvan\Desktop\MERN\node\modules\path.js\qspider\jspiders
C:\Users\bhuvan\Desktop\MERN\node\modules

C:\Users\bhuvan\Desktop\MERN\node\modules>
```

Extension name
Return the extension of the path, from the last '.' to end of string in the last portion of the path. If there is no '.' in the last portion of the path or the first character of it is '.', then it returns an empty string

```
console.log(path.extname("shashi.java"));//.java is the output
```

Base name
Return the last portion of a path. Similar to the Unix basename command. Often used to extract the file name from a fully qualified path.

```
console.log(path.basename("akas/man/virus"));//virus
```

Normalize
Normalize a string path, reducing '..' and '.' parts. When multiple slashes are found, they're replaced by a single one; when the path contains a trailing slash, it is preserved. On Windows backslashes are used.
It just reduces the number of /////// to / (a single one).

```
console.log(path.normalize('//public///////shashi///'));
```

Output
\\public\shashi\

Resolve
The right-most parameter is considered {to}. Other parameters are considered an array of {from}.Starting from leftmost {from} parameter, resolves {to} to an absolute path.
To-->from

```
console.log(path.resolve("bhuvan","hyper.txt"));
```

Output
C:\Users\bhuvan\Desktop\MERN\node\modules\bhuvan\hyper.txt

Parse
Returns an object from a path string - the opposite of format()

```
console.log(path.parse(__filename));
```

Output

```
C:\Users\bhuvan\Desktop\MERN\node\modules>node path.js
{
  root: 'C:\\',
  dir: 'C:\\Users\\bhuvan\\Desktop\\MERN\\node\\modules',
```

```
  base: 'path.js',
  ext: '.js',
  name: 'path'
}
```

Dirname
Return the directory name of a path. Similar to the Unix dirname command.

```
console.log(path.dirname(__filename));
```

Ouput
C:\Users\bhuvan\Desktop\MERN\node\modules

Relative path
(method) path.PlatformPath.relative(from: string, to: string): string. Solve the relative path from {from} to {to}. At times we have two absolute paths, and we need to derive the relative path from one to the other. This is actually the reverse transform of path.resolve.opposite of resolve (from-->to)

```
console.log(path.relative('../../shashi','bhuvan'));
```

Ouput
C:\Users\bhuvan\Desktop\MERN\node\modules>node path.js
..\node\modules\bhuvan

Path

1	Extname	Retuns extension of path
2	Basename	Returns last portion of path
3	Normalize	Remove excecss … and /// and returns only what . Or / is neccesary
4	Resolve	Returns absolute path
5	Join	Join two or more path
6	Dirname	Returns directory name
7	Relative	

Events

Friday, January 28, 2022 12:16 PM

Events
Provides a class called EventEmitter using which we can create our own events.

```
const events = require("events");
// to ways to create event
// 1. by using constructor function
const eventEmitter = new events.EventEmitter();
//2. by using extends keyword
class eventEmitter extends events.EventEmitter() { };
```

Registering events and emitting them

```
const events = require("events");
let eventEmitter = new events.EventEmitter();
//register the event
eventEmitter.on("ddclick", () => {
    console.log("clicked!!!!");
});
//emitting the events
eventEmitter.emit("ddclick");
```

```
For dynamically chatting appl

const { privateEncrypt } = require('crypto');
const events = require('events');
const util = require('util');
const Users = function (name) {
    this.name = name;
}
util.inherits(Users, events.EventEmitter);
let girish = new Users("Girish");
let pavan = new Users("Pavan");
let allUsers = [girish, pavan];
allUsers.forEach(user => {
    // add custom events
    user.on("speak", function (message) {
        console.log(user.name + " said " + message);
    });
});
girish.emit("speak", "hii i am girish");
pavan.emit("speak", "hii i am pavan");
```

Mongo db

Friday, January 28, 2022 2:37 PM

nodemon is a command-line interface (CLI) utility developed by @rem that wraps your Node app, watches the file system, and automatically restarts the process. In this article, you will learn about installing, setting up, and configuring nodemon

npm install -g nodemon

Where it installs

```
C:\Users\bhuvan\AppData\Roaming\npm>cd node_modules

C:\Users\bhuvan\AppData\Roaming\npm\node_modules>dir
 Volume in drive C has no label.
 Volume Serial Number is 8EF5-A35D

 Directory of C:\Users\bhuvan\AppData\Roaming\npm\node_modules

01/28/2022  02:40 PM    <DIR>          .
01/28/2022  02:40 PM    <DIR>          ..
01/11/2022  01:17 PM    <DIR>          live
01/28/2022  02:40 PM    <DIR>          nodemon
01/11/2022  01:17 PM    <DIR>          server
               0 File(s)              0 bytes
               5 Dir(s)  113,274,941,440 bytes free

C:\Users\bhuvan\AppData\Roaming\npm\node_modules>
```

Yarn is developed by facebook

npm install -g yarn

projectwise
If npm => npm install package name
If yarn => yarn add package name

Machine wise
For global install
If npm=> npm install global package name || npm i -g packagename
If yarn=> yarn add global package name

In package.json
Change this

```
 "main": "server.js",
```

Remove

```
"test": "echo \"Error: no test specified\" && exit 1"
```

And type this

```
"start":"nodemon server.js"
```

To start the server use npm start

MongoClient

The MongoClient class is a class that allows for making Connections to MongoDB.

Connect() is used to connecting to mongodb URL

Create a new database name

```
const mongodb = require("mongodb").MongoClient;
let mongodbURL = "mongodb://localhost:27017";
//?The MongoClient class is a class that allows for making Connections to MongoDB.
```

```
console.log(mongodb);
mongodb
  .connect(mongodbURL) //this is a promise
  .then(db => {
    console.log("database connected");
//create a new database name
    let dbName = db.db("MERN");
    //create collections
    dbName.createCollection("students");
  })
  .catch(err => {
    console.log(err);
  });
```

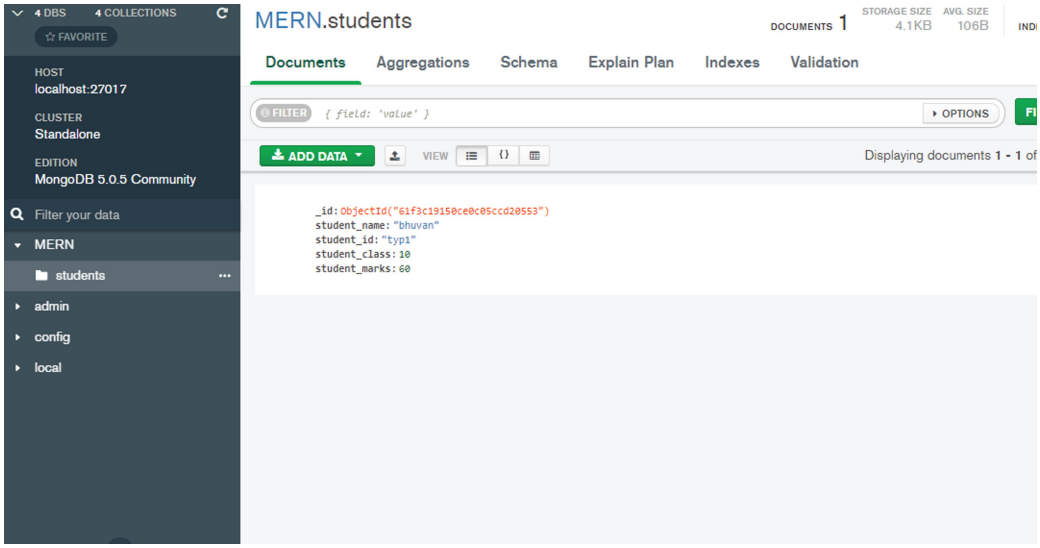
Using async and await

```
// using async
const mongodb = require("mongodb").MongoClient;
let mongodbURL = "mongodb://localhost:27017";
let mongodbconnection = async () => {
  try {
    let connections = await mongodb.connect(mongodbURL);
    //createdb name
    let dbName = connections.db("MERN");
    //create collection
    dbName.createCollection("students");
    console.log("successfully db connected");
  } catch (error) {
    console.log(error);
  }
};
mongodbconnection();
```

Inserting data into DB

¥

```
// using async
const mongodb = require("mongodb").MongoClient;
let mongodbURL = "mongodb://localhost:27017";
let mongodbconnection = async () => {
  try {
    let connections = await mongodb.connect(mongodbURL);
    //createdb name
    let dbName = connections.db("MERN");
    //insert data into collections
    let students_obj = {
      student_name: "bhuvan",
      student_id: "typ1",
      student_class: 10,
      student_marks: 60,
    };
    //create collection
    let data = await dbName.collection("students").insertOne(students_obj);
    console.log("successfully db connected");
    console.log("data is inserted");
  } catch (error) {
    console.log(error);
  }
};
mongodbconnection();
```


4 DBS
4 COLLECTIONS
FAVORITE
HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 5.0.5 Community
Filter your data
MERN
students
admin
config
local
MERN.students
DOCUMENTS 1
STORAGE SIZE 4.1KB
AVG. SIZE 106B
Documents
Aggregations
Schema
Explain Plan
Indexes
Validation
FILTER { field: 'value' }
OPTIONS
ADD DATA
VIEW
Displaying documents 1 - 1 of
_id: ObjectId("61f3c19150ce0c05ccd20553")
student_name: "bhuvan"
student_id: "typ1"
student_class: 10
student_marks: 60

NPM

Tuesday, February 1, 2022 4:38 PM

There are two ways to install a package using nom: globally and locally

Globally: this method is generally used to install development tools and CLI based packages.
To install a package globally, use the following code.

Npm install -g <package-name>

Locally: This method is generally used to install frameworks and libraries.
A locally installed package can be used only within the directory It is installed.
To install a package locally, use the same command as above without -g flag.

Npm install <package-name>

Npm packages and source control:
All the main dependecies will be in package.json

OS

Wednesday, February 2, 2022 12:17 PM

Os
Os.arch() give architecture of processor x32 x64
Os.cpus() details of processors=> Returns objects based on the processors counts
Os.userInfo() returns object name of the computer, etc
Os.hostname() returns the host name
Os.networkInterfaces() returns IP4 or IP6
Os.platform() returns linus or windows or ubuntu etc

Spawn	Fork
This starts sending data back to a parent process from the child process as soon as the child process starts executing.	This does not send data automatically, but we can use a global module name process to send data from the child process and in the parent module, using the name of the child the process to send to the child process.
It creates a new process through command rather than running on the same node process.	It makes several individual processes (child processes) but all of them run on the same node process as the parent.
In this, no new V8 instance is created.	In this, a new V8 instance is created.
It is used when we want the child process to return a large amount of data back to the parent process.	It is used to separate computation-intensive tasks from the main event loop.

Fs

Tuesday, February 1, 2022 6:28 PM

Synchronous functions

1. readFileSync( )
2. writeFileSync()
3. mkdirSync( )
4. unlinkSync( ) delete file
5. rmdirSync( ) delete folder
6. renameSync( ) rename file

Asynchronous functions

1. readFile( )
2. writeFile( )
3. mkdir( )
4. unlink( )
5. rmdir( )
6. rename( )

readFile("filename")
"filename" as first parameter
Encoding as second parameter
Cb call back function as third parameter

writeFile( )
Created Filename as first parameter
Reading file as second parameter
Third parameter call back function

mkdir( )
First parameter is folder name to be created
Second parameter call back function

unlink( )
To remove files in directory
First parameter folderpath/filename in " "
Second parameter is call back function

rmdir( )
To remove the whole folder
Same as unlink parameters

Important ques

Tuesday, March 29, 2022 5:39 PM

Node js Theory Questions.

1.What is Node JS?
2.What are the features of NodeJS?
3.Why do we use NodeJS instead of other frameworks?
4.What is waterfall model?
5.What is agile model?
6.what is promise?
7.what is closures?
8.what is callback hell?
9.what is callback?
10.how to avoid call back hell?
11. what is async and await?
12. what is rest and soap?
13. what is diff btw return and throw?
14. what is diff btw asynchronous and non blocking in node js?
15. what is the workflow of node js?
16.what if fork?
    In fork a new instance of V8 engine is created, in spawning no new instance of V8 engine is created.
17. what is middleware and type of middleware?
18. features of es6
19. default packages of node js
20. what is libuv and threads in libuv
21. what are web services
22. what are micro function
23. what are event loops and event queues in node js
24. why do you need callback
25. what are jwt tokens and how to use it
26. how to integrate backend with frontend
27. why do you use package.json
28. what is clusters
29. what is multipart form data and multer
30. what is node API
31. what is the diff btw authentication and authorization
32. what is REPL
33. why do we use express instead of other frameworks
34. What is the current version of node & express
35. which application we use node js.
36.Why node js is single threaded?
37.What is the role of single threads?
38.Disadvantages of node js?

github link for node js interview questions

1.https://github.com/learning-zone/nodejs-interview-questions

2.https://www.simplilearn.com/tutorials/nodejs-tutorial/nodejs-interview-questions

Pratical Questions

1. In the given array filter the alpha numeric values
Input----->var abc = ["78ty", "190ab", "ui", "U1", "Wtrl", "U2"];
OutPut----->["78ty", "190ab","U1","U2"]

2.In the given Object if name is Bhaskar then form_filters need to be false and if value is keelu then form_filters need to be true

```
let array = [
{
key: "fn2",
name: "bhaskar",
form_filters: true,
},
{
key: "ln2",
value: "keelu",
form_filters: false,
},
{
name: "Arun",
value: "Kumar",
form_filters: false,
},
];
```

3. Output should be the number of times the place occurs(dynamically)
EX-->punjab=1
mumbai=3
banglore=1

```
let objects = [
{
name: "yakshit",
place: "punjab",
},
{
name: "omkar",
place: "mumbai",
},
{
name: "gagan",
place: "banglore",
},
{
name: "kiran",
place: "mumbai",
},
{
name: "shivu",
place: "mumbai",
},

];
```

4.Hashing the password and object

5.['Orange', 'Mango', 'Apple', 'Orange', 'Banana', 'Mango', 'Kiwi']

- Return only duplicate values

OU: ['Orange', 'Mango']

6.arr = [8,10,21,29,2,9,7,3,19,16,6,13]

op->8, 10, 2, 16, 6];

7.There is a room with a door (closed) and three light bulbs. Outside the room, there are three switches, connected to the bulbs. You may manipulate the switches as you wish,
but once you open the door you can’t change them. Identify each switch with its bulb. All bulbs are in working condition.

Shashi sir important questions

1. Process.nextTick
2. setImmediate
3. Different node js timings we have
4. In mongoose what is populate
5. In aggreagations how to filter, sort
6. In AWS how to use S3 bucket
7. In socket.io
8. Mongodb