NODE JS - SCALAR

1. ***Introduction*** :

SCALER

What is Node.js ?

Node.js is an Open source , cross platform JavaScript runtime Environment for executing code outside of a Browser

Node.js is widely used to build back-end services for Client Side Application, like creating APIs , building Authentication Systems , File handling and talking to the Database.

Node.js can be used to build data-Intensive , highly Scalable and real time web apps.

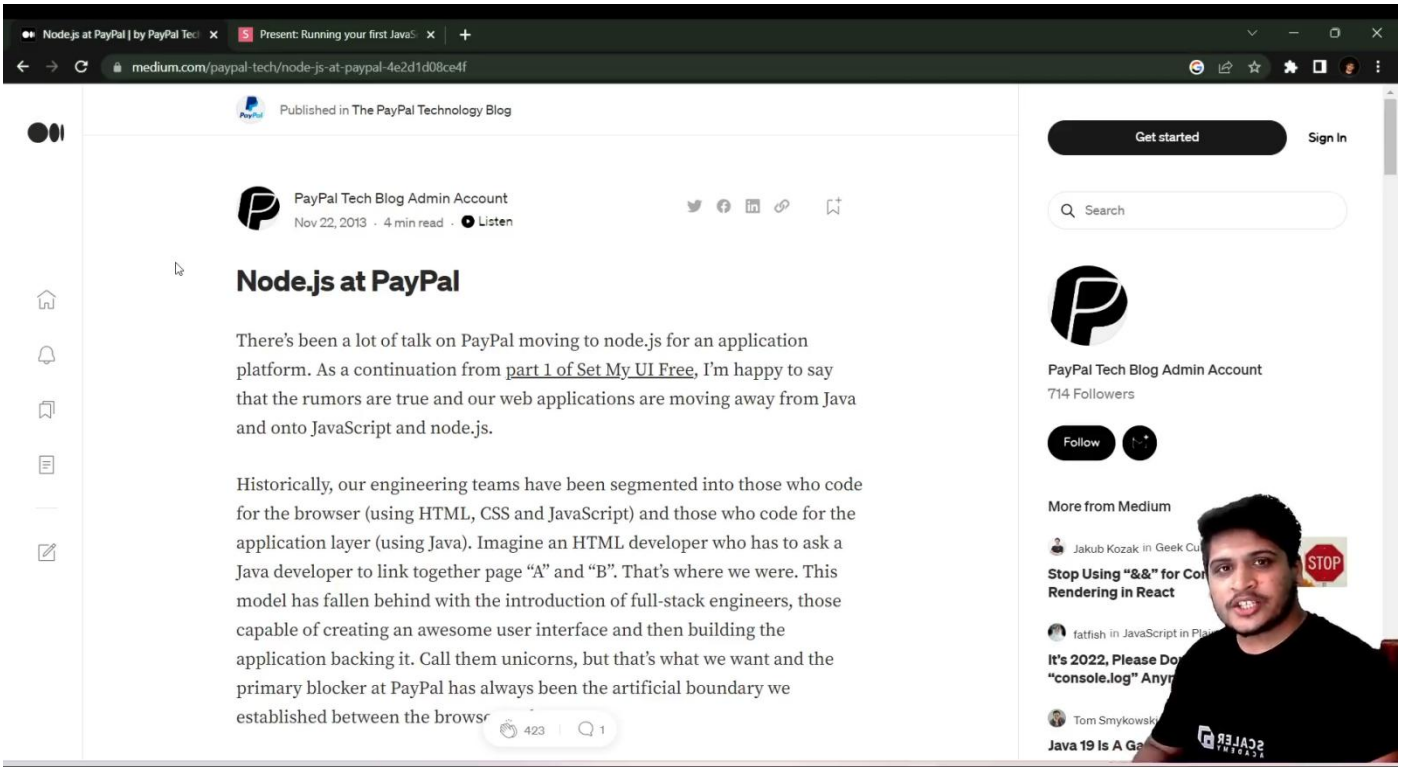
Ryan Dahl

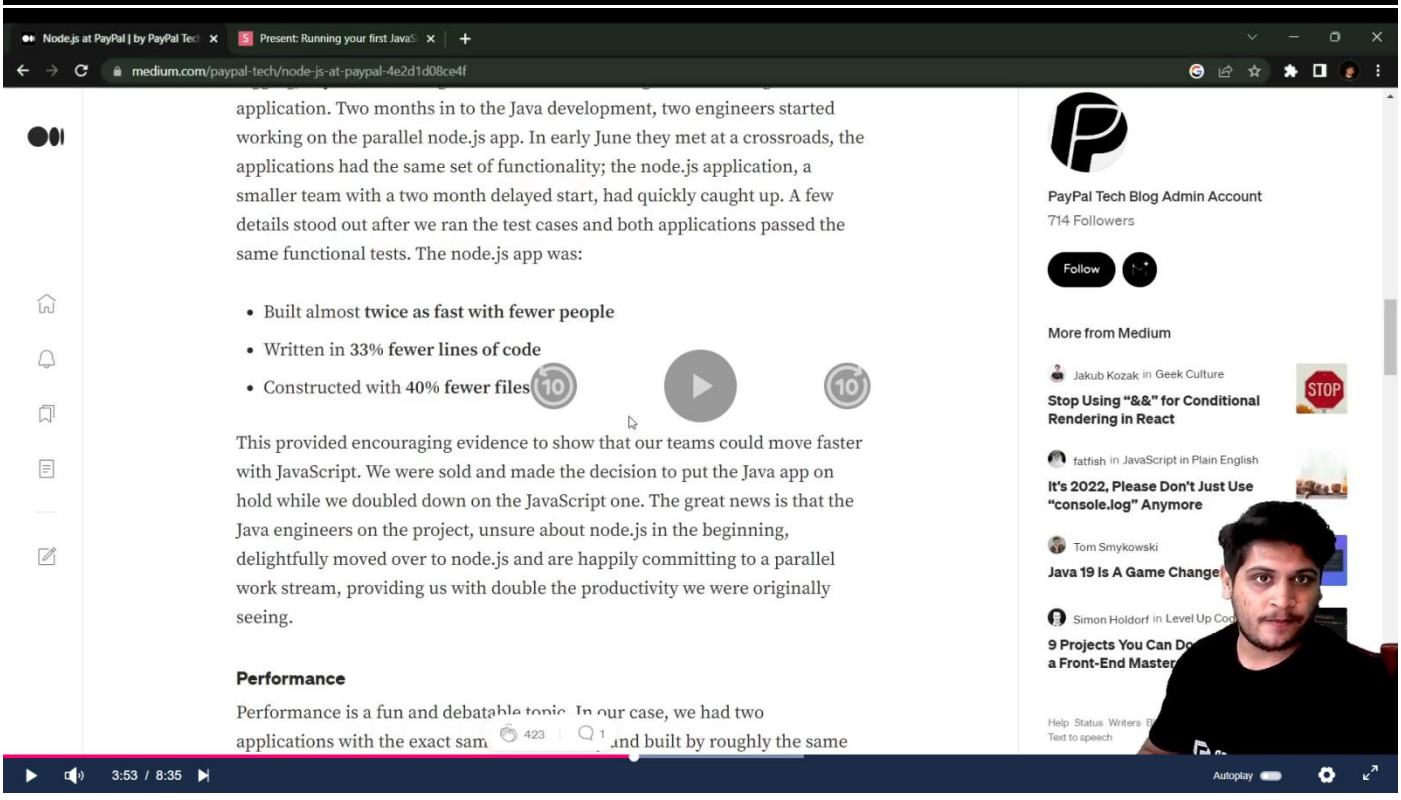
Creator of Node.js

SCALER

Why to use Node.js?

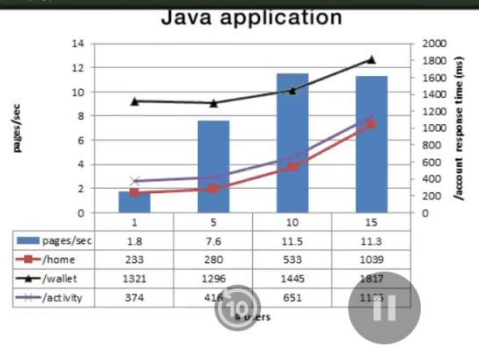
1. **Superfast and Highly Scalable**
2. **JavaScript Everywhere**
3. **Consistent and clean codebase**
4. **Huge Community Support**
5. **Large Ecosystem of Open Source Libs (NPM)**


Node.js at PayPal | by PayPal Tech
Present: Running your first JavaS
medium.com/paypal-tech/node-js-at-paypal-4e2d1d08ce4f
Published in The PayPal Technology Blog
Get started
Sign In
PayPal Tech Blog Admin Account
Nov 22, 2013 · 4 min read · Listen
Search
Node.js at PayPal
There's been a lot of talk on PayPal moving to node.js for an application platform. As a continuation from part 1 of Set My UI Free, I'm happy to say that the rumors are true and our web applications are moving away from Java and onto JavaScript and node.js.
Historically, our engineering teams have been segmented into those who code for the browser (using HTML, CSS and JavaScript) and those who code for the application layer (using Java). Imagine an HTML developer who has to ask a Java developer to link together page "A" and "B". That's where we were. This model has fallen behind with the introduction of full-stack engineers, those capable of creating an awesome user interface and then building the application backing it. Call them unicorns, but that's what we want and the primary blocker at PayPal has always been the artificial boundary we established between the brows
423
1
PayPal Tech Blog Admin Account
714 Followers
Follow
More from Medium
Jakub Kozak in Geek Cu
Stop Using "&&" for Co
Rendering in React
fatfish in JavaScript in Pla
It's 2022, Please Do
"console.log" Anyr
Tom Smykowski
Java 19 Is A Ga

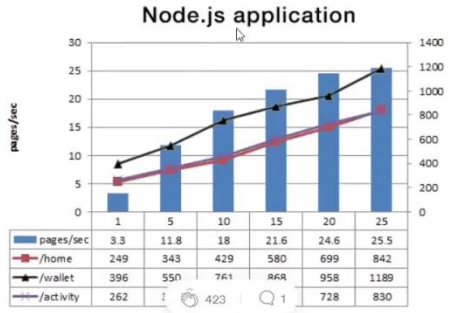

Node.js at PayPal | by PayPal Tech
Present: Running your first JavaS
medium.com/paypal-tech/node-js-at-paypal-4e2d1d08ce4f
application. Two months in to the Java development, two engineers started working on the parallel node.js app. In early June they met at a crossroads, the applications had the same set of functionality; the node.js application, a smaller team with a two month delayed start, had quickly caught up. A few details stood out after we ran the test cases and both applications passed the same functional tests. The node.js app was:
Built almost twice as fast with fewer people
Written in 33% fewer lines of code
Constructed with 40% fewer files
10
10
This provided encouraging evidence to show that our teams could move faster with JavaScript. We were sold and made the decision to put the Java app on hold while we doubled down on the JavaScript one. The great news is that the Java engineers on the project, unsure about node.js in the beginning, delightfully moved over to node.js and are happily committing to a parallel work stream, providing us with double the productivity we were originally seeing.
Performance
Performance is a fun and debatable topic. In our case, we had two applications with the exact sam
nd built by roughly the same
423
1
PayPal Tech Blog Admin Account
714 Followers
Follow
More from Medium
Jakub Kozak in Geek Culture
Stop Using "&&" for Conditional Rendering in React
fatfish in JavaScript in Plain English
It's 2022, Please Don't Just Use "console.log" Anymore
Tom Smykowski
Java 19 Is A Game Change
Simon Holdorf in Level Up Cod
9 Projects You Can Do
a Front-End Master
Help Status Writers B
Text to speech
3:53 / 8:35
Autoplay

Node JS Certification Course - Master the Fundamentals

medium.com/paypal-tech/node-js-at-paypal-4e2d1d08ce4f



	1	5	10	15
pages/sec	1.8	7.6	11.5	11.3
/home	233	280	533	1039
/wallet	1321	1296	1445	1617
/activity	374	416	651	1153



	1	5	10	15	20	25
pages/sec	3.3	11.8	18	21.6	24.6	25.5
/home	249	343	429	580	699	842
/wallet	396	550	761	868	958	1189
/activity	262	423	1		728	830

PayPal Tech Blog Admin Account
714 Followers
Follow

More from Medium

Jakub Kozak in Geek Culture
Stop Using "&&" for Conditional Rendering in React

fatfish in JavaScript in Plain English
It's 2022, Please Don't Just Use "console.log" Anymore

Tom Smykowski
Java 19 Is A Game Chan

Simon Holdorf in Level Up
9 Projects You Can Do a Front-End Master

Help Status Writers Bl
Text to speech

4:01 / 8:35

Autoplay

Press Esc to exit full screen

SCALER

Features of Node.js-

1. **Single-Threaded**
2. **Asynchronous**
3. **Event Driven**
4. **NPM**
5. **Highly Scalable**
6. **Performance (C++ and V8 Engine)**

2. *Getting Started with Node JS*:

a) Install Node.js and VScode.

b) First Node program :

i. Create a js file with a function:p1_test.js

```
console.log('Hello World');

function sayHello(){
    console.log('Hello from anshad');
}
sayHello();
```

ii. To Execute type "node name_of_program.js" : node p1_test.js

c) 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]
}
```

3. *Node Module system*:

a) 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
```

b) Modules and Modularity :

i. 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
}
```

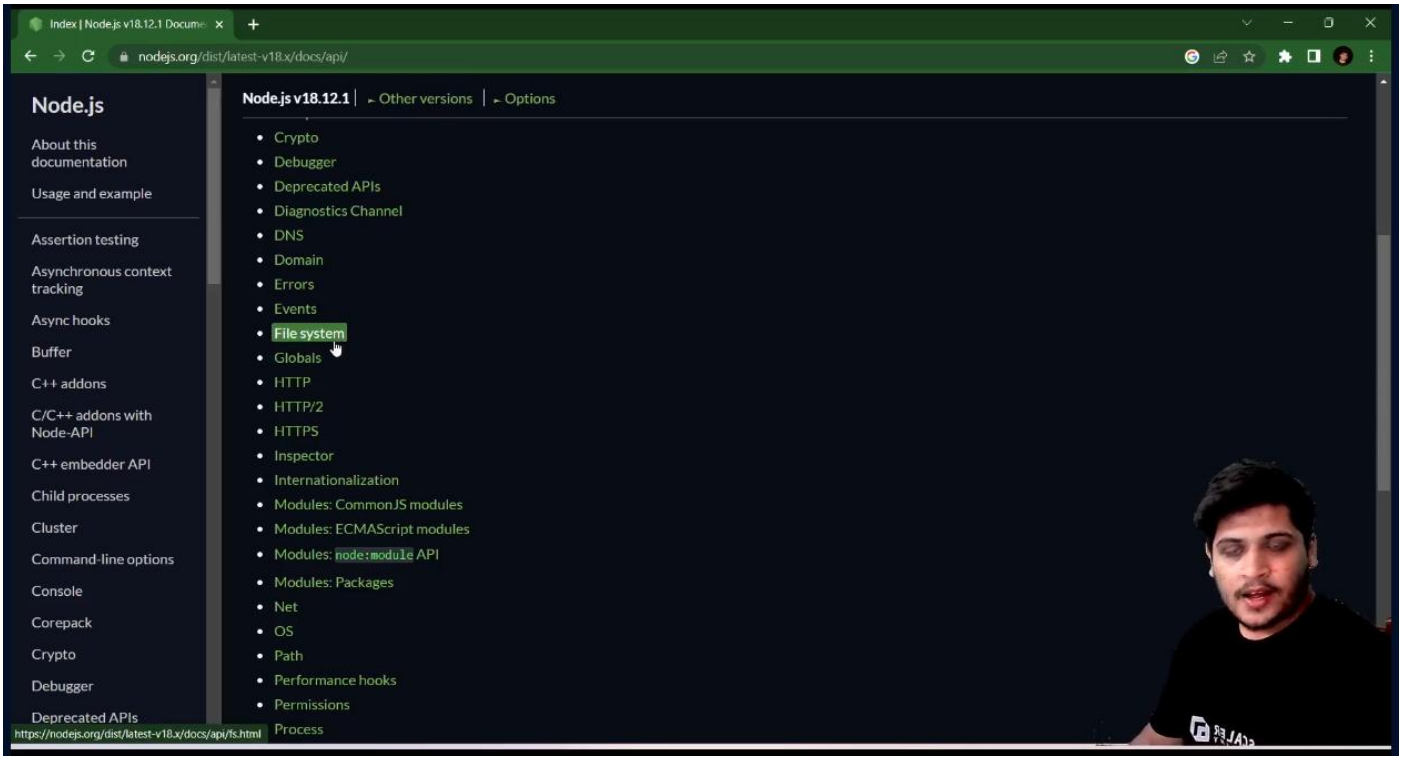
ii. Create main file 'modularity.js' file :

```
//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);
```

c) Introduction to Node Modules :



d) 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.
```

e) OS Module :

Code	Output

`//OS Module :(to get the information of your current system)`
`const os = require('os');//Import OS module`
	`PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_3> node .\osModule.js`
	`x64`
	`win32`

```
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.
```

```
{
  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
```

f) 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.t
xt');
```

```
//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.t
xt');
```

```
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
*/
```

g) 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
*/
```

h) FS Module with Directories :

```
// //FS Module With Directory:

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

```
// 1.First Create Directory [mkdirSync()] :
fs.mkdirSync('myDirectory');
```

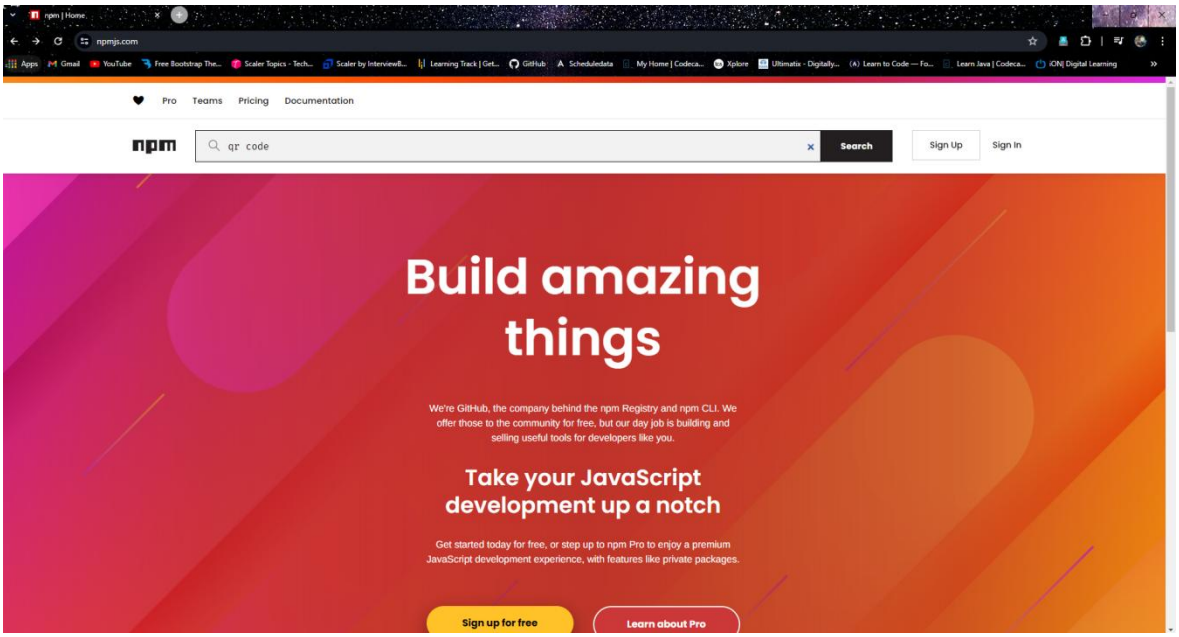
```
// //2.Check the content inside of a  Directory [readdirSync()]:
let folderPath =
'E:\\MCA\\COURSES\\NODE_JS\\nodejs_certification@scalar\\Module_3\\myDirectory';
let folderContent = fs.readdirSync(folderPath);
console.log('Folder Content : ' , folderContent);
/*OUTPUT :
Folder Content :  [ 'f1.txt', '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');
```

4. Node Package Manager(NPM):[Third party packages that we can use in JS]

a) Introduction to NPM :(We can use npm install different packages)



> **Note:** to download the latest version of npm, on the command line, run the following command:
>
> `npm install -g npm`

b) 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)
```

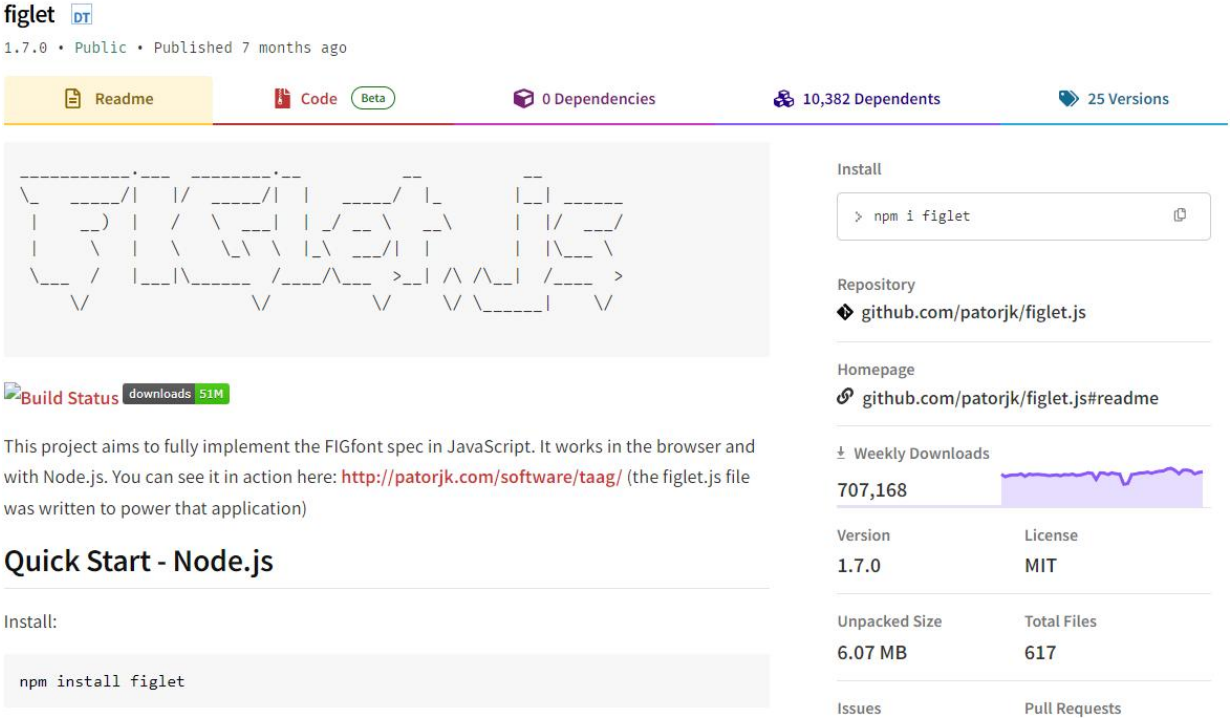
```
About to write to
E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_4\npm_package\package.json:
{
  "name": "npm_package",
  "version": "1.0.0",
  "description": "it is my package",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
Is this OK? (yes)
```

2) "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"
}
```

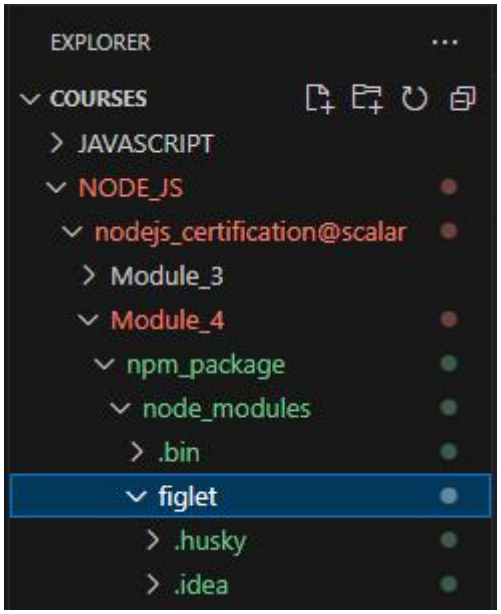
3) Example : Installing 'figlet' package :

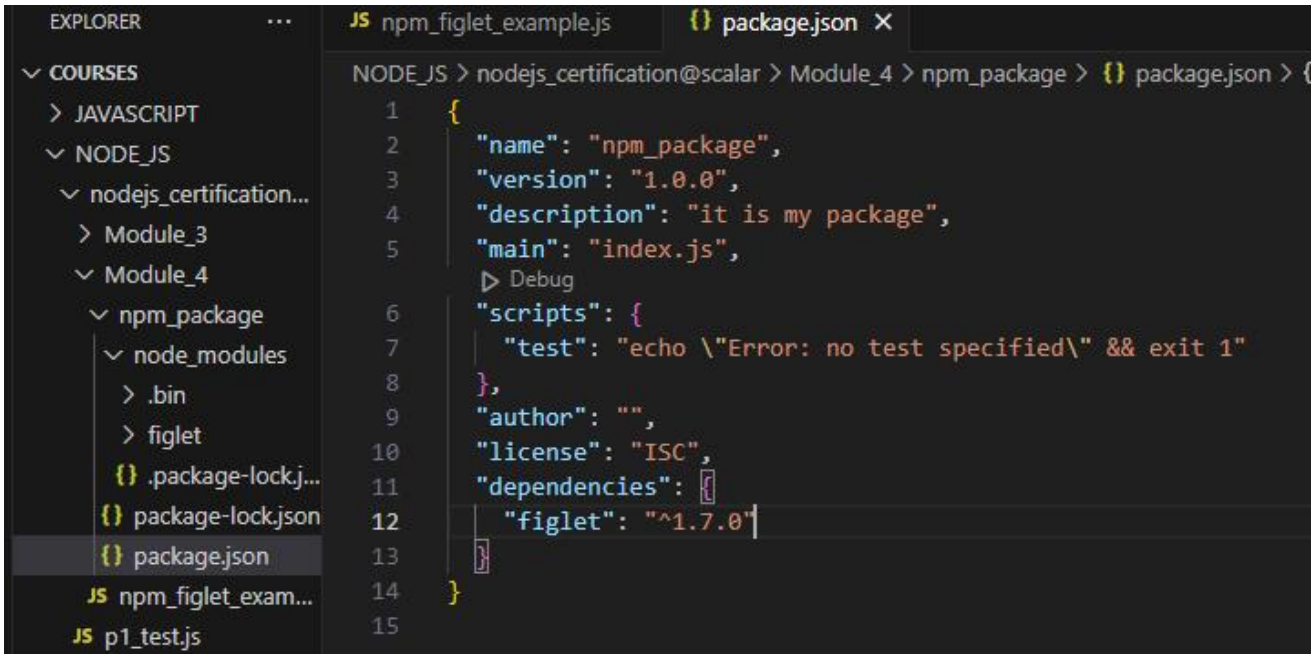
i. Goto npm site and type figlet :



ii. Install figlet :

```
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_4\npm_package> npm i figlet

added 1 package, and audited 2 packages in 4s

found 0 vulnerabilities
```





iii. Add example code in a new file 'npm_figlet_example.js' :

Quick Start - Node.js

Install:

```
npm install figlet
```

Simple usage:

```
var figlet = require("figlet");

figlet("Hello World!!", function (err, data) {
  if (err) {
    console.log("Something went wrong...");
    console.dir(err);
    return;
  }
  console.log(data);
});
```

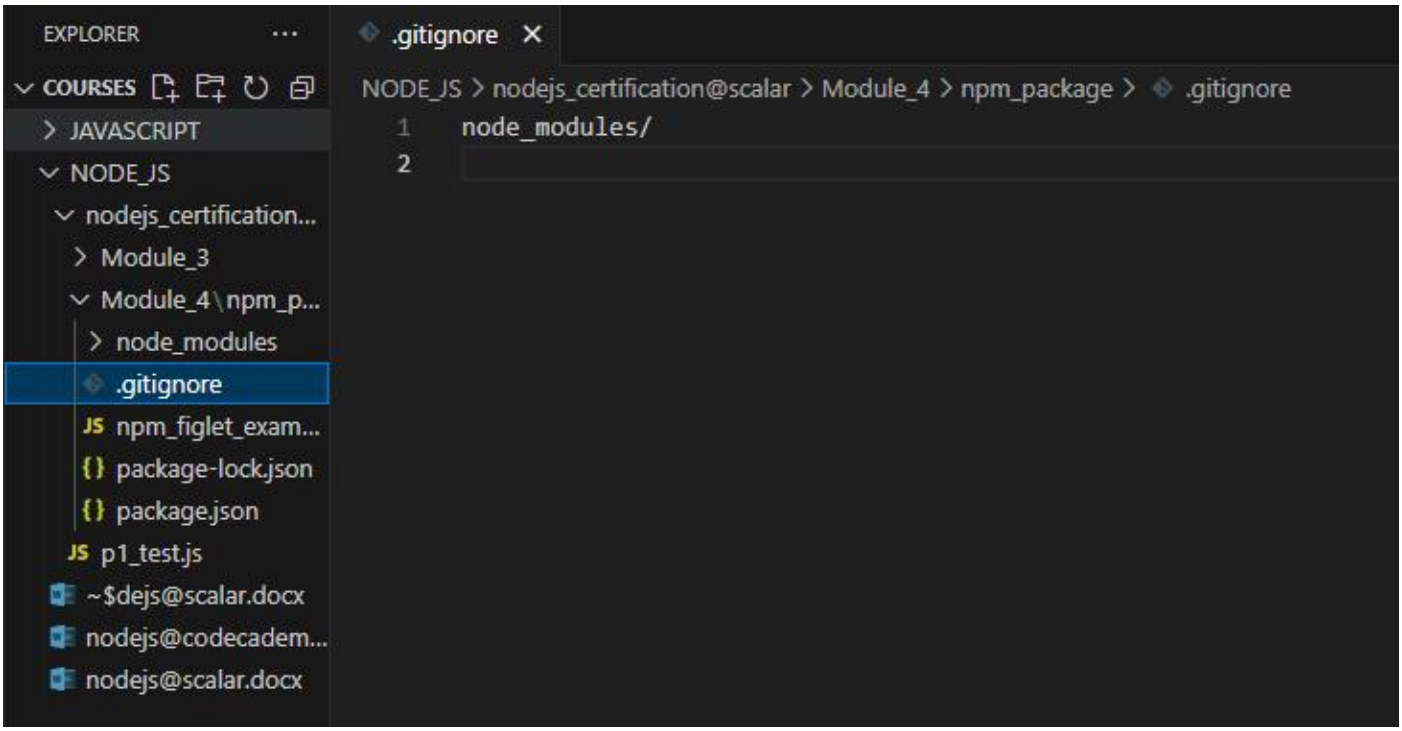
That should print out:

```
  _   _          _ _                  __        __         _     _ _ _
 | | | | ___ | | | ___   \ \      / /__  _ __| | __| | | |
 | |_| |/ _ \ | |/ _ \   \ \ /\ / / _ \| '__| |/ _` | | |
 |  _  |  __/ | | (_) |   \ V  V / (_) | |  | | (_| |_|_|
 |_| |_|\___|_|_|\___/     \_/\_/ \___/|_|  |_|\__,_(_|_)
```

```
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>
```

c) All about '.gitignore ' :



Git Ignore

Change Platform

GitHub

Bitbucket

GitLab

When sharing your code with others, there are often files or parts of your project, you do not want to share.

Examples

- log files
- temporary files
- hidden files
- personal files
- etc.

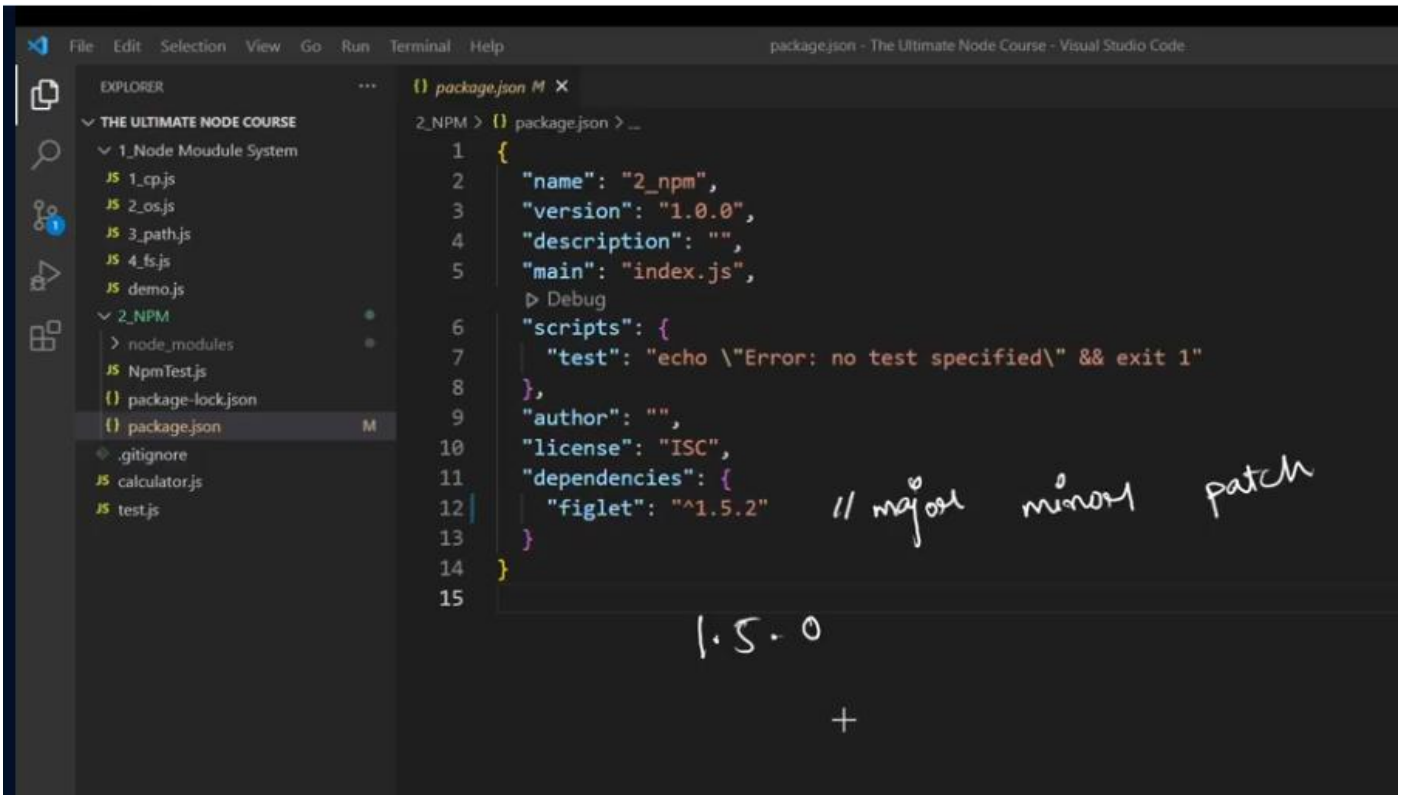
Git can specify which files or parts of your project should be ignored by Git using a `.gitignore` file.

Git will not track files and folders specified in `.gitignore`. However, the `.gitignore` file itself **IS** tracked by Git.

Example

```
# ignore ALL .log files
*.log

# ignore ALL files in ANY directory named temp
temp/
```

d) Semantic Versioning :(major , minor , patching)



e) Publishing your own NPM package :

i. First create a folder ‘scalarCalc’ .

ii. Then install npm : ‘npm init’.

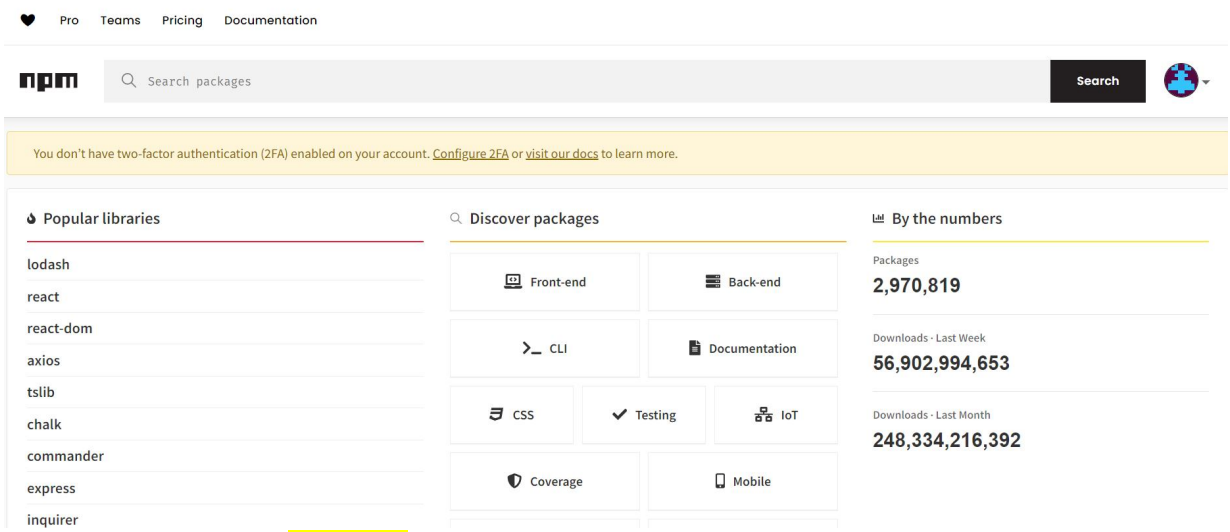
```
package name: (scalercalc)
version: (1.0.0)
description: This is a calculator module developed during node course by scalar topics.
entry point: (index.js)
test command:
git repository:
keywords:
author: Anshad
```

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

```
npm adduser
```

iv. Now Login to that .

```
npm login
```



v. 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
}
```

vi. Now publish it :

```
npm publish
```

Now Package is published to npm.

vii. Now GOTO npm site and type your package name : 'scalarCalc' .

viii. Where you can see the published npm.

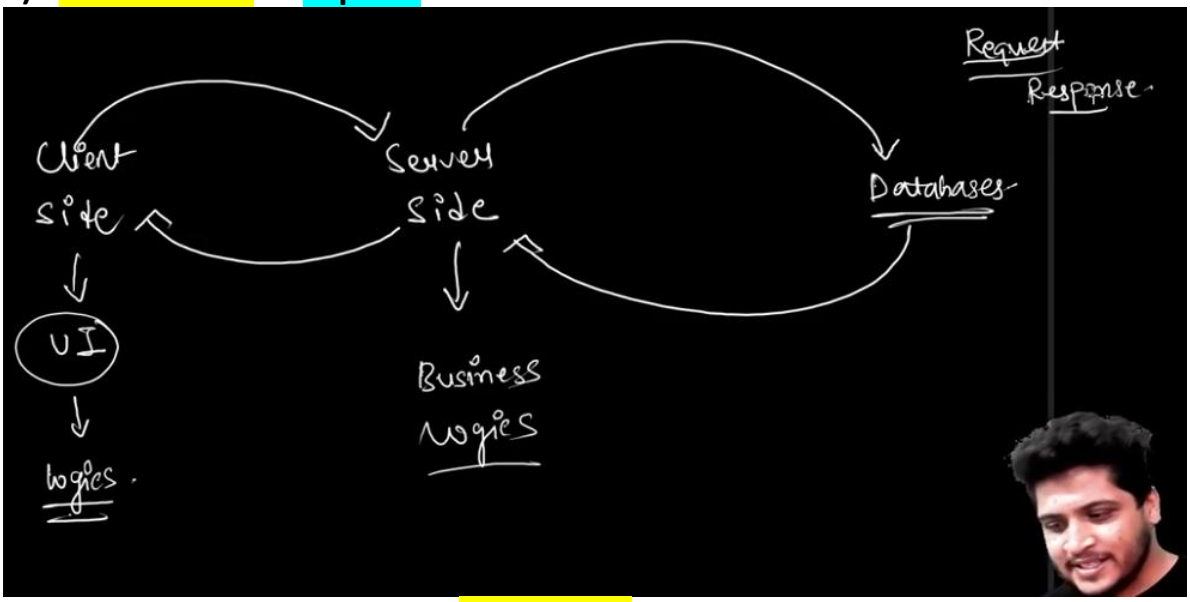
ix. Now You can use it by :

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');
```

5. Getting Started with Express :

a) 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'));
```

3) **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!'); });
```

b) Express Installation :

i. Create a Folder 'Module_5_Express'.

ii. Open Terminal and type :

```
npm init --yes
```

iii. GOTO npm website and search for 'express' and use the code to install express.

```
npm i express
```

iv. or You can go directly to express webpage and use the code to install.

```
$ npm install express --save
```

```
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express> npm init --yes
Wrote to E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express\package.json:

{
  "name": "module_5_express",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express> npm i express

added 64 packages, and audited 65 packages in 5s

12 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express>
```

c) Let's Use Express :

i. 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'));
```

Example 1: Below is the code example of **app.get()** Function:

javascript

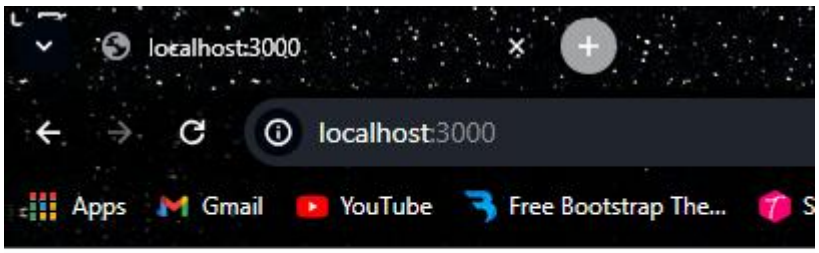
```
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
    res.send("GET Request Called")
})

app.listen(PORT, function (err) {
    if (err) console.log(err);
    console.log("Server listening on PORT", PORT);
});
```

ii. **Now run app.js :**

```
node app.js
```

```
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express> node app.js
Port is Running on 3000
```

iii. **Goto browser and type 'localhost:3000'.[which will display the request message we passed]:**



Hello from Scalar Topics

iv. **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');
```

```
})
```

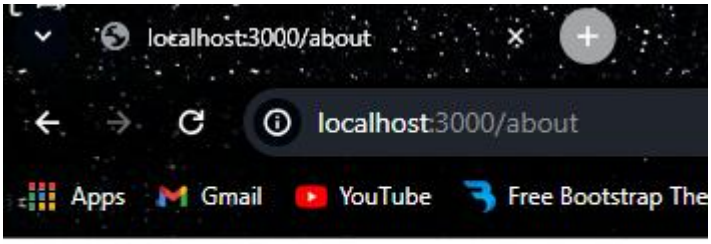
```
//Now if you run the code you will not see anything in output .
//This is bcs you have to Specify the PORT :
app.listen(3000,()=> console.log('Port is Running on 3000'));
```

v. **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
```

```
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express> node app.js
Port is Running on 3000
```

vi. **Goto browser and type 'localhost:3000/about':**



We create Impact

d) **Nodemon :(**Used to automatically restarting the node application when file changes in the directory are detected.**)**

i. **To use First install Nodemon:['npm install -g nodemon'-To install globally]**

```
PS E:\MCA\COURSES\NODE JS\nodejs certification@scalar\Module 5 Express> npm install -g nodemon

added 29 packages in 3s

4 packages are looking for funding
  run `npm fund` for details
```

\

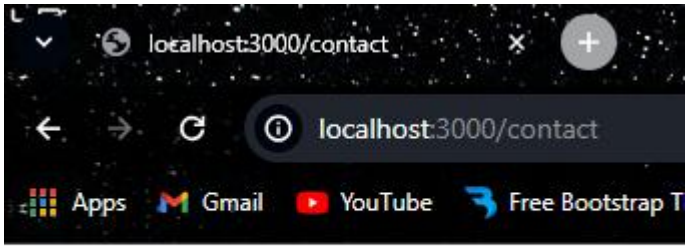
ii. **Now you have to run your js file using nodemon : ['nodemon app.js']**

```
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express> nodemon app.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Port is Running on 3000
```

iii. **Now make changes and See it automatically updates :**

```
app.get('/contact',(req , res) =>{
    res.send('Contact us on abcd@gmail.com ');
})
```

```
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_5_Express> nodemon app.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Port is Running on 3000
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Port is Running on 3000
```



Contact us on abcd@gmail.com

e) 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**].**

i. **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'));
```

ii. **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].**

iii. **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}`));
```

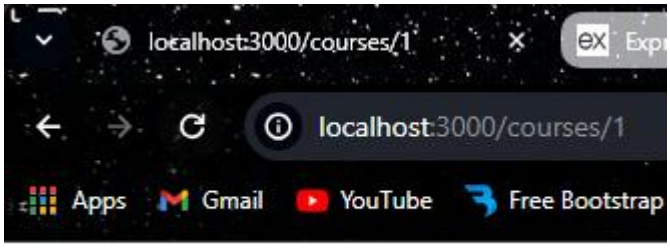
f) Routes Parameters :

i. Route parameters are named URL segments that are used to capture the values specified at their position in the URL .

ii. 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.

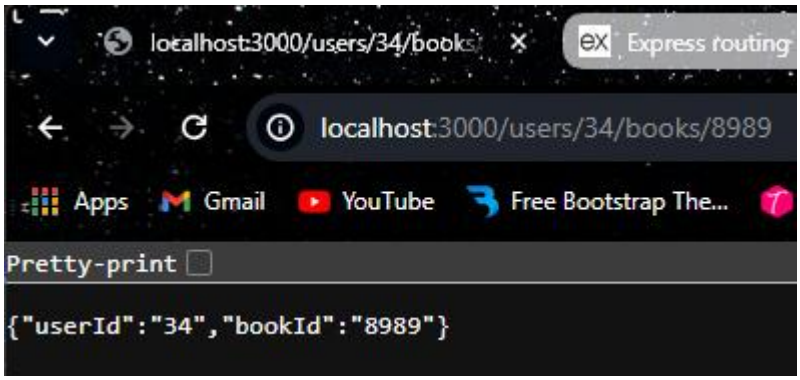
iii. Create same "app3_route_para.js" with same code and add new route with parameters:

```
//Route Parameters :
app.get('/courses/:id',(req , res) =>{
    res.send(req.params.id); //To get id as response
    console.log(req.params);
})
```



1

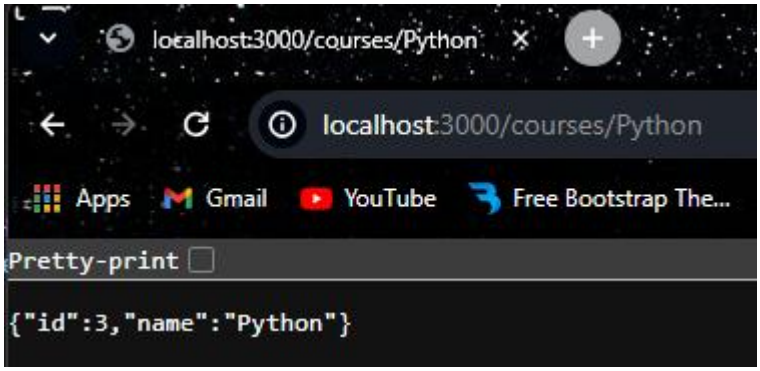
```
//Example 2 :
app.get('/users/:userId/books/:bookId', (req, res) => {
    res.send(req.params)
  })
```



g) Handling Multiple Routes :

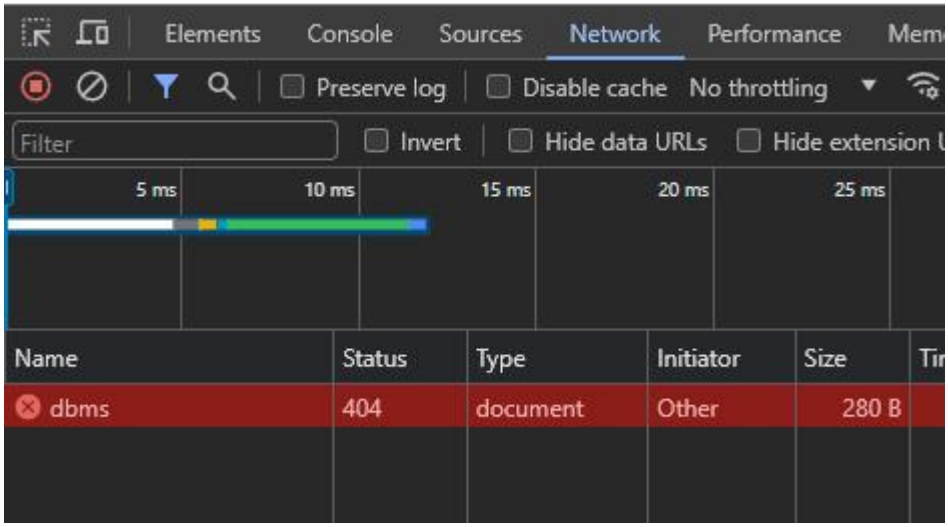
i. Set objects inside an array with id and name. And then define route:

```
//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);
})
```



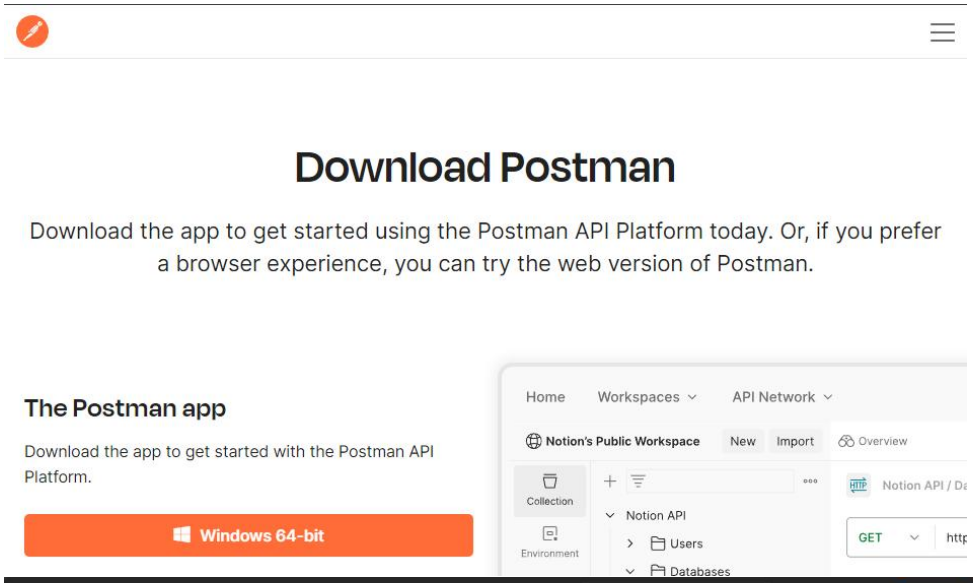
Handling Errors:

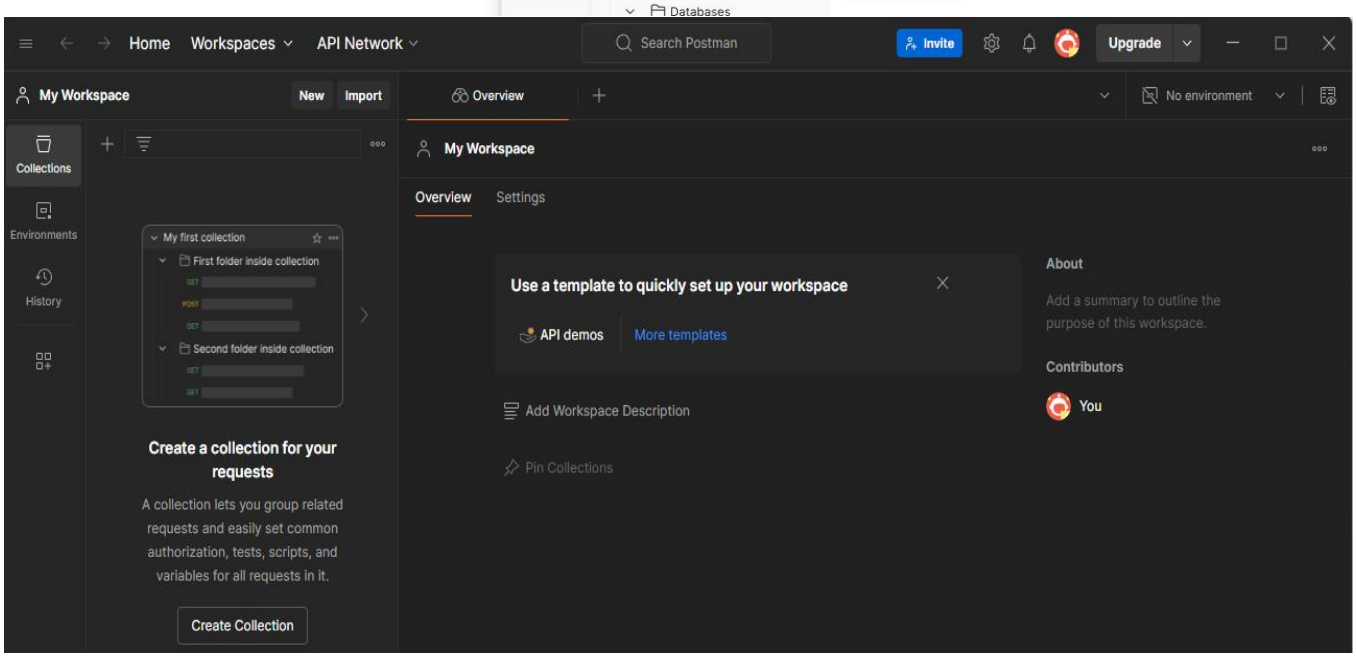
The course you are looking for does not exist



h) **Postman** :(Postman is an application that allows the testing of web APIs.)

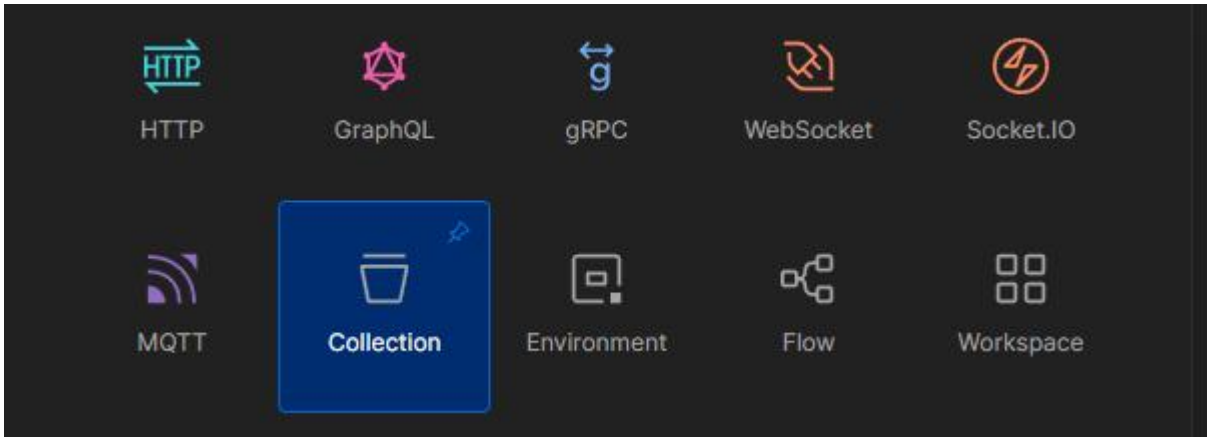
i. Goto postman website,signup and Download it.



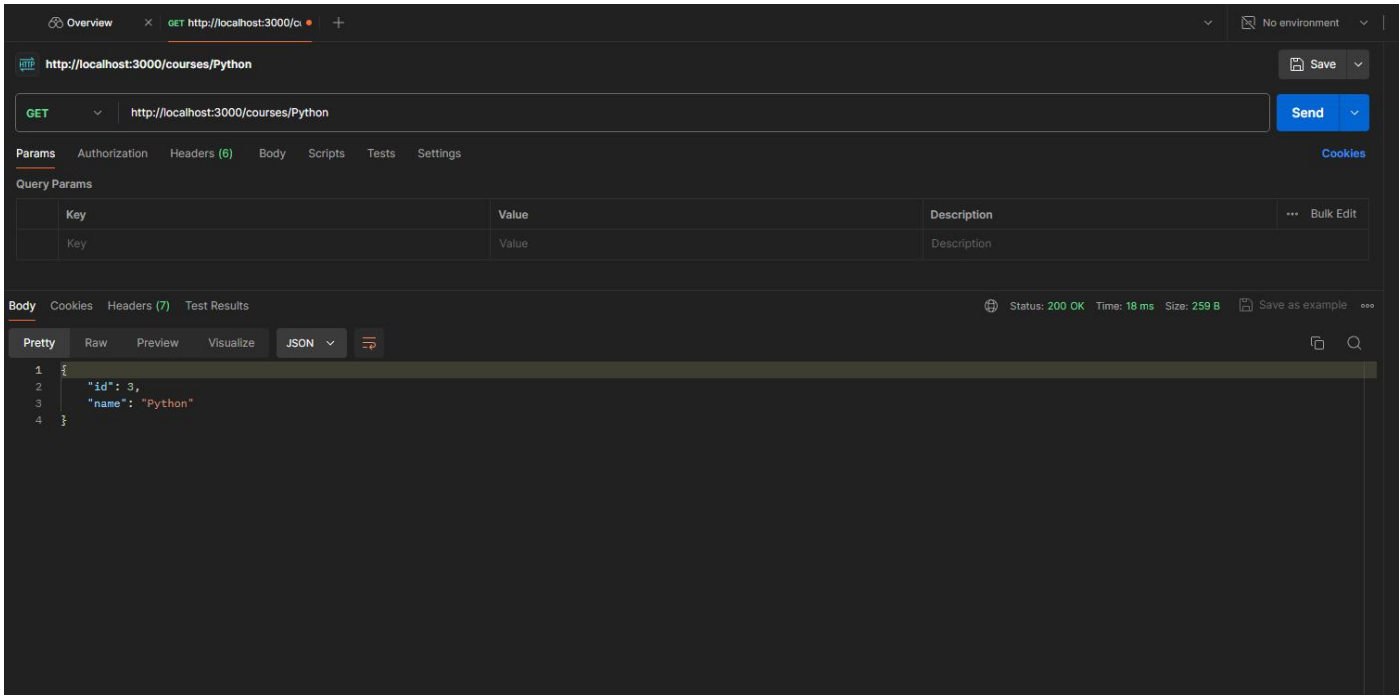


ii. How to Use ?

- Click 'New':



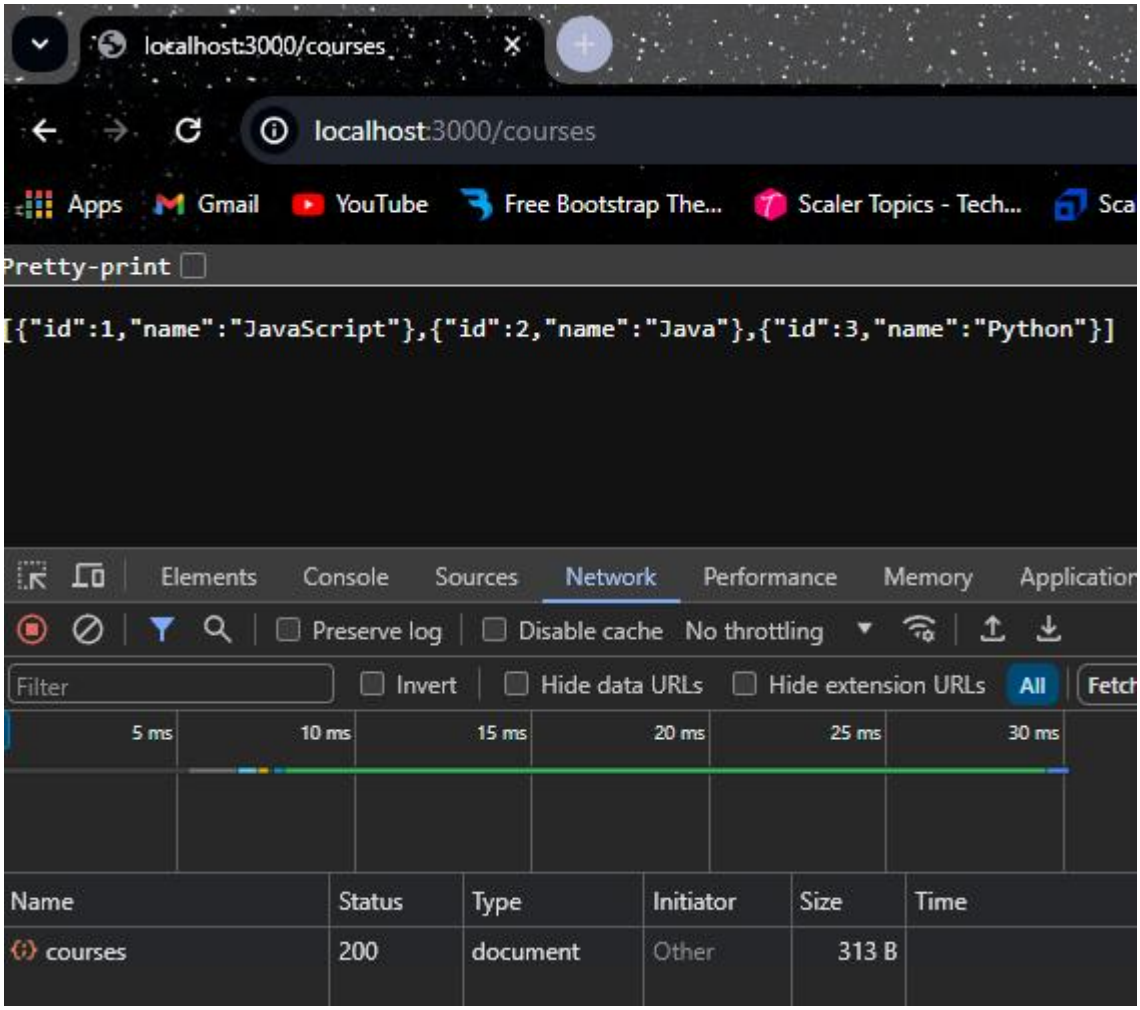
- Then Paste link and press send:



i) Http Post Method :[To Create]

i. 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);
})
```



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

iii. To pass you need to use a Middleware.

iv. 'app.use()' to use middleware:

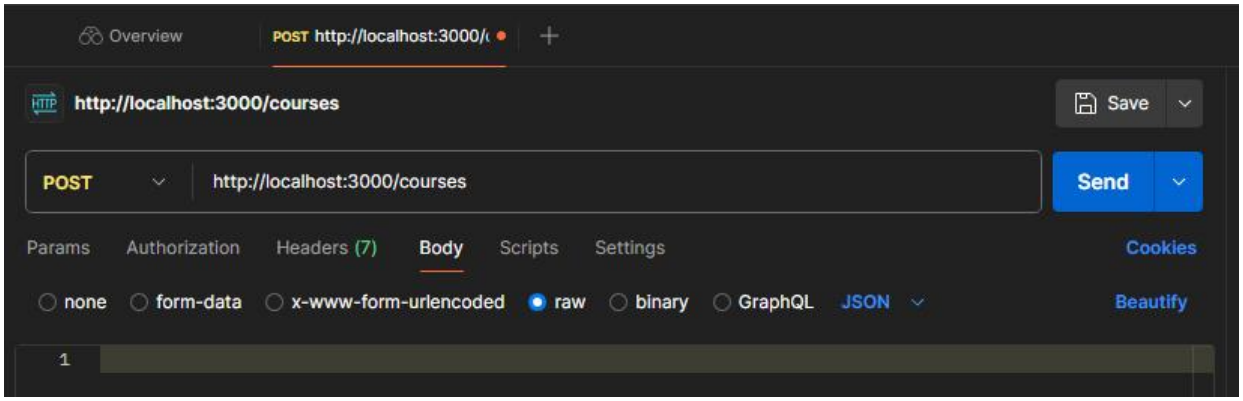
```
//To use middleware:
app.use(express.json());
```

v. Pushing to courses collection :

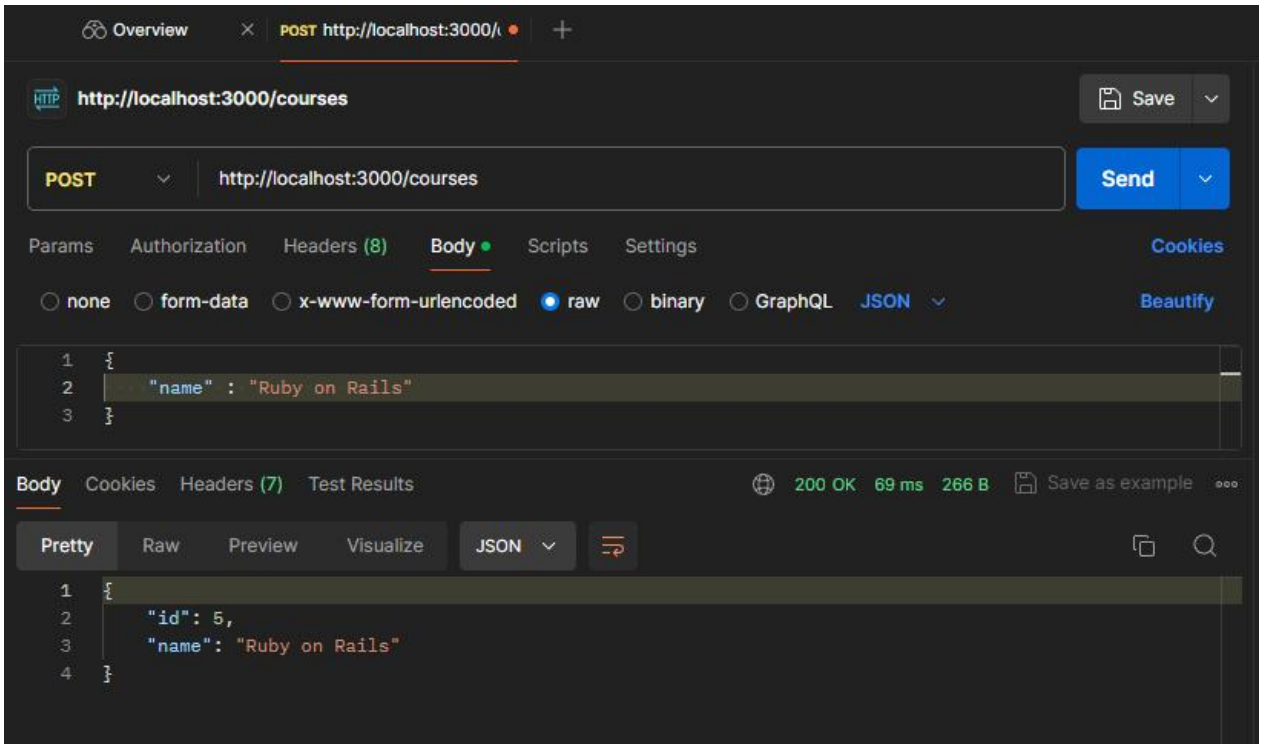
```
//post()
//Whenever we are  using post(whenever we need to create data),when we are
using Express you have pass to JSON
//To pass you need to use a Middleware.
app.post('/courses' , (req , res) =>{
    const course = {
        id : courses.length +1,
        name : req.body.name
```

```
    }
    courses.push(course);//pushing courses collection
    res.send(course);
})
```

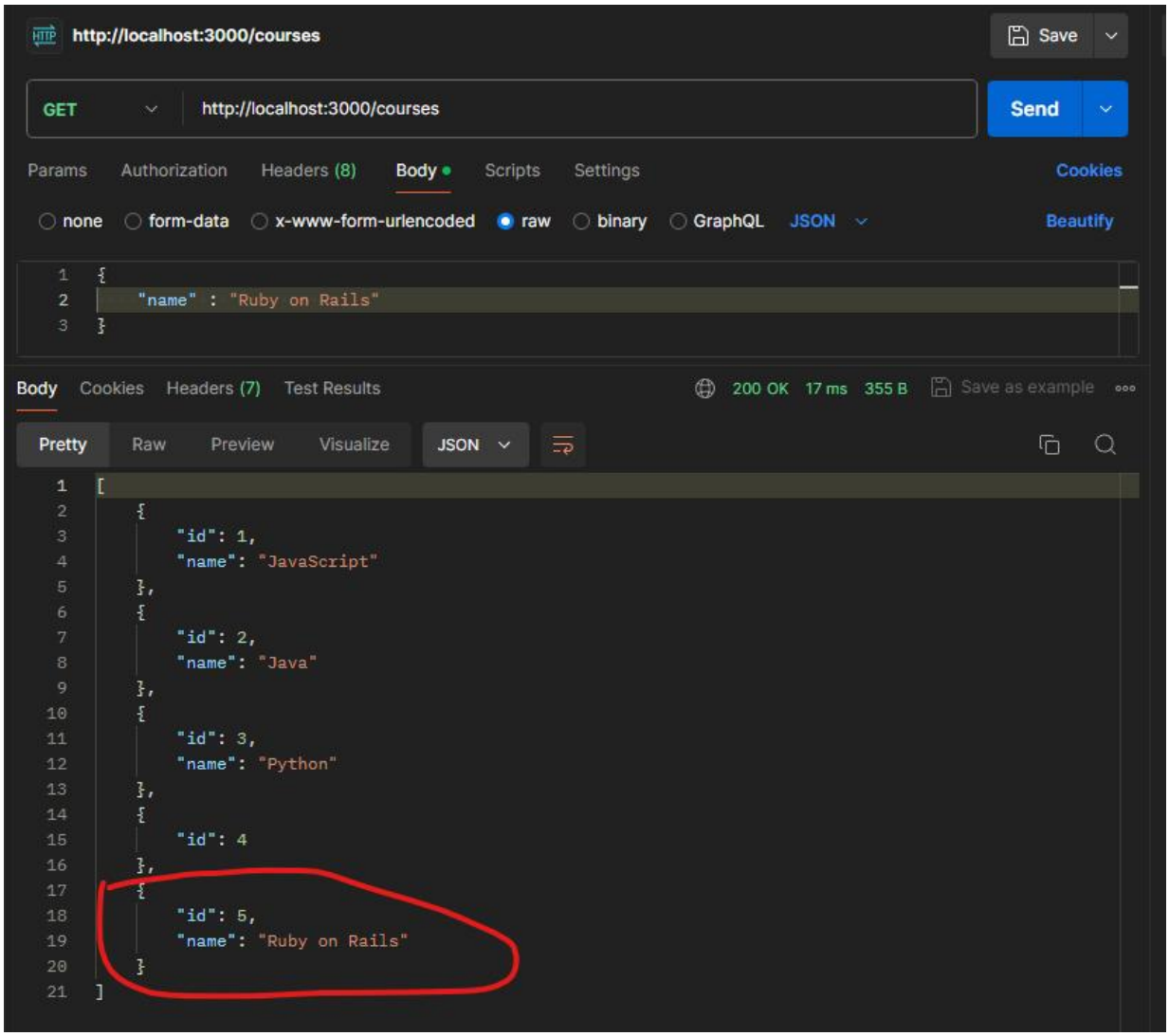
vi. Now GOTO postman and test this(send).



- Goto 'Body' and select 'raw':



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



j) Http Put Method :[To UPDATE]-(Used to update already existing entries)

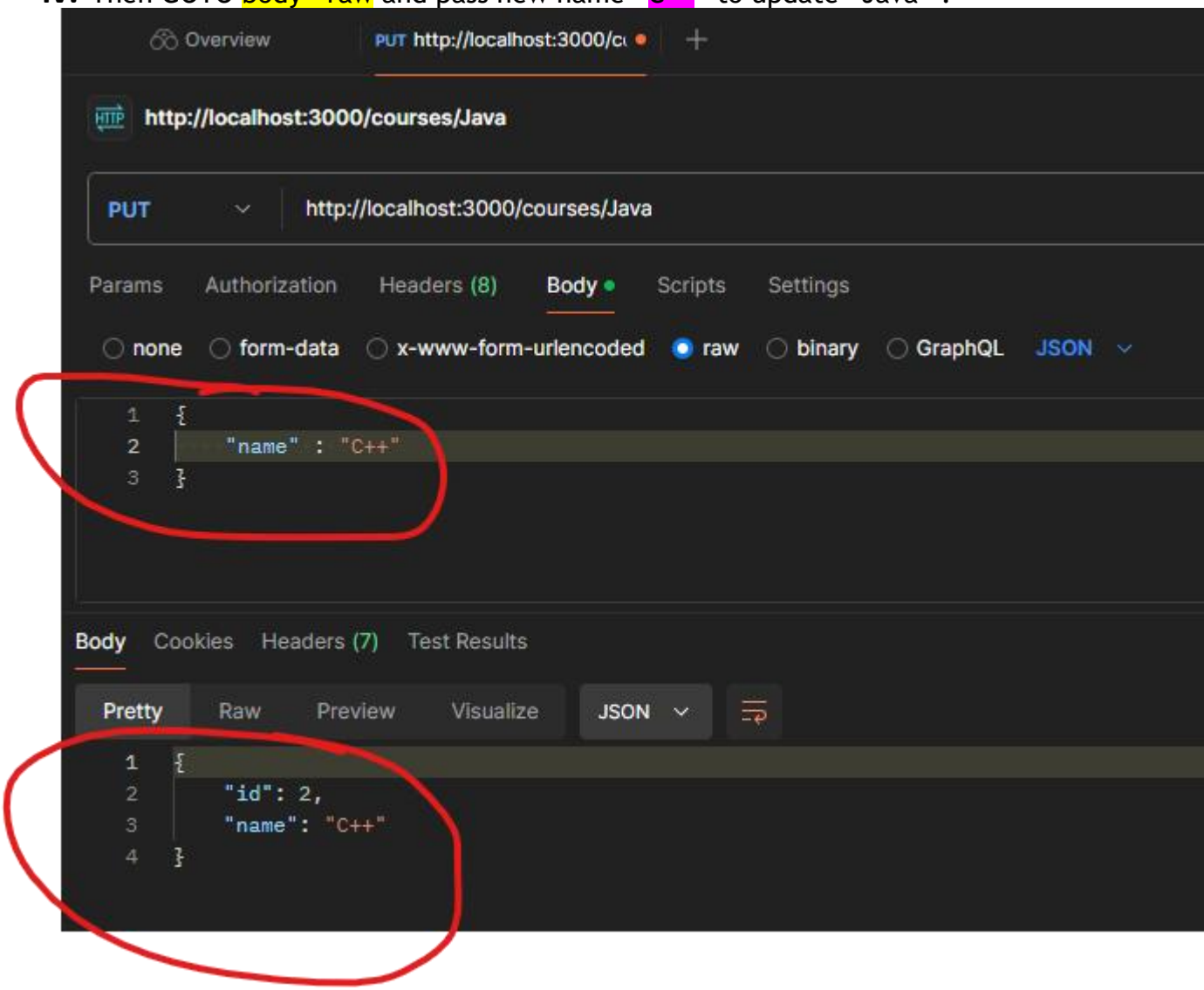
i. Add a 'app.put()' route:

```
// 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);
})
```

ii. Now run 'nodemon app5_http_post.js' .

iii. Then Open Postman and select PUT at :(http://localhost:3000/courses/Java).

iv. Then GOTO body->raw and pass new name "C++" to update "Java" .



k) Http Delete Method :

i. First change courses array from 'const' to 'let':

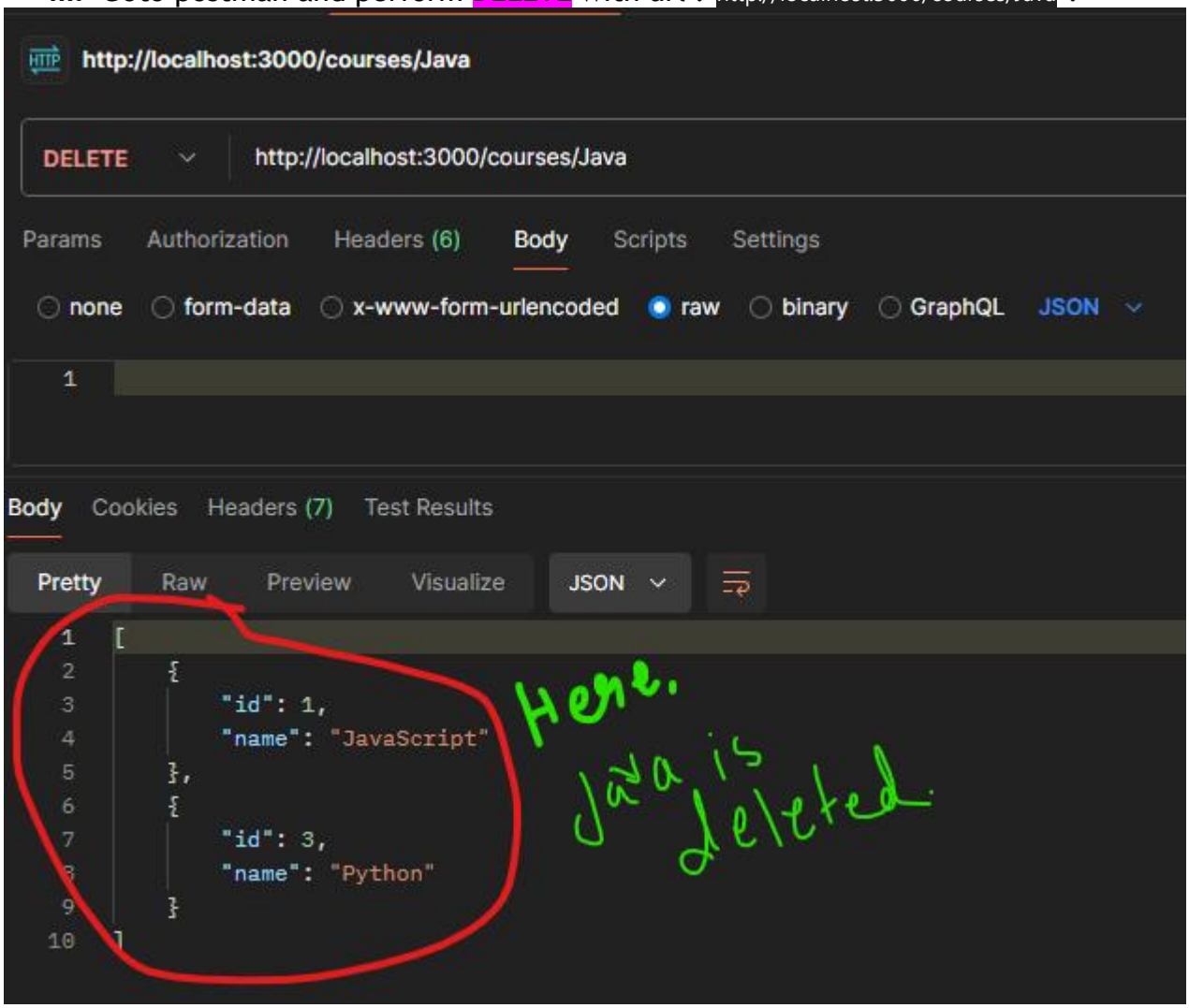
```
//Handling Multiple routes:
let courses = [ //change const to let for delete
    {id:1 , name : 'JavaScript'},
    {id:2 , name : 'Java'},
    {id:3 , name : 'Python'},
]
```

ii. 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);
})
```

iii. Goto postman and perform DELETE with url :'http://localhost:3000/courses/Java'.

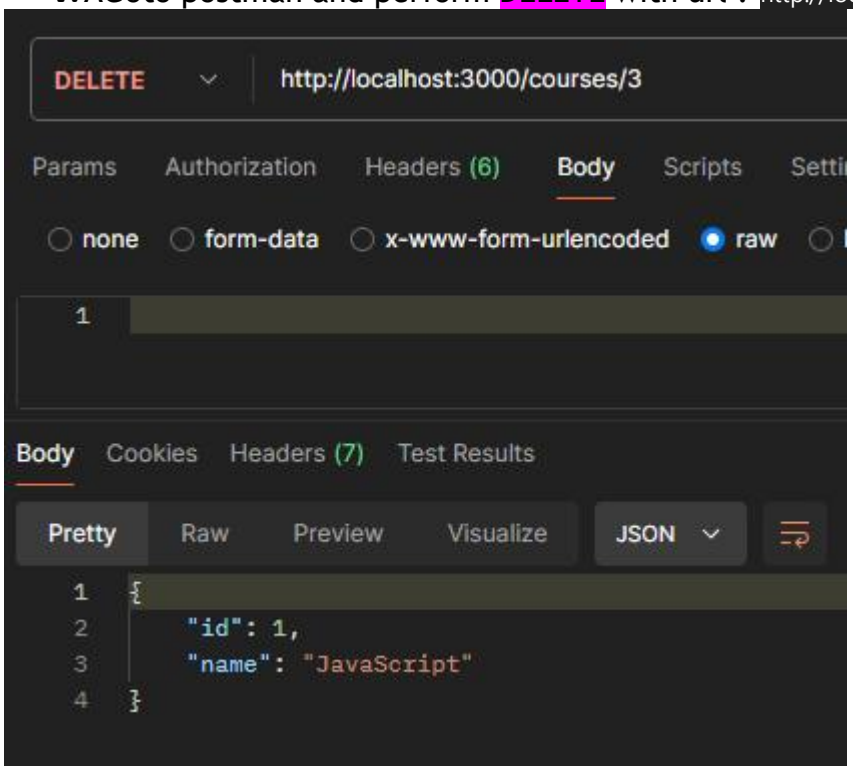


iv. Here we passed string(name) to DELETE content.But it is not not recommended.So lets use 'id':

```
//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) return 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);
})
```

v. 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}`));
```

6. Middlewares in Express :

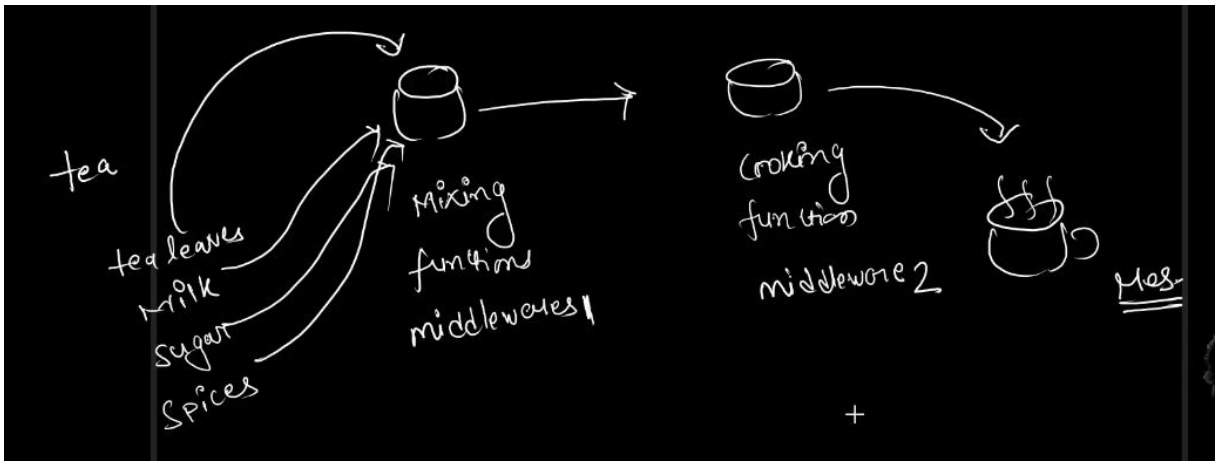
```
app.use();//To use middleware
```

a) What are Middlewares ?.

i. 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.

ii. Middleware functions provide access to the request (req) and response (res) objects. They are the perfect way to modify the req and res objects.

iii. For example, we can fetch user details after the user has logged in and save these details in the request object.

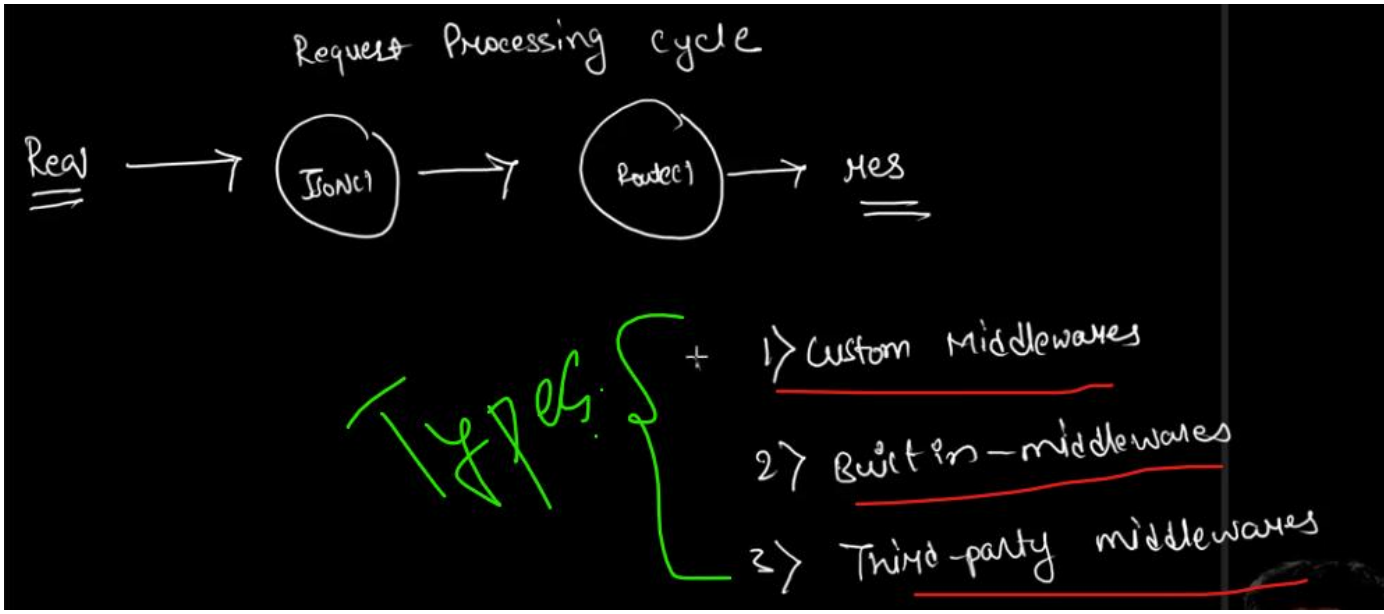


iv. Json() as Middleware : In post() request , We have used 'json()', to pass the data into 'json()' ,That is we passed value by typing in body>raw ('req.body.name') which were passed to json() and used to create the object .SNIPPET CODE:

```
//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);
})
```

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


Request Processing cycle
Req
JSON()
Route()
Res
Types:
1) Custom Middlewares
2) Built in - middlewares
3) Third-party middlewares

b) Custom Middlewares.

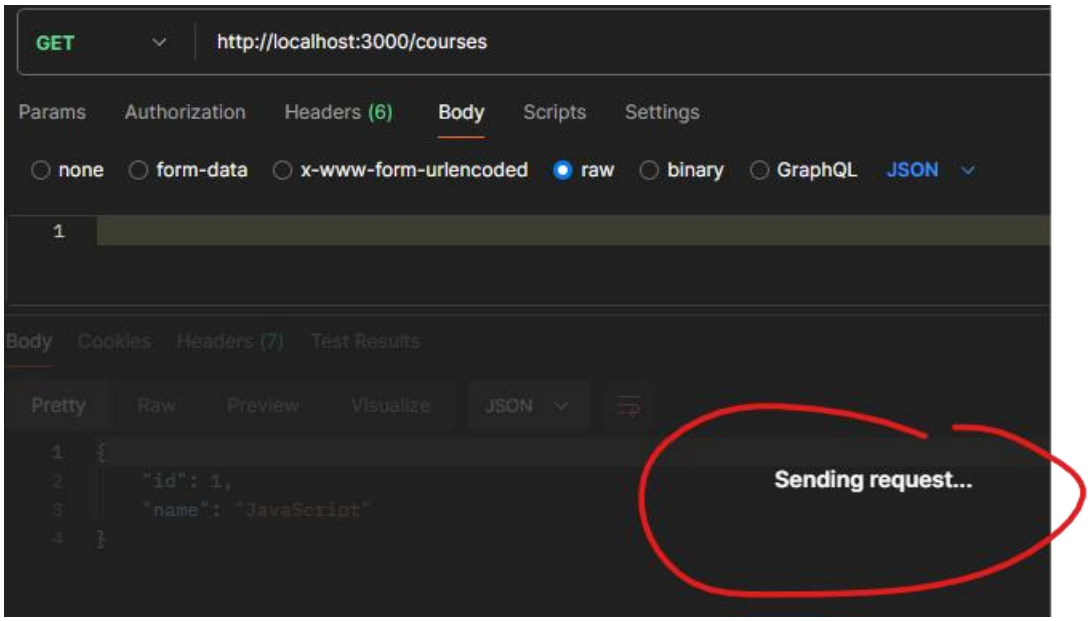
i. First define a custom middleware.

```
//Custom middleware :
app.use(function(req ,res ){
    console.log('I am Custom Middleware');
});
```

ii. Then run it : 'nodemon app.js':

iii. Then Goto postman and try to GET using url :'http://localhost:3000/courses'.

iv. Here we doesn't get any response.Its just keep sending request.



v. And in console output we will have the output:

```
Port is Running on 3000
I am Custom Middleware
```

vi. 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();
});
```

vii. Now if you use GET in postman you will get the response.

viii. Here we should not specify custom middlewares like this.Instead we have to do in separate file .

ix. For that create a separate folder 'middlewares' .

x. 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;
```

xi. Now import those files to 'app.js':

```
const myMiddlewareFunction = require('./middlwares/middle');//Importing custom
middleware from folder
const myMiddlewareFunction2 = require('./middlwares/middle2');//Importing
middle2
```

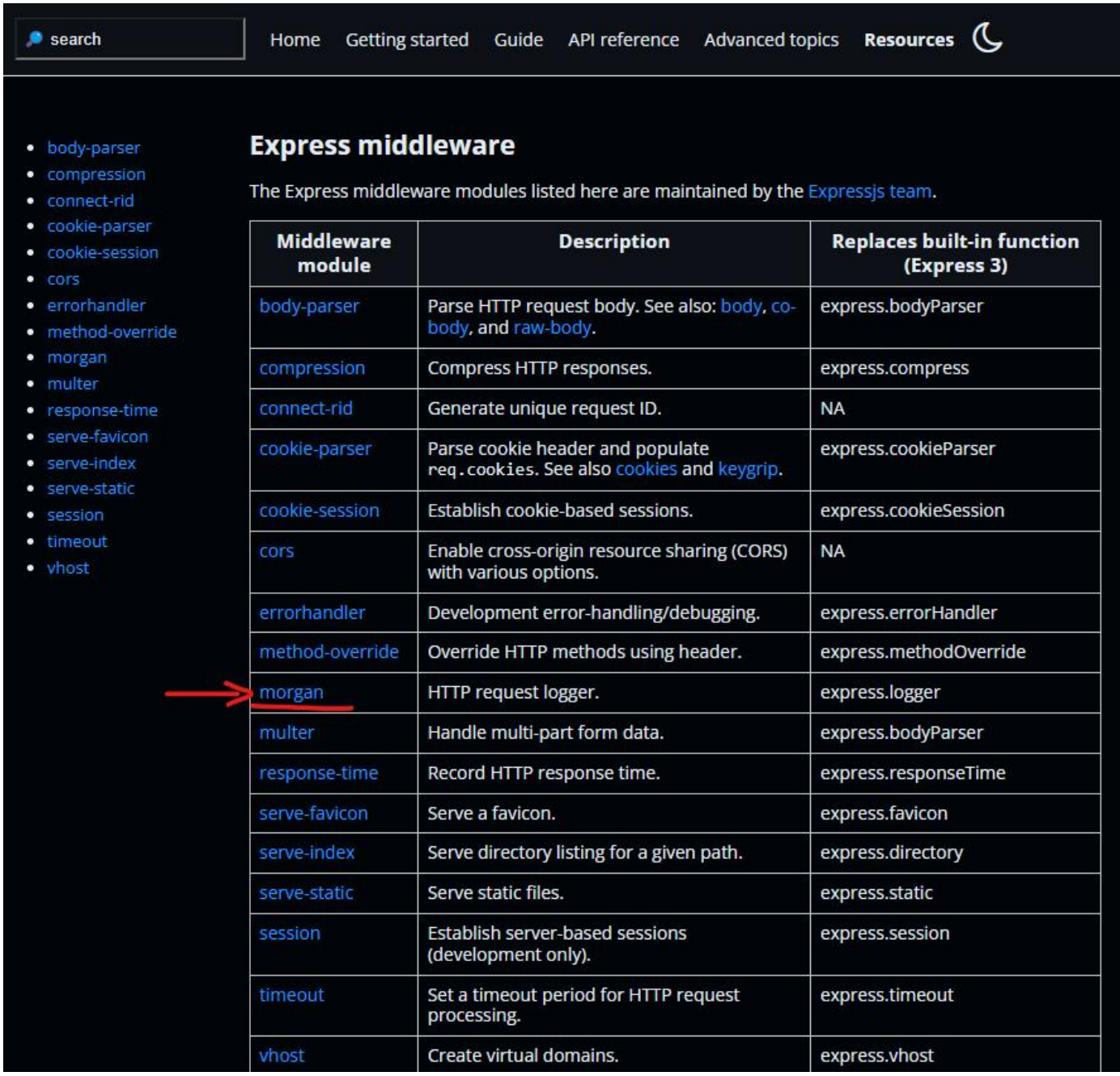
xii. Now specify it in your 'app.js' :

```
//Custom middleware :
app.use(myMiddlewareFunction);
app.use(myMiddlewareFunction2);
```

xiii. Now if you go to postman and check.It will be running good.

c) Third Party Middlewares.

i. Goto “Express webiste >resources >Middleware” :



ii. Lets try “morgan”:[morgan is a HTTP request logger.]

1) First install morgan.

```
npm install morgan
```

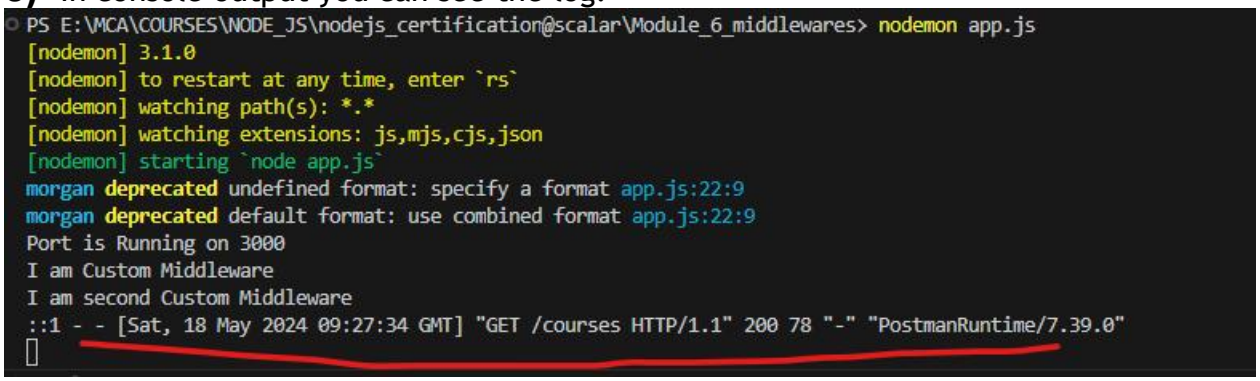
2) Then import it in your ‘app.js’ :

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

3) Now use morgan(specify):

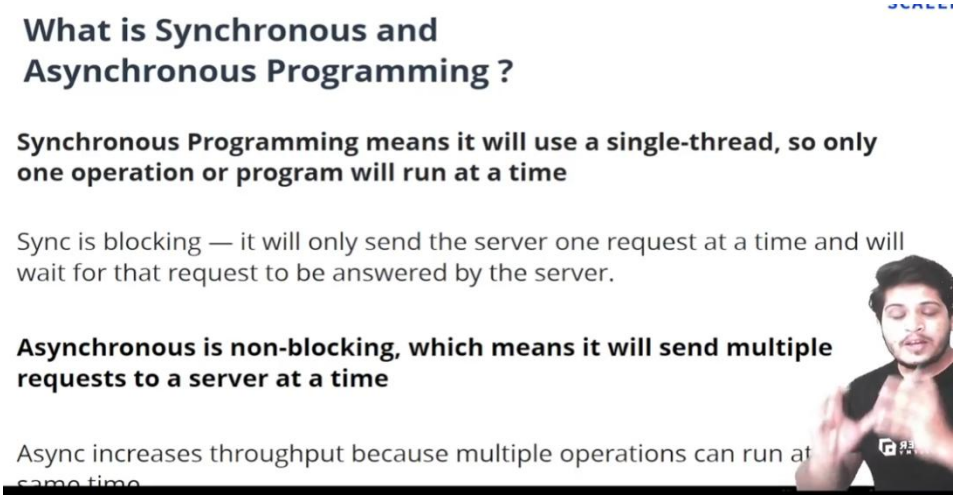
```
//user third partym middleware-morgan
app.use(morgan());
```

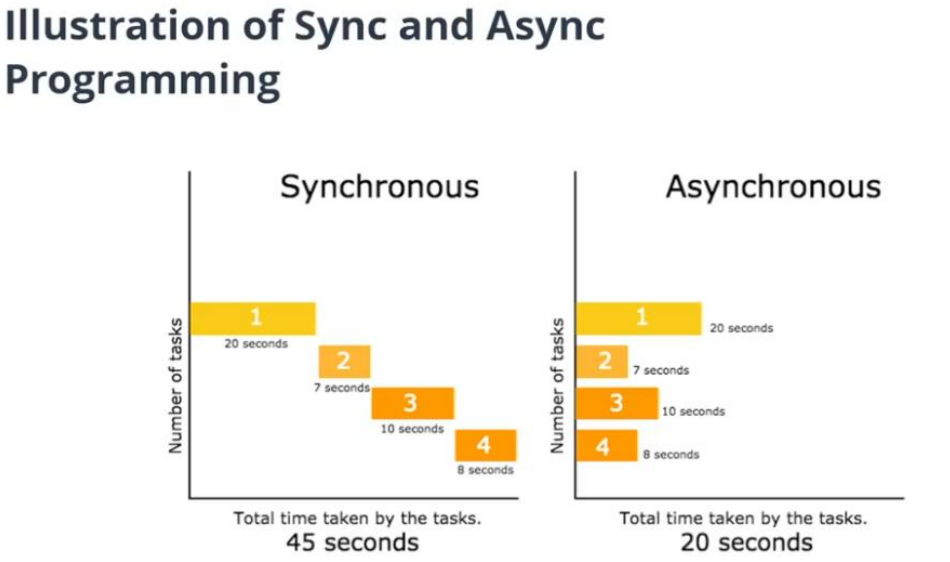
4) Then Run 'nodemon app.js'.
5) Now goto postman and Test.
6) In console output you can see the log.

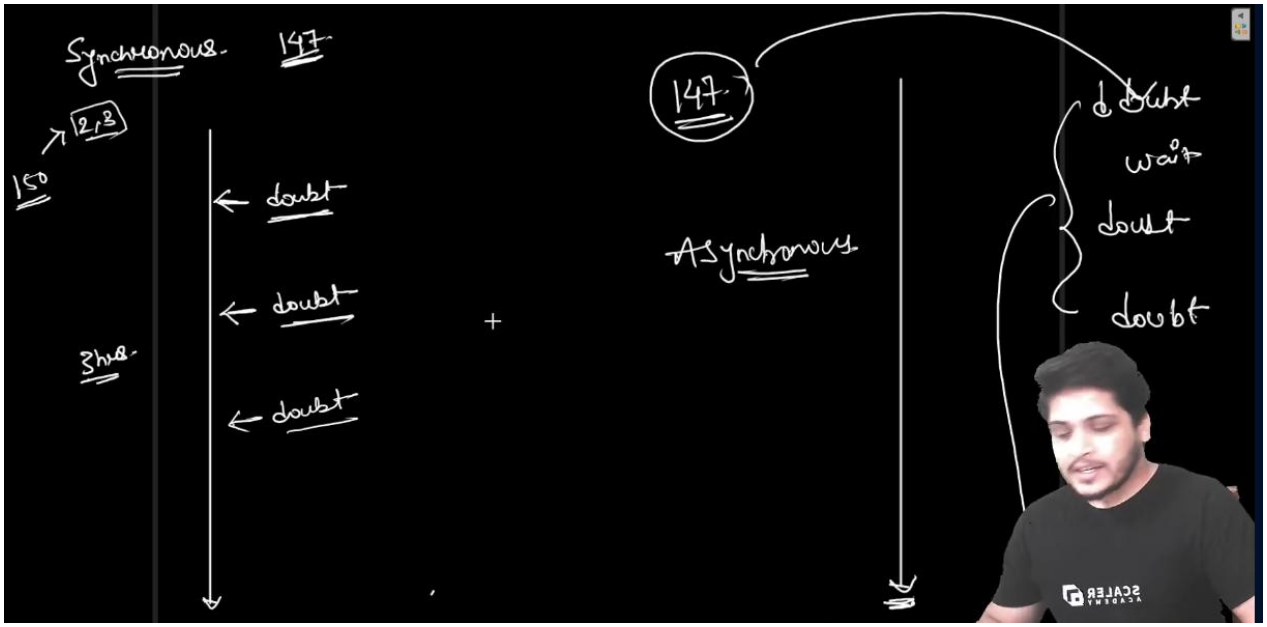


7. Asynchronous JavaScript :

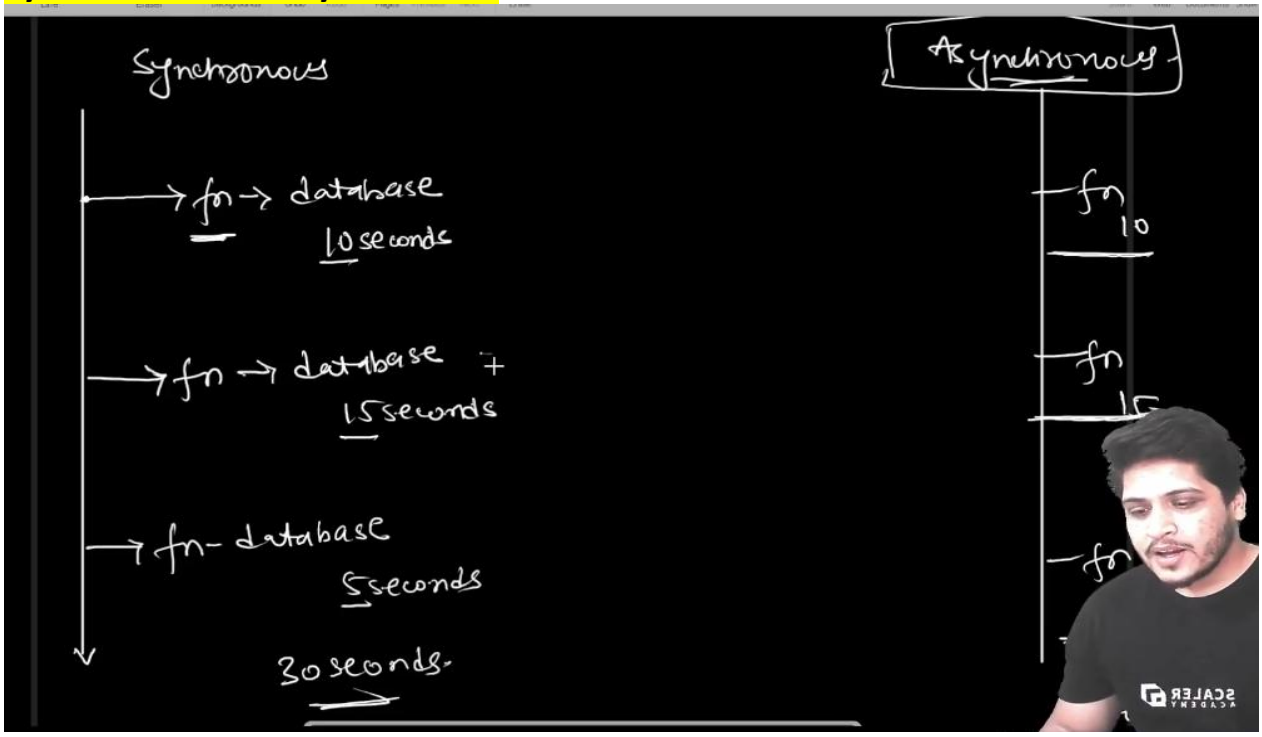
a) Introduction to Asynchronous Programming.







Synchronous and Asynchronous in JS:



Topics that we will cover-

1. What is Async Programming?
2. Callbacks (setTimeout , setInterval etc)
3. Callback queue and the Event Loop
4. Promises
5. async/await
6. microtask Queue
7. Sequential and Parallel Execution of code

b) 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')
```

2) Create ‘f1.txt ‘ file.

```
I am file 1
```

3) OUTPUT :

```
First Line
File 1 Data -> I am file 1
Last Line
```

c) 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')
```

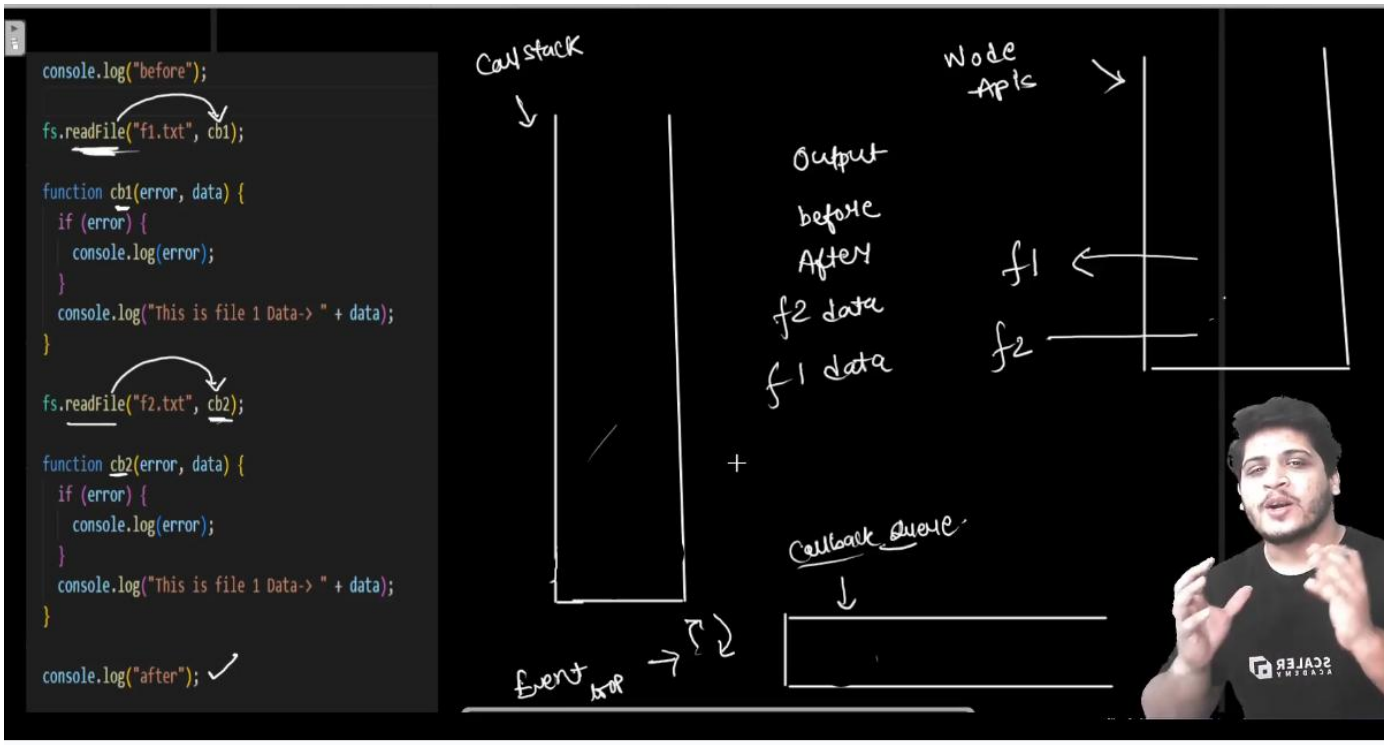
2) Create ‘f2.txt’ file.

```
I am file 2
```

3) 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
```

d) The Event Loop and Everything.



e) 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
}
```

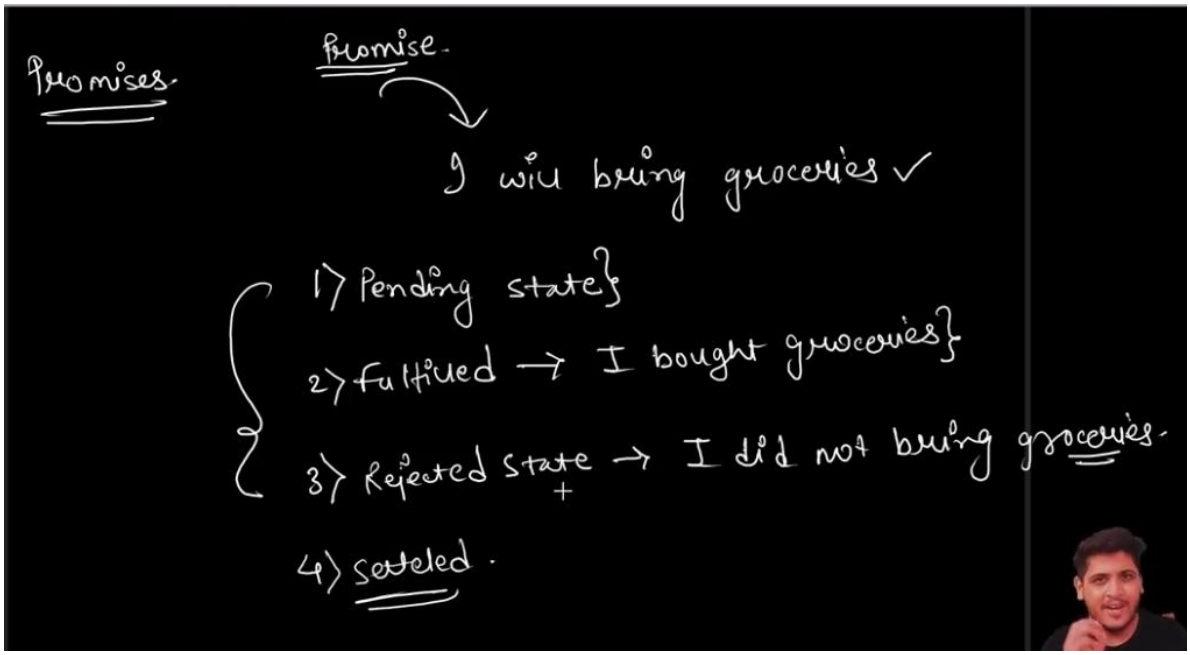
```
function cb3(err,data){
    if(err){
        console.log(err);
    }
```

```
    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
*/
```

f) Promises in JS.



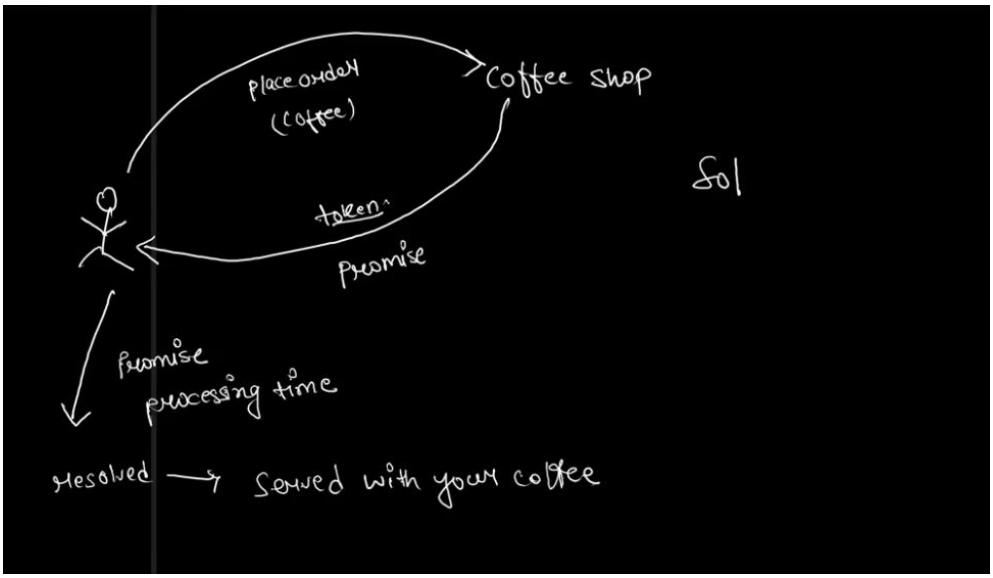

```
//Promises .
//How to produce a promise:

//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
```

g) 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 {
```

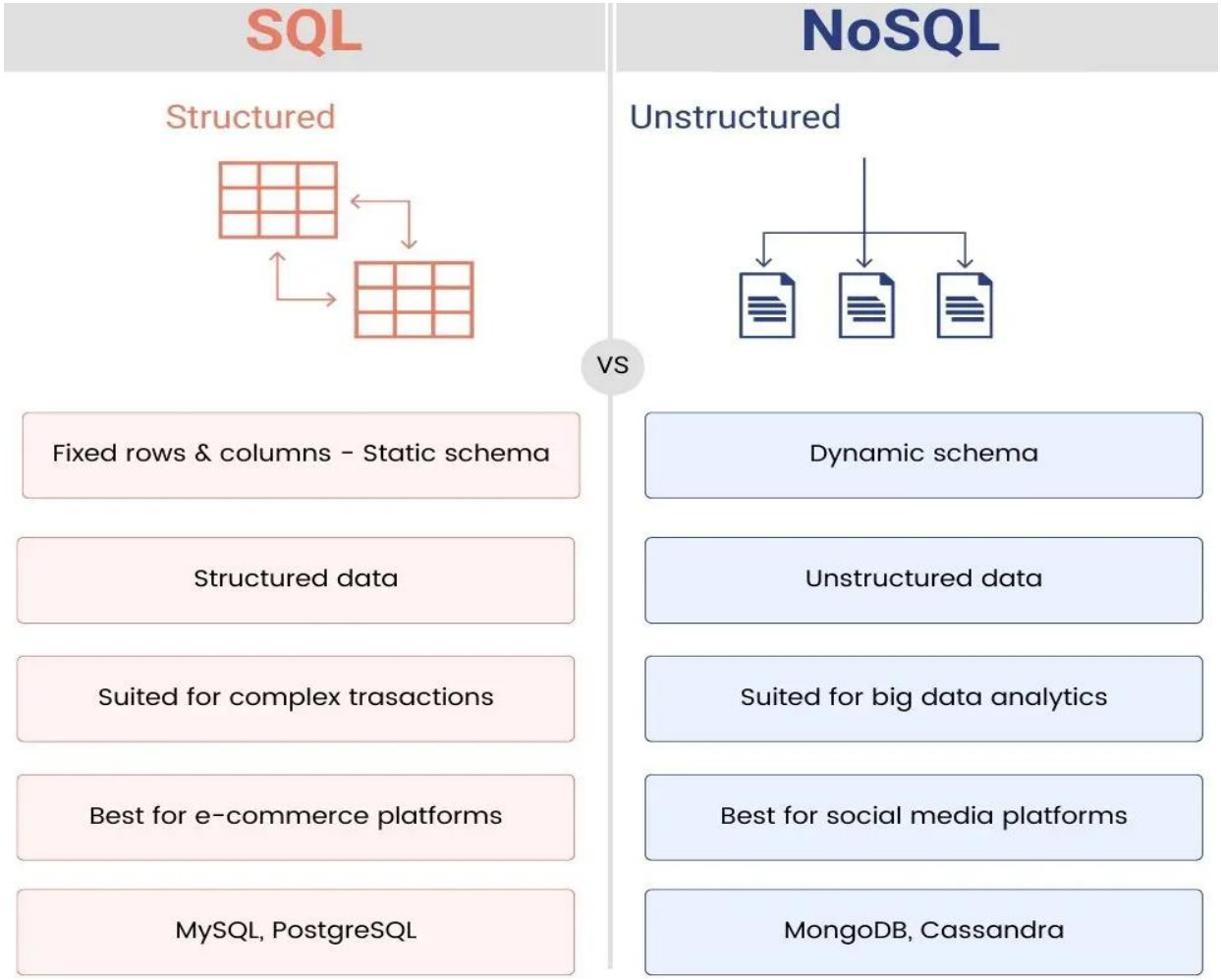
```
        let orderPlaced = await placeOrder('coffee');
        console.log(orderPlaced);
        let processedOrder =await processOrder(orderPlaced);
        console.log(processedOrder);
    }
    catch (error) {
        console.log(error);
    }

}
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.
*/
```

8. CRUD with Mongose and MongoDB.

a) Getting started with MongoDB.

i. MongoDB is a NoSQL DBMS.
ii. NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables.
iii. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph.
iv. It stores data in a type of 'JSON' format called 'BSON'.
v. MongoDB is a document database(non-relational database) and can be installed locally or hosted in the cloud.



b) Installation and Configuration of MongoDB.

i. There are two versions of MongoDB :

1) **MongoDB Atlas -[For Cloud].**
2) **MongoDB Community Server- [For Local System].**

ii. To Download GOTO MongoDB website>Community Edition and Download it.

iii. Then install it including 'compass'.

iv. Now Add path to Environement variables - 'C:\Program Files\MongoDB\Server\7.0\bin' :

v.Now open cmd and run : 'mongod'.

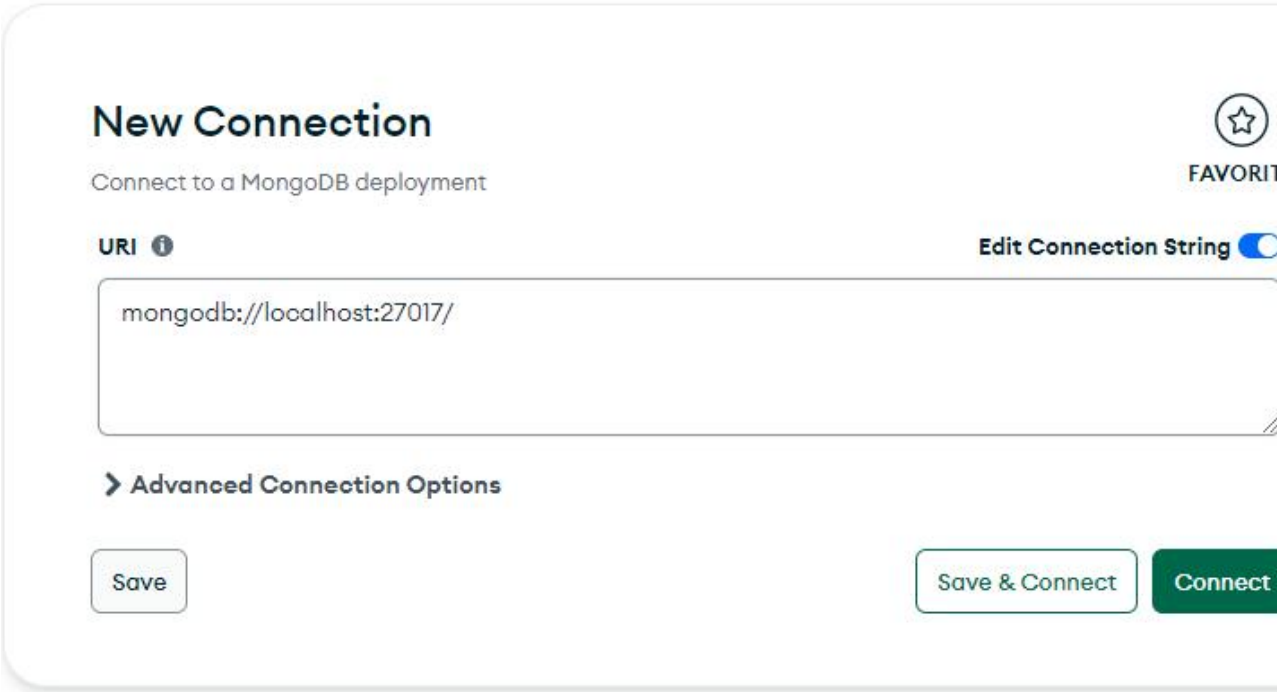
vi. After running we can see and Error :

```
,"s":"E",  "c":"CONTROL",  "id":20557,   "ctx":"initandlisten","msg":"DBEx
{"error":"NonExistentPath: Data directory C:\\data\\db\\ not found. Create
 using (1) the --dbpath command line option, or (2) by adding the 'storage
```

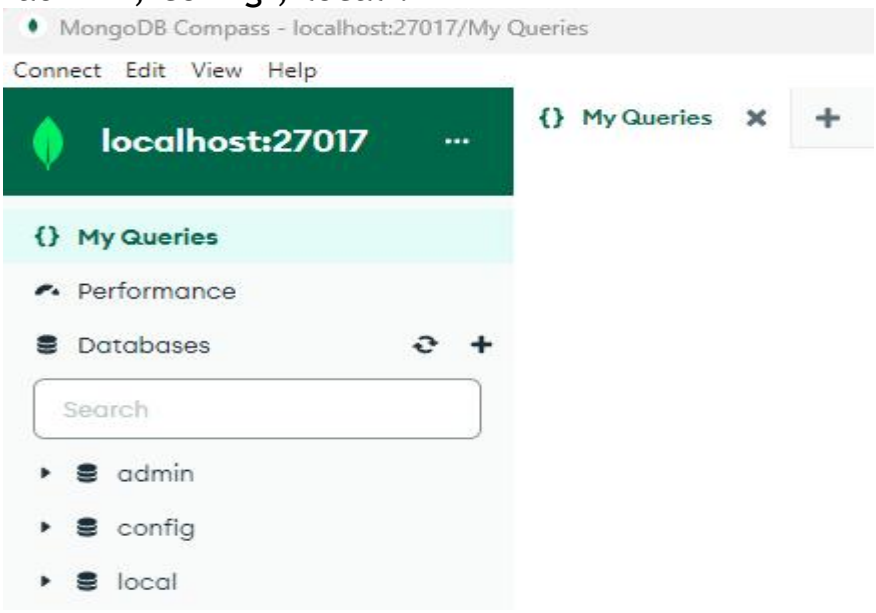
vii. This says it is not able to find the path.It is the path were MongoDb store its data.So we have to create this path .For that run command : 'md C:\\data\\db\\'.

viii. Then 'mongod' again.Now you wont find error.

ix. Now open MongoDB compass .Don't change defaultly given url.Just press 'connect'.



x.After connection you can see some random collections : 'admin','config','local'.



xi. How to connect our application to MongoDB compass ?

1) **Goto VScode and create New folder.**
2) **Open that in terminal and Install 'Mongoose'.(Is a Tool that gives API.Acts as Bridge between MongoDB and Express Application.)**
3) **Mongoose is a MongoDB ODM i.e (Object database Modelling) that used to translate the code and its representation from MongoDB to the Node. js server.**
4) **To Install 'Mongoose' type in terminal : 'npm install mongoose'.**

```
npm i mongoose
```

5) **After Installation,create a new file 'index.js' :**

```
//1.Import mongoose :
const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1/testDatabase') //To connect to DB.(The
url with default localhost adrress :127.0.0.1 and dbname:testDatabase)
.then(() => console.log('Connection is Successfull')) //If connection is
successfull
.catch(err => console.log('Coud not connect to mongodb' , err))
```

6) **Then run 'index.js' :**

```
PS E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD> nodemon index.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Connection is Successfull
```

c) Schemas and Models.

i. What is a schema?

A schema is a JSON object that defines the structure and contents of your data

ii. Schema creation Process :

1) First create a schema :

```
//1.How to Create a Schema ?

const courseSchema = new mongoose.Schema({
    name : String,
    creator : String,
    publishedDate : {type:Date ,default:Date.now },//to set multiple
properties use '{}'
    isPublished : Boolean
})
//Right now this schema wont work becuase we dont have a 'Model' for that.
```

2) Then Create a Model:

```
//2.How to create a Model ?
const Course = mongoose.model('Course' , courseSchema) //specify model
name,schema name inside.
```

3) Then Create some Datasets for Model:

```
//3.Now Create some Datasets for this particular Model :'Course'.
const course =  new Course({
    name : 'JavaScript',
    creator : 'Anshad',
    isPublished : true

})
```

4) Now Save using .save() :

```
    //4.Now Save these datas into Database(.save() method is used):
    const result = await course.save();
    console.log(result);
```

5) Full code :

```
//Import mongoose :
const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1/testDatabase') //To connect to DB.(The url with default localhost adrress :127.0.0.1 and dbname:testDatabase)
    .then(() => console.log('Connection is Successfull')) //If connection is successfull
    .catch(err => console.log('Coud not connect to mongodb', err))
```

```
//Schema - (pattern or structure)
//1.How to Create a Schema ?
const courseSchema = new mongoose.Schema({
    name: String,
    creator: String,
    publishedDate: { type: Date, default: Date.now },//to set multiple properties use '{}'
    isPublished: Boolean
})
```

```
//Right now this schema wont work becuase we dont have a 'Model' for that.
//2.How to create a Model ?
const Course = mongoose.model('Course', courseSchema) //specify model name,schema name inside.
```

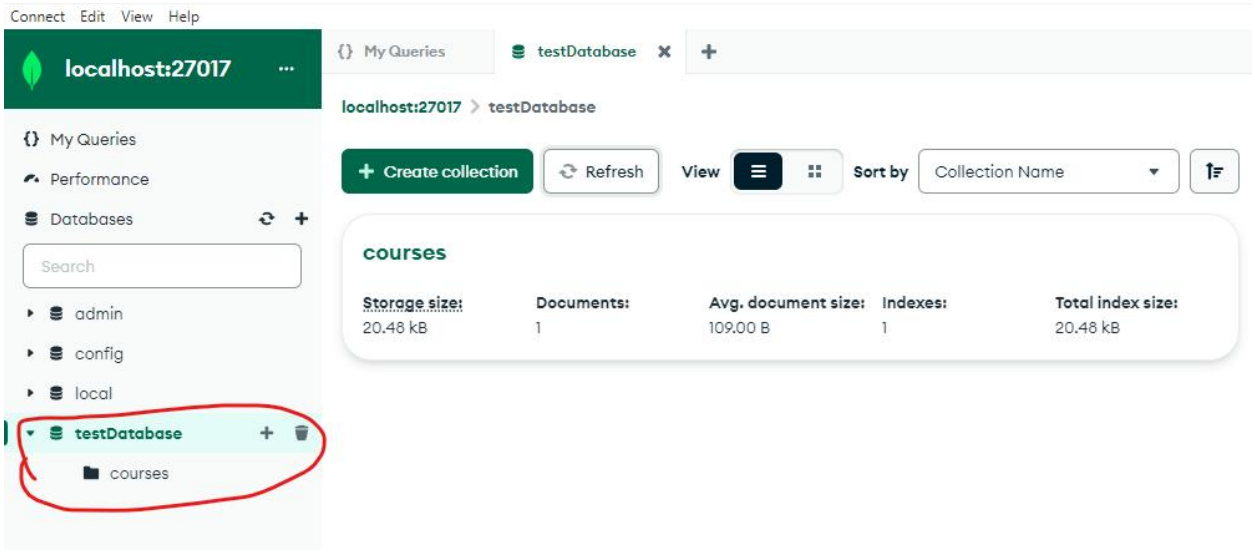
```
async function createCourse() {//make function 'async' for 'await'.
    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'JavaScript',
        creator: 'Anshad',
        isPublished: true
    })
    //4.Now Save these datas into Database(.save() method is used):
    const result = await course.save();
    console.log(result);
}
```

```
createCourse();//calling function
```

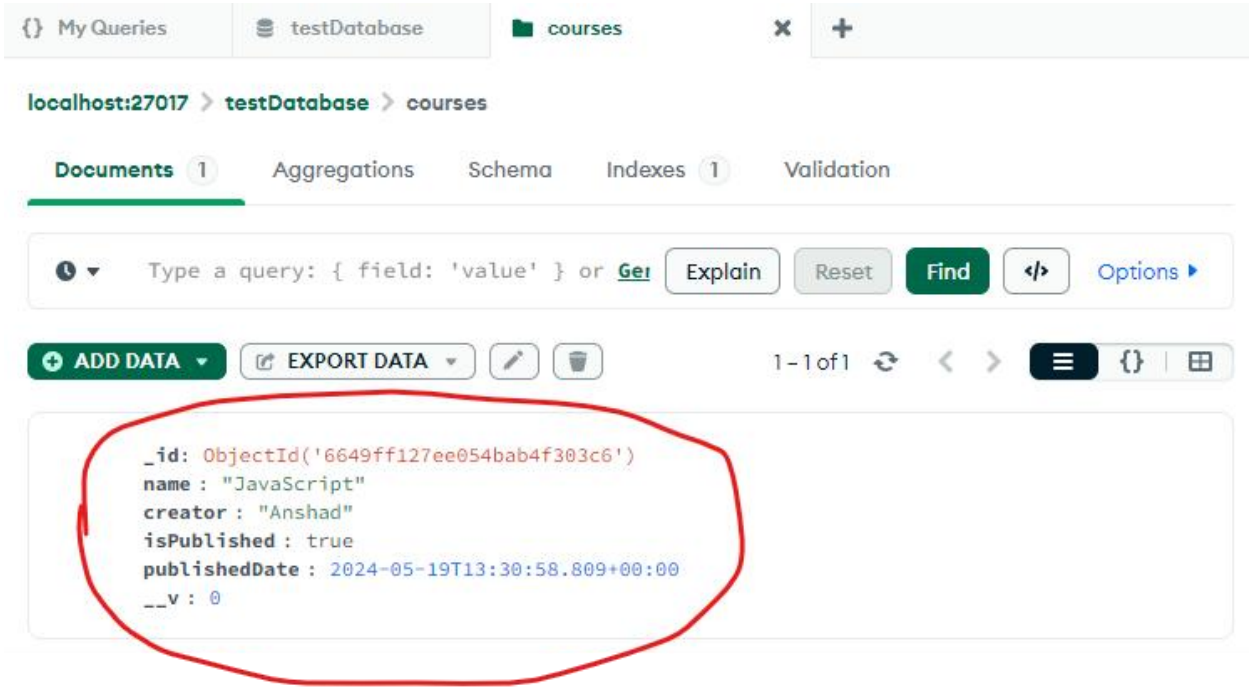
Output :

```
Connection is Successfull
{
  name: 'JavaScript',
  creator: 'Anshad',
  isPublished: true,
  _id: new ObjectId('6649ff127ee054bab4f303c6'),
  publishedDate: 2024-05-19T13:30:58.809Z,
  __v: 0
}
```

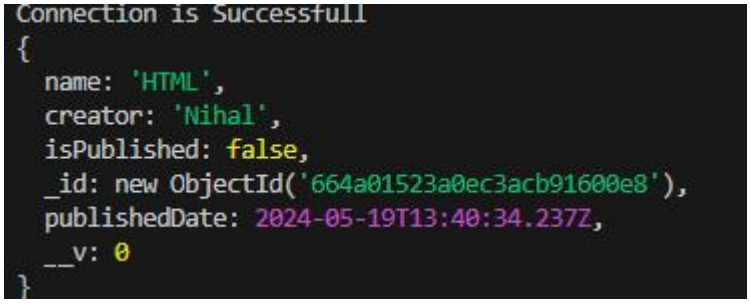
6) **Now goto MongoDB compass and check whether the collection is there or not :**



When you click that :



7) **Similarly if you got and edit course in vscode to add new data it will also shown in compass:**




```
_id: ObjectId('6649ff127ee054bab4f303c6')
name : "JavaScript"
creator : "Anshad"
isPublished : true
publishedDate : 2024-05-19T13:30:58.809+00:00
__v : 0
```

```
_id: ObjectId('664a01523a0ec3acb91600e8')
name : "HTML"
creator : "Nihal"
isPublished : false
publishedDate : 2024-05-19T13:40:34.237+00:00
__v : 0
```

d) How to query for documents.

i. Here,each dataset or collection is Known as ‘Documents’.

ii. Code to get the data :

```
//How to Query for Documents :
async function getCourses(){//Function to get courses
    const courses =await Course.find({creator:'Anshad'});//Finds course whose creator is 'Anshad'.
    console.log(courses);//displays found course
}
getCourses();//calling
```

OUTPUT :

```
Connection is Successfull
[
  {
    _id: new ObjectId('6649ff127ee054bab4f303c6'),
    name: 'JavaScript',
    creator: 'Anshad',
    isPublished: true,
    publishedDate: 2024-05-19T13:30:58.809Z,
    __v: 0
  }
]
```

iii. If you want only the course Name :(use .select())

```
//How to Query for Documents :
async function getCourses(){//Function to get courses
    const courses =await
Course.find({creator:'Anshad'}).select({name :1});//Finds course whose creator
is 'Anshad'.
    console.log(courses);//displays found course
}
getCourses();//calling
```

OUTPUT :

```
Connection is Successfull
[
  { _id: new ObjectId('6649ff127ee054bab4f303c6'), name: 'JavaScript' }
]
```

iv. If you wan to sort :

```
//How to Query for Documents :
async function getCourses(){//Function to get courses
    const courses =await
Course.find({creator:'Nihal'}).select({name :1}).sort({name : 1});
    console.log(courses);//displays found course
}
getCourses();//calling
```

OUTPUT :

```
Connection is Successfull
[
  { _id: new ObjectId('664a091c746c0afda6c27a31'), name: 'C++' },
  { _id: new ObjectId('664a0927658ff958659c385f'), name: 'DBMS' },
  { _id: new ObjectId('664a01523a0ec3acb91600e8'), name: 'HTML' }
]
```

e) Comparison Query Operator.($eq,$gt,$gte,$lt,$lte,$in,$not in)

i. First Add rating property to schema to work with comparison.

```
const courseSchema = new mongoose.Schema({
    name: String,
    creator: String,
    publishedDate: { type: Date, default: Date.now },//to set multiple properties use '{}'
    isPublished: Boolean,
    rating : Number
})
```

ii. Before drop database from Compass.

iii. Then add some datasets:(one's rating is 3)

```
_id: ObjectId('664a0b255a3b2d29cedd1d3b')
name : "CSS"
creator : "Majo"
isPublished : false
rating : 4.5
publishedDate : 2024-05-19T14:22:29.747+00:00
__v : 0
```

```
_id: ObjectId('664a0b41aebef132b5b0b87d')
name : "Java"
creator : "Nihal"
isPublished : true
rating : 4.2
publishedDate : 2024-05-19T14:22:57.807+00:00
__v : 0
```

```
_id: ObjectId('664a0b5609eb2533c3e37405')
name : "HTML"
creator : "Anshad"
isPublished : true
rating : 4.3
publishedDate : 2024-05-19T14:23:18.587+00:00
__v : 0
```

```
_id: ObjectId('664a0bc13c34fc875767e623')
name : "JavaScript"
creator : "Hari"
isPublished : true
rating : 3
publishedDate : 2024-05-19T14:25:05.219+00:00
__v : 0
```

iv. How to get those courses which has rating 4 and Above ?.

v.For that we have Comparison Operators.

1. **eq - (for equals)**
2. **gt - (for greater than)**
3. **gte - (greater than and equalsto)**
4. **lt - (less than)**
5. **lte - (less than and equalsto)**
6. **in**
7. **not in.**

vi. Solution :

```
async function getCourses(){//Function to get courses
    const courses =await Course.find({rating : {$gte : 4 }}).select({name :1 ,
publishedDate :1});
    console.log(courses);//displays found course
}
getCourses();//calling
```

OUTPUT:

```
Connection is Successfull
[
  {
    _id: new ObjectId('664a0b255a3b2d29cedd1d3b'),
    name: 'CSS',
    publishedDate: 2024-05-19T14:22:29.747Z
  },
  {
    _id: new ObjectId('664a0b41aebef132b5b0b87d'),
    name: 'Java',
    publishedDate: 2024-05-19T14:22:57.807Z
  },
  {
    _id: new ObjectId('664a0b5609eb2533c3e37405'),
    name: 'HTML',
    publishedDate: 2024-05-19T14:23:18.587Z
  }
]
```

vii. Suppose if you want Courses with Rating 3 and 4.2 :

```
async function getCourses(){//Function to get courses
    const courses =await Course.find({rating : {$in :
[3 ,4.2] }}).select({name :1 , publishedDate :1});
    console.log(courses);//displays found course
}
getCourses();//calling
```

OUTPUT :

```
Connection is Successfull
[
  {
    _id: new ObjectId('664a0b41aebef132b5b0b87d'),
    name: 'Java',
    publishedDate: 2024-05-19T14:22:57.807Z
  },
  {
    _id: new ObjectId('664a0bc13c34fc875767e623'),
    name: 'JavaScript',
    publishedDate: 2024-05-19T14:25:05.219Z
  }
]
```

f) Logical Query Operators.($or,$and)

i. $or:

```
//Logical Query Operators:
//1.or
//2.and
async function getCourses2(){//Function to get courses
    const courses =await Course.find({rating : {$in : [3 ,4.2 , 4.5 ,
4.3] }}).select({name :1 , publishedDate :1})
    .or([{creator:'Anshad'},{rating : 4.5}])
    console.log(courses);//displays found course
}
getCourses2();//calling
```

(Above code displays the courses whose creator is 'Anshad' or rating is 4.5)

OUTPUT :

```
Connection is Successfull
[
  {
    _id: new ObjectId('664a0b41aebef132b5b0b87d'),
    name: 'Java',
    publishedDate: 2024-05-19T14:22:57.807Z
  },
  {
    _id: new ObjectId('664a0bc13c34fc875767e623'),
    name: 'JavaScript',
    publishedDate: 2024-05-19T14:25:05.219Z
  }
]
[
  {
    _id: new ObjectId('664a0b255a3b2d29cedd1d3b'),
    name: 'CSS',
    publishedDate: 2024-05-19T14:22:29.747Z
  },
  {
    _id: new ObjectId('664a0b5609eb2533c3e37405'),
    name: 'HTML',
    publishedDate: 2024-05-19T14:23:18.587Z
  }
]
```

ii. $and :

```
async function getCourses2(){//Function to get courses
    const courses =await Course.find({rating : {$in : [3 ,4.2 , 4.5 ,
4.3] }}).select({name :1 , publishedDate :1})
    .and([{creator:'Anshad'},{rating : 4.5}])
    console.log(courses);//displays found course
}
getCourses2();//calling
```

OUTPUT :

```
Connection is Successfull
[]
```

g) How to Update a Document.

```
//How to Update a Document :
async function updateCourse(id){//parameter 'id' used to update using id
    let course = await Course.findById(id)

    if(!course) return;//returns if specific id not found.
```

```
    course.name = 'Python';
    course.creator = 'Steve';
    const updatedCourse = await course.save();
    console.log(updatedCourse);
```

```
}
```

```
updateCourse('664a0b255a3b2d29cedd1d3b');//calling by passing an id.(you will
get id from Compass)
```

OUTPUT:

```
Connection is Successfull
{
  _id: new ObjectId('664a0b255a3b2d29cedd1d3b'),
  name: 'Python',
  creator: 'Steve',
  isPublished: false,
  rating: 4.5,
  publishedDate: 2024-05-19T14:22:29.747Z,
  __v: 0
}
```

h) How to Delete a Document.

```
//How to Delete a Document :
async function deleteCourse(id){//parameter 'id' used to delete using id
    let course = await Course.findByIdAndDelete(id);//this will find the course of
specific id and delete it.
    console.log(course);

}
```

```
deleteCourse('664a0b255a3b2d29cedd1d3b');
```

OUTPUT:

```
Connection is Successfull
{
  _id: new ObjectId('664a0b255a3b2d29cedd1d3b'),
  name: 'Python',
  creator: 'Steve',
  isPublished: false,
  rating: 4.5,
  publishedDate: 2024-05-19T14:22:29.747Z,
  __v: 0
}
```

9. Data validation in MongoDB .

a) Data validation in MongoDB :[By adding 'required' keyword in schema]

i. Add 'required' keyword in schema :

```
//Schema - (pattern or structure)
//1.How to Create a Schema ?

const courseSchema = new mongoose.Schema({
    name: {type : String, required : true},
    creator:{type : String, required : true},
    publishedDate: { type: Date, default: Date.now },//to set multiple properties
use '{}'
    isPublished: {type : String, required : true},
    rating : Number
})
//Right now this schema wont work becuase we dont have a 'Model' for that.
//2.How to create a Model ?
const Course = mongoose.model('Course', courseSchema) //specify model name,schema
name inside.\
```

ii. Now if you try to create a dataset only by providing name,lets see what happens:

```
async function createCourse() {//make function 'async' for 'await'.

    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'Ruby',
    })
    //4.Now Save these datas into Database(.save() method is used):
    const result = await course.save();
    console.log(result);
}
createCourse();//calling function
```

Output :[Error: Validation Error]

```
[nodemon] starting `node index2.js`
E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD\node_modules\mongoose\lib\do
cument.js:3289
    this.$__.validationError = new ValidationError(this);
                               ^

ValidationError: Course validation failed: isPublished: Path `isPublished` is required., crea
tor: Path `creator` is required.
    at Document.invalidate (E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD\
node_modules\mongoose\lib\document.js:3289:32)
    at E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD\node_modules\mongoose
\lib\document.js:3050:17
    at E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD\node_modules\mongoose
\lib\schemaType.js:1388:9
    at process.processTicksAndRejections (node:internal/process/task_queues:77:11) {
  errors: {
    isPublished: ValidatorError: Path `isPublished` is required.
        at validate (E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD\node_mo
```

iii. You can handle these Error using try{} catch{} block:

```
async function createCourse() {//make function 'async' for 'await'.

    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'Ruby',
    })
    try {
        //4.Now Save these datas into Database(.save() method is used):
        const result = await course.save();
        console.log(result);
    } catch (error) {
        console.error(error.message);
    }
}
createCourse();//calling function
```

OUTPUT :(Now specific error is shown)

```
[nodemon] starting `node index2.js`
Course validation failed: isPublished: Path `isPublished` is required., creator: Path `creato
r` is required.
Connection is Successfull
```

iv. There is another methode in try{}catch{}: ‘await course.validate()’:

```
async function createCourse() {//make function 'async' for 'await'.

    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'Ruby',
    })
    try {
        await course.validate();
        // //4.Now Save these datas into Database(.save() method is used):
        // const result = await course.save();
        // console.log(result);
    } catch (error) {
        console.error(error.message);
    }
}
createCourse();
```

b) Inbuilt Data Validators:

i. Lets add function in required of rating.So that when isPublished is true then, rating should be required else not required.

```
const courseSchema = new mongoose.Schema({
    name: {type : String, required : true},
    creator:{type : String, required : true},
    publishedDate: { type: Date, default: Date.now },//to set multiple properties
use '{}'
    isPublished: {type : String, required : true},
    rating : {type : Number, required : function(){return this.isPublished}}
})

//Right now this schema wont work becuase we dont have a 'Model' for that.
//2.How to create a Model ?
const Course = mongoose.model('Course', courseSchema) //specify model name,schema
name inside.
async function createCourse() {//make function 'async' for 'await'.
    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'Ruby',
        creator : 'Jibin',
        isPublished : true
    })
    try {
        await course.validate();
        // //4.Now Save these datas into Database(.save() method is used):
        // const result = await course.save();
        // console.log(result);
    } catch (error) {
        console.error(error.message);
    }
}
createCourse();//calling function
```

ii. minlength,maxlength,enum :(enum :the value we are giving should be one amoung enum values)

```
//1.How to Create a Schema ?
const courseSchema = new mongoose.Schema({
    name: {type : String, required : true , minlength : 5 , maxlength :200},
    category : {
        type :String,
        required :true,
        enum : ['DSA','Web','Mobile','Data Science']
    },
    creator:{type : String, required : true},
    publishedDate: { type: Date, default: Date.now },//to set multiple properties
use '{}'
    isPublished: {type : String, required : true},
    rating : {type : Number, required : function(){return this.isPublished}}
})
```

```
//Right now this schema wont work becuase we dont have a 'Model' for that.
//2.How to create a Model ?
const Course = mongoose.model('Course', courseSchema) //specify model name,schema
name inside.
async function createCourse() {//make function 'async' for 'await'.
    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'MongoDB',
        creator : 'Jibin',
        category : 'DSA',
        isPublished : false,
        rating : 4.5
    })
    try {
        // await course.validate();
        // //4.Now Save these datas into Database(.save() method is used):
        const result = await course.save();
        console.log(result);
    } catch (error) {
        console.error(error.message);
    }
}
createCourse();//calling function
```

```
Connection is Successfull
{
  name: 'MongoDB',
  category: 'DSA',
  creator: 'Jibin',
  isPublished: 'false',
  rating: 4.5,
  _id: new ObjectId('664b0fc44467b4d4777f9495'),
  publishedDate: 2024-05-20T08:54:28.055Z,
  __v: 0
```

c) Custom Data Validators:

i. Implementing tags :

```
//VALIDATION : [By adding 'required' keyword in schema ]

const mongoose = require('mongoose');//Import mongoose.
```

```
mongoose.connect('mongodb://127.0.0.1/testDatabase') //To connect to DB.(The url with default localhost adrress :127.0.0.1 and dbname:testDatabase)
    .then(() => console.log('Connection is Successfull')) //If connection is successfull
    .catch(err => console.log('Coud not connect to mongodb', err))
```

```
//Schema - (pattern or structure)
//1.How to Create a Schema ?
```

```
const courseSchema = new mongoose.Schema({
    name: {type : String, required : true , minlength : 5 , maxlength :200},
    tags : {type : Array , validate :{
        validator : function(tags){
            return tags.length > 1
        }
    }},
    category : {
        type :String,
        required :true,
        enum : ['DSA','Web','Mobile','Data Science']
    },
    creator:{type : String, required : true},
    publishedDate: { type: Date, default: Date.now },//to set multiple properties use '{}'
    isPublished: {type : String, required : true},
    rating : {type : Number, required : function(){return this.isPublished}}
})
```

```
//Right now this schema wont work becuase we dont have a 'Model' for that.
//2.How to create a Model ?
const Course = mongoose.model('Course', courseSchema) //specify model name,schema name inside.
async function createCourse() {//make function 'async' for 'await'.
```

```
    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'MongoDB',
        tags : ['express','mongodb'],
        creator : 'Jibin',
        category : 'DSA',
        isPublished : false,
        rating : 4.5
    })
    try {
        // await course.validate();
        // //4.Now Save these datas into Database(.save() method is used):
```

```
        const result = await course.save();
        console.log(result);
    } catch (error) {
        console.error(error.message);
    }
}
createCourse();//calling function
```

d) Error Validators:(Use error validators to see each fields of error)

i. Now Try Running by passing only one value in tags:(you will get error)

```
async function createCourse() {//make function 'async' for 'await'.

    //3.Now Create some Datasets for this particular Model :'Course'.
    const course = new Course({
        name: 'MongoDB',
        tags : ['express'],
        creator : 'Jibin',
        category : 'DSA',
        isPublished : false,
        rating : 4.5
    })
    try {
        // await course.validate();
        // //4.Now Save these datas into Database(.save() method is used):
        const result = await course.save();
        console.log(result);
    } catch (error) {
        for(field in error.errors){
            console.log(error.errors[field])
        }
    }
}
createCourse();//calling function
```

OUTPUT:

```
[nodemon] starting `node index2.js`
ValidatorError: Validator failed for path `tags` with value `express`
    at validate (E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD\node_module
s\mongoose\lib\schemaType.js:1385:13)
    at SchemaType.doValidate (E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRU
D\node_modules\mongoose\lib\schemaType.js:1369:7)
    at E:\MCA\COURSES\NODE_JS\nodejs_certification@scalar\Module_8_CRUD\node_modules\mongoose
\lib\document.js:3042:18
    at process.processTicksAndRejections (node:internal/process/task_queues:77:11) {
  properties: {
    validator: [Function: validator],
    message: 'Validator failed for path `tags` with value `express`',
    type: 'user defined',
    path: 'tags',
    fullPath: undefined,
    value: [ 'express' ]
  },
  kind: 'user defined',
  path: 'tags',
  value: [ 'express' ],
  reason: undefined,
  [Symbol(mongoose#validatorError)]: true
}
Connection is Successfull
```

10. Model Relationships by Building APIs .

a) Initializing the Project : (PROJECT NAME : Online E-Learning Platform)

i. Create a New Folder 'Module_5_PROJECT' and open it in Terminal.

ii. Install npm into that folder:

```
npm init --yes
```

iii. Install Express :

```
npm i express
```

iv. Install mongoose (Is a Tool that gives API.Acts as Bridge between MongoDB and Express Application.):

```
npm i mongoose
```

v. Create a file 'app.js' :(import,specify get,post,put,delete)

```
const express = require('express');//Importing express.

const app = express();//calling express.

app.use(express.json()); // Middleware to parse JSON bodies.

const categories = [
    {id:1 , name : 'Web'},
    {id:2 , name : 'Mobile'},
    {id:3 , name : 'Photography'},
]
//GET:
app.get('/api/categories',(req , res ) => {
    res.send(categories);
});
//POST :Create
app.post('/api/categories',(req , res ) => {
    const category = {
        id : categories.length+1, //new id (3+1=4)
        name : req.body.name //Whatever the name we pass as req in postman body>raw
it will be set as the name field of categories array
    };
    categories.push(category);
    res.send(category);
```

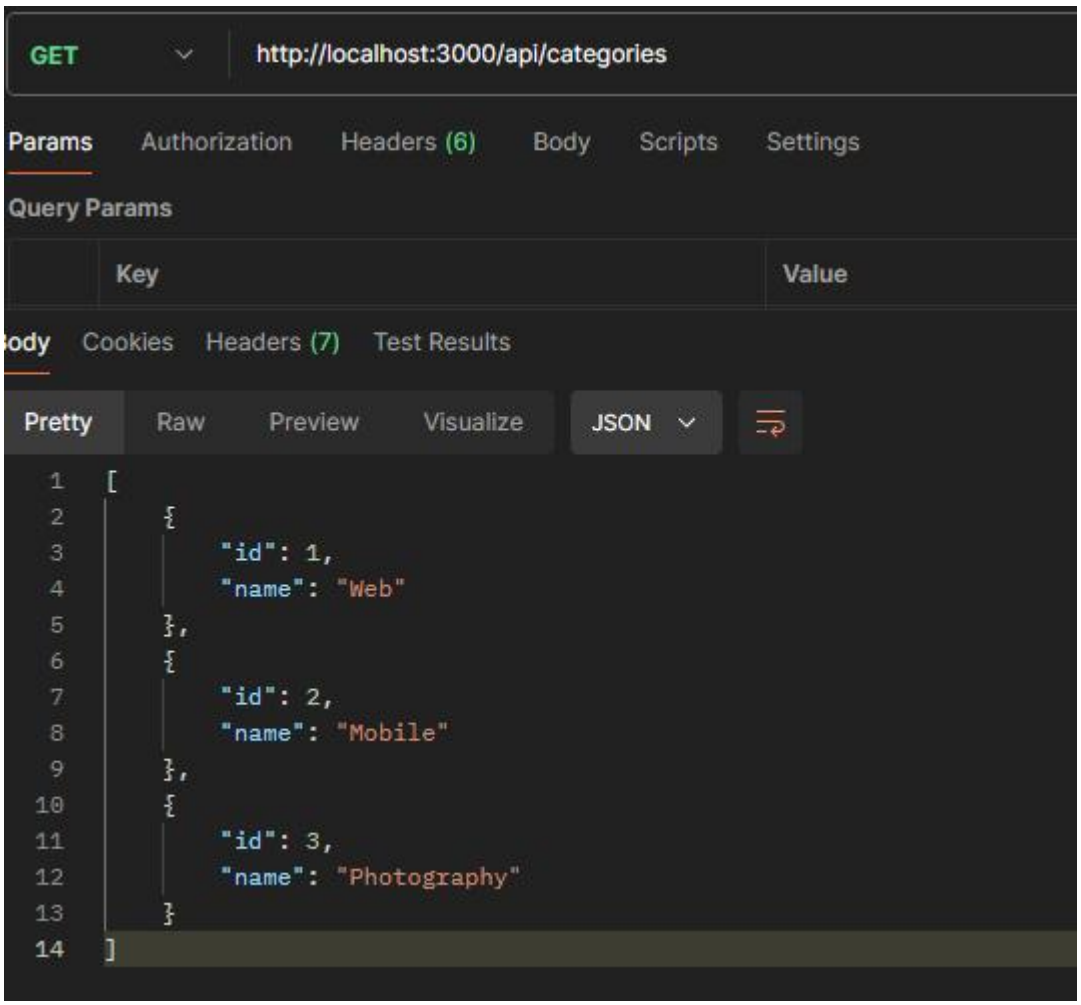
```
});
//PUT : Update
app.put('/api/categories/:id', (req , res)=> {
    const category = categories.find(c => c.id === parseInt(req.params.id));//Finds
the particular course with id.
     if(!category) return res.status(404).send('The category with given ID was not
found');
    //  if(error) return res.status(400).send(error.details[0].message);
     category.name = req.body.name;//Updating name
     res.send(category);
})
```

```
//DELETE : delete
app.delete('/api/categories/:id',(req , res)=>{
    const  category = categories.find(c => c.id === parseInt(req.params.id));
//Finds the particular course with id.
    if(!category) return res.status(404).send('The genre with the given ID was not
found.');
    const index = courses.indexOf(category);//To find the index .
    category.splice(index , 1);//To delete that index from array.
    res.send(category);
})
app.get('/api/categories/:id', (req , res)=> {
    const category = categories.find(c => c.id === parseInt(req.params.id));//Finds
the particular course with id.
     if(!category) return res.status(404).send('The genre with given ID was not
found');
     res.send(category);
})

//ENVIRONMENT VARIABLE - for PORTs
const port = process.env.PORT || 3000//for static

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

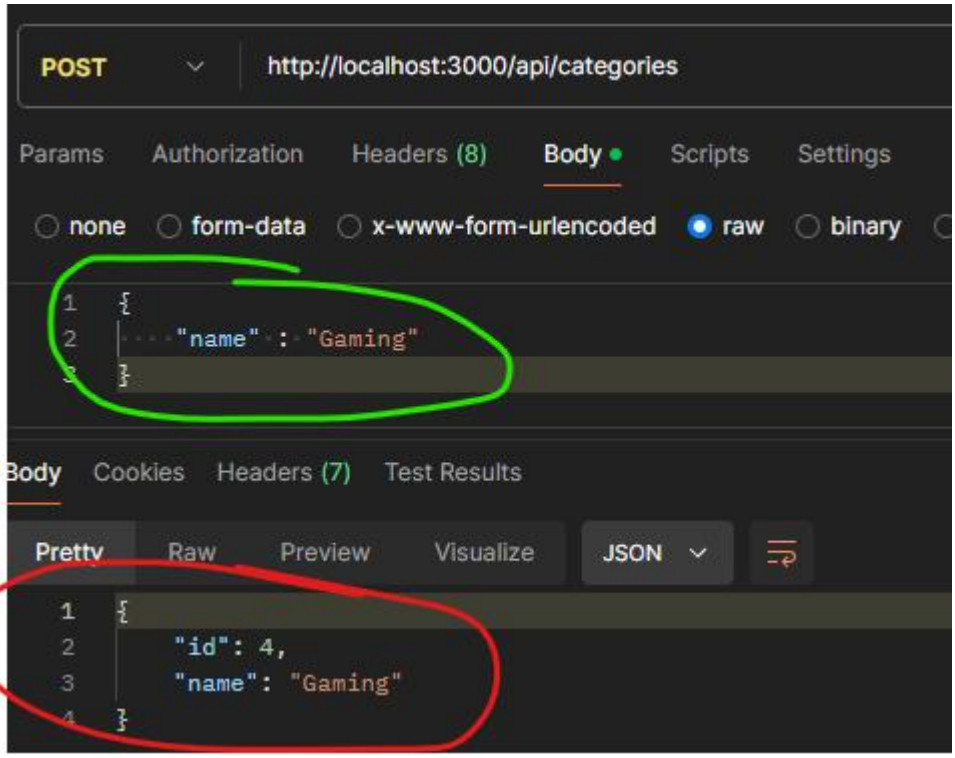
vi. Run ‘app.js’ -> ‘nodemon app.js’ :

```
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Port is Running on 3000
```

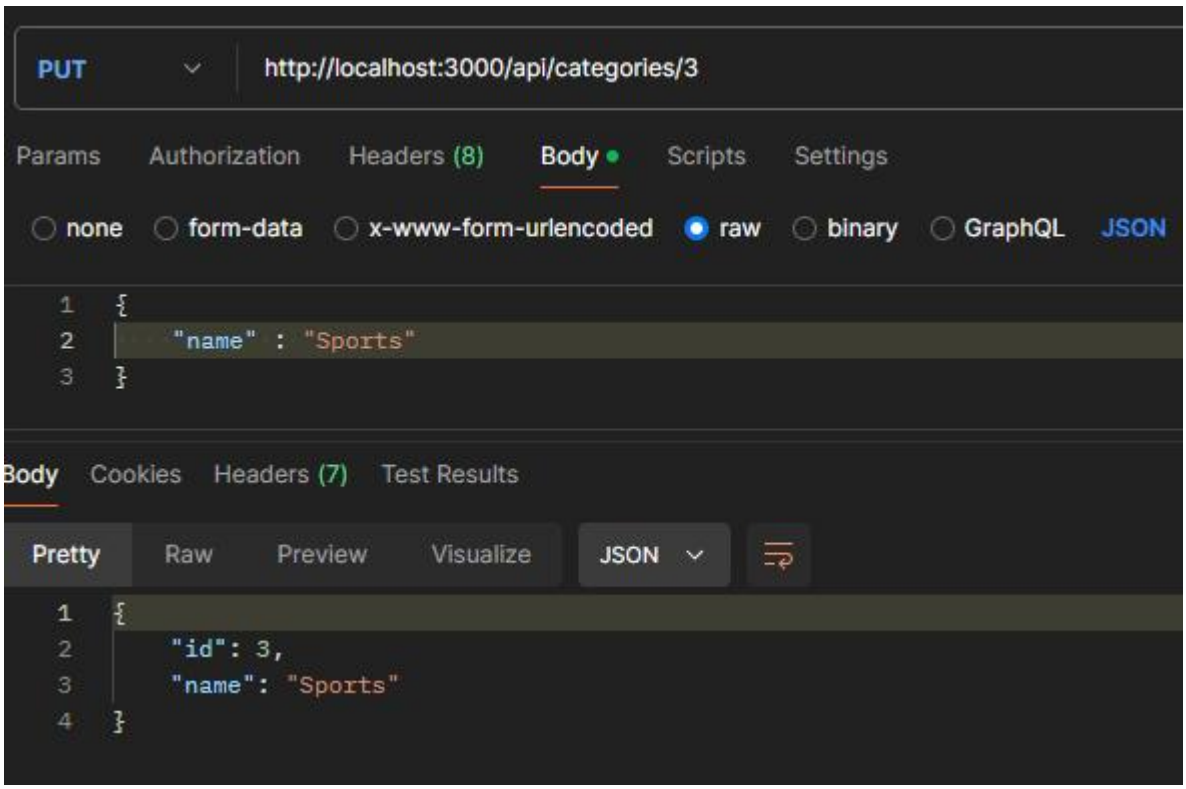
vii. Open ‘postman’ and send GET ‘`http://localhost:3000/api/categories`’:



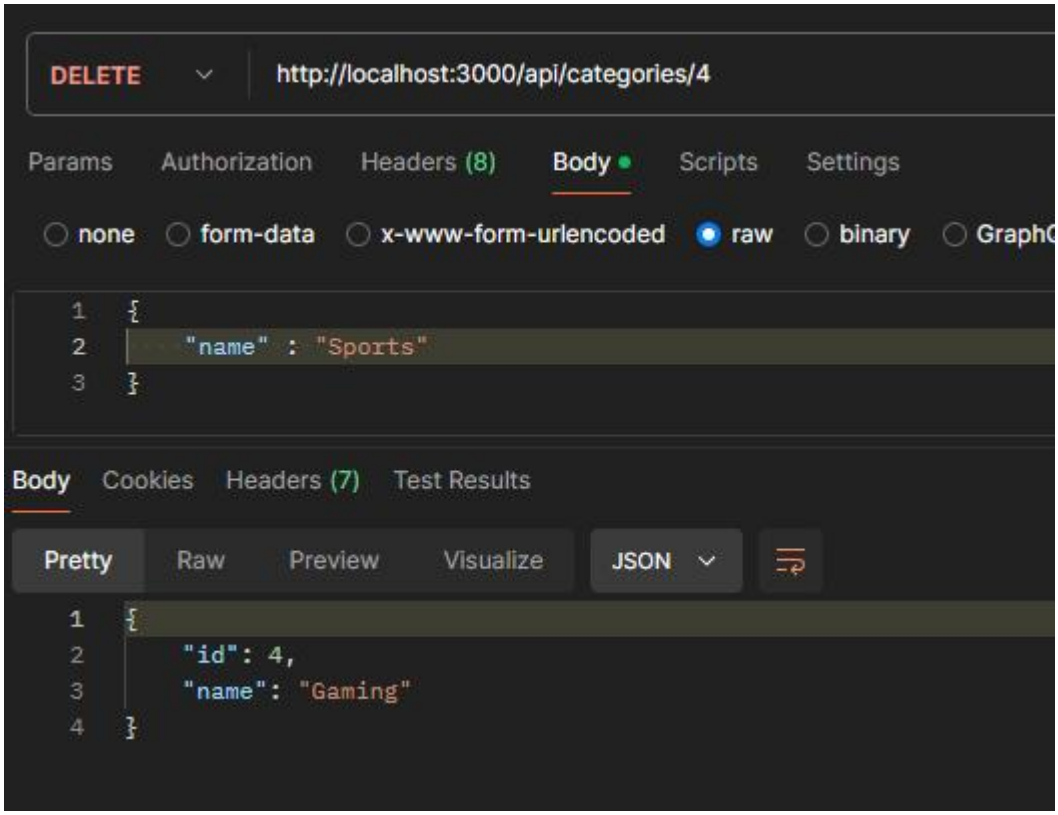
viii. Add another category .for that send POST ‘`http://localhost:3000/api/categories`’ in postman :



ix. Update category 'photography' to 'Sports'.For that send PUT :'http://localhost:3000/api/categories/3'



x.Delete category 'Gaming' .For that send DELETE :


DELETE
http://localhost:3000/api/categories/4
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
{
"name" : "Sports"
}
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 4,
"name": "Gaming"
}

b) Managing Routes :

i. Create a separate Folder 'Routes'.And in that create a file 'caregories.js' to store all routes.

ii. Cut and paste codes from 'app.js' to 'categories.js' , import express,set route ,change all 'app' to 'router' and export router:

```
const express = require('express');//Importing express.

const router = express.Router();//setting route

const categories = [
    {id:1 , name : 'Web'},
    {id:2 , name : 'Mobile'},
    {id:3 , name : 'Photography'},
]
//GET:
router.get('/api/categories',(req , res ) => {
    res.send(categories);
});
//POST :Create
router.post('/api/categories',(req , res ) => {
    const category = {
        id : categories.length+1, //new id (3+1=4)
        name : req.body.name //Whatever the name we pass as req in postman body>raw
it will be set as the name field of categories array
    };
    categories.push(category);
    res.send(category);
});
//PUT : Update
router.put('/api/categories/:id', (req , res)=> {
    const category = categories.find(c => c.id === parseInt(req.params.id));//Finds
the particular course with id.
     if(!category) return res.status(404).send('The category with given ID was not
found');
    //  if(error) return res.status(400).send(error.details[0].message);
     category.name = req.body.name;//Updating name
     res.send(category);
})
//DELETE : delete
router.delete('/api/categories/:id',(req , res)=>{
    const  category = categories.find(c => c.id === parseInt(req.params.id));
//Finds the particular course with id.
    if(!category) return res.status(404).send('The genre with the given ID was not
found.');
    const index = categories.indexOf(category);//To find the index .
    categories.splice(index , 1);//To delete that index from array.
    res.send(category);
})
```

```
router.get('/api/categories/:id', (req , res)=> {
    const category = categories.find(c => c.id === parseInt(req.params.id));//Finds
the particular course with id.
     if(!category) return res.status(404).send('The genre with given ID was not
found');
     res.send(category);
})
module.exports = router ;//exporting router
```

iii. Import routes in 'app.js' :

```
const express = require('express');//Importing express.

const categories = require("./Routes/categories")

const app = express();//calling express.

app.use(categories);
app.use(express.json()); // Middleware to parse JSON bodies.

//ENVIRONMENT VARIABLE - for PORTs
const port = process.env.PORT || 3000//for static
app.listen(port,()=> console.log(`Port is Running on ${port}`));
```

c) Data Validation with Joi :

i. We have a specific package called 'Joi' for validation.

ii. Install 'Joi' : 'npm install joi@13.1.0 ' :

```
npm install joi@13.1.0
```

iii. Import 'joi' in 'categories.js' :

```
const Joi = require('joi');//import joi
```

iv. Create a function for validation in 'categories.js' :

```
//VALIDATION PART
function validateData(category){
    const schema = {
        name : Joi.string().min(3).required() //name should be with min length
3
    }

    return Joi.validate(category , schema);
}
```

v.Then use that function inside POST method :

```
//POST :Create
router.post('/api/categories',(req , res ) => {

    const {error} = validateData(req.body)
    if(error) res.send(400).send(error.details[0].message)
    const category = {
        id : categories.length+1, //new id (3+1=4)
        name : req.body.name //Whatever the name we pass as req in postman body>raw
it will be set as the name field of categories array
    };
    categories.push(category);
    res.send(category);
});
```

vi. Now test it in postman by sending POST request :

d) Integrating the Database :

i. To Integrate First import 'mongoose' to 'app.js'.

```
const mongoose = require('mongoose');//import mongoose
```

ii. Then specify the connection code to connect to DB with DB name 'learningPlatform':

```
mongoose.connect('mongodb://127.0.0.1/learningPlatform') //To connect to
DB.(The url with default localhost adrress :127.0.0.1 and dbname:testDatabase)
    .then(() => console.log('Connection is Successfull')) //If connection is
successfull
    .catch(err => console.log('Coud not connect to mongodb', err))
```

iii. Then open 'MongoDB' and connect to localhost and run 'app.js' .

```
[nodemon] starting `node app.js`
Port is Running on 3000
Connection is Successfull
```

iv. For now we have static working of data.Whenever we use a POST method data is added .But when you run program again it goes back default data.

v.So now we need actual database for to store these data.

vi. For that first Import 'mongoose' in 'categories.js' also :

```
const mongoose = require('mongoose');//import mongoose
```

vii. Then create a 'categorySchema' :

```
//Create a schema:
const categorySchema = new mongoose.Schema({
    name : {type : String , required : true , minlength : 3 , maxlength :
30 }
})
```

viii. Then Create a Model 'Category' for that :

```
//Create a Model :
const Category = mongoose.model('Category' , categorySchema);//'Category' is
model based on schema 'categorySchema' .
```

ix. Remove the old static categories array.

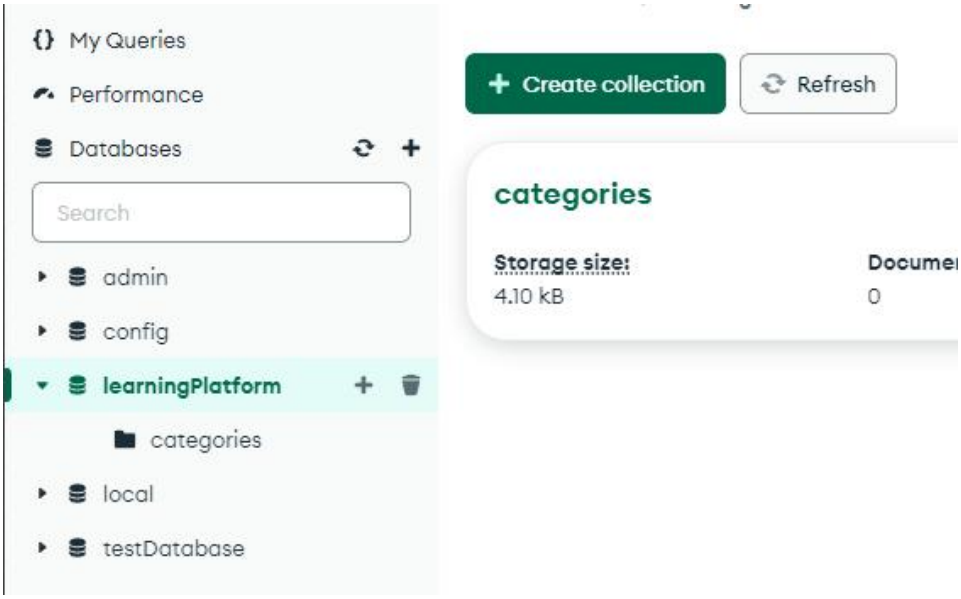
```
// const categories = [
//     {id:1 , name : 'Web'},
//     {id:2 , name : 'Mobile'},
//     {id:3 , name : 'Photography'},
// ]
//GET:
```

x.Now add database code to GET method (add async in function to use await):

```
//GET:
router.get('/api/categories',async (req , res ) => {
    let categories = await Category.find();//finds data from database
    res.send(categories);
});
```

xi. In POST method remove id section (since in MongoDB id is auto assigned) and edit to objects.And Remove push() code and Add save() code instead,also add async to use await:

```
//POST :Create
router.post('/api/categories',async (req , res ) => {

    const {error} = validateData(req.body)
    if(error) res.send(400).send(error.details[0].message)
    const category = new Category({
        name : req.body.name
})
    await category.save();
    res.send(category);
});
```



e) Testing the Categories Route :

i. Comment from PUT method last GET request :

```
//PUT : Update
// router.put('/api/categories/:id', (req , res)=> {
//     const category = categories.find(c => c.id ===
parseInt(req.params.id));//Finds the particular course with id.
//      if(!category) return res.status(404).send('The category with given ID was
not found');

//     //  if(error) return res.status(400).send(error.details[0].message);
```

```
//      category.name = req.body.name;//Updating name
//      res.send(category);
// })
```

```
// //DELETE : delete
// router.delete('/api/categories/:id',(req , res)=>{
//     const  category = categories.find(c => c.id === parseInt(req.params.id));
//Finds the particular course with id.
//     if(!category) return res.status(404).send('The genre with the given ID was
not found.');
```

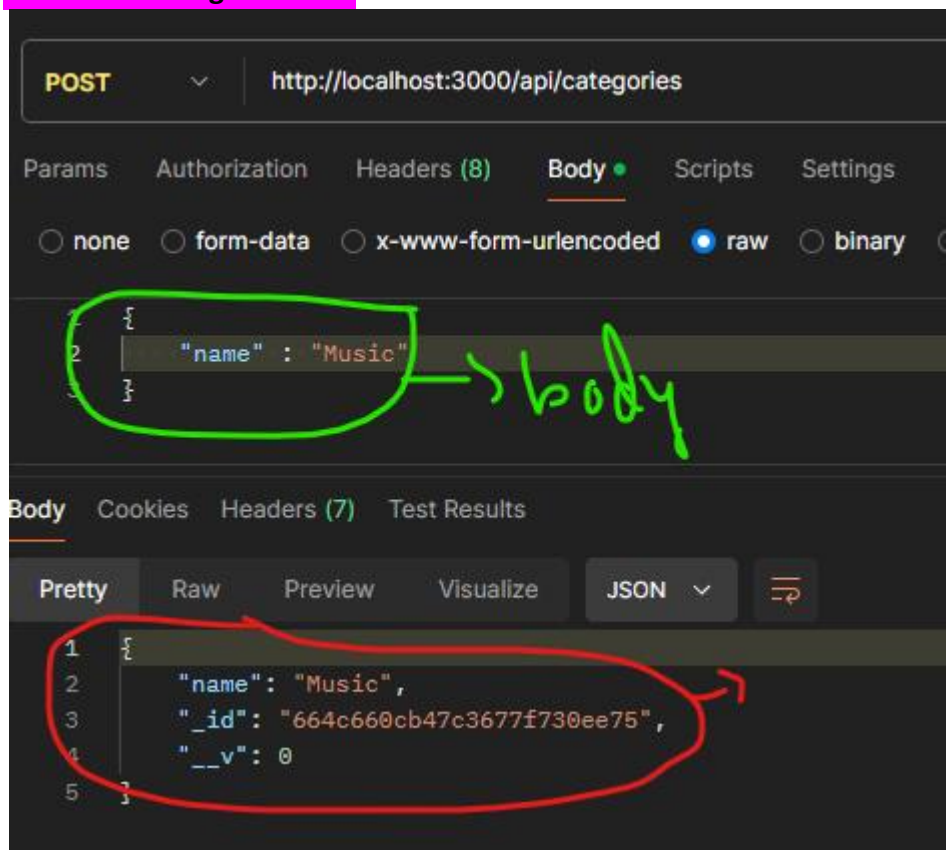
```
//     const index = categories.indexOf(category);//To find the index .
//     categories.splice(index , 1);//To delete that index from array.
//     res.send(category);
// })
```

```
// router.get('/api/categories/:id', (req , res)=> {
//     const category = categories.find(c => c.id ===
parseInt(req.params.id));//Finds the particular course with id.
//      if(!category) return res.status(404).send('The genre with given ID was not
found');
//      res.send(category);
// })
```

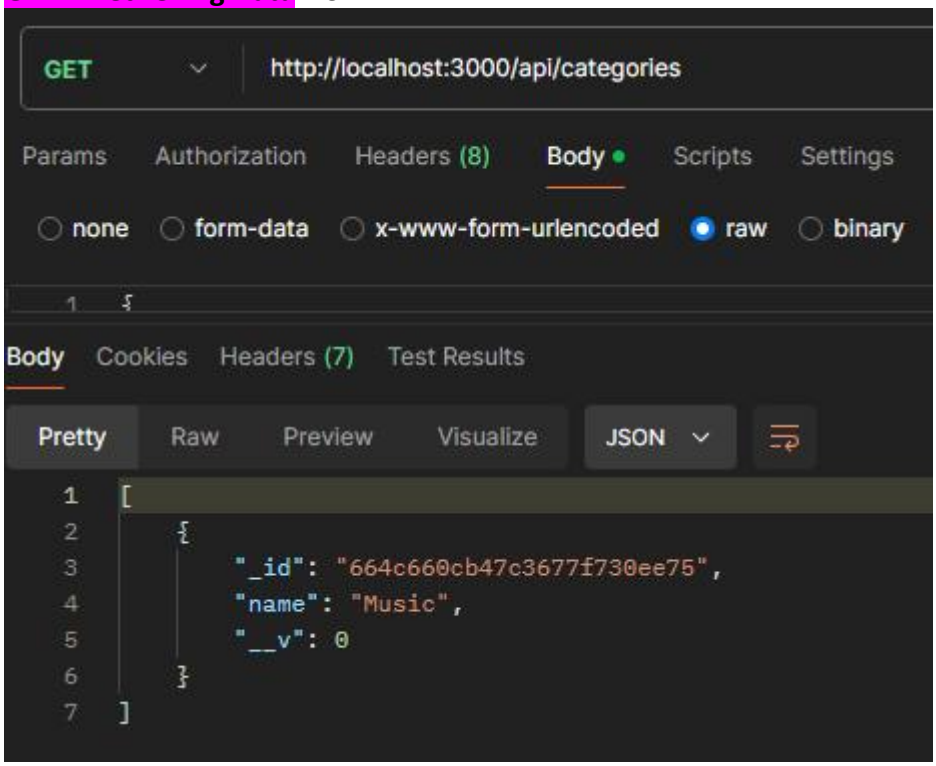
ii. Then run ‘app.js’.

iii. Then open postman and test the GET and POST method we already created.

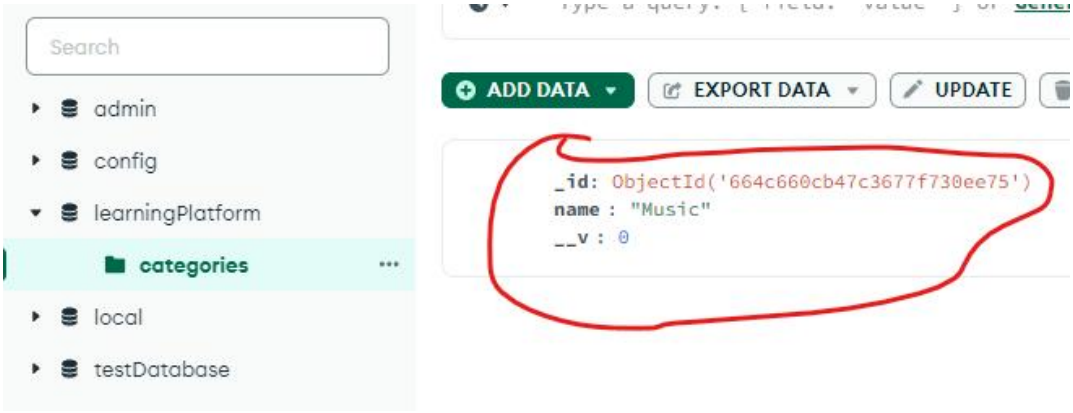
POST -> Creating new data 'Music' DB:



GET ->Retrieving Data from DB :



iv. Now if you Goto MongoDB Compass you can see created data there:



v.Like this Add one more data 'Web Development':

```
_id: ObjectId('664c660cb47c3677f730ee75')
name : "Music"
__v : 0
```

```
_id: ObjectId('664c67fe40b271c017046cf8')
name : "Web Development"
__v : 0
```

vi. Now UNComment the PUT request first .

vii. Remove finding course with id line(since this was used when not working with DB) :

```
    const category = categories.find(c => c.id ===
parseInt(req.params.id));//Finds the particular course with id.
```

viii. Instead of that Add a new line to find name by id and update it:(add async in function to use await)

```
//Updating by id in DB
    const category = await Category.findByIdAndUpdate(req.params.id , {name :
req.body.name} ,{new : true})
```

ix. Add validation part :

```
//validation part:
    const {error} = validateData(req.body)
    if(error) res.send(400).send(error.details[0].message)
```

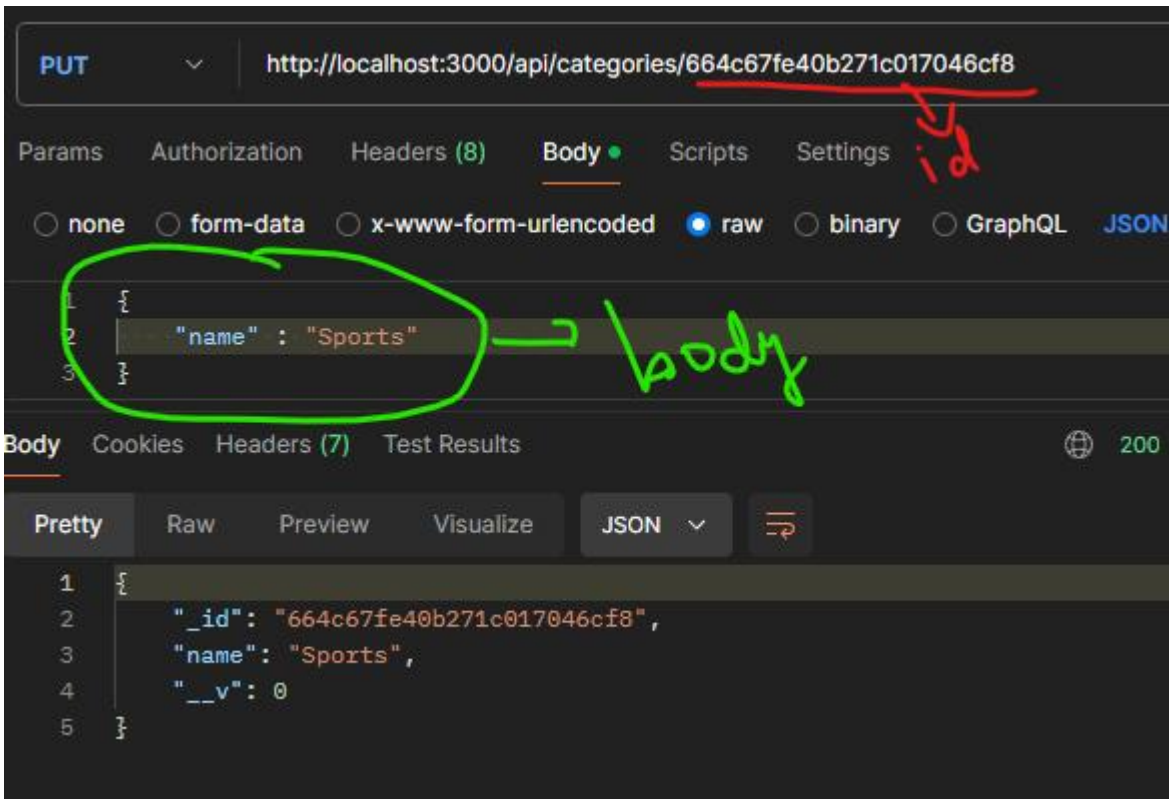
x.Remove old line :

```
 category.name = req.body.name;//Updating name
```

xi. Final code of PUT request:

```
// //PUT : Update
router.put('/api/categories/:id', async (req , res)=> {
    //validation part:
    const {error} = validateData(req.body)
    if(error) res.send(400).send(error.details[0].message)
    //Updating by id in DB:
    const category = await  Category.findByIdAndUpdate(req.params.id , {name :
req.body.name} ,{new : true})
     if(!category) return res.status(404).send('The category with given ID was not
found');
     res.send(category);
})
```

xii. Now Goto postman and try Updating 'Web Develpoment' to 'Sports' :



xiii. Now Goto MongoDB Compass and check for updation:

xiv. Now UNComment the DELETE request and make changes.

```
//DELETE : delete
router.delete('/api/categories/:id', async (req , res)=>{

    const category = await  Category.findByIdAndDelete(req.params.id)//category with
passed id will be deleted here
    if(!category) return res.status(404).send('The genre with the given ID was not
found.');
    res.send(category);
})
```

xv. Now UNComment the last GET request and make changes :

```
router.get('/api/categories/:id', async (req , res)=> {
    const category = await  Category.findById(req.params.id)//Finds the particular
course with id.
     if(!category) return res.status(404).send('The category with given ID was not
found');
     res.send(category);
})
```

xvi. Now Goto Postman and test these chnages.

f) Building the Students API :

i. We will creating an API end point or route for students.

ii. First create a file 'students.js' inside folder 'Routes'.

iii. Then Goto 'app.js' and add the route to 'students.js' .

```
const students = require("./Routes/students")//import students from Routes folder
```

NOTE:

- In 'categories.js' ->Instead of having route url like '/api/categories' we can keep it as just '/'.And specify the url in 'app.js' :

```
app.use('/api/categories',categories);
```

- So change all route in 'categories.js' from '/api/categories' to just '/' .
- And then add that url in app.use() of 'app.js' :

```
app.use('/api/categories',categories);
```

iv. And Use that imported 'students.js' :

```
app.use('/api/students',students);
```

v. Goto 'students.js' and paste all contents of 'categories.js' and edit it to perform a student details:

```
const express = require('express');//Importing express.
const router = express.Router();//setting route
const mongoose = require('mongoose');//import mongoose
const Joi = require('joi');//import joi
```

```
//Create a schema:
const studentSchema = new mongoose.Schema({
    name : {type : String , required : true , minlength : 3 , maxlength : 30 } ,
    isEnrolled : {
        type : Boolean,
        default : false
    },
```

```
    Phone :{
        type : String,
        required : true,
        minlength : 10 ,
        maxlength : 25
    }
})
```

- Like mentioned in NOTE, Do it for ‘students.js’ also:

```
//Create a Model :
const Student = mongoose.model('Student' , studentSchema);//'Student' is model based
on schema 'studentSchema' .
```

```
//GET:
router.get('/',async (req , res ) => {
    let students = await Student.find();//finds data from database
    res.send(students);
});
```

```
//POST :Create
router.post('/',async (req , res ) => {

    const {error} = validateData(req.body)
    if(error) res.send(400).send(error.details[0].message)
    const student = new Student({
        name : req.body.name,
        isEnrolled : req.body.isEnrolled,
        Phone : req.body.Phone
})
    await student.save();
    res.send(student);
});
```

```
// //PUT : Update
router.put('/:id', async (req , res)=> {
    //validation part:
    const {error} = validateData(req.body)
    if(error) res.send(400).send(error.details[0].message)
    //Updating by id in DB:
    const student = await  Student.findByIdAndUpdate(req.params.id , {name :
req.body.name , Phone : req.body.Phone , isEnrolled : req.body.isEnrolled} ,{new :
true})
     if(!student) return res.status(404).send('The student with given ID was not
found');
     res.send(student);
})
```

```
//DELETE : delete
router.delete('/:id', async (req , res)=>{

    const student = await Student.findByIdAndDelete(req.params.id)//student with
passed id will be deleted here
    if(!student) return res.status(404).send('The student with the given ID was not
found.');
    res.send(student);
})
```

```
router.get('/:id', async (req , res)=> {
    const student = await  Student.findById(req.params.id)//Finds the particular
student with id.
     if(!student) return res.status(404).send('The student with given ID was not
found');
     res.send(student);
})
```

```
//VALIDATION PART
function validateData(student){
    const schema = {
        name : Joi.string().min(3).max(50).required(), //name should be with min
length 3 and max 50
        Phone : Joi.string().min(10).max(50).required(),
        isEnrolled : Joi.boolean()
    }

    return Joi.validate(student , schema);
}
module.exports = router ;//exporting router
```

g) Organizing the App :

i. Single Responsibility Principle -(It says that as we have created 'categories.js' and 'student.js' , it should not have the Schema and Models such codes.So we have to keep that in separate file.).

ii. For that first create a folder 'models' inside project folder.

iii. Inside that folder create a file 'categoriesModel.js' .

iv. Now cut and paste all schema, model ,validation , mongoose and joi codes from 'categories.js' to 'categoriesModel.js'.Add export code also.

```
const mongoose = require('mongoose');//import mongoose

const Joi = require('joi');//import joi
//Create a schema:
const categorySchema = new mongoose.Schema({
    name : {type : String , required : true , minlength : 3 , maxlength : 30 }
})
//Create a Model :
const Category = mongoose.model('Category' , categorySchema);//'Category' is model
based on schema 'categorySchema' .
//VALIDATION PART
function validateData(category){
    const schema = {
        name : Joi.string().min(3).required() //name should be with min length 3
    }
    return Joi.validate(category , schema);
}
//exporting to 'categories.js'
exports.Category = Category
exports.validate = validateData
```

v.Now we have import these in 'categories.js' file :

```
const {Category , validate} =require('../models/categoriesModel');//imporing the
ones exported from 'categoriesModel'
```

vi. Now wherever you have 'validateData' replace it with 'validate'.

vii. Similarly Do these for 'students.js' also.

viii. Inside the folder ‘models’ create a file ‘studentsModel.js’.

ix. Now cut and paste all schema, model ,validation , mongoose and joi codes from ‘students.js’ to ‘studentsModel.js’.Add export code also.

```
const mongoose = require('mongoose');//import mongoose
const Joi = require('joi');//import joi

//Create a schema:
const studentSchema = new mongoose.Schema({
    name : {type : String , required : true , minlength : 3 , maxlength : 30 } ,
    isEnrolled : {
        type : Boolean,
        default : false
    },

    Phone :{
        type : String,
        required : true,
        minlength : 10 ,
        maxlength : 25
    }
})
//Create a Model :
const Student = mongoose.model('Student' , studentSchema);//'Student' is model based
on schema 'studentSchema' .

//VALIDATION PART
function validateData(student){
    const schema = {
        name : Joi.string().min(3).max(50).required(), //name should be with min
length 3 and max 50
        Phone : Joi.string().min(10).max(50).required(),
        isEnrolled : Joi.boolean()
    }
    return Joi.validate(student , schema);
}
//exporting to 'students.js'
exports.Student = Student
exports.validate = validateData
```

x.Now we have import these in ‘students.js’ file.

```
const {Student , validate} =require('../models/studentsModel');//imporing the ones
exported from 'studentsModel'
```

xi. Now wherever you have ‘validateData’ replace it with ‘validate’.

h) Building the course API :

i. Inside the folder ‘models’ create a file ‘courseModel.js’:

```
const mongoose = require('mongoose');//import mongoose
const Joi = require('joi');//import joi
const { Course } = require('./coursesModel');

//Create a schema:
const courseSchema = new mongoose.Schema({
    title : {
        type : String ,
         required : true ,
         trim : true ,
         minlength : 5 ,
         maxlength : 255
        } ,
    category : { //Here we will embbed another document(we have to select from
already created category)
    },

    creator :{
        type : String,
        required : true,
    },
    rating :{
        type : Number,
        required : true,
    }
})
//Create a Model :
const Course = mongoose.model('Course' , courseSchema);//'Course' is model based on
schema 'courseSchema' .

//VALIDATION PART
function validateCourse(course){
    const schema = {
        title : Joi.string().min(5).max(255).required(),
        categoryId : Joi.string().required(),
        creator : Joi.string().min(5).required(),
        rating : Joi.number().min(0).required()
    }
    return Joi.validate(course , schema);
}
//exporting to 'courses.js'
exports.Course = Course;
exports.validate = validateCourse;
```

ii. Now ,In 'categoriesModel.js' export its schema :

```
exports.categorySchema = categorySchema
```

iii. Then , Goto 'courseModel.js Import the exported schema from 'categoriesModel.js':

```
//import categorySchema from categoriesModel:
const {categorySchema} = require('../models/categoriesModel');
```

iv. Then,Inside courseSchema add 'type:categorySchema' :

```
    category : { //Here we will embbed another document(we have to select from
already created category)
        type : categorySchema,
        required : true
    },
```

v.Create a file 'courses.js' inside Routes folder .

```
const express = require('express');//Importing express.
const {Course , validate} = require('../models/coursesModel');//imporing the ones
exported from 'coursesModel'
//import category to get category while entering student details:
const {Category , validate} = require('../models/categoriesModel');

const router = express.Router();//setting route
```

```
//GET:
router.get('/',async (req , res ) => {
    let courses = await Course.find();//finds data from database
    res.send(courses);
});
```

```
//POST :Create
router.post('/',async (req , res ) => {
    const {error} = validate(req.body)
    if(error) res.send(400).send(error.details[0].message)
    //First find category by id from Category
    const category = await Category.findById(request.body.categoryId)
    if(!category) return res.status(400).send('Invalid ID ');
    let course = new Course({
        title : req.body.title,
        category: {
            _id : category._id,
            name : category.name
        },
        creator : req.body.creator,
        rating : req.body.rating
})
```

```
    await course.save();
    res.send(course);
});

// //PUT : Update
router.put('/:id', async (req , res)=> {
    //validation part:
    const {error} = validate(req.body)
    if(error) res.send(400).send(error.details[0].message)
    //First find category by id from Category
    const category = await Category.findById(request.body.categoryId)
    if(!category) return res.status(400).send('Invalid ID ');
    //Updating by id in DB:
    const course = await  Course.findByIdAndUpdate(req.params.id ,
        {
            title : req.body.title,
            category: {
            _id : category._id,
            name : category.name
            },
            creator : req.body.creator,
            rating : req.body.rating
        } ,{new : true})
     if(!course) return res.status(404).send('The course with given ID was not
found');
     res.send(course);
})

//DELETE : delete
router.delete('/:id', async (req , res)=>{
    const course = await Course.findByIdAndDelete(req.params.id)//course with passed
id will be deleted here
    if(!course) return res.status(404).send('The course with the given ID was not
found.');
    res.send(course);
})
router.get('/:id', async (req , res)=> {
    const course = await  Course.findById(req.params.id)//Finds the particular
course with id.
     if(!course) return res.status(404).send('The course with given ID was not
found');
     res.send(course);
})
module.exports = router ;//exporting router
```