**1St- What is Node JS**

* Node is not a language.
* Node is a server-side environment.
* Node JS can connect with database.
* Code and syntax very similar to JavaScript but not exactly the same.
* Node is free and open-source.
* Node JS use chrome’s v8 engine to execute code.

**Why do we use Node**

* Node js is mostly used for API.
* So, we can connect the same database with web App, Mobile app.
* Node is easy to understand who knows JavaScript.
* Node is super-fast for APIs.
* With node and JavaScript, you can become full stack developer.

**JavaScript and node are the same?**

* JavaScript and node js code syntax is same.
* If you know JavaScript you can easily understand node.
* But both are not exactly the same.
* You cannot connect JavaScript with database.
* Node can connect with database.
* Node js run on the server side.
* JavaScript runs on the browser.

**Why you should learn Node?**

* If you’re JavaScript developer then you can easily learn node.
* You can become a full stack developer.

**2nd – What developers make with node js?**

* Developer make api with node js.
* We can connect server with client side.
* Node can make api for web, android and iOS etc.
* Node can also make websites.

**3rd- Installation and setup Node JS**

* Install node js from nodejs.org

**4th- First Script with node**

var a=20;

var b=30;

var c=40;

console.log(a+b+c);

to run the node file type **node index.js** in command prompt (index.js shows your file name).

**note: - The script running on the node or browser are not same.**

**5th – fundamentals we need for node js**

* JavaScript fundamentals for node js.
* Conditions, loops and arrays.
* The important function and variables from another file.
* Try filter function on an array of numbers.

**6th – core modules in node js**

* There are 2 types of core modules.
* Global modules and non-global modules.

//this is a global module because we don't need to import it.

console.log ("hello world!")

//this is a non-global module because we need to import it.

const fs = require("fs");

fs.writeFileSync("hello.txt","code with me")//this will make a new file as we run node commands and it will write code with me in txt file.

That was the first way for file system.

//this is the second way for file system.

var gd = require("fs").writeFileSync;

gd("abc.txt","some code written")//this will make a new file as we run node commands and it and you can give any name to variable.

That was the second way for file system.

**7th – Making basic server using node**

const http = require('http');

http.createServer((req,res)=>{

res.write("Hello this is Rahul Singh Chauhan!");

res.end();

}).listen(4500);

//to run the server type node fileName(node Create\_server.js) and then go to http://localhost4500 in your browser.

**8th – All about package.json file**

* What is package.json file?

The package. json file is **the heart of any Node project**. It records important metadata about a project which is required before publishing to NPM, and also defines functional attributes of a project that npm uses to install dependencies, run scripts, and identify the entry point to our package.

* How to make it?  
  just write **npm init** in your terminal.
* Check the file in detail.
* Installing external packages.  
  to install any external package in your node file just **type npm I packagename  
  (npm I nodemon).**
* node js is single threaded or multi-threaded…so it’s single threaded.

**9th – what if?**

* what if node modules folder deleted by any chance?  
  to recover it just type **npm init** in your terminal for already made modules.it will recover all your dependencies.
* Do not ever delete your **package.json** file otherwise you will not able to recover your dependencies or modules.
* Don’t ever push node Modules folder to your repository.
* To ignore node modules or any other files from pushing just make a file called **.gitignore**

And then write the folder or file name in gitignore file. Example given below.

/node\_modules

**10th – Nodemon|timesaving Package**

* What is nodemon?  
  Nodemon is **a popular tool that is used for the development of applications based on node.** **js**. It simply restarts the node application whenever it observes the changes in the file present in the working directory of your project.
* How to install it?  
  just write **npm I nodemon** in terminal.
* How to use it?  
  write like given below in package.json file where you will find “scripts” and then do it.
* "scripts": {
* "start": "nodemon index.js"
* },

And then type **npm start** in your terminal.

**Note: if you have installed nodemon globally then to run your file just type nodemon index.js (nodemon filename.js) in your terminal.**

**11th – Simple and basic API using Node JS**

const http = require('http');

http.createServer((req,res)=>{

res.write("Hello this is Rahul Singh Chauhan!");

res.end();

}).listen(4500);

//to run the server type node fileName(node Create\_server.js) and then go to http://localhost4500 in your browser.

**12th – getting input from command line**

**To create a file from command line**

To run a command just type **node 12.js apple.txt "i am a sweet fruit"**apple.txt is to create a file and “ I am a sweet fruit” is data to fill into new file.

//getting input from command line

//to create a file we need to import filesystem

const fs = require('fs');

const input = process.argv; //to take input from command line

fs.writeFileSync(input[2],input[3]); //first input is file name and second is data in file.

//now run the command.

**To remove a file from command line**

//getting input from command line

//to create a file we need to import filesystem

const fs = require('fs');

const input = process.argv;//to take input from command line

if(input[2]=='add')

{

    fs.writeFileSync(input[3],input[4])//first input is file name and second is data in file.

}

else if(input[2]=='remove')

{

    fs.unlinkSync(input[3])//to remove a file we need to type node 12.js remove(filename.txt) and then it will be removed from the filesystem.

}

else

{

    console.log("invalid input ")//To remove a file from command line just type node currentfile.js(12.js) and then remove (filename) orange or whatever you created as txt file.

}

//node 12.js add data.txt "this is a color and a fruit." this is a command to add a file and data into file

//node remove 12.js data.txt "this is a command to remove a file from command  line input"

To remove a file from command line just type node currentfile.js(12.js) and then remove (filename) orange or whatever you created as txt file.

**13th – Displaying file list from folder and creating files dynamically from the command line**

//creating files in loop from command line and displaying file list from command line.

const fs= require('fs');//to import the file system

const path=require('path');//to get a path of the folder where we want to create files..

const dirPath= path.join(\_\_dirname,'13files');//to get the path of the current directory.

console.log(dirPath)

for(i=0;i<5;i++)//to create files in loop from coommand line it will create files in given folder named 13files.

 {

   fs.writeFileSync(`${dirPath}/hello${i}.txt`,"some simple text in file")

//to make dynamically generated files just add ${i}.

 }

// fs.readdir(dirPath,(err,files)=>{//to get the files in console

//     files.forEach((item)=>{

//         console.warn("file name is : ",item)

//     });

// }

// )

**Note: you can’t access other folders in node you can only access current folder or current directory. Because when you work on the node it behaves as a server not as a local machine.  
  
14th – CRUD operations with file system**

**C = create**

**R = read**

**U = Update  
D = Delete  
and we will also perform renaming a file name.**

const fs = require("fs"); //to import filesystem into our file.

const path = require("path");// to select the path where you want to create new files.

const dirPath=path.join(\_\_dirname,"crud"); // to give the name of path

const filePath = `${dirPath}/apple.txt`;

//fs.writeFileSync(filePath,'This is a simple text file');// to "create" a file run command "node index.js"  in your terminal.

//to read a file "to read a file as normal text you have to write utf8.

// fs.readFile(   filePath, 'utf8',(err,item)=>{

//     console.log(item)

// });

// //to update a file

// fs.appendFile(filePath, 'and file name is apple.txt',(err)=>{//this will and this text into the same file.

//     if(!err) console.log("file is updated successfully")//if there is no error

// });

// //to rename the file name

// fs.rename(filePath,`${dirPath}/fruit.txt`,(err)=>{

//     if(!err) console.log("filename is updated successfully")

// });

//to delete the file

// fs.unlinkSync(`${dirPath}/fruit.txt`)

**What is buffer?**

Buffer is a temporary memory location. Whenever we create file or perform any operation on file system at that time nodejs needs some memory so that node can perform that operation.

**15th – Asynchronous basic in Node js**

**Synchronous VS Asynchronous**

Synchronous operations perform only one task at a time it means it performs only one task at a time.  
In asynchronous operations 2nd or upcoming task do not wait for the first task to get finished.

// console.log("I am an asynchronous task");

// setTimeout(()=>{

// console.log(" i am an asynchronous task but will get completed after 2 Ms after execution of those tasks whom don't have setTimeout");

// },2000);

// console.log("I am also an asynchronous task");

//drawback of asynchronous programming

let a=10;

let b=0;

setTimeout(() => {

    b=20

}, 2000);

console.log(a+b); //in that case the result will be 10 because here console.log doesn't have any setTimeout so it will take variables declared before setTimeout.

**Drawback of async programming is given in the example.**

**16th – How to handle asynchronous data?**

// //drawback of asynchronous programming

// let a=10;

// let b=0;

// setTimeout(() => {

//     b=20

// }, 2000);

// console.log(a+b); //in that case the result will be 10 because here console.log doesn't have any setTimeout so it will take variables as declared  before setTimeout.

//now we will solve that problem with promises...

let a=10;

let b=0;

let waitingdata= new Promise((resolve,reject)=> {

setTimeout(() => {

    resolve(20);

}, 2000);

})

waitingdata.then((data)=> {

b=data

console.log(a+b);

});

**17th – How Node js works?**

1. Call stack
2. Node API
3. CallBack queue   
   **there will be a Main () function by default in call stack .**

**What is call stack ?**

Call Stack is a mechanism to keep track of the function calls.

A call stack is a way for the JavaScript engine to keep track of its place in code that calls multiple functions. It has the information on what function is currently being run and what functions are invoked from within that function…

Also, the JavaScript engine uses a **call stack** to manage [execution contexts](https://www.javascripttutorial.net/javascript-execution-context/):

* Global execution context
* function execution contexts

The call stack works based on the LIFO principle i.e., last-in-first-out.

When you execute a script, the JavaScript engine creates a global execution context and pushes it on top of the call stack.

Whenever a function is called, the JavaScript engine creates a function execution context for the function, pushes it on top of the call stack, and starts executing the function.

If a function calls another function, the JavaScript engine creates a new function execution context for the function that is being called and pushes it on top of the call stack.

When the current function completes, the JavaScript engine pops it off the call stack and resumes the execution where it left off.

The script will stop when the call stack is empty.

**JavaScript call-back example**

function add(a, b) {

  return a + b;

}

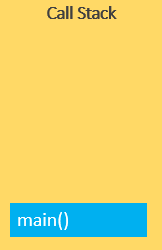
function average(a, b) {

  return add(a, b) / 2;

}

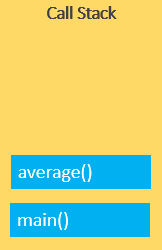
let x = average(10, 20);

When the JavaScript engine executes this script, it places the global execution context (denoted by main() or global() function on the call stack.



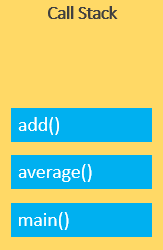
The global execution context enters the creation phase and moves to the execution phase.

The JavaScript engine executes the call to the average(10, 20) function and creates a function execution context for the average() function and pushes it on top of the call stack:

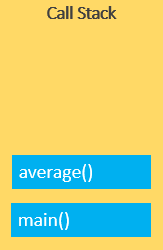


The JavaScript engine starts executing the average() since because the average() function is on the top of the call stack.

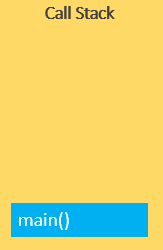
The average() function calls add() function. At this point, the JavaScript engine creates another function execution context for the add() function and places it on the top of the call stack:



JavaScript engine executes the add() function and pops it off the call stack:



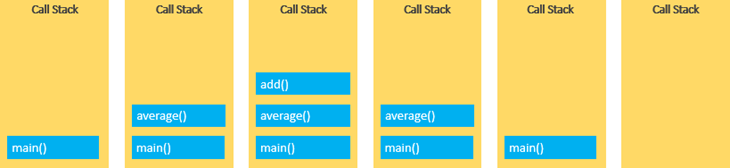
At this point, the average() function is on the top of the call stack, JavaScript engine executes and pops it off the call stack.



Now, the call stack is empty so the script stops executing:



The following picture illustrates the overall status of the Call Stack in all steps:



**Stack overflows**

The call stack has a fixed size, depending on the implementation of the host environment, either the web browser or Node.js.

If the number of the execution contexts exceeds the size of the stack, a stack overflow error will occur.

For example, when you execute a [recursive function](https://www.javascripttutorial.net/javascript-recursive-function/) that has no exit condition, the JavaScript engine will issue a stack overflow error:

function fn() {

    fn();

}

fn(); // stack overflow

**Asynchronous JavaScript**

JavaScript is the single-threaded programming language. This means that the JavaScript engine has only one call stack. Therefore, it only can do one thing at a time.

When executing a script, the JavaScript engine executes code from top to bottom, line by line. In other words, it is synchronous.

Asynchronous means the JavaScript engine can execute other tasks while waiting for another task to be completed. For example, the JavaScript engine can:

* Request for data from a remote server.
* Display a spinner
* When the data is available, display it on the webpage.

To do this, the JavaScript engine uses an event loop

* JavaScript engine uses a call stack to manage execution contexts.
* The call stack uses the stack data structure that works based on the LIFO (last-in-first-out) principle.

**What is Node-API ?**

Node-API is **a toolkit introduced in Node 8.0.** **0 that acts as an intermediary between C/C++ code and the Node JavaScript engine**. It permits C/C++ code to access, create, and manipulate JavaScript objects as if they were created by JavaScript code. Node-API is built into Node versions 8.0.

**What is Call-back queue ?**

A call-back queue is **a queue of tasks that are executed after the current task**. The call-back queue is handled by the JavaScript engine after it has executed all tasks in the microtask queue.

**18th –** **introduction to Express Js**

Express is a node js web application framework that provides broad features for building web and mobile applications. It is used **to build a single page, multipage, and hybrid web application**. It's a layer built on the top of the Node js that helps manage servers and routes.

**Benefits of Express Js**

* Express was created to make APIs and web applications with ease,
* It saves a lot of coding time almost by half and still makes web and
* mobile applications are efficient.
* Another reason for using express is that it is written in JavaScript as JavaScript is an easy language even if you don't have a previous
* knowledge of any language. Express lets so many new developers enter the field of web development.

The reason behind creating an express framework for node js is:

* Time-efficient
* Fast
* Economical
* Easy to learn
* Asynchronous

**How to Install Express ?**

To install express js in your project type **npm I express** in terminal.

const express = require("express"); //to import the express module into our project.

const app = express();

//method for adding routes

app.get("/", (req, res) => {

  res.send("Welcome , This is home page");

});

app.get("/about", (req, res) => {

  res.send("Welcome , This is about page");

});

app.get("/help", (req, res) => {

  res.send("Welcome , This is help page");

});

app.listen(5000); // to start the server on port 5000

**19th – Routing params || request and response**

const express = require('express'); //to import the express module into our project.

const app = express();

//method for adding routes

app.get('/', (req, res) => {

    res.send("Welcome , This is home page");

});

app.get('/about', (req, res) => {

    res.send("Welcome , This is about page");

});

app.get('/help', (req, res) => {

    res.send("Welcome , This is help page");

});

app.listen(5000);// to start the server on port 5000

**20th – Render html and json**

const express = require("express"); //to import the express module into our project.

const app = express();

//method for adding routes

app.get("/", (req, res) => {

  res.send(`

    <h1>Welcome , This is home page</h1> <a href="/about"> Go to the homepage</a>`); //to rendor html on page load and to go to another page.

});

app.get("/about", (req, res) => {

  res.send(`

    <input type="text" placeholder="user name" "value=${req.query.name}"/>

    <button>click me</button>

    `); //to get the value of the user name.

});

app.get("/help", (req, res) => {

  //to rendor json on page load.

  res.send({

    name: "anil",

    email: "rahulsinghchauhanrj@gmail.com",

  });

});

app.listen(5000); // to start the server on port 5000

**21st – Loading full html page using node js**

const express = require('express');

const path = require('path');

const app = express();

const publicPath=path.join(\_\_dirname,'public')

app.use(express.static(publicPath));

app.listen(5000);

**use of path module**

it helps us to access the required path of folder we want to access.

**Static method**

It helps us to load static content or static pages using node js.

**22nd – Remove extension from url and showing 404 page if URL is not valid**

Removing extension means that when we load our html page using node so there should not be a .html written in address of our document.

const express = require("express");

const path = require("path");

const app = express();

const publicPath = path.join(\_\_dirname, "public");

// app.use(express.static(publicPath));

app.get("", (\_, resp) => {

  resp.sendFile(`${publicPath}/index.html`); //TO REMOVE THE .html extension from our document address

});

app.get("/contact", (\_, resp) => {

  resp.sendFile(`${publicPath}/about.html`);

});

app.get("/help", (\_, resp) => {

  resp.sendFile(`${publicPath}/help.html`);

});

app.get("\*", (\_, resp) => {

  resp.sendFile(`${publicPath}/nopage.html`);

}); //here we are using \* so that when ever user types a url which is not valid so user will be redirected to the error or 404 pages as we given in path

app.listen(5000);

to remove the extension from URL use **resp.sendFile** method .

**23rd – EJS Template Engine**

EJS (Embedded JavaScript Templating) is **one of the most popular template engines for JavaScript**. As the name suggests, it lets us embed JavaScript code in a template language that is then used to generate HTML , and it is used to make dynamic pages and this is a NPM package.

const express = require("express");

const path = require("path");

const app = express();

const publicPath = path.join(\_\_dirname, "public");

app.set("view engine", "ejs"); //to tell nodejs that we're using template engine the type of engine is view engine and the name of the engine is EJS. and whenever we use view engine or ejs so by default we have to make a views folder  and then we can make a document inside that folder and we have to make sure that we are using .ejs extension.

app.get("", (\_, resp) => {

  resp.sendFile(`${publicPath}/index.html`);

});

//to render the .ejs extension file.

app.get("/profile", (\_, resp) => {

  const user = {

    name: "Peter",

    email: "peter@test.com",

    country: "USA",

  };

  resp.render("profile", { user });

});

app.get("/contact", (\_, resp) => {

  resp.sendFile(`${publicPath}/about.html`);

});

app.get("/help", (\_, resp) => {

  resp.sendFile(`${publicPath}/help.html`);

});

app.get("\*", (\_, resp) => {

  resp.sendFile(`${publicPath}/nopage.html`);

});

app.listen(5000);

To tell the node js that we are using template engine

app.set("view engine", "ejs"); //to tell nodejs that we're using template engine the type of engine is view engine and the name of the engine is EJS. and whenever we use view engine or or ejs so by default we have to make a views folder  and then we can make a document inside that folder and we have to make sure that we are using .ejs extension.

To render the .ejs file

//to render the .ejs extension file.

app.get("/profile", (\_, resp) => {

  const user = {

    name: "Peter",

    email: "peter@test.com",

    country: "USA",

  };

  resp.render("profile", { user });

});

The .ejs file

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <title>Profile Page</title>

  </head>

  <body>

    <h1>Welcome, <%= user.name %></h1>

    <!-- to load the data from index.js file into the profile page.ejs we use <%= user.name %> (or whatever value you want) -->

    <h3><%= user.email %></h3>

    <h3><%= user.country %></h3>

  </body>

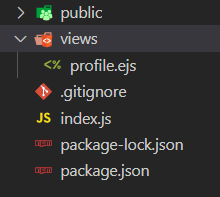
</html>

Note :-

<!-- to load the data from index.js file into the profile.ejs we use <%= user.name %> (or whatever value you want) -->

Note :-

whenever we use view engine or ejs so by default we have to make a views folder  and then we can make a document inside that folder and we have to make sure that we are using .ejs extension.

The folder structure will look like this -  


**24th – Dynamic page with EJS**

**To load the data from .ejs file using loop**

<!-- to get data using loop syntax given below -->

    <ul>

        <% user.skills.forEach((item)=>{%>

        <li><%= item %></li>

        <% })%>

    </ul>

**To load the common .ejs file into your ejs file.**

 <!-- to load a common file into your.ejx file, syntax is given below , where common is a folder name and header is a file name which we want to load commonly-->

    <!-- and note that you have you use hyphn (-) not equal to sign != to load common file -->

    <%- include('common/header'); %>

**25th – Express js middleware**

**What is Middleware ?**

Middleware is the function , which is used only with routs, with the help of middleware we can access request and response , or we can say that we can modify the request and response .

**Types of middleware**

* Application based middleware
* Route level middleware
* Error handling middleware
* Built in middleware
* Third party middleware

const express = require("express");

const app = express();

//adding a middleware or creating a new middleware.and note that middleware has 3 parameters.

const reqFilter = (req, resp, next) => {

  if (!req.query.age) {

    resp.send("Please provide your age");

  } else if (req.query.age < 18) {

    resp.send("You are under aged");

  } else {

    next(); //we have to call next() if we want to move further.

  }

};

app.use(reqFilter); // to use the middleware function.

//adding routs

app.get("/", (res, resp) => {

  resp.send("Welcome to Home page");

});

app.get("/users", (res, resp) => {

  resp.send("Welcome to Users page");

});

app.listen(5000);

**26th – Route level middleware**

const express = require("express");

const reqFilter = require("./middleware"); //to import the middleware functions from another file

const app = express();

const route = express.Router(); // importing routers

// app.use(reqFilter);// it will apply middleware function on all routs

route.use(reqFilter); //to use the middleware functions and to apply the routes

app.get("/", (res, resp) => {

  resp.send("Welcome to Home page");

});

app.get("/users", (res, resp) => {

  resp.send("Welcome to Users page");

});

route.get("/about", (res, resp) => {

  //use route.get to apply the route at this page

  resp.send("Welcome to About page");

});

route.get("/contact", (res, resp) => {

  //use route.get to apply the route at this page

  resp.send("Welcome to contact page");

});

app.use("/", route); // to use route instance.

app.listen(5000);

**Middleware is made in separate file**

//to export the middleware from this file to another file

//and this is a middleware function

module.exports = (req, resp, next) => {

    if (!req.query.age) {

        resp.send("Please provide your age")

    }

    else if (req.query.age<18) {

        resp.send("You are under aged")

    }

    else {

        next();

    }

}

**Note :-** Route level Middleware gets applied on only given routes and application-level middleware gets applied on all routs.

**27th – Installing mongo db**

Install mongo DB from here :- [**https://www.mongodb.com/try/download/community**](https://www.mongodb.com/try/download/community)

**28th – Mongo db Basics**

**What is mongoDB ?**

* MongoDB is a NoSQL database.
* The data is stored as collection or document.
* Collection don’t have any row and columns.
* Data is stored in the form of object.

**29th – Mongo crud**

Try crud operations using mongo DB compass.

**30th – Connecting node js with mongo db**

const {MongoClient} = require('mongodb')//to import mongodb instance.

const url= 'mongodb://localhost:27017';//giving the address of the database

const databaseName='e-comm'//givind the name of the database from where we are fetching the data

const client= new MongoClient(url);//to fetch the data from mongodb

async function getData()

{

    let result = await client.connect();//to connect with client

    db= result.db(databaseName);

    collection = db.collection('products');//from which collection we want to fetch data

    let data = await collection.find({}).toArray();//to get data from collection

    console.log(data)

}

getData();

**31st – Connecting node js with mongo db**

**Separate file to connect with database**

const {MongoClient} = require('mongodb')

const url= 'mongodb://localhost:27017';

const databaseName='e-comm'

const client= new MongoClient(url);

async function dbConnect()

{

    let result = await client.connect();

    db= result.db(databaseName);

    return db.collection('products');

}

module.exports= dbConnect;

**To read the data from the database**

const dbConnect= require('./mongodb');

dbConnect().then((resp)=>{

resp.find({name:'nord'}).toArray().then((data)=>{

console.log(data)

})

})

const main=async ()=>{

   let data = await dbConnect();

   data = await data.find({name:'nord'}).toArray();

   console.log(data)

}

main()

**32nd – To insert the data into mongodb**

const dbConnect = require("./mongodb");

const insertData = async () => {

  let data = await dbConnect();

  let result = await data.insert([

    { name: "max 5", brand: "micromax", price: 420, category: "mobile" },

    { name: "max 6", brand: "micromax", price: 520, category: "mobile" },

    { name: "max 7", brand: "micromax", price: 620, category: "mobile" },

  ]);

  if (result.acknowledged) {

    console.warn("data is inserted");

  }

};

insertData();

**33rd – Updating the data into mongodb**

const dbConnect = require("./mongodb");

const updateData = async () => {

  let data = await dbConnect();

  let result = await data.update(

    //to update single data write update one for multiple updates write only update as written in code.

    { name: "max 5" },//to find which product we want to update

    {

      $set: { name: "max pro 5", price: 1000 },// to update the value of the data.

    }

  );

  console.log(result);

};

updateData();

**34th – Deleting the data from mongodb**

const dbConnect = require("./mongodb");

const deleteData = async () => {

  let data = await dbConnect();

  let result = await data.deleteMany({ name: "max 7" }); //to delete single data write deleteone or more than one write deleteone

  console.log(result);

};

deleteData();

**35th – Basic get API with mongo DB**

const express = require('express');

const dbConnect = require('./mongodb')

const app = express();

app.get('/',async(req,resp)=>{

let data = await dbConnect();

data = await data.find().toArray();

resp.send(data);

});

app.listen(5000)

**36th – Post API | insert data in mongoDB**

const dbConnect = require("./mongodb");

const express = require("express");

const app = express();

app.use(express.json());

//to get the data or to read the data from the database

app.get("/", async (res, resp) => {

  let data = await dbConnect();

  data = await data.find().toArray();

  resp.send(data);

});

//to send the data or to create the data into the database.

app.post("/", async (req, resp) => {

  let data = await dbConnect();

  let result = await data.insert(req.body);

  resp.send(result);

});

app.listen(5000);

**37th – Put API | updating data in mongoDB**

const dbConnect = require("./mongodb");

const express = require("express");

const app = express();

app.use(express.json());

//to get the data or to read the data from the database

app.get("/", async (res, resp) => {

  let data = await dbConnect();

  data = await data.find().toArray();

  resp.send(data);

});

//to send the data or to create the data into the database.

app.post("/", async (req, resp) => {

  let data = await dbConnect();

  let result = await data.insert(req.body);

  resp.send(result);

});

//to update the data

app.put("/:name", async (req, resp) => {

  console.log(req.body);

  const data = await dbConnect();

  let result = data.updateOne({ name: req.params.name }, { $set: req.body });

  resp.send({ status: "updated" });

});

app.listen(5000);

**38th – Delete API | Deleting data in mongoDB**

const dbConnect = require("./mongodb");

const express = require("express");

const mongodb = require("mongodb");

const app = express();

app.use(express.json());

app.get("/", async (res, resp) => {

  let data = await dbConnect();

  data = await data.find().toArray();

  resp.send(data);

});

app.post("/", async (req, resp) => {

  let data = await dbConnect();

  let result = await data.insert(req.body);

  resp.send(result);

});

app.put("/:name", async (req, resp) => {

  const data = await dbConnect();

  let result = data.updateOne({ name: req.params.name }, { $set: req.body });

  resp.send({ status: "updated" });

});

//to delete the existing data from the database.

app.delete("/:id", async (req, resp) => {

  console.log(req.params.id);

  const data = await dbConnect();

  const result = await data.deleteOne({

    \_id: new mongodb.ObjectId(req.params.id),

  });

  resp.send(result);

});

app.listen(5000);

**39th – Mongoose with node**

To insert this package into your project just type **npm I mongoose** in your terminal.

**Schema**

Schemas validates the fields of database ..**now the question is how it validates the fields of our database?** So in our database we have to define that which fields we want to enter in our database.  
when you define your desired fields in schema after that you can’t enter any other fields into your data.

For example if you have created a database and in that database you want to add **name as “string” mobile number as number 8602399308 ..** if you have defined that feels in schema then you can’t enter any other fields than that. Let’s think if you have added 5 fields in schema then you can’t add any other field other than that five fields.

**And you can define type validation of your data fields .**

type validation means that you want to store data as string “ ” or as number “000” or maybe as Boolean “true/false”.

**What does model do ?**

It connects node js and mongo db by using schemas.

const mongoose = require('mongoose')

mongoose.set('strictQuery', false);

const main = async () => {

    await mongoose.connect("mongodb://localhost:27017/e-comm")

    const productSchema = new mongoose.Schema({

        name: String

    });

    const productsModel = mongoose.model('Products', productSchema)

    let data=new productsModel ({name:"m8"})

    let result = await data.save();

}

main()

**40th – Crud with Mongoose**

const mongoose = require("mongoose");

mongoose.connect("mongodb://localhost:27017/e-comm");

const productSchema = new mongoose.Schema({

  name: String,

  price: Number,

  brand: String,

  category: String,

});

const saveInDB = async () => {

  const Product = mongoose.model("products", productSchema);

  let data = new Product({

    name: "max 100",

    price: 200,

    brand: "maxx",

    category: "Mobile",

  });

  const result = await data.save();

  console.log(result);

};

const updateInDB = async () => {

  const Product = mongoose.model("products", productSchema);

  let data = await Product.updateOne(

    { name: "max 6" },

    {

      $set: { price: 550, name: "max pro 6" },

    }

  );

  console.log(data);

};

const deleteInDB = async () => {

  const Product = mongoose.model("products", productSchema);

  let data = await Product.deleteOne({ name: "max 100" });

  console.log(data);

};

const findInDB = async () => {

  const Product = mongoose.model("products", productSchema);

  let data = await Product.find({ name: "max pro 611" });

  console.log(data);

};

findInDB();

**41st – Post(Insert) API with Mongoose**

const express = require('express');

require("./config");

const Product = require('./product');

const app = express();

app.use(express.json());

app.post("/create", async (req, resp) => {

    let data = new Product(req.body);

    const result = await data.save();

    resp.send(result);

});

app.listen(5000)

**42nd – Post,GET, Delete and Put API with Mongoose**

const express = require("express");

require("./config");

const Product = require("./product");

const app = express();

app.use(express.json());

app.post("/create", async (req, resp) => {

  let data = new Product(req.body);

  const result = await data.save();

  resp.send(result);

});

app.get("/list", async (req, resp) => {

  let data = await Product.find();

  resp.send(data);

});

app.delete("/delete/:\_id", async (req, resp) => {

  console.log(req.params);

  let data = await Product.deleteOne(req.params);

  resp.send(data);

});

app.put("/update/:\_id", async (req, resp) => {

  console.log(req.params);

  let data = await Product.updateOne(req.params, { $set: req.body });

  resp.send(data);

});

app.listen(5000);

**43rd – Search API with multiple fields**

const express = require("express");

require("./config");

const Product = require("./product");

const app = express();

app.use(express.json());

app.get("/search/:key", async (req, resp) => {

  let data = await Product.find({

    $or: [

      { name: { $regex: req.params.key } },

      { brand: { $regex: req.params.key } },

    ],

  });

  resp.send(data);

});

app.listen(5000);

**44th – Uploading files using node js**

First of all we have to download a package called multer from npm using command **npm I multer.**

const express = require("express");

const multer = require("multer");

const app = express();

const upload = multer({

  storage: multer.diskStorage({

    destination: function (req, file, cb) {

      cb(null, "uploads");

    },

    filename: function (req, file, cb) {

      cb(null, file.fieldname + "-" + Date.now() + ".jpg");

    },

  }),

}).single("file\_name");

app.post("/upload", upload, (req, resp) => {

  resp.send("file upload");

});

app.listen(5000);

**45th – OS modules in node js**

The node:os module **provides operating system-related utility methods and properties**. It can be accessed using: const os = require('node:os'); .

const os = require("os");

// console.log(os.arch());

// console.log(os.freemem()/(1024\*1024\*1024));

// console.log(os.totalmem()/(1024\*1024\*1024));

// console.log(os.hostname());

// console.log(os.platform());

console.log(os.userInfo());

**46th – Events and event Emitter in Node js**

const express = require("express");

const EventEmitter = require("events"); // event emitter is basically a class that’s why the first alphabet of its is Capital.. classes starts with capital always.

app = express();

const event = new EventEmitter(); //its an object , when ever we create a new object we have to use "new" keyword for it.

let count = 0;

//to handle an event.

event.on("countAPI", () => {

  count++;

  console.log("event called", count);

});

app.get("/", (req, resp) => {

  resp.send("api called");

  event.emit("countAPI"); //here we are generating an event.

});

app.get("/serch", (req, resp) => {

  resp.send("Search api called");

  event.emit("countAPI");

});

app.get("/update", (req, resp) => {

  resp.send("Update api called");

  event.emit("countAPI");

});

app.listen(5000);

**47th – REPL**

**Read-Eval-Print-Loop**

REPL is a command line tool of node js by which we can execute codes of node js and JavaScript.

**48th – Connect node js with MySQL(npm I mysql)**

const mysql= require("mysql");

const con= mysql.createConnection({

  host:"localhost",

  user:"root",

  password:"",

  database:"test"

});

// con.connect((err)=>{

//   if(err)

//   {

//     console.warn("not connect")

//   }else{

//     console.warn("connected!!!")

//   }

// })

con.query("select \* from users",(err,result)=>{

  if(err)

  {

    console.warn("some error")

  }

  else{

    console.warn(result)

  }

})

**49th – Get API with MySQL**

**Configuration file**

const mysql = require("mysql");

const con = mysql.createConnection({

  host: "localhost",

  user: "root",

  password: "",

  database: "test",

});

con.connect((err) => {

  if (err) {

    console.warn("error in connection");

  }

});

module.exports = con;

**main file**

const express = require("express");

const con = require("./config");

const app = express();

app.get("/", (req, resp) => {

  con.query("select \* from users", (err, result) => {

    if (err) { resp.send("error in api") }

    else { resp.send(result) }

  })

});

app.listen("5000")

**50th – Post API with MySQL**

const express = require("express");

const con = require("./config");

const app = express();

app.use(express.json());

app.get("/", (req, resp) => {

  con.query("select \* from users", (err, result) => {

    if (err) { resp.send("error in api") }

    else { resp.send(result) }

  })

});

app.post("/",(req,resp)=>{

  const data=req.body;

  con.query("INSERT INTO users SET?",data,(error,results,fields)=>{

    if(error) throw error;

    resp.send(results)

  })

});

app.listen("5000")

**51st – Put API with MySQL**

const express = require("express");

const con = require("./config");

const app = express();

app.use(express.json());

app.get("/", (req, resp) => {

  con.query("select \* from users", (err, result) => {

    if (err) {

      resp.send("error in api");

    } else {

      resp.send(result);

    }

  });

});

app.post("/", (req, resp) => {

  const data = req.body;

  con.query("INSERT INTO users SET?", data, (error, results, fields) => {

    if (error) throw error;

    resp.send(results);

  });

});

app.put("/:id", (req, resp) => {

  const data = [

    req.body.name,

    req.body.password,

    req.body.user\_type,

    req.params.id,

  ];

  con.query(

    "UPDATE users SET name = ?, password = ?, user\_type = ? WHERE id = ?",

    data,

    (error, results, fields) => {

      if (error) throw error;

      resp.send(results);

    }

  );

});

app.listen("5000");

**52nd – Delete API with MySQL**

const express = require("express");

const con = require("./config");

const app = express();

app.use(express.json());

app.get("/", (req, resp) => {

  con.query("select \* from users", (err, result) => {

    if (err) {

      resp.send("error in api");

    } else {

      resp.send(result);

    }

  });

});

app.post("/", (req, resp) => {

  const data = req.body;

  con.query("INSERT INTO users SET?", data, (error, results, fields) => {

    if (error) throw error;

    resp.send(results);

  });

});

app.put("/:id", (req, resp) => {

  const data = [

    req.body.name,

    req.body.password,

    req.body.user\_type,

    req.params.id,

  ];

  con.query(

    "UPDATE users SET name = ?, password = ?, user\_type = ? WHERE id = ?",

    data,

    (error, results, fields) => {

      if (error) throw error;

      resp.send(results);

    }

  );

});

app.delete("/:id", (req, resp) => {

  con.query("DELETE FROM users WHERE id = " + req.params.id, (error, result));

  if (error) throw error;

  resp.send(results);

});

app.listen("5000");