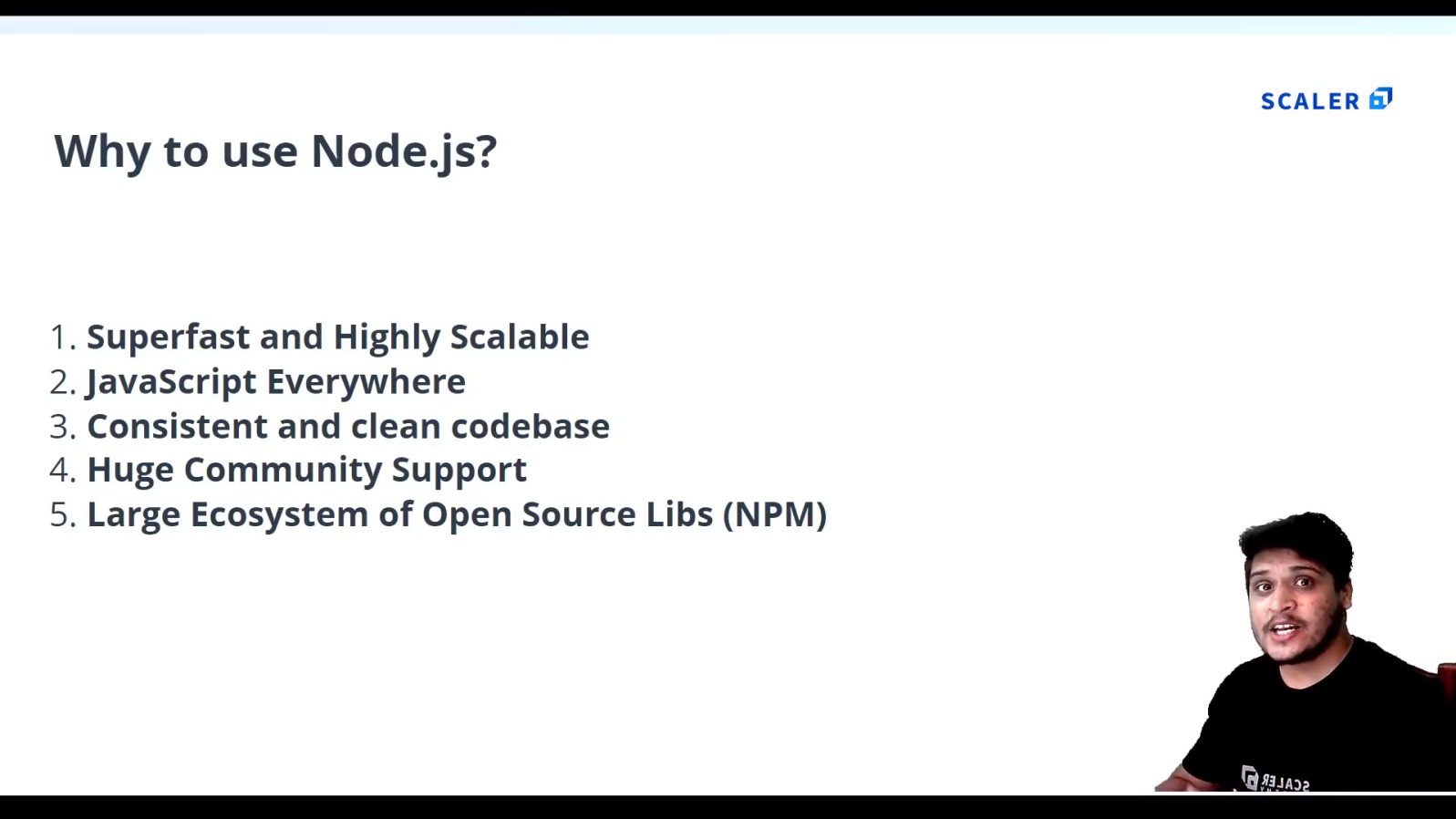
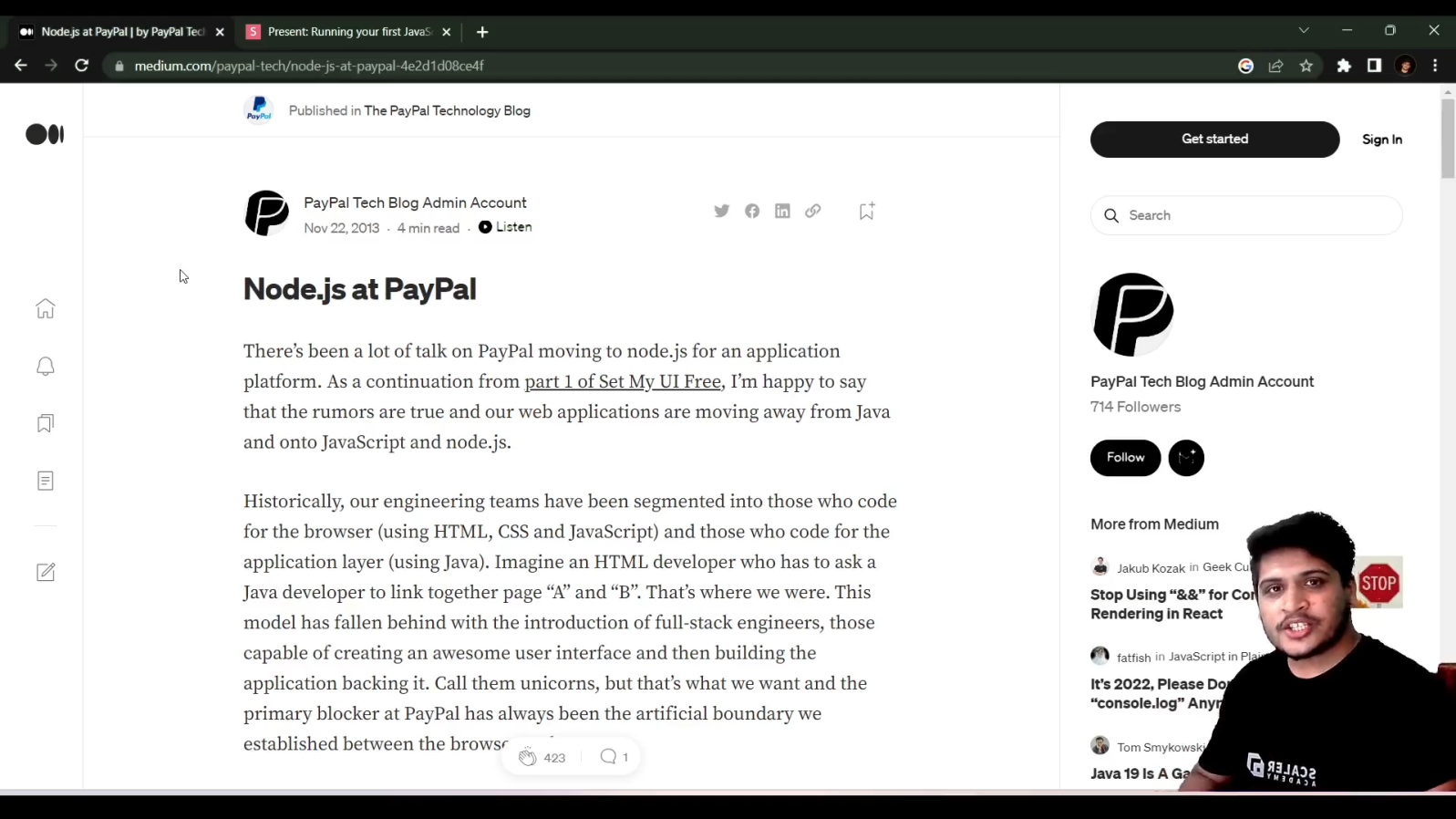
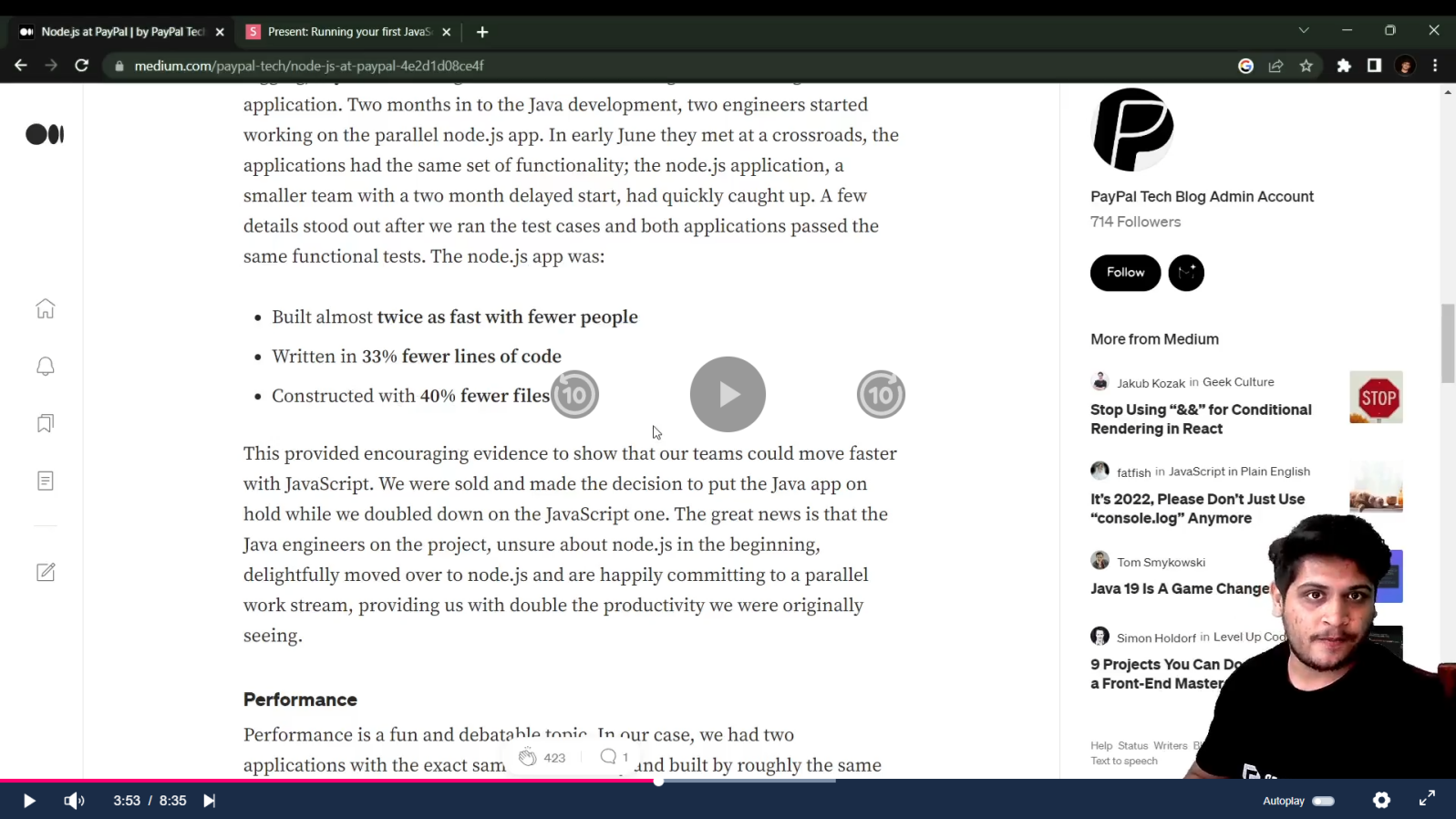
**NODE JS - SCALAR**

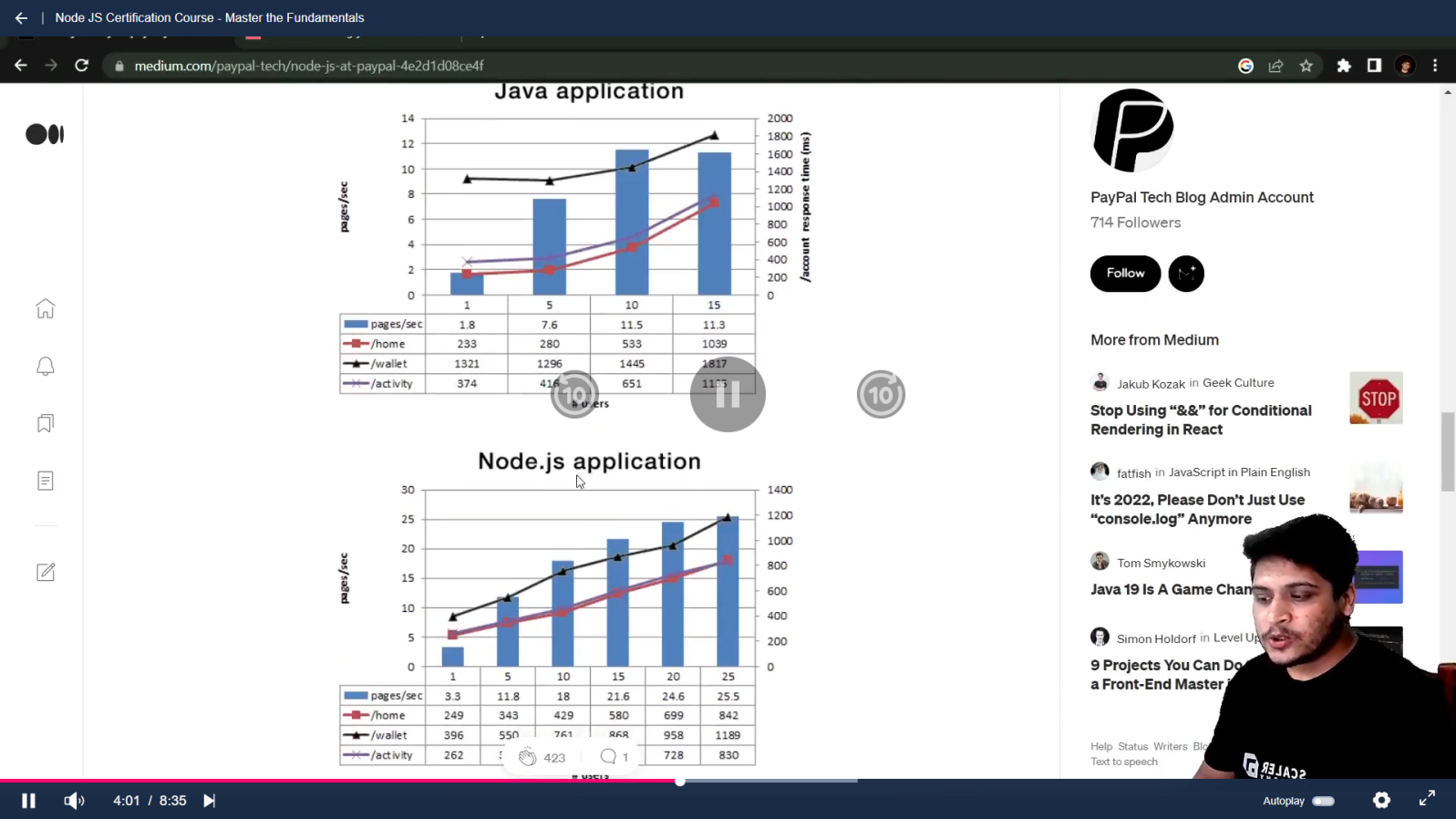
1. ***Introduction* :**

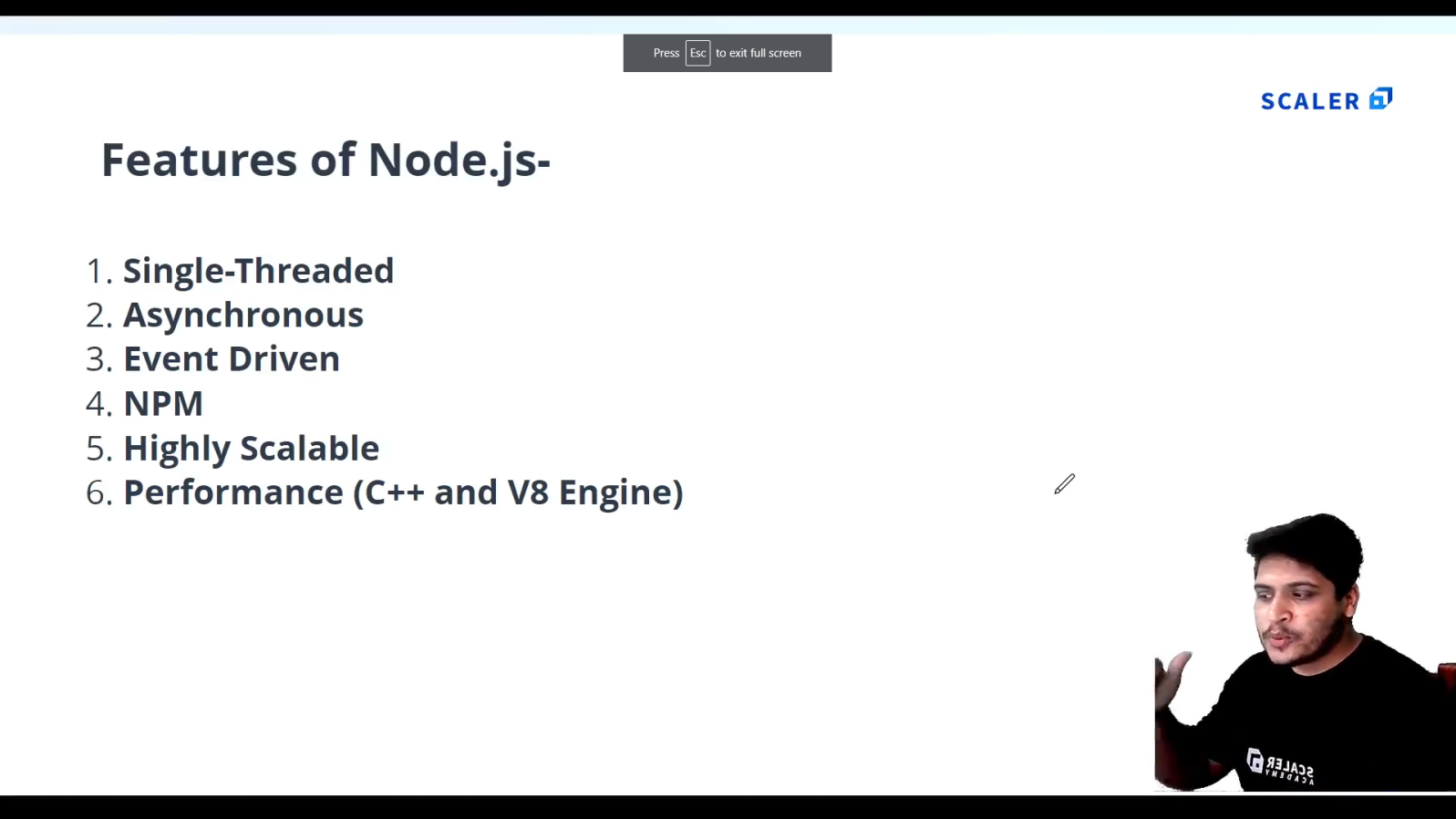
****

****

****

****

****

****

1. ***Getting Started with Node JS*:** 
   1. **Install Node.js and VScode.**
   2. **First Node program :**
      1. **Create a js file with a function:p1\_test.js**

console.log('Hello World');

function sayHello(){

    console.log('Hello from anshad');

}

sayHello();

* + 1. **To Execute type “node name\_of\_program.js” : node p1\_test.js**
  1. **There is no “window” object in Node js instead we have “global” object.**

console.log(global);

PS E:\MCA\COURSES\NODE JS\nodejs\_certification@scalar> node p1\_test.js

<ref \*1> Object [global] {

  global: [Circular \*1],

  clearImmediate: [Function: clearImmediate],

  setImmediate: [Function: setImmediate] {

    [Symbol(nodejs.util.promisify.custom)]: [Getter]

  },

  clearInterval: [Function: clearInterval],

  clearTimeout: [Function: clearTimeout],

  setInterval: [Function: setInterval],

  setTimeout: [Function: setTimeout] {

    [Symbol(nodejs.util.promisify.custom)]: [Getter]

  },

  queueMicrotask: [Function: queueMicrotask],

  structuredClone: [Function: structuredClone],

  atob: [Getter/Setter],

  btoa: [Getter/Setter],

  performance: [Getter/Setter],

  fetch: [Function: value],

  crypto: [Getter]

}

1. ***Node Module system*:**
   1. **The “global” Object :**

//GLOBAL OBJECT

//are built-in objects that are part of the JavaScript and can be used directly in the application without importing any particular module.

let name = 'Anshad';

console.log(global.name);//undefined

* 1. **Modules and Modularity :**

1. **Create ‘calculator.js’ file :**

//CALCULATOR :

function add(a , b ){

    console.log( a + b );

}

function sub(a , b ){

    console.log( a - b );

}

function mul(a , b ){

    console.log( a \* b );

}

function div(a , b ){

    console.log( a / b );

}

//Exporting functions:

module.exports = {

    addition : add ,

    subtraction : sub,

    multiplication : mul,

    division : div

}

1. **Create main file ‘modularity.js’ file :**

//1.create a seperate file 'calculator.js'.

//Modularity lets us use those contents in calculator.js in this file.

const calculator = require('./calculator');//import calculator.js

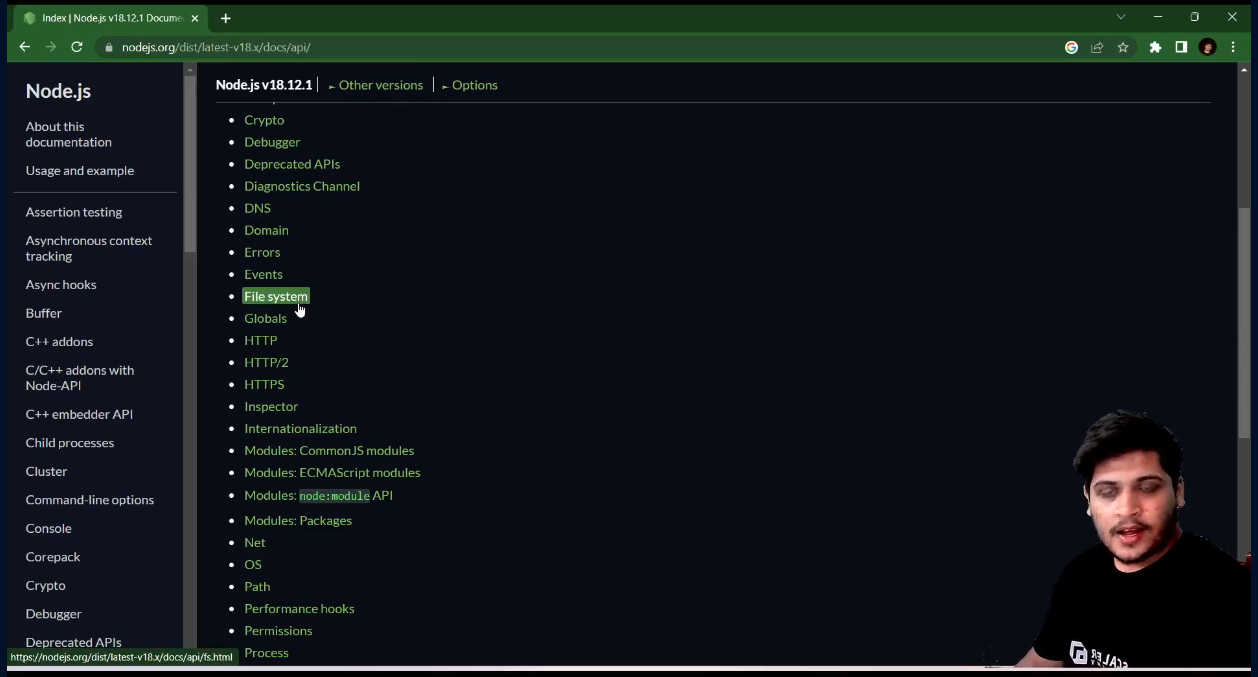
calculator.addition(3 , 4);//Calls the function in calculator.js by passing values to that.

calculator.subtraction( 5 ,2);

calculator.multiplication( 3 , 4);

calculator.division(10 , 2);

* 1. **Introduction to Node Modules :**



* 1. **Child Process Module :**

//Child process is a node module used to create sub process within a script .

//You can perform different tasks with your script by just using some methods.

const cp = require('child\_process');//Importing child-process module.

cp.execSync('calc') ; //Opens a calculator.['calc' is short code for opening calculator.]

cp.execSync('start chrome'); //opens chrome browser.

cp.execSync('start chrome https://www.google.com/');//opens google page directly

console.log('Output '+cp.execSync('node p1\_test.js'));//prints output from a specified File.

* 1. **OS Module :**

|  |  |
| --- | --- |
| **Code** | **Output** |
| //OS Module :(to get the information of your current system)  const os = require('os');//Import OS module  console.log(os.arch());//Displays Architecture of your OS (64 or 32)  console.log(os.platform());//Displays the platform of your OS  console.log(os.networkInterfaces());//Displays the network informations of your Os  console.log(os.cpus());//Displays the cpu's details (graphics,processor,configurations)  console.log('Total Memory : ',os.totalmem());//Displays the total memory .  console.log('Free Memory : ',os.freemem());//Displays the free memory. | PS E:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_3> node .\osModule.js  x64  win32  {  Ethernet: [  {  address: '192.168.1.12',  netmask: '255.255.255.0',  family: 'IPv4',  mac: 'f8:0d:ac:29:a8:91',  internal: false,  cidr: '192.168.1.12/24'  }  ],  'Loopback Pseudo-Interface 1': [  {  address: '::1',  netmask: 'ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff',  family: 'IPv6',  mac: '00:00:00:00:00:00',  internal: true,  cidr: '::1/128',  scopeid: 0  },  {  address: '127.0.0.1',  netmask: '255.0.0.0',  family: 'IPv4',  mac: '00:00:00:00:00:00',  internal: true,  cidr: '127.0.0.1/8'  }  ]  }  [  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 101765, nice: 0, sys: 115328, idle: 1025421, irq: 13640 }  },  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 73468, nice: 0, sys: 51812, idle: 1117078, irq: 1656 }  },  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 94734, nice: 0, sys: 97968, idle: 1049656, irq: 1468 }  },  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 81906, nice: 0, sys: 47234, idle: 1113218, irq: 1312 }  },  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 58578, nice: 0, sys: 38953, idle: 1144828, irq: 1046 }  },  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 59406, nice: 0, sys: 34625, idle: 1148328, irq: 859 }  },  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 55937, nice: 0, sys: 33125, idle: 1153296, irq: 843 }  },  {  model: 'AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx ',  speed: 2096,  times: { user: 61718, nice: 0, sys: 31375, idle: 1149265, irq: 875 }  }  ]  Total Memory : 14901878784  Free Memory : 6341722112 |

* 1. **Path Module :**

//PATH Module :(We need a File to work with path Eg:f1.txt )

//NOTE : USE DOUBLE SLASHES in path.

const path = require('path');//import path Module.

//1.extname : To know the extension of a File in a Path:

let ext = path.extname('E:\\MCA\\COURSES\\NODE\_JS\\nodejs\_certification@scalar\\Module\_3\\f1.txt');

//2.basename : To know the extension of a File in a Path:

let base = path.basename('E:\\MCA\COURSES\\NODE\_JS\\nodejs\_certification@scalar\\Module\_3\\f1.txt');

console.log(ext);

console.log(base);

//3.To display the path of current file (this file)

console.log(\_\_filename);

//4.To display the directory of current file (this file)

console.log(\_\_dirname);

/\*

output :

.txt

f1.txt

e:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_3\pathModule.js

e:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_3

\*/

* 1. **FS Module with Files [readFileSync(),writeFileSync(),appendFileSync(),unlinkSync() ].**

//FS Module with files :(TO handle Files)

const fs = require('fs');//import fs.

//Create f1.txt ,f2.txt and f3.txt

//1.Reading a file [readFileSync()]:

let fileContent = fs.readFileSync('f1.txt');

console.log('Data of FIle 1 - >' + fileContent);//Use '+' to convert buffer to string data.

//2.Writing in a File [writeFileSync()],(Data inside will be overwritten) :

fs.writeFileSync('f2.txt','File 2 is Overwritten');//File 2 will be overwritten

//Even if f2.txt is not there it will create it.

console.log('File has been written');

//3.Append to a File[appendFileSync()],(Updating a File) :

fs.appendFileSync('f3.txt','Updating File 3');

console.log('File has been appended');

// 4.Delete a FIle [unlinkSync()] :

fs.unlinkSync('f2.txt');

console.log('File has been deleted.');

/\*OUTPUT:

PS E:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_3> node fs.js

Data of FIle 1 - >Hi i am file 1

File has been written

File has been appended

\*/

* 1. **FS Module with Directories :**

// //FS Module With Directory:

const fs = require('fs');//import fs.

// 1.First Create Directory [mkdirSync()] :

fs.mkdirSync('myDirectory');

// //2.Check the content inside of a  Directory [readdirSync()]:

let folderPath = 'E:\\MCA\\COURSES\\NODE\_JS\\nodejs\_certification@scalar\\Module\_3\\myDirectory';

let folderContent = fs.readdirSync(folderPath);

console.log('Folder Content : ' , folderContent);

/\*OUTPUT :

Folder Content :  [ 'f1.txt', 'f3.txt' ]

\*/

//3.Check particular directory exists or not [existSync()]:

let doesExist = fs.existsSync('myDirectory');

console.log(doesExist);//true

//4.Remove Directory []:

//Before that empty the directory.

fs.rmdirSync('myDirectory');

console.log('Directory Has been Deleted');

1. **Node Package Manager(NPM):[Third party packages that we can use in JS]**
   1. **Introduction to NPM :(We can use npm install different packages)**





* 1. **How to Install and use an NPM Package :**

1. **Create a folder and open it in terminal and run ‘npm init’:**

npm init

PS E:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_4\npm\_package> npm init

This utility will walk you through creating a package.json file.

It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields

and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and

save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (npm\_package)

version: (1.0.0)

description: it is my package

entry point: (index.js)

test command:

git repository:

keywords:

author:

license: (ISC)

About to write to E:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_4\npm\_package\package.json:

{

  "name": "npm\_package",

  "version": "1.0.0",

  "description": "it is my package",

  "main": "index.js",

  "scripts": {

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

  },

  "author": "",

  "license": "ISC"

}

Is this OK? (yes)

1. **“package.json” file will be created inside that folder :**

{

  "name": "npm\_package",

  "version": "1.0.0",

  "description": "it is my package",

  "main": "index.js",

  "scripts": {

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

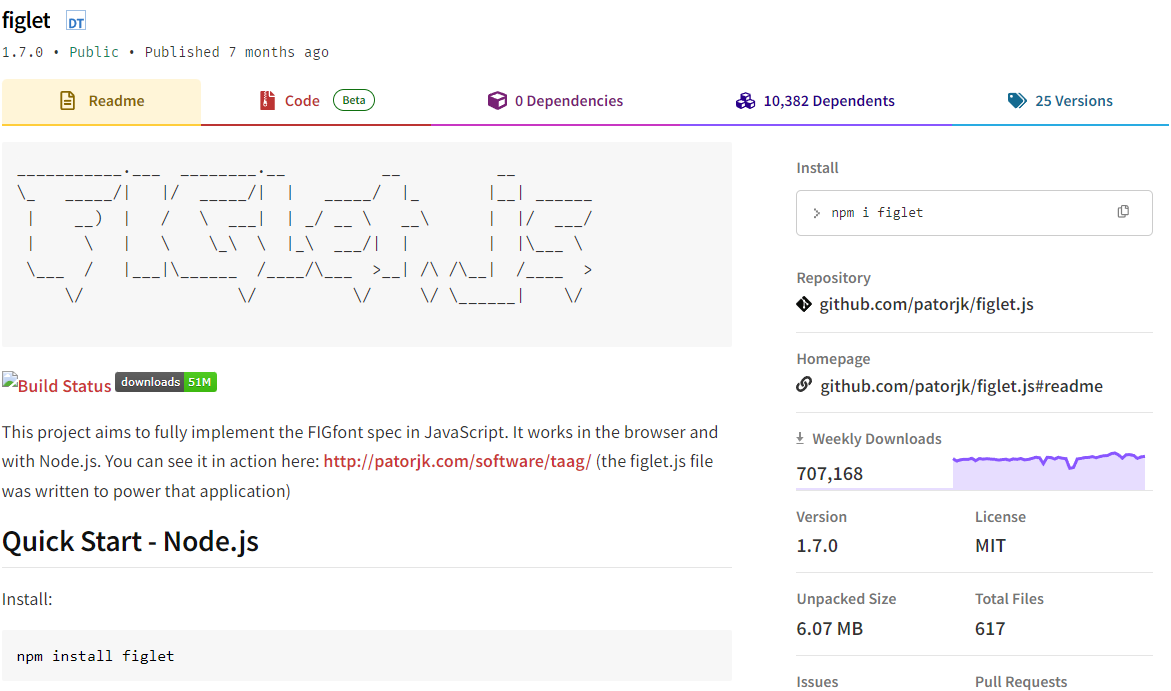
  },

  "author": "",

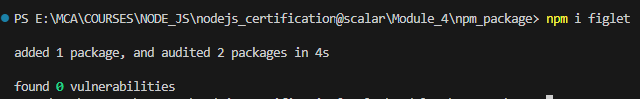
  "license": "ISC"

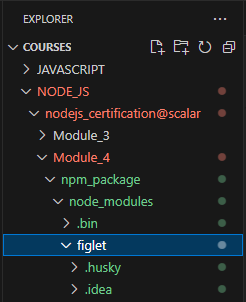
}

1. **Example : Installing ‘figlet’ package :**
2. **Goto npm site and type figlet :**



1. **Install figlet :**







1. **Add example code in a new file ‘npm\_figlet\_example.js’ :**



const figlet = require('figlet'); //figlet package which we installed is  imported.

figlet("Hello World!!", function (err, data) {

  if (err) {

    console.log("Something went wrong...");

    console.dir(err);

    return;

  }

  console.log(data);

});

PS E:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_4\npm\_package> node .\npm\_figlet\_example.js

  \_   \_      \_ \_        \_\_        \_\_         \_     \_ \_ \_

 | | | | \_\_\_| | | \_\_\_   \ \      / /\_\_  \_ \_\_| | \_\_| | | |

 | |\_| |/ \_ \ | |/ \_ \   \ \ /\ / / \_ \| '\_\_| |/ \_` | | |

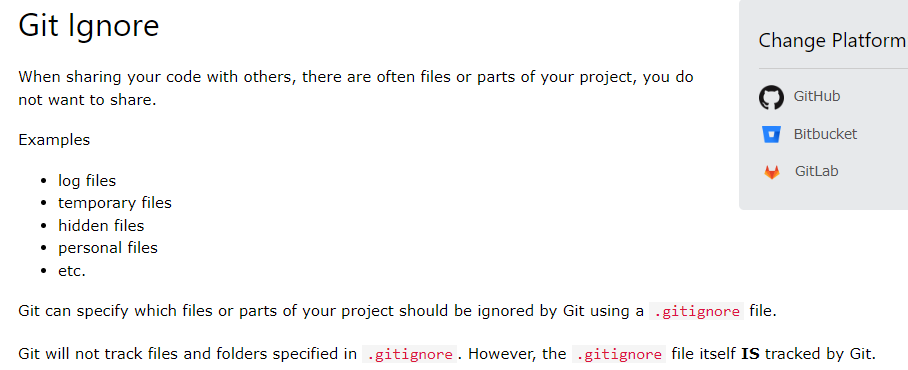
 |  \_  |  \_\_/ | | (\_) |   \ V  V / (\_) | |  | | (\_| |\_|\_|

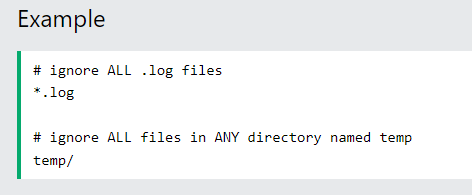
 |\_| |\_|\\_\_\_|\_|\_|\\_\_\_/     \\_/\\_/ \\_\_\_/|\_|  |\_|\\_\_,\_(\_|\_)

PS E:\MCA\COURSES\NODE\_JS\nodejs\_certification@scalar\Module\_4\npm\_package>

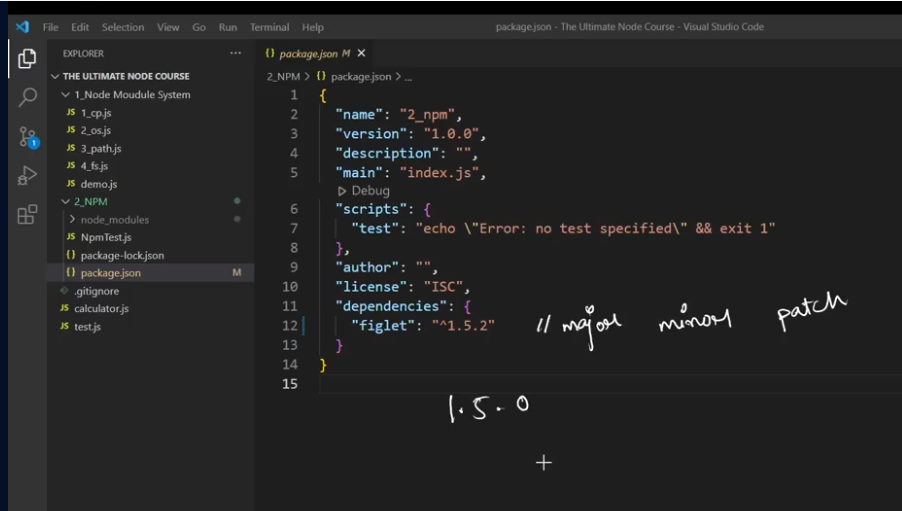
* 1. **All about ‘.gitignore ‘ :**



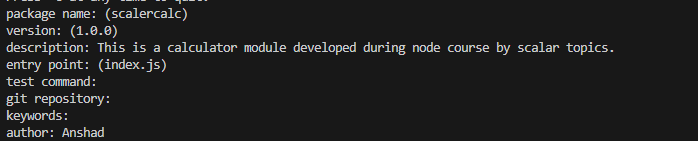




* 1. **Semantic Versioning :(major , minor , patching)**



* 1. **Publishing your own NPM package :**
     1. **First create a folder ‘scalarCalc’ .**
     2. **Then install npm : ‘npm init’.**

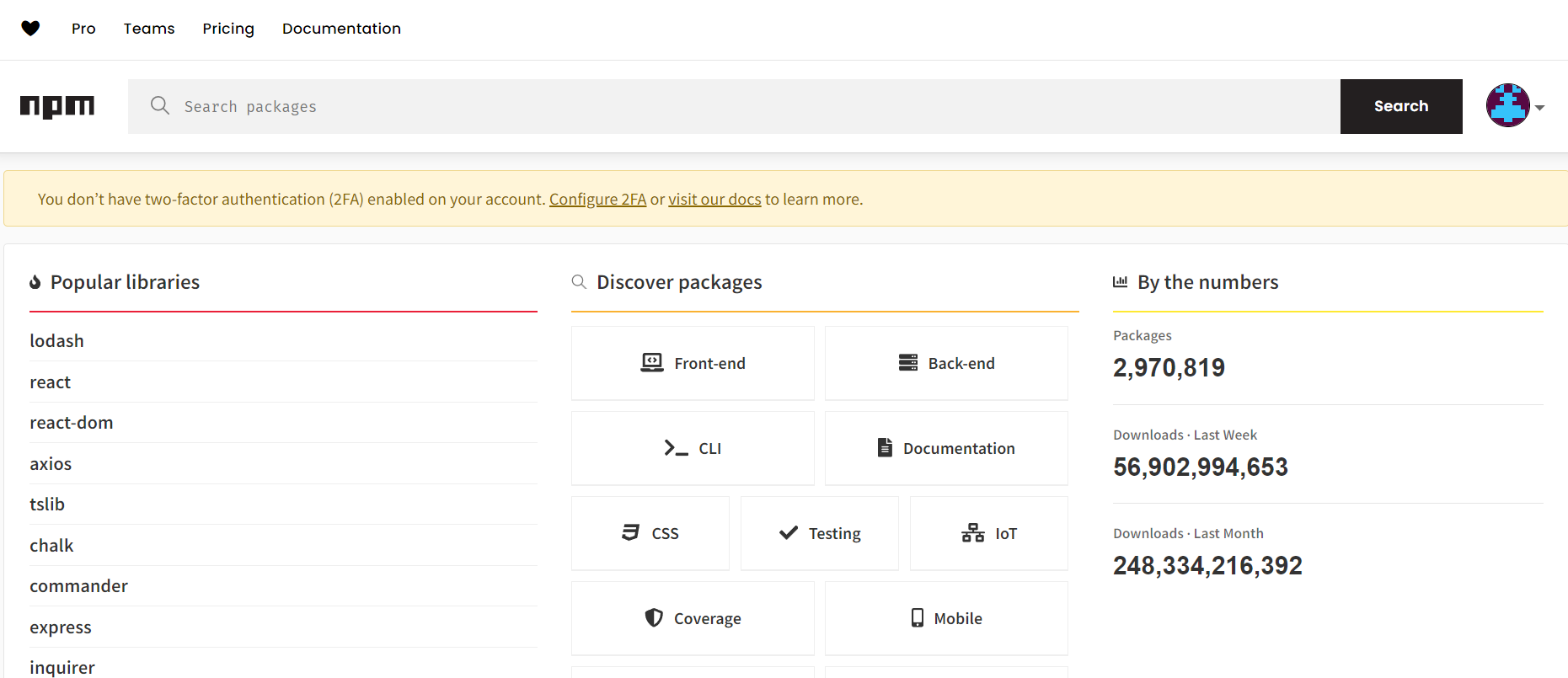


* + 1. **Now create an Account in npm .(Adding a user).It will take us into a page to create a user.**

npm adduser

* + 1. **Now Login to that .**

npm login



* + 1. **Create a file ‘index.js’ inside ‘scalarCalc’ :**

//CALCULATOR :

function add(a , b ){

    console.log( a + b );

}

function sub(a , b ){

    console.log( a - b );

}

function mul(a , b ){

    console.log( a \* b );

}

function div(a , b ){

    console.log( a / b );

}

//Exporting functions:

module.exports = {

    addition : add ,

    subtraction : sub,

    multiplication : mul,

    division : div

}

* + 1. **Now publish it :**

npm publish

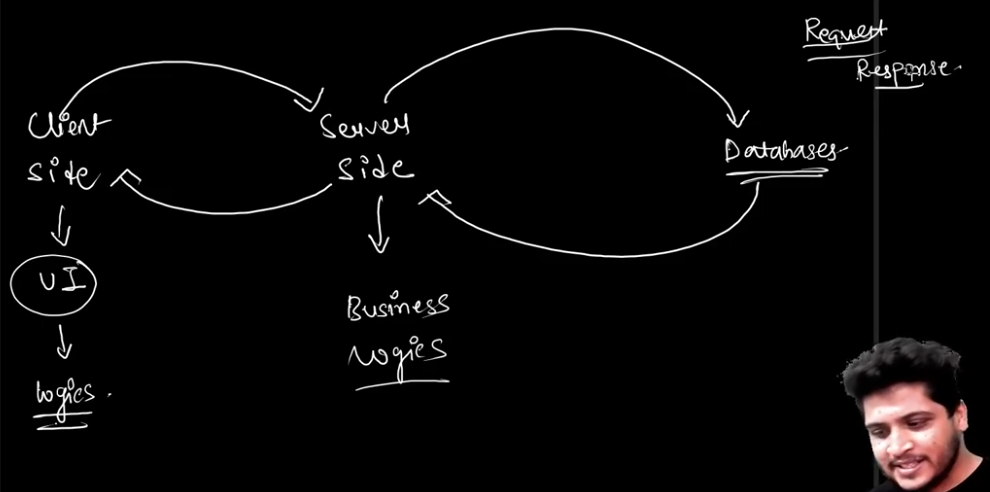
**Now Package is published to npm.**

* + 1. **Now GOTO npm site and type your package name : ‘scalarCalc’ .**
    2. **Where you can see the published npm.**
    3. **Now You can use it by :**

1. **Create a test folder.**
2. **“npm init --yes”.**
3. **“npm i scalarCalc”.**
4. **Now we can use it.by importing**

const calc = require('scalarCalc');

1. **Getting Started with Express :**
   1. **Introduction to Express :**



* ***We will be focusing on Server-side.***
* ***For that we will be using Node Js with Express.js .***



* ***What is Express.js* ?**
* **Express.js is a Framework that is built for Node JS.**
* **Node JS is not a Framework.**
* **Node JS is Runtime Environment (Server side)for JAVASCRIPT by  Ryan Dalh.**
* **Node JS uses ‘V8’ Engine.**
* **‘V8’ : is a JavaScript and WebAssembly engine developed by Google for its Chrome browser.**
* ***Features of Express.js* :**

1. **Fast and Robust Appliacations.(When you are using just Node, without using Express, you have to create to many Boilerplate codes and Configurations.You have to consider so many things.So Express is here to help you to Remove all these Boilerplate codes and Configurations)**
2. **Middlewares :( are the middle processes that execute between processes. In terms of web development, when we store passwords in a database using a server, we use middleware to encrypt our passwords to make them secure. But Node JS does not contain any middleware by default, but we can create our own custom middleware in it. Instead of Node JS, Express.js contains built-in middleware like express.static())**

app.use(express.static('public'));

1. **Routing :(Routing is the process of handling an HTTP request that defines which kind of response will be sent to the client on which particular request.**

* **In Node JS, we have a module called ‘http’ to create a server, where we create a server using http.createServer and pass a callback function to http.createServer, where we get requests and responses as parameter and using if else and URL, we setup routes).**

**NODE JS**

if (method === 'GET' && url === '/') {  
 res.end('Hello, World!');  
 }

* **In Express JS, routing and creating servers is an inbuilt feature, we don’t need to setup if else statements to setup routes. We can directly use the simple methods of Express JS to setup routes.)**

**Express.js**

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

* 1. **Express Installation :**
     1. **Create a Folder ‘Module\_5\_Express’.**
     2. **Open Terminal and type :**

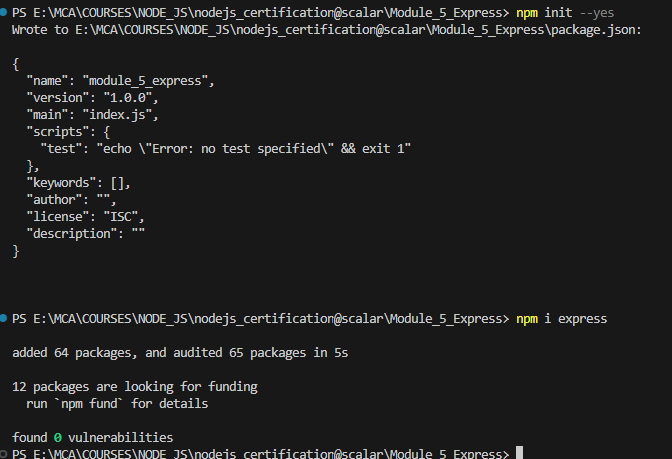
npm init --yes

* + 1. **GOTO npm website and search for ‘express’ and use the code to install express.**

npm i express

* + 1. **or You can go directly to express webpage and use the code to install.**

$ npm install express --save



* 1. **Let’s Use Express :**
     1. **Create ‘app.js’ .**

//First import Express:

const express = require('express');

//This function will return lot of methods.

//To get those methods define another variable:

const app = express();

//Methods : get() , post() , put() ,delete().

//1 app.get() :To READ[To route HTTP GET requests to the specified path]

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

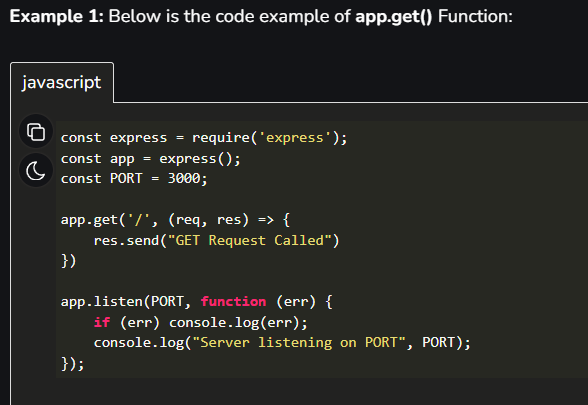
    res.send('Hello from Scalar Topics');

})

//Now if you run the code you will not see anything in output .

//This is bcs you have to Specify the PORT :

app.listen(3000,()=> console.log('Port is Running on 3000'));

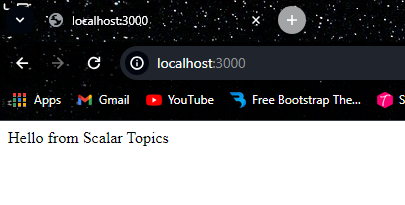


* + 1. **Now run app.js :**

node app.js



* + 1. **Goto browser and type ‘localhost:3000’.[which will display the request message we passed]:**



* + 1. **Now make some changes(add another route of ‘about’) :**

//First import Express:

const express = require('express');

//This function will return lot of methods.

//To get those methods define another variable:

const app = express();

//Methods : get() , post() , put() ,delete().

//1 app.get() :To READ[To route HTTP GET requests to the specified path]

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

    res.send('Hello from Scalar Topics');

})

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

    res.send('We create Impact');

})

//Now if you run the code you will not see anything in output .

//This is bcs you have to Specify the PORT :

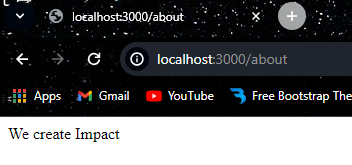
app.listen(3000,()=> console.log('Port is Running on 3000'));

* + 1. **Now to see the changes , you have to STOP THE SERVER and RUN AGAIN.(But this is not a good thing.Each time whenever we make updations we have to re run the server.Solution : ‘Nodemon’ which we will discuss in next part)**

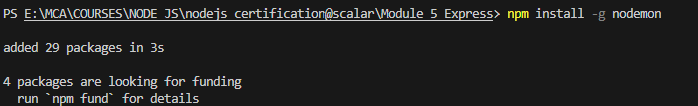
node app.js



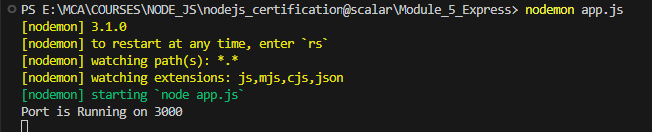
* + 1. **Goto browser and type ‘localhost:3000/about’:**



* 1. **Nodemon :(Used to automatically restarting the node application when file changes in the directory are detected.)**
     1. **To use First install Nodemon:[‘npm install -g nodemon’-To install globally]**

\

* + 1. **Now you have to run your js file using nodemon : [‘nodemon app.js’]**

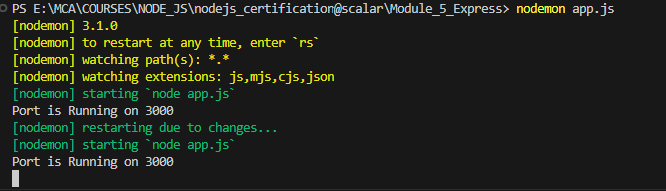


* + 1. **Now make changes and See it automatically updates :**

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

    res.send('Contact us on abcd@gmail.com ');

})





* 1. **Environment Variables and PORT :[**The variable that will change according to whatever environment you are working-that is if you are working in local environment or hosted environment**].**
     1. **Here we are assigning PORT number statically.But in Production Environment PORT is assigned Dynamically.**

app.listen(3000,()=> console.log('Port is Running on 3000'));

* + 1. **So here we will use Environment Variables.[The variable that will change according to whatever environment you are working-that is if you are working in local environment or hosted environment].**
    2. **To Use Env variables ,use process object for that :**

//ENVIRONMENT VARIABLE - for PORTs

const port = process.env.PORT || 3000//for static

const port = process.env.PORT //for dynamic

app.listen(port,()=> console.log(`Port is Running on ${port}`));

* 1. **Routes Parameters :**
     1. Route parameters are named URL segments that are used to capture the values specified at their position in the URL .
     2. The captured values are populated in the “req.params” object, with the name of the route parameter specified in the path as their respective keys.
     3. Create same “app3\_route\_para.js” with same code and add new route with parameters:

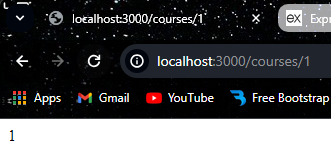
//Route Parameters :

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

    res.send(req.params.id); //To get id as response

    console.log(req.params);

})

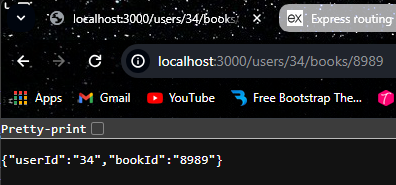


//Example 2 :

app.get('/users/:userId/books/:bookId', (req, res) => {

    res.send(req.params)

  })



* 1. **Handling Multiple Routes :**
     1. **Set objects inside an array with id and name. And then define route:**

//Handling Multiple routes:

const courses = [

    {id:1 , name : 'JavaScript'},

    {id:2 , name : 'Java'},

    {id:3 , nmae : 'Python'},

]

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

    console.log(req.params.coursename);

    let course = courses.find(course => course.name === req.params.coursename);//parseInt to convert string to int

    //To handle Error when specific course is not there.

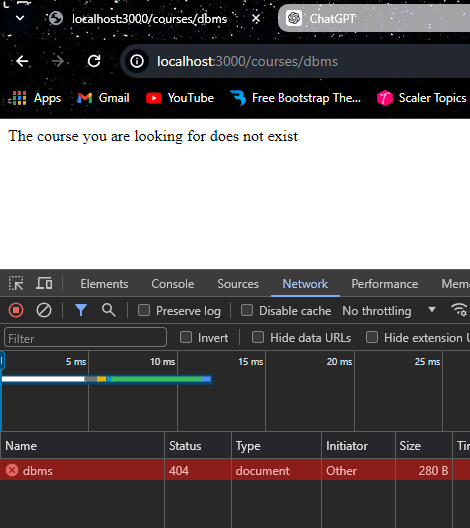
    if(!course) res.status(404).send('The course you are looking for does not exist')

    res.send(course);

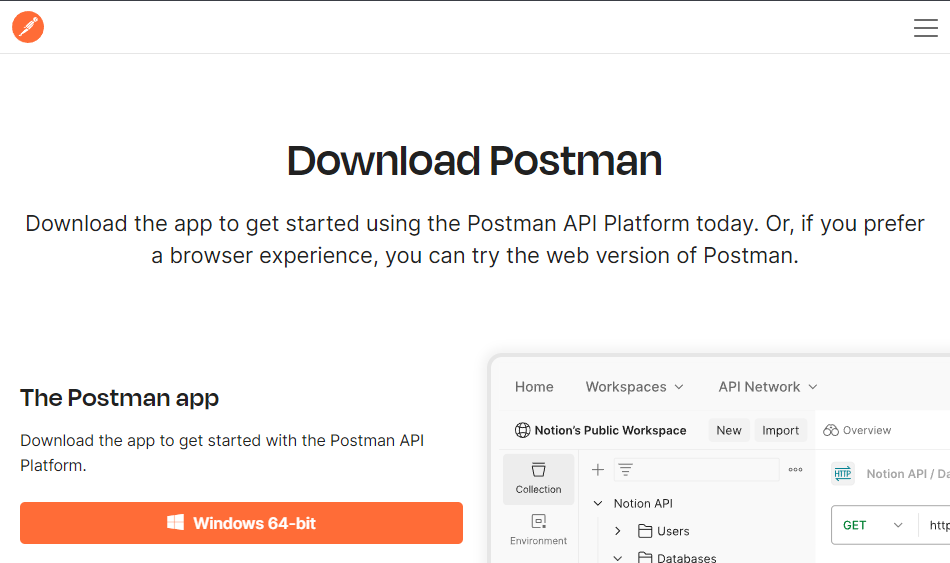
})

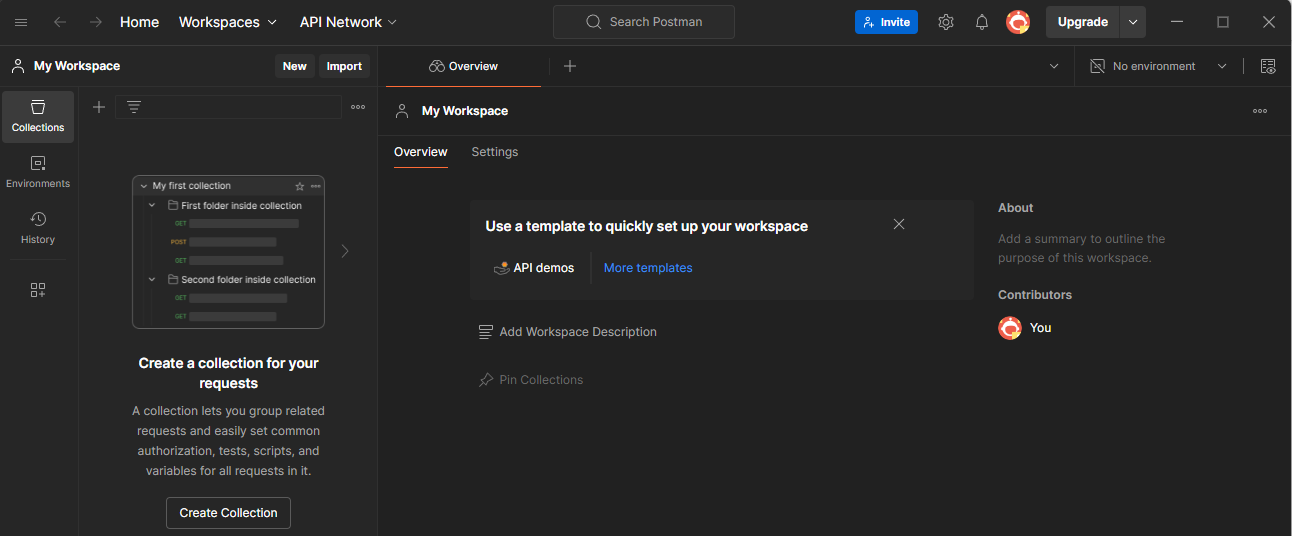


**Handling Errors:**

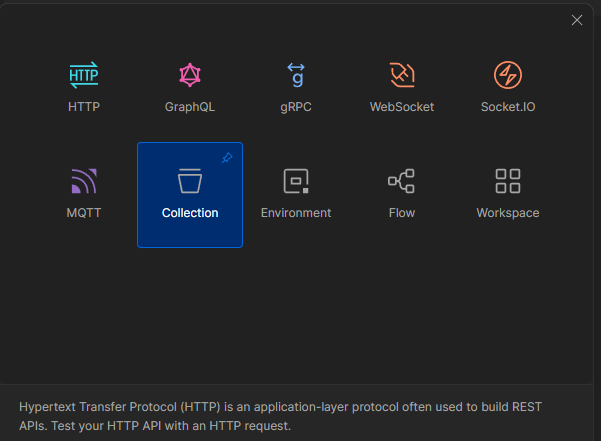


* 1. **Postman :(**Postman is an application that allows the testing of web APIs**.)**
     1. Goto postman website,signup and Download it.

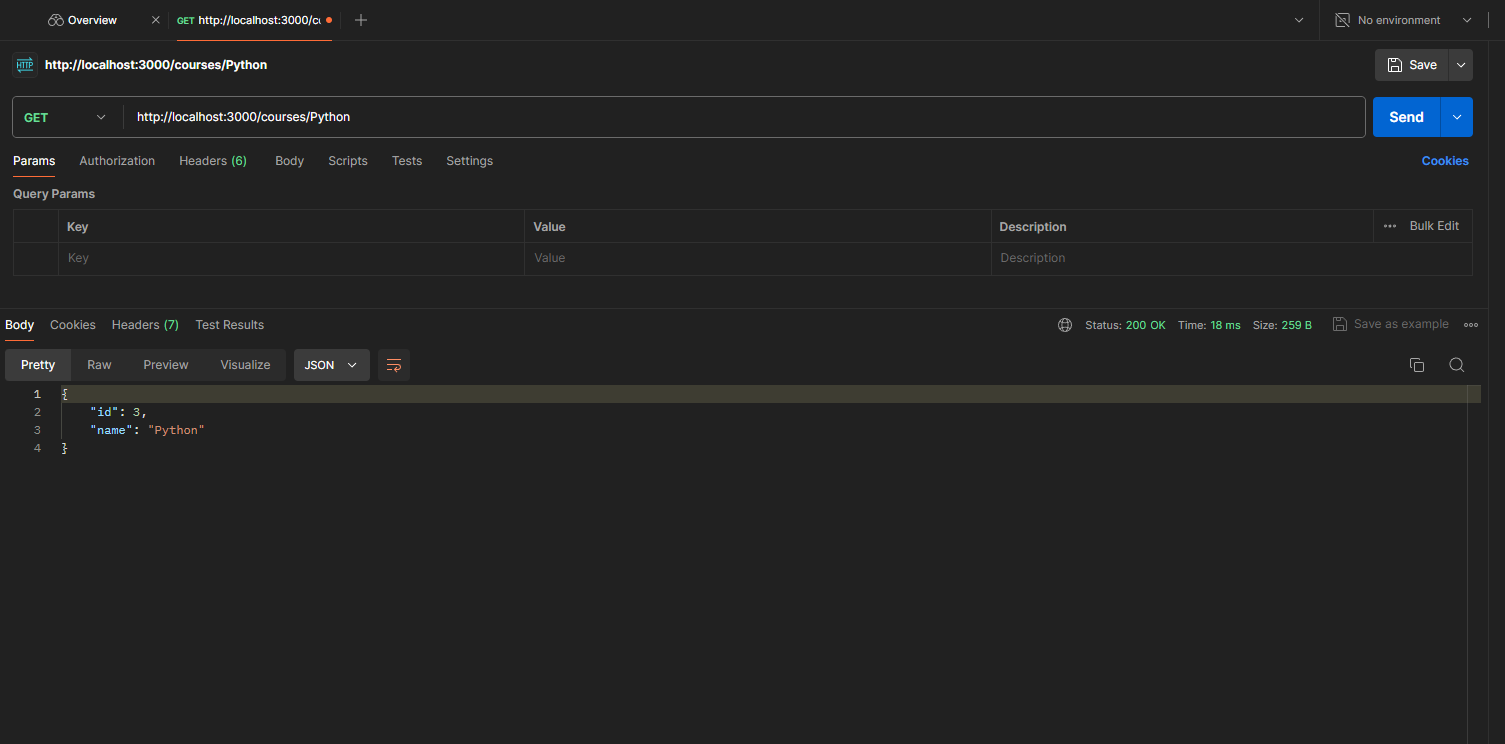




* + 1. How to Use ?
* Click ‘New’:



* Then Paste link and press send:



* 1. **Http Post Method :[To Create]**
     1. First define a get route for getting all courses :

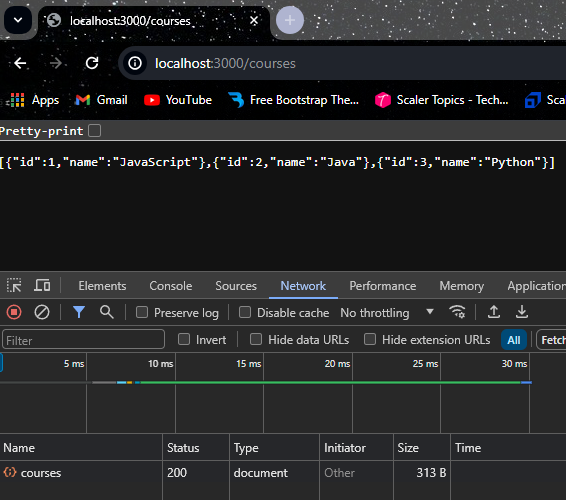
//2. app.post() :

//Before using post ,define a get route for getting all courses:

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

    res.send(courses);

})



* + 1. Whenever we are using post(whenever we need to create data),when we are using Express you have pass to JSON.
    2. To pass you need to use a Middleware.
    3. ’app.use()’ to use middleware:

//To use middleware:

app.use(express.json());

* + 1. Pushing to courses collection :

//post()

//Whenever we are  using post(whenever we need to create data),when we are using Express you have pass to JSON

//To pass you need to use a Middleware.

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

    const course = {

        id : courses.length +1,

        name : req.body.name

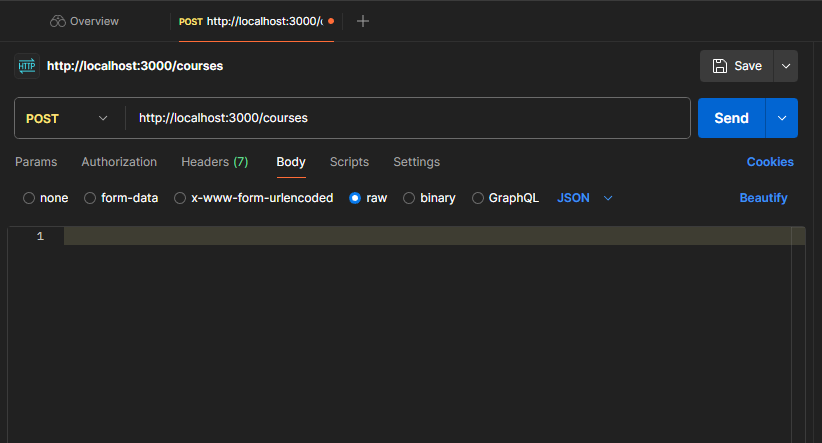
    }

    courses.push(course);//pushing courses collection

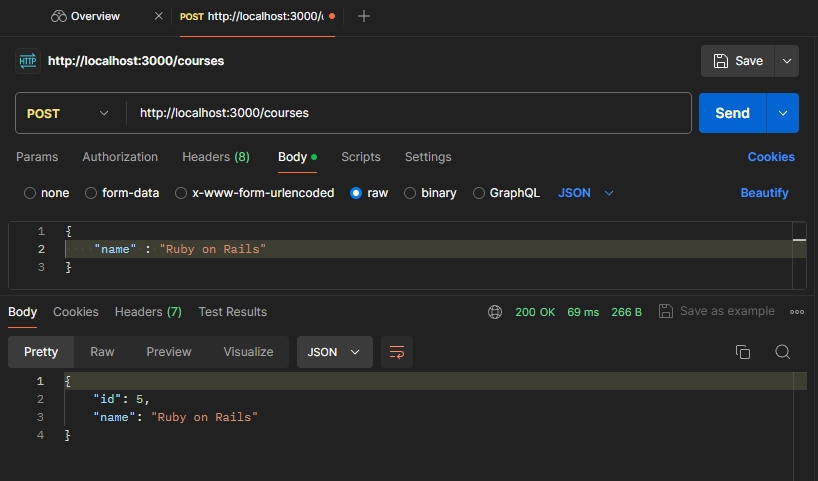
    res.send(course);

})

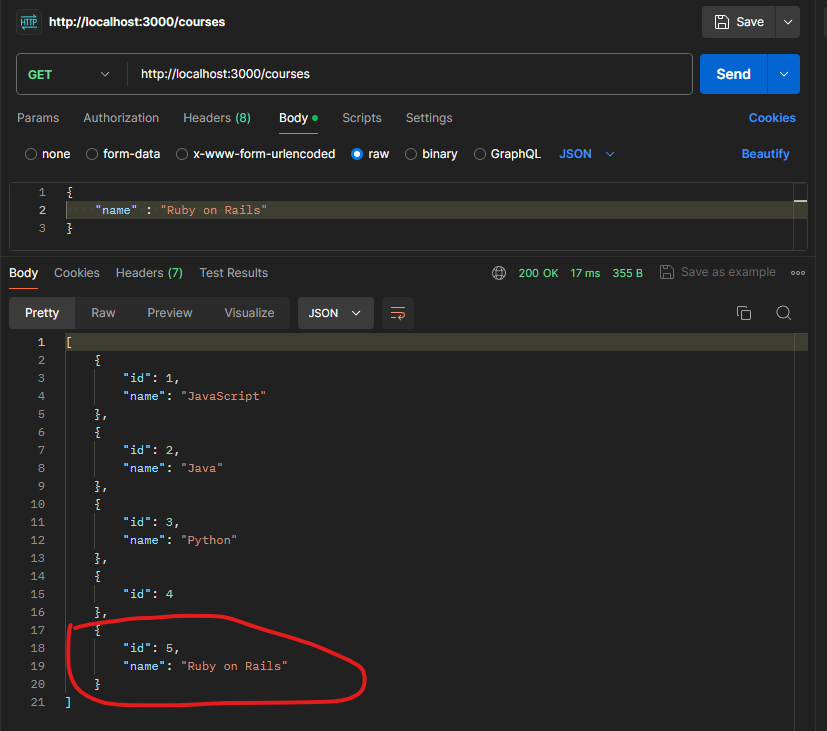
* + 1. Now GOTO postman and test this(send).



* Goto ‘Body’ and select ‘raw’:



* Now use GET to check whether the send data is recieved:



* 1. **Http Put Method :[To UPDATE]-(**Used to update already existing entries**)**
     1. Add a ‘app.put()’ route:

// 3. app.put() :TO UPDATE

//used to update already existing entries.

app.put('/courses/:coursename', (req , res)=> {

    let course = courses.find(course => course.name === req.params.coursename);

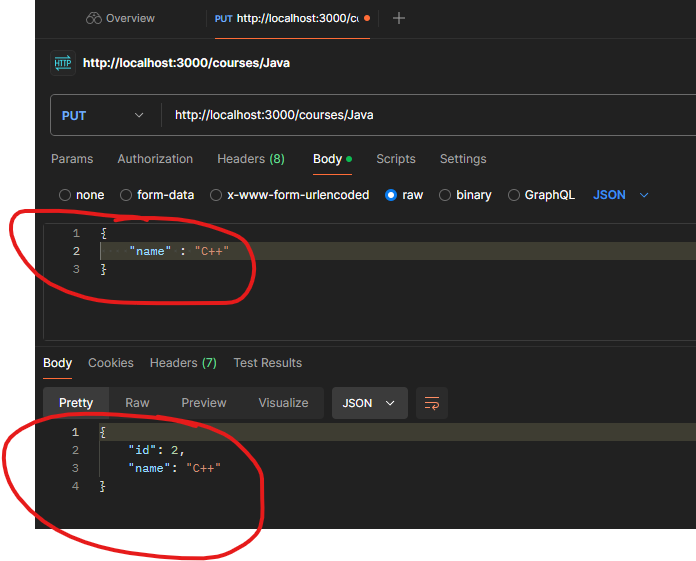
     if(!course) res.status(404).send('The course you are looking for does not exist');

     course.name = req.body.name;//whatever the name we give in body in postman will update above courses array object.

     res.send(course);

})

* + 1. Now run ‘nodemon app5\_http\_post.js’ .
    2. Then Open Postman and select PUT at :(http://localhost:3000/courses/Java).
    3. Then GOTO body->raw and pass new name “C++” to update “Java” .



* 1. **Http Delete Method :**
     1. First change courses array from ‘const’ to ‘let’:

//Handling Multiple routes:

let courses = [ //change const to let for delete

    {id:1 , name : 'JavaScript'},

    {id:2 , name : 'Java'},

    {id:3 , name : 'Python'},

]

* + 1. Now specify the route :

//4 .app.delete() : To Delete

app.delete('/courses/:coursename',(req , res)=>{

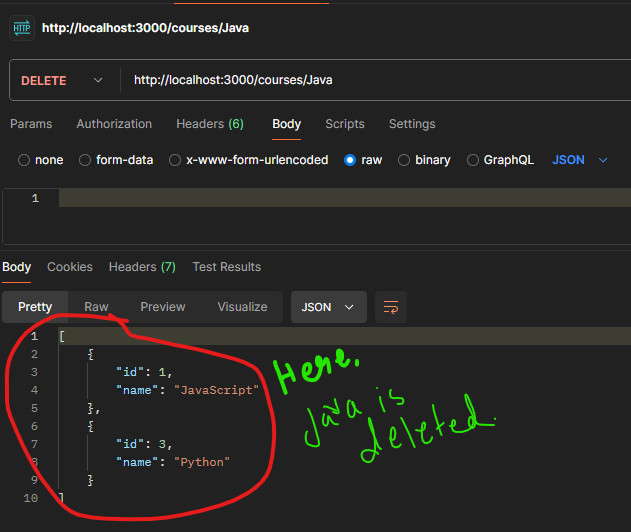
    let UpdatedCourses = courses.filter(course => course.name !== req.params.coursename);

    courses = UpdatedCourses;

    req.send(courses);

})

* + 1. Goto postman and perform DELETE with url :’http://localhost:3000/courses/Java’.



* + 1. Here we passed string(name) to DELETE content.But it is not not recommended.So lets use ‘id’:

//DELETE using id :

app.delete('/courses/:id',(req , res)=>{

    let course = courses.find(course => course.id !== req.params.id); //Finds the particular course with id.

    if(!course) res.status(404).send('The course you are looking for does not exist');//if not found

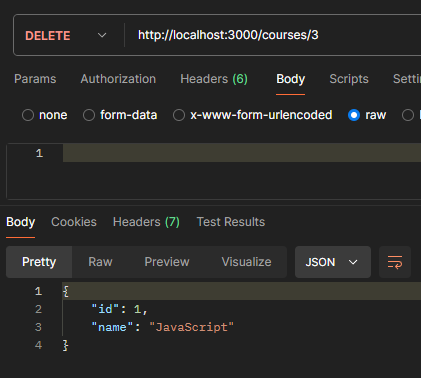
    const index = courses.indexOf(course);//To find the index .

    courses.splice(index , 1);//To delete that index from array.

    res.send(course);

})

* + 1. AGoto postman and perform DELETE with url :’http://localhost:3000/courses/3’:



//Http delete() : .

//First import Express:

const express = require('express');

const app = express();

//Methods : get() , post() , put() ,delete().

//1 app.get() :To READ[To route HTTP GET requests to the specified path]:

//To use middleware:

app.use(express.json());

//Handling Multiple routes:

let courses = [ //change const to let for delete

    {id:1 , name : 'JavaScript'},

    {id:2 , name : 'Java'},

    {id:3 , name : 'Python'},

]

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

    res.send('Hello from Scalar Topics');

})

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

    res.send('We create Impact');

})

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

    res.send('Contact us on abcd@gmail.com ');

})

//Route Parameters :

app.get('/users/:userId/books/:bookId', (req, res) => {

    res.send(req.params)

  })

//2. app.post() : TO CREATE

//Before using post ,define a get route for getting all courses:

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

    res.send(courses);

})

//post()

//Whenever we are  using post(whenever we need to create data),when we are using Express you have pass to JSON

//To pass you need to use a Middleware.

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

    const course = {

        id : courses.length +1,

        name : req.body.name

    }

    courses.push(course);//pushing courses collection

    res.send(course);

})

// 3. app.put() :TO UPDATE

//used to update already existing entries.

app.put('/courses/:coursename', (req , res)=> {

    let course = courses.find(course => course.name === req.params.coursename);

     if(!course) res.status(404).send('The course you are looking for does not exist');

     course.name = req.body.name;//whatever the name we give in body in postman will update above courses array object.

     res.send(course);

})

//Handling Multiple routes-continue:

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

    // console.log(req.params.coursename);

    let course = courses.find(course => course.name === req.params.coursename);//parseInt to convert string to int

    //To handle Error when specific course is not there.

    if(!course) res.status(404).send('The course you are looking for does not exist')

    res.send(course);

})

//4 .app.delete() : To Delete

// app.delete('/courses/:coursename',(req , res)=>{

//     let UpdatedCourses = courses.filter(course => course.name !== req.params.coursename);

//     courses = UpdatedCourses;

//     res.send(courses);

// })

//DELETE using id :

app.delete('/courses/:id',(req , res)=>{

    let course = courses.find(course => course.id !== req.params.id); //Finds the particular course with id.

    if(!course) res.status(404).send('The course you are looking for does not exist');//if not found

    const index = courses.indexOf(course);//To find the index .

    courses.splice(index , 1);//To delete that index from array.

    res.send(course);

})

//ENVIRONMENT VARIABLE - for PORTs

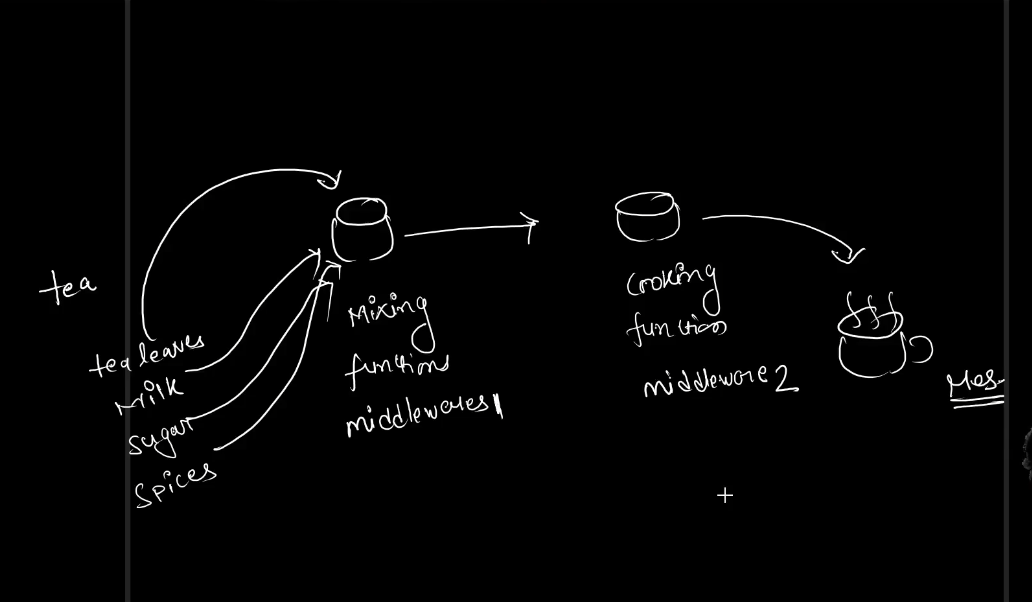
const port = process.env.PORT || 3000//for static

app.listen(port,()=> console.log(`Port is Running on ${port}`));

1. **Middlewares in Express :**

app.use();//To use middleware

* 1. **What are Middlewares ?.**
     1. It is a function that will have all the access for requesting an object, responding to an object, and moving to the next middleware function in the application request-response cycle.
     2. Middleware functions provide access to the request (req) and response (res) objects. They are the perfect way to modify the req and res objects.
     3. For example, we can fetch user details after the user has logged in and save these details in the request object.



* + 1. **Json() as Middleware** : In post() request , We have used ‘json()’, to pass the data into ‘json()’ ,That is we passed value by typing in body>raw (‘req.body.name’) which were passed to json() and used to create the object .SNIPPET CODE:

//To use middleware:

app.use(express.json());

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

    const course = {

        id : courses.length +1,

        name : req.body.name

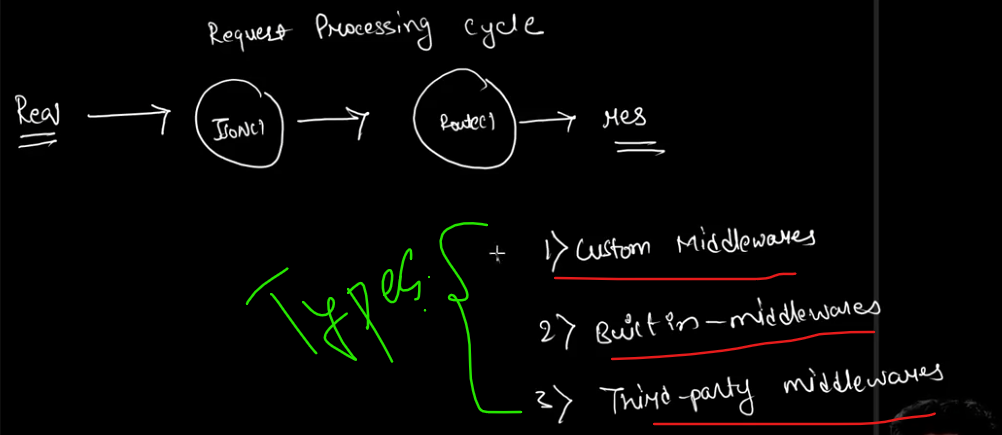
    }

    courses.push(course);//pushing courses collection

    res.send(course);

})

* + 1. app.get() also acts as a middleware which returns the response.Every step in express is a middleware.



* 1. **Custom Middlewares.**
     1. First define a custom middleware.

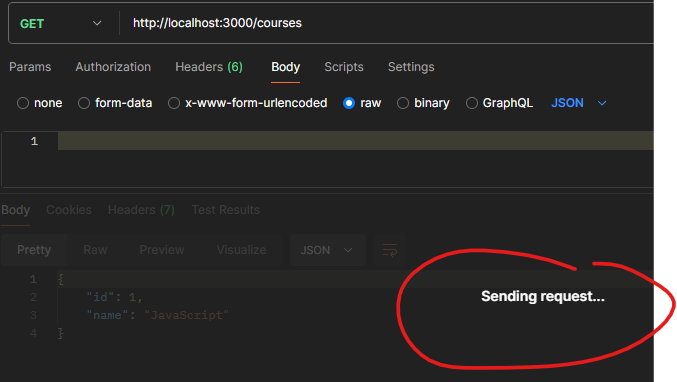
//Custom middleware :

app.use(function(req ,res ){

    console.log('I am Custom Middleware');

});

* + 1. Then run it : ‘nodemon app.js’:
    2. Then Goto postman and try to GET using url :’http://localhost:3000/courses’.
    3. Here we doesn’t get any response.Its just keep sending request.



* + 1. And in console output we will have the output:



* + 1. Here actually the middleware ‘I am Custom Middleware’ is executed.But it is not letting to go for next middleware.So for that we have to pass from one middleware to another middleware.For that we use the ‘next()’ method.

//Custom middleware :

app.use(function(req , res , next ){

    console.log('I am Custom Middleware');

    next();

});

app.use(function(req , res , next ){

    console.log('I am second Custom Middleware');

    next();

});

* + 1. Now if you use GET in postman you will get the response.
    2. Here we should not specify custom middlewares like this.Instead we have to do in separate file .
    3. For that create a separate folder ‘middlewares’ .
    4. Then create a file ‘middle.js’ and ‘middle2.js’ [for each middleware use separate file] :

middle.js

function myMiddleware(req , res , next ){

    console.log('I am Custom Middleware');

    next();

}

 //exporting

module.exports = myMiddleware;

middle2.js

function myMiddleware2(req , res , next ){

    console.log('I am second Custom Middleware');

    next();

}

//exporting

module.exports = myMiddleware2;

* + 1. Now import those files to ‘app.js’:

const myMiddlewareFunction = require('./middlwares/middle');//Importing custom middleware from folder

const myMiddlewareFunction2 = require('./middlwares/middle2');//Importing middle2

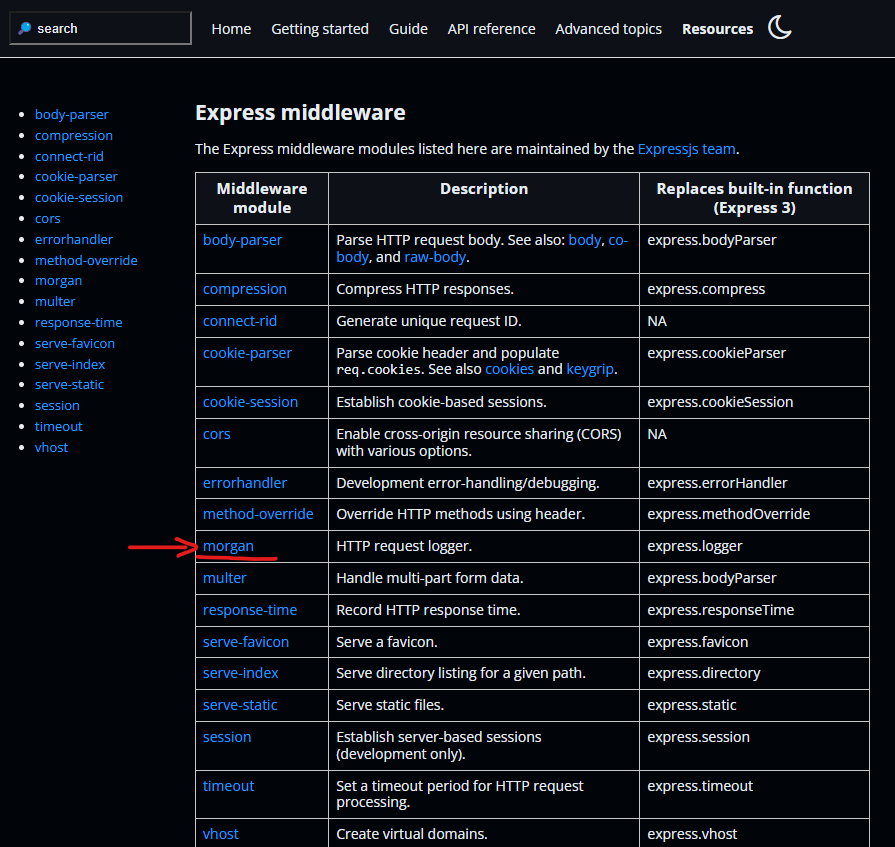
* + 1. Now specify it in your ‘app.js’ :

//Custom middleware :

app.use(myMiddlewareFunction);

app.use(myMiddlewareFunction2);

* + 1. Now if you go to postman and check.It will be running good.
  1. **Third Party Middlewares.**
     1. Goto “Express webiste >resources >Middleware” :



* + 1. Lets try “morgan”:[morgan is a HTTP request logger.]

1. First install morgan.

npm install morgan

1. Then import it in your ‘app.js’ :

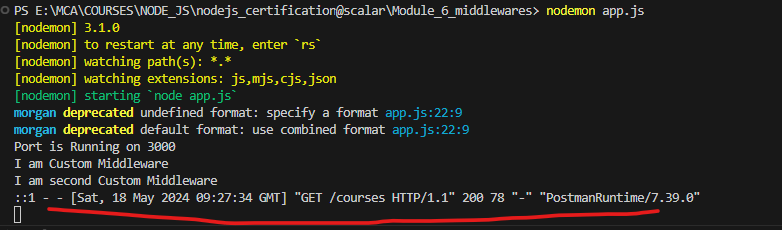
const morgan = require('morgan');//Importing morgan

1. Now use morgan(specify):

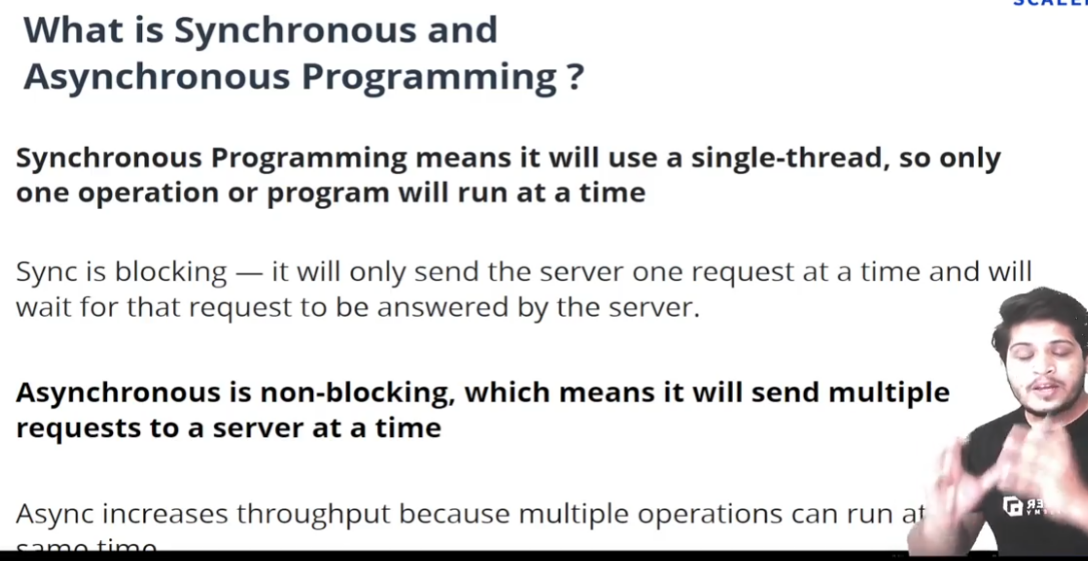
//user third partym middleware-morgan

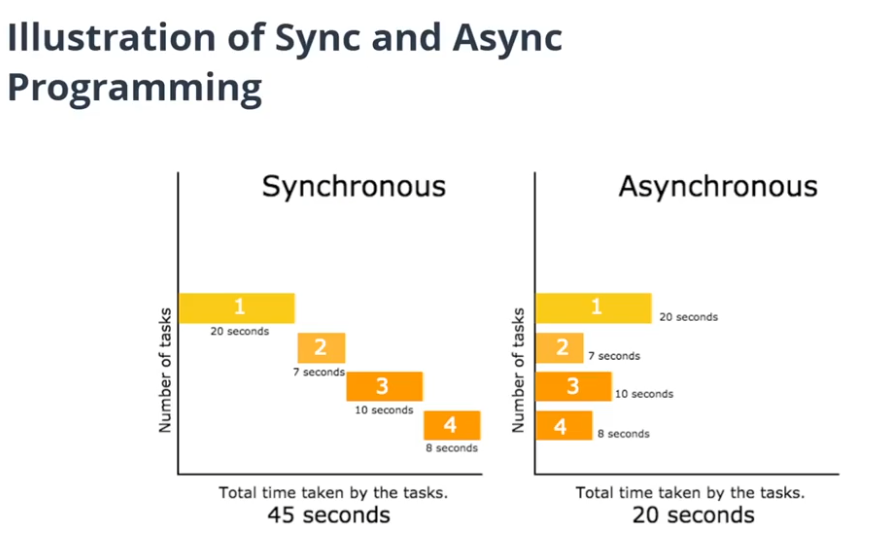
app.use(morgan());

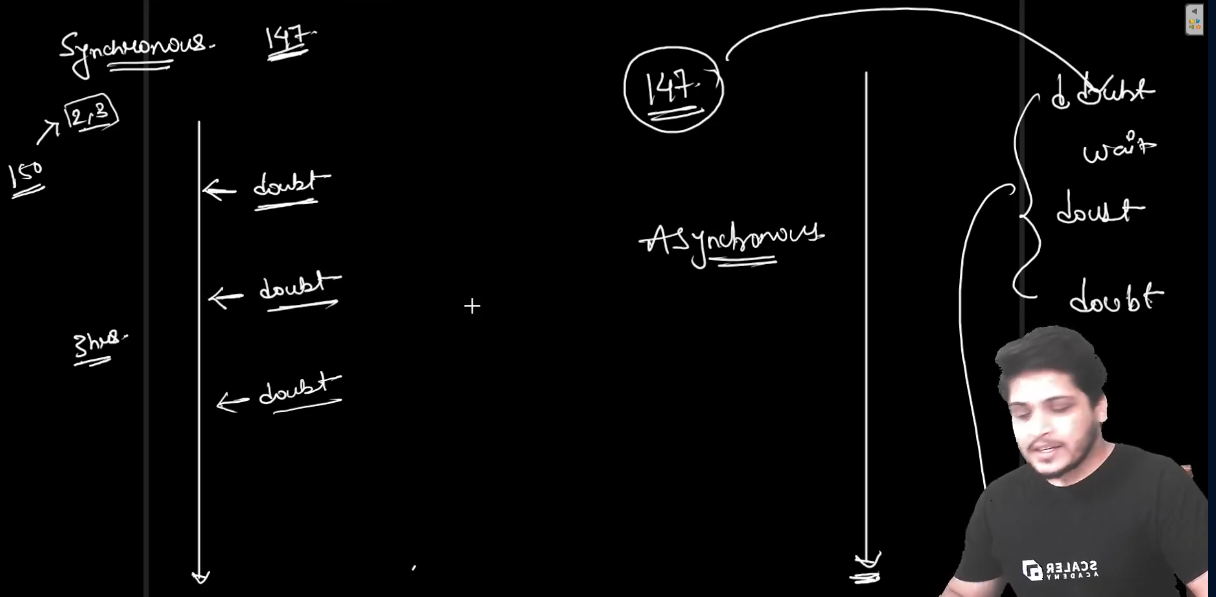
1. Then Run ‘nodemon app.js’.
2. Now goto postman and Test.
3. In console output you can see the log.



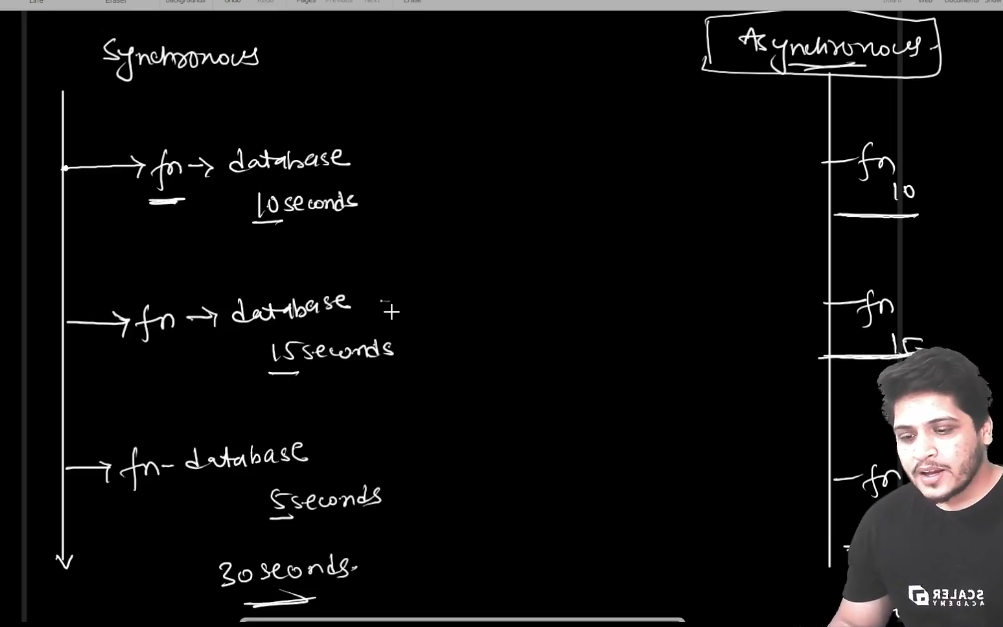
1. **Asynchronous JavaScript :**
   1. **Introduction to Asynchronous Programming.**

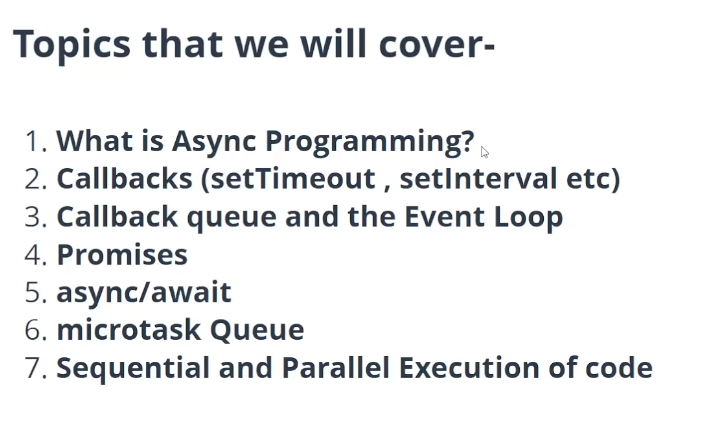






**Synchronous and Asynchronous in JS:**





* 1. **Reading a File Synchronously.**

1. **Create ‘synchronous.js’ file.**

//Synchronous Approach :[Using 'readFileSync()' method]

const fs = require('fs');

console.log('First Line');

let data =fs.readFileSync('f1.txt');

console.log('File 1 Data -> '+ data);

console.log('Last Line')

1. **Create ‘f1.txt ‘ file.**

I am file 1

1. **OUTPUT :**

First Line

File 1 Data -> I am file 1

Last Line

* 1. **Read a File Asynchronously.**

1. **Create ‘Asynchronous.js’ file.**

//Asynchronous Approach :[Using 'readFile()' method]

const fs = require('fs');

console.log('First Line');

// let data1 =fs.readFileSync('f1.txt');

// console.log('File 1 Data -> '+ data1);

// let data2 =fs.readFileSync('f2.txt');

// console.log('File 1 Data -> '+ data2);

fs.readFile('f1.txt',cb1); //Reading file Asynchronously

//While reading if there is any error it will return to 'err' ,

//if no error return data to 'data'.

function cb1(err,data){

    if(err){

        console.log(err);

    }

    console.log('File 1 data ->' + data);

}

fs.readFile('f2.txt', cb2);

function cb2(err,data){

    if(err){

        console.log(err);

    }

    console.log('File 2 data ->' + data);

}

console.log('Last Line')

1. **Create ‘f2.txt’ file.**

I am file 2

1. **Output :**

PS E:\MCA\COURSES\JAVASCRIPT\Js\_certification@scalar\Module\_8> node Asynchronous.js

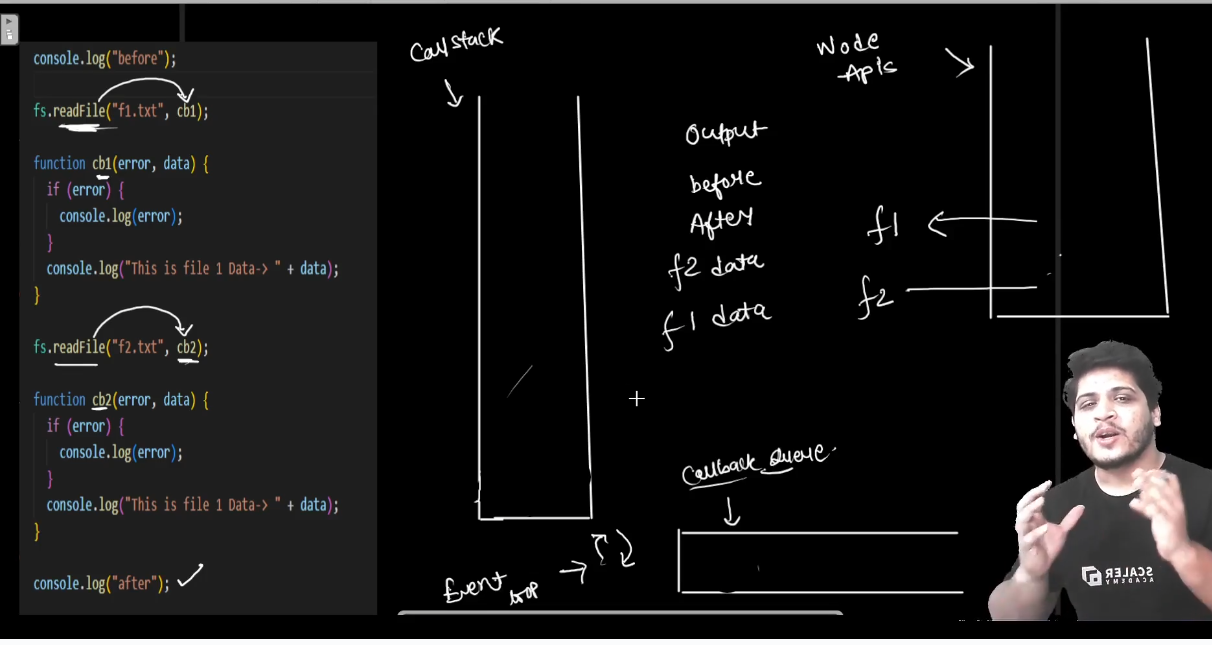
First Line

Last Line

File 1 data ->I am file 1

File 2 data ->I am file 2

* 1. **The Event Loop and Everything.**



* 1. **Serial Execution of Async code.**

//Serial Execution :(You have make something dependent)

const fs = require('fs');

console.log('First Line');

fs.readFile('f1.txt',cb1); //Reading file Asynchronously

function cb1(err,data){

    if(err){

        console.log(err);

    }

    console.log('File 1 data ->' + data);

    fs.readFile('f2.txt', cb2);//readfile of 2nd file is put inside file 1

}

function cb2(err,data){

    if(err){

        console.log(err);

    }

    console.log('File 2 data ->' + data);

    fs.readFile('f3.txt', cb3);//readfile of 3rd file is put inside here

}

function cb3(err,data){

    if(err){

        console.log(err);

    }

    console.log('File 3 data ->' + data);

}

console.log('Last Line')

/\*OUTPUT : (How many times you run the code it will be in order)

PS E:\MCA\COURSES\JAVASCRIPT\Js\_certification@scalar\Module\_8> node serialExec.js

First Line

Last Line

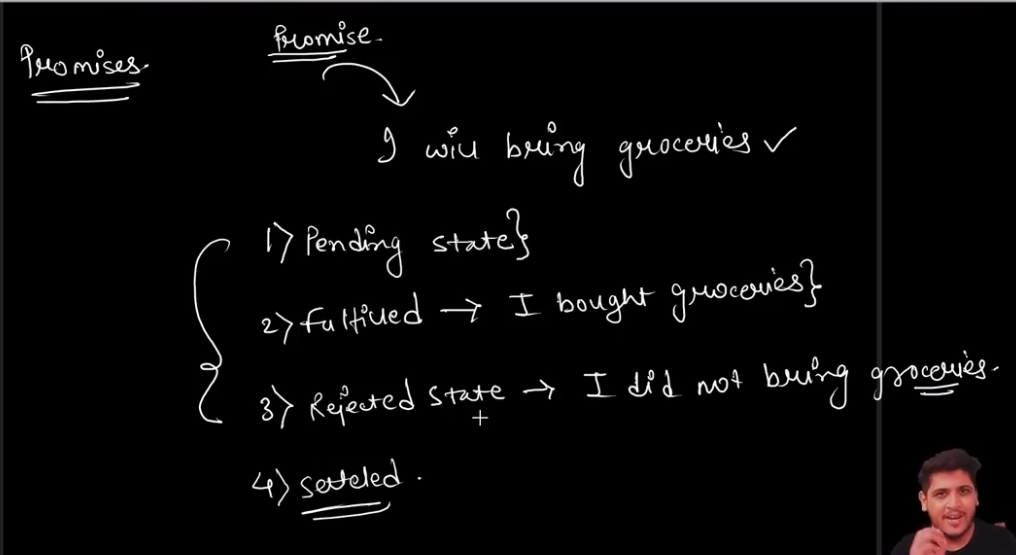
File 1 data ->I am file 1

File 2 data ->I am file 2

File 3 data ->This is File 3

\*/

* 1. **Promises in JS.**



//Promises .

//How to produce a promise:

//State 1 : Pending State.

// let myPromise = new Promise(function(resolve , reject){

//     const a = 4;

//     const b = 5;

//     setTimeout(()=>{

//         if(a === b){

//             resolve();

//         }

//         else{

//             reject();

//         }

//     },2000)

// })

// console.log(myPromise);

/\*OUTPUT :

Promise { <pending> }

\*/

//State 2 : Fulfilled State.(using 'then' method)

let myPromise = new Promise(function(resolve , reject){

    const a = 4;

    const b = 5;

    setTimeout(()=>{

        if(a === b){

            resolve('The Values are Equal');

        }

        else{

            reject('The values were not equal');

        }

    },2000)

})

myPromise.then(function(result){

    console.log(result);

})//State 2:FULFILLED

//We have to catch exception while Promise is rejected using 'catch()' method .

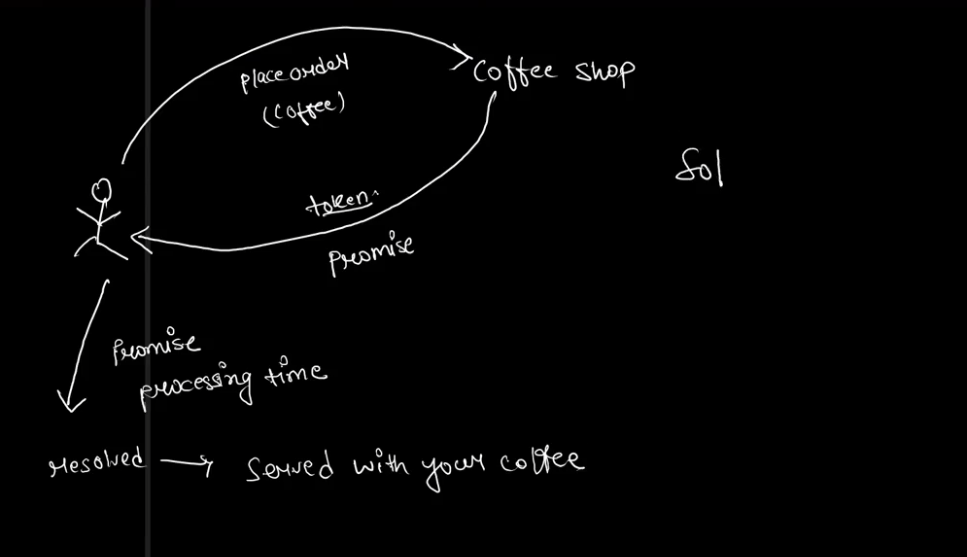
myPromise.catch(function(failedResult){

    console.log(failedResult);

})//State 3 :REJECTED

//State 4 :Promise is SETTLED

* 1. **Promises and Async Wait.**



//COFFEE SHOP-Promise

// function placeOrder(drink){

//     return new Promise(function(resolve , reject){

//         if(drink === 'coffee'){

//             resolve('Order For Coffee recieved');

//         }

//         else{

//             reject('Other orders Rejected');

//         }

//     })

// }

// //function for proccessing order time:

// function processOrder(order){

//     return new Promise(function(resolve){

//         console.log('Order is being Processed');

//         resolve(`${order} and is Served.`);

//     })

// }

// //two 'then's used because we have two Promises.

// placeOrder('coffee').then(function(orderPlaced){

//     console.log(orderPlaced);

//     let orderIsProcessed = processOrder(orderPlaced);//for catch exception in reject state.

//     return orderIsProcessed;

// }).then(function(processedOrder){//chaining of Promise

//     console.log(processedOrder);

// })

//catch:

//WHAT if you have Lots of Promise ?

//Solution=> 'Async Wait' :

//--------------------------------->

function placeOrder(drink){

    return new Promise(function(resolve , reject){

        if(drink === 'coffee'){

            resolve('Order For Coffee recieved');

        }

        else{

            reject('Other orders Rejected');

        }

    })

}

//function for proccessing order time:

function processOrder(order){

    return new Promise(function(resolve){

        console.log('Order is being Processed');

        resolve(`${order} and is Served.`);

    })

}

async function serveOrder(){//'async' keyword

    try {

        let orderPlaced = await placeOrder('coffee');

        console.log(orderPlaced);

        let processedOrder =await processOrder(orderPlaced);

        console.log(processedOrder);

    }

    catch (error) {

        console.log(error);

    }

}

serveOrder();

/\*OUTPUT :

node "e:\MCA\COURSES\JAVASCRIPT\Js\_certification@scalar\Module\_8\coffeeShop.js"

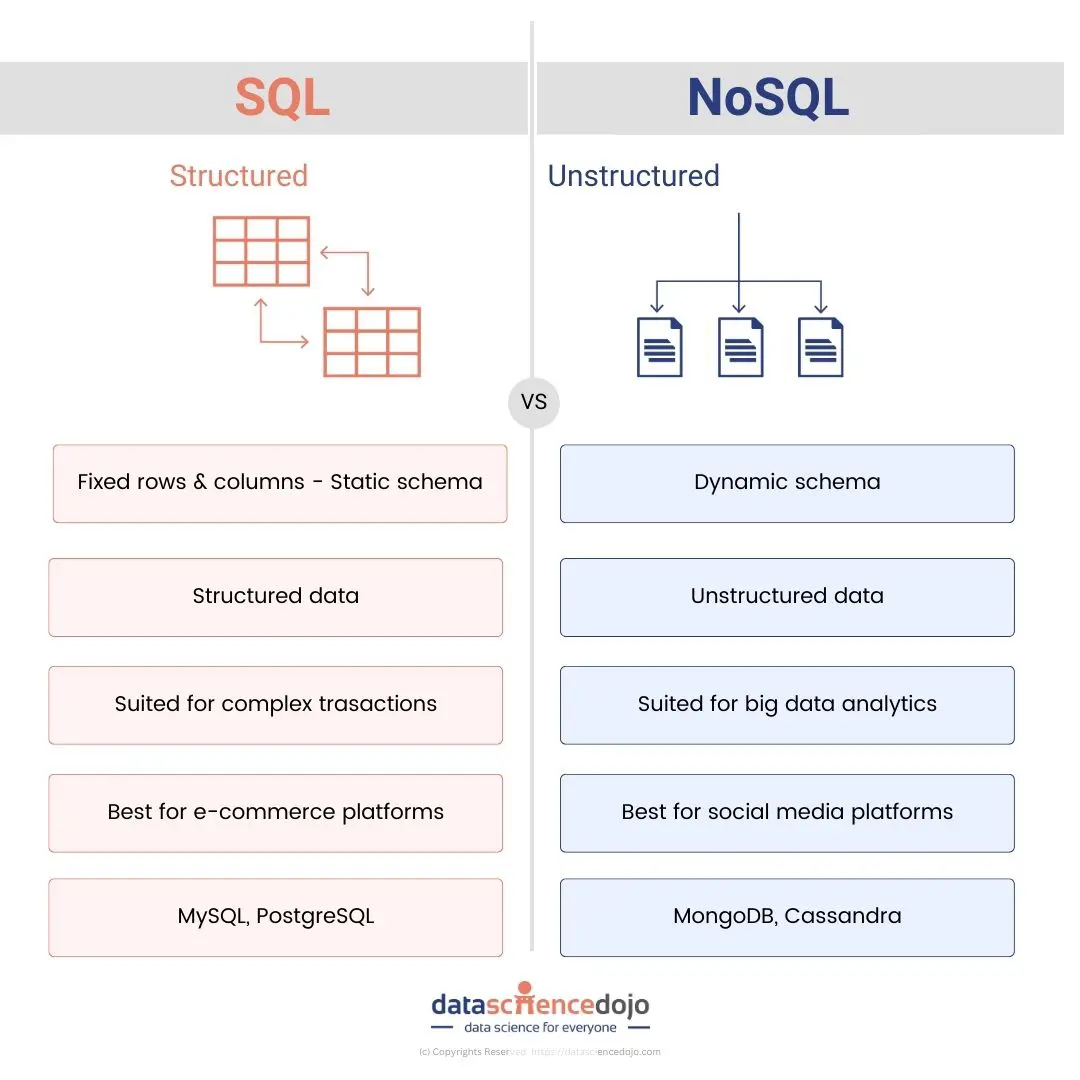
Order For Coffee recieved

Order is being Processed

Order For Coffee recieved and is Served.

\*/

1. **CRUD with Mongose and MongoDB.**
   1. **Getting started with MongoDB.**
      1. MongoDB is a NoSQL DBMS.
      2. NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables.
      3. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph.
      4. It stores data in a type of ‘JSON’ format called ‘BSON’.
      5. MongoDB is a document database(non-relational database) and can be installed locally or hosted in the cloud.



* 1. **Installation and Configuration of MongoDB.**
     1. goD
     2. Sdds
     3. Sd
     4. Sd
     5. Sd
     6. Sd
     7. Ds
     8. Sd
  2. **Schemas and Models.**
  3. **How to query for documents.**
  4. **Comparison Query Operator.**
  5. **Logical Query Operators.**
  6. **How to Update a Document.**
  7. **How to Delete a Document.**

1. **Asda**
2. **asd**