

NodeJS

Setup Installation of the NodeJS's:

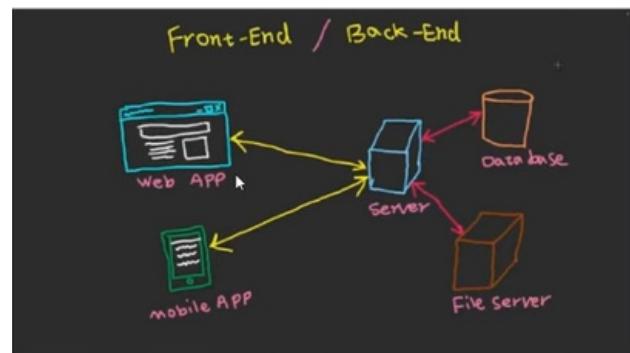
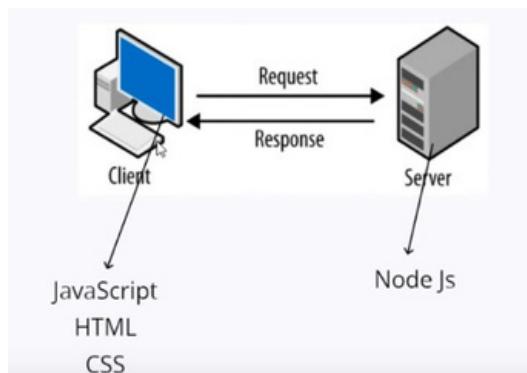
- Click here to download
-
- Use the sudo su mode:
- curl -o- <https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh> | bash
- source ~`/.bashrc`
- nvm list-remote
- nvm install <nvm ->version>
- nvm install **22.13.0**
- node -v || node -- version [//To Check](#)

- YouTube Channel: Code Step By Step

Features:

- - Node is using JavaScript internally and using V8 engine.
- - NodeJS is a server side environment and JS run on the browser side environment
- - You are able to connect to the Database
- - It mostly used for API. So, we can connect to the same database with Web-App and Mobile App
- - It is super – fast for APIs

Basic Before getting into:



- It communicates between them through the APIs

- Open the Directory with super-user:

- codium . --no-sandbox --puser-data-dir /home/**your_user_name**/dir

- Always create a file with the name of the – “index.js”

Code:

```
// Printing the program
console.log("Madara Uchiha");

// Taking the var as int, float and double
var a = 20.5; // "a" is a Float
var b = 10; // "b" is a Int
var c = "Hello"; // "c" is a String

var f = true; // "f" is a boolean

var d = a + b;

console.log(d); // For printing.
```

- Is "console.log()" from browser and IDE's same?

-> No. (Reason – Search it on internet)

Fundamentals of Node JS:

Code of Data-types:

```
var x = 20; // "var" can be re-declared anywhere, and it is mostly used in the global declarationn of  
the variables
```

```
console.log(x);
```

```
var x = 40; //Here, you can see that we are re-declaring the x = 40.
```

```
console.log(x);
```

```
// Now, let's talk about the "let":
```

```
let y = 10; //It is used in the local declaration in the functions or loops
```

```
// let y = 20; //You can't re-declare
```

```
y = 90; //But, you can re-assign the value to that one.
```

```
console.log(y);
```

```
// Now, "const":
```

```
const a = 80; //It is some similar to "let", but has very different properties:
```

```
// const a = 30; //Here, you can't re-declare the that const with same name again.
```

```
// a = 90; //Even though you can't re-assign the value to that same name's variable
```

```
console.log(a);
```

```
// But, you can make changes in the "object" of the "const"
```

```
const mad = {n: 1};
```

```
console.log(mad); //It gives an output - { n: 1 }
```

```
// Here, you can change the object's value -
```

```
mad.n = 3;
```

```
console.log(mad); //Here, it gives an output { n: 3}
```

- Output:

```
[root@parrot]~[~/home/keren/Code]  
● └─#node "/home/keren/Code/Node-1.0/second.js"  
20  
40  
90  
80  
{ n: 1 }  
{ n: 3 }
```

Code of Conditions:

```
var x = 20;  
if(x === 20){  
    console.log("matched");  
}
```

// Here, we are using the "===" , for the "Strict Equality":

// In the above it got matched and the answer is "matched"

// Now, if we compare the String with value then it doesn't give an answer:

```
var y = '229';  
if(y === 229){  
    console.log("Matched");  
}  
  
else{  
    console.log("Not matched");  
}
```

// Here, answer is - "not matched" because we are comparing that one for the Strict equality

```
var z = 30;  
if(z == '39'){  
    console.log("Done");  
}
```

// The above one doesn't give an answer

console.log(z == '39'); //Here, value is compared and give an answer in the comparision form.

```
console.log(5 == '54');
```

console.log(5 == '5'); // true (number 5 is converted to string '5')

console.log(true == 1); // true (boolean true is converted to number 1)

console.log(null == undefined); // true (null and undefined are considered equal)

console.log([1, 2] == '1,2'); // true (array is converted to string '1,2')

Output:

```
node "/home/keren/Code/Node-1.0/third.js"
[root@parrot-/home/keren/Code]
● └─ #node "/home/keren/Code/Node-1.0/third.js"
matched
Not matched
false
false
true
true
true
true
```

Code of an Array:

```
let numbers = [1, 2, 3, 4, 5]
```

```
let fruits = ["apple", "banana", "cherry"]
```

```
let mixedArray = [1, "hello-world", true, {key:"value"}, [1, 2, 3]]; // var, string, boolean and "key: value", and multidimensional array
```

// Here, we can add ";" at the end or not, it's our choice

```
console.log(numbers)
```

```
console.log(fruits)
```

```
console.log(mixedArray)
```

```
console.log(mixedArray[1])
```

Output:

```
[x] [root@parrot]-/home/keren/Code]
● └─ #node "/home/keren/Code/Node-1.0/forth.js"
[ 1, 2, 3, 4, 5 ]
[ 'apple', 'banana', 'cherry' ]
[ 1, 'hello-world', true, { key: 'value' }, [ 1, 2, 3 ] ]
hello-world
```

- We can't exports directly in the NodeJS like we exports in the JS.
- Here, we are trying to export and import from the two files:

```

sixth.js - Code - VSCodium [Superuser] (as superuser)
File Edit Selection View Go Run Terminal Help
JS fifth.js x ...
Node-1.0 > JS fifth.js
1 import {x, y} from './sixth'
2
3 console.log(x);
4 console.log(y);

JS sixth.js x ...
Node-1.0 > JS sixth.js > [e] y
1 export let x = 20
2 export let y = 30

```

- But, it gives an error:

```

[x]~[root@parrot]~[/home/keren/Code]
⑥ └─#node "/home/keren/Code/Node-1.0/fifth.js"
node:internal/modules/esm/resolve:275
  throw new ERR_MODULE_NOT_FOUND(
    ^

Error [ERR_MODULE_NOT_FOUND]: Cannot find module '/home/keren/Code/Node-1.0/sixth' imported from /home/keren/Code/Node-1.0/fifth.js
  at finalizeResolution (node:internal/modules/esm/resolve:275:11)
  at moduleResolve (node:internal/modules/esm/resolve:932:10)
  at defaultResolve (node:internal/modules/esm/resolve:1056:11)
  at ModuleLoader.defaultResolve (node:internal/modules/esm/loader:654:12)
  at #cachedDefaultResolve (node:internal/modules/esm/loader:603:25)
  at ModuleLoader.resolve (node:internal/modules/esm/loader:586:38)
  at ModuleLoader.getModuleJobForImport (node:internal/modules/esm/loader:242:38)
  at ModuleJob._link (node:internal/modules/esm/module_job:135:49) {
    code: 'ERR_MODULE_NOT_FOUND',
    url: 'file:///home/keren/Code/Node-1.0/sixth'
  }

```

Node.js v22.13.0

- This error is happened because in the NodeJS, it uses some little bit old JavaScript. That's why it can't handle that.
- To solve this we are using some different methods.

Sixth.js's file's code:

```

module.exports = {
  x: 10,
  y: 20,
  z: function () {
    return 11;
  },
}

```

```
}
```

//Here, we are making an object and putting all those value inside it
// Here, we are exporting the file's code to another file.

Fifth.js's file's code:

```
const app = require('./sixth.js')

console.log(app.x);
console.log(app.y);
console.log(app.z);
console.log(app);

//Here are updating the values with Particular object
// Here, we are importing the other file's object's value.
```

Output:

```
[root@parrot ~]# node "/home/keren/Code/Node-1.0/fifth.js"
10
20
[Function: z]
{ x: 10, y: 20, z: [Function: z] }
```

- What is the use of the **filter()** function?

-> filter is used to filter out with a particular condition, and it is going to printout that particular value.

E.g.: Code -

```
const arr = [2, 4, 7, 1, 3, 8, 3]

// Here, we are using an Arrow Function () => {}

let result = arr.filter((item) => {
  return item > 4;
})

console.log(result);
```

Output:

```
[root@parrot ~]# node "/home/keren/Code/Node-1.0/seventh.js"
[ 7, 8 ]
```

- Here, we are taking the values which were greater than 4 and it gives an output.

Core Module in Node Js

- It is like Library, which is known as the Core Module in the Node Js
- It doesn't require the additional installation
 - http and https: Create HTTP and HTTPS servers and clients
 - fs (File System): Interact with the file systems (Read, Write, Update and Delete files)
 - path: Handle and transform the paths
 - os: Provide operating system – related utility methods and properties
 - url: Parse and format URLs, etc.
- The module that need to import that is known as the **non – global module**. And vice – versa known as the **global module**.

Code:

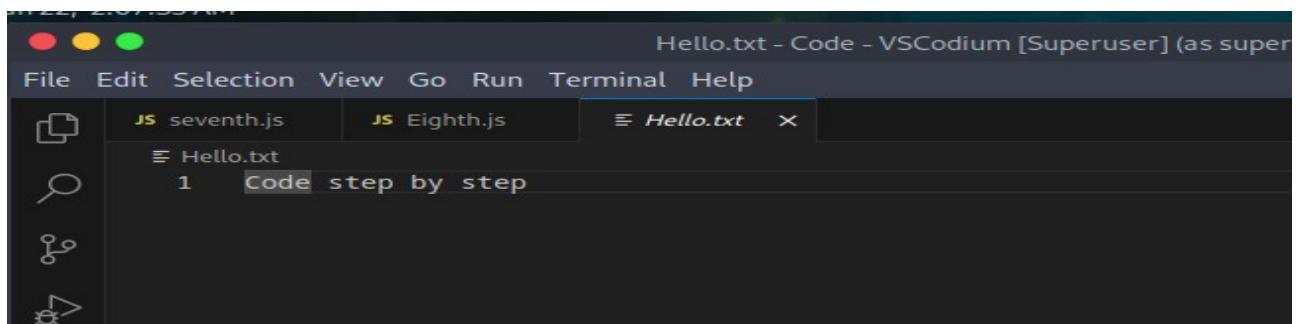
```
const fs = require('fs');

fs.writeFileSync("Hello.txt", "Code step by step");
```

Output:

```
[root@parrot]~[~/home/keren/Code]
● └─#node "/home/keren/Code/Node-1.0/Eighth.js"
```

- It creates a file named as the – “Hello.txt”, and text written inside that file is the next text that we have provided.



Code:

```
// Now, we are printing the name of the directory where we are performing these operations:
console.log(__dirname) //The output is the: /home/keren/Code/Node-1.0

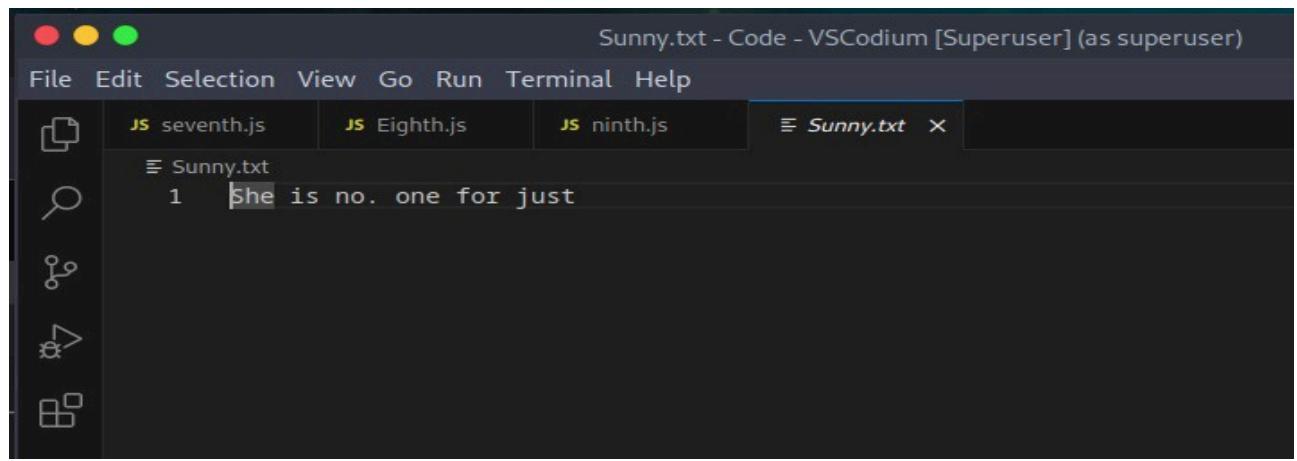
// Another way to create a write file:
```

```
const gs = require('fs').writeFileSync  
gs("Sunny.txt", "She is no. one for just")
```

Output:

```
[root@parrot]~[~/home/keren/Code]  
● └─#node "/home/keren/Code/Node-1.0/ninth.js"  
/home/keren/Code/Node-1.0
```

- It creates a file named as the Sunny.txt and here is the text inside that one:



-We can give any name of the variable (const) after const, like instead of “fs”, we can put the “gs”:

- `const gs = require('fs').writeFileSync;`

Basic Server:

Code:

```
const ht = require('http');

// ht.createServer().listen(4500);

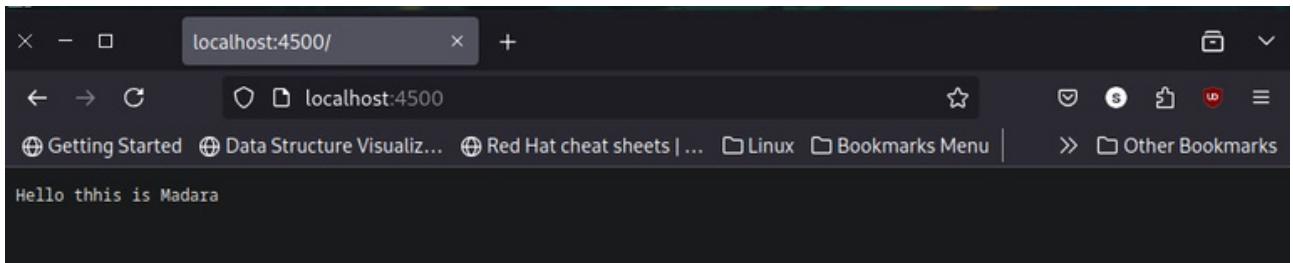
// Instead of passing variables, we can even pass the full function inside for the calculation
// test(a,b){
// }

// Now, we are using "an Arrow", function for passing a data.
// Here, we are taking this because we want to store the "request" and gave a "response", according
// to that

ht.createServer((req, resp) => {
  resp.write("Hello thhis is Madara");

  resp.end();
}).listen(4500)
```

Output:



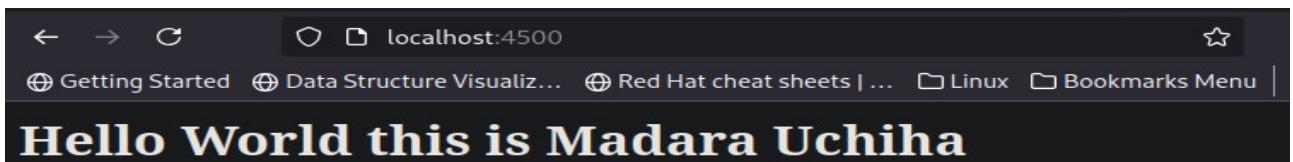
Now, we are creating a one function and add some HTML's elements:

Code:

```
// Now, we are runnnning our server for the HTML file:

function first(req, resp) {
  resp.write("<h1>Hello World this is Madara Uchiha </h1>")
  resp.end();
}

ht.createServer(first).listen(4500)
```



- Remember that you have to run the code again after changing some things in the Code.
- Try to create an Arrow function.

All About Package Json file:

It stores all the information about Project's imported module version, command's and packages details.

- To create a packages:

```
npm init #To create a packages
```

```
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
$ 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: (new-one)
version: (1.0.0)
description: This just an first projects example
entry point: (index.js) index.js
test command:
git repository: https://github.com/SujalBedre4/NodeJS
keywords:
author: Keren
license: (ISC)
About to write to /home/keren/Code/Node-1.0/New One/package.json:

{
  "name": "new-one",
  "version": "1.0.0",
  "description": "This just an first projects example",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/SujalBedre4/NodeJS.git"
  },
  "author": "Keren",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/SujalBedre4/NodeJS/issues"
  },
  "homepage": "https://github.com/SujalBedre4/NodeJS#readme"
}

Is this OK? (yes) yes
```

- Output of this file as follows:

- package.json:

```
{
  "name": "new-one",
  "version": "1.0.0",
```

```
"description": "This just an first projects example",
"main": "index.js",
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
"repository": {
  "type": "git",
  "url": "git+https://github.com/SujalBedre4/NodeJS.git"
},
"author": "Keren",
"license": "ISC",
"bugs": {
  "url": "https://github.com/SujalBedre4/NodeJS/issues"
},
"homepage": "https://github.com/SujalBedre4/NodeJS#readme"
}
```

- After installing the packages it will automatically add in this packages file:
- Here, we are using the "**yarn**" instead of the "**npm**" package manager.
- Now, add the colors's package:

```
- npm install colors | npm i colors
```

OR

```
- yarn add colors
```

- Output:

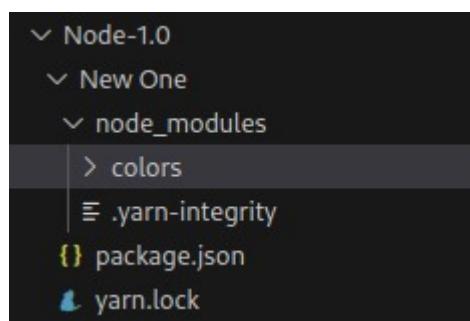
```
{
  "name": "new-one",
  "version": "1.0.0",
  "description": "This just an first projects example",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
```

```

"url": "git+https://github.com/SujalBedre4/NodeJS.git"
},
"author": "Keren",
"license": "ISC",
"bugs": {
"url": "https://github.com/SujalBedre4/NodeJS/issues"
},
"homepage": "https://github.com/SujalBedre4/NodeJS#readme",
"dependencies": {
"colors": "^1.4.0"
}
}

```

- Here, you can see that after adding the **colors** package it got updated at the end, means **dependencies section**.
- Even one directory also added after adding this package, which is known as the - **node-modules**.



- Here, we are using the **"yarn"**, that's why it creates a file named as the - **.yarn+integrity**
- This will stores all the details about the packages:

```
{
"systemParams": "linux-x64-127",
"modulesFolders": [
"node_modules"
],
"flags": [],
"linkedModules": [],
"topLevelPatterns": [
"colors@^1.4.0"
]
```

```
], "lockfileEntries": { "colors@^1.4.0": "https://registry.yarnpkg.com/colors/-/colors-1.4.0.tgz#c50491479d4c1bdaed2c9ced32cf7c7dc2360f78" }, "files": [], "artifacts": [] }
```

- Now, we are using a colors packages for doing some work:

Code:

```
const color = require('colors')

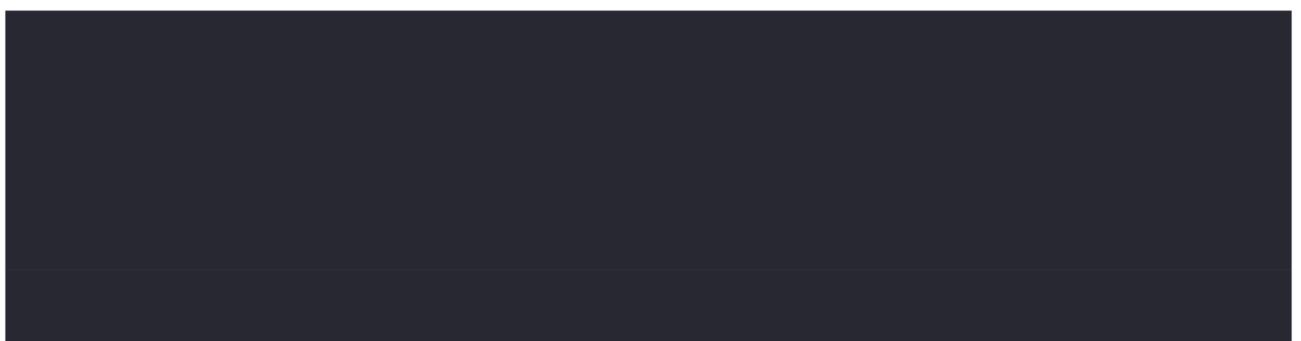
console.log("Hello".red);

console.log("Hello".blue);

console.log("Hello".gray);

console.log("Hello".yellow);
```

Output:



- Here, you can see that the console's log values has been changed.

- Questions:

- NodeJS is Single Threaded or Multi-threaded?

-> Single – threaded

Small Challenges for you:

- import chalk packages:

```
yarn add chalk
```

Code:

```
import chalk from 'chalk'

console.log(chalk.yellow('Hello world'));
```

Output:

```
[root@parrot]~[/home/keren/Code]
└─#node "/home/keren/Code/Node-1.0/New One/node_modules/index.js"
Hello world
```

- If someone deleted the node-modules directory, then we can re-add that directory by using the command:

```
yarn || npm install
```

```
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
└─#ls
node_modules package.json yarn.lock
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
└─#rm -rf node_modules/
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
└─#ls
package.json yarn.lock
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
└─#yarn
yarn install v1.22.22
warning .../package.json: No license field
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

Done in 0.07s.
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
└─#ls
node_modules package.json yarn.lock
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
└─#
```

- Here, you can see that we are able to re – add that directory.
- Never push the **node_modules** in the GitHub’s repository.
- To avoid to push this file in the GitHub, we have to create a `.gitignore` file for the better working.

```
touch .gitignore
```

Now, add the directory and other files that you don’t wanna push in the GitHub.

```
/node_modules
```

- Now, push the code to the GitHub’s repo:

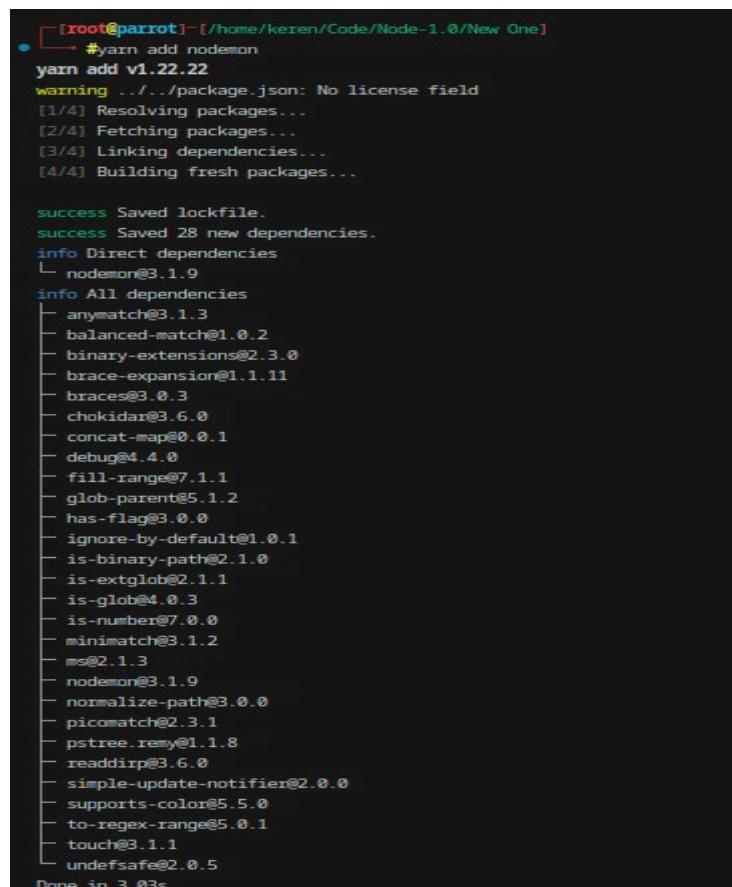
The screenshot shows a GitHub repository page for a private repository named 'NodeJS'. At the top, there are navigation links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights. Below the header, the repository name 'NodeJS' is displayed with a 'Private' badge. A notification bar indicates that a 'Node_module' had recent pushes 41 seconds ago, with a 'Compare & pull request' button. The main content area shows a commit history with four commits by 'First One': 'root' (7a0fd55), '.gitignore', 'package.json', and 'yarn.lock', all pushed 5 minutes ago. On the right side, there are sections for 'About', 'Activity' (1 watching, 0 stars, 0 forks), 'Releases' (no releases published, Create a new release), and 'Packages' (no packages published, Publish your first package).

- Here, you can see that we have pushed the code to the GitHub’s repository

Nodemon | time saving module:

- It is like a live – server, and it saves a time. It automatically runs the code.

- yarn add nodemon



```
[root@parrot]~/home/kezen/Code/Node-1.0/New One]
└─ #yarn add nodemon
yarn add v1.22.22
warning .../package.json: No license field
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 28 new dependencies.
info Direct dependencies
└── nodemon@3.1.9
info All dependencies
├── anymatch@3.1.3
├── balanced-match@1.0.2
├── binary-extensions@2.3.0
├── brace-expansion@1.1.11
├── braces@3.0.3
├── chokidar@3.6.0
├── concat-map@0.0.1
├── debug@4.4.0
├── fill-range@7.1.1
├── glob-parent@5.1.2
├── has-flag@3.0.0
├── ignore-by-default@1.0.1
├── is-binary-path@2.1.0
├── is-extglob@2.1.1
├── is-glob@4.0.3
├── is-number@7.0.0
└── minimatch@3.1.2
└── ms@2.1.3
└── nodemon@3.1.9
└── normalize-path@3.0.0
└── picomatch@2.3.1
└── ptree.remy@1.1.8
└── readdirp@3.6.0
└── simple-update-notifier@2.0.0
└── supports-color@5.5.0
└── to-regex-range@5.0.1
└── touch@3.1.1
└── undefsafe@2.0.5
Done in 3.03s.
```

- Here, how you can install with the – yarn package manager instead of the npm.

- To add globally packages through the “yarn”, we are using the given below command:

yarn global add nodemon

Or using the npm:

npm i nodemon -g

Index.js Code:

```
console.log("F")
console.log("F")
console.log("F")
console.log("F")
```

Output:

- After saving the code it will automatically run inside the – nodemon.

```
[root@parrot]~[/home/keren/Code/Node-1.0/New One]
└── #nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
F
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
F
F
F
F
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
F
F
F
F
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
```

- Node JS is async.
- In the Sync, it will run the command step – by – step. But, in the “**async**”, it will directly run without thinking of the other modules.

Make a Simple API:

- Here, we are creating a Static data model on the web – server.

First initialize the packages:

```
yarn init
```

```
[root@parrot]~[/home/keren/Code/Node-2.0]
└─# yarn init
yarn init v1.22.22
warning .../package.json: No license field
question name (Node-2.0): Simple_API
question version (1.0.0):
question description: Here, we are making a Simple API
question entry point (index.js): index.js
question repository url: https://github.com/SujalBedre4/NodeJS
question author: Sujal
question license (MIT):
question private:
success Saved package.json
Done in 66.53s.
```

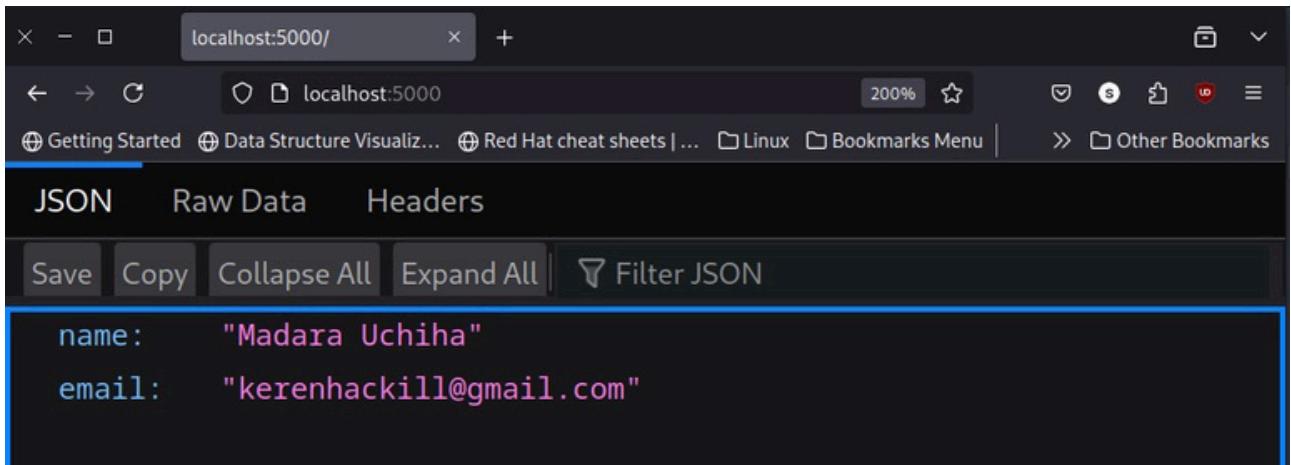
```
touch index.js
```

Code:

```
const http = require('http')
http.createServer((req, resp) => {
  resp.writeHead(200, { 'Content-Type': 'application/json' })
  resp.write(JSON.stringify({
    name: 'Madara Uchiha',
    email: 'kerenhackill@gmail.com',
  }))
  resp.end();
}).listen(5000, () => {
  console.log("Server is running on the localhost: http://localhost:5000")
})
```

- Output:

- Here, we are writing a static data and then passing the JSON format, for an output in the web – browser.



A screenshot of a web browser window titled "localhost:5000/". The address bar shows "localhost:5000". The page content is a JSON object:

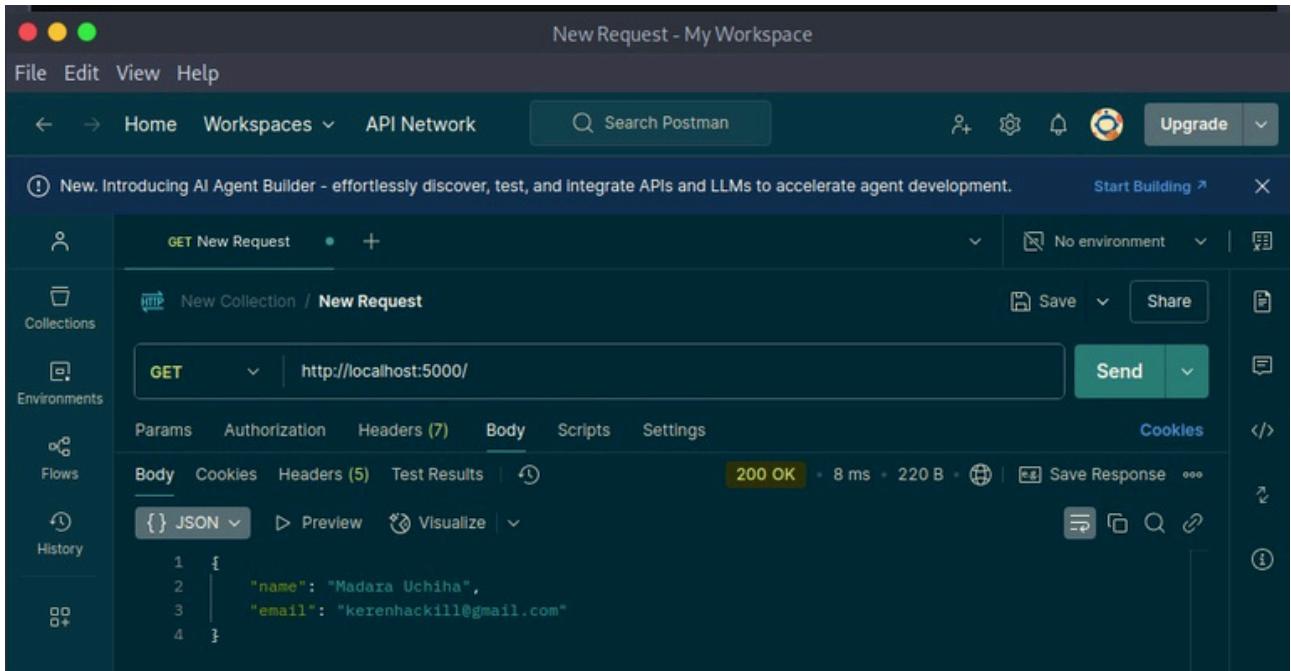
```
name: "Madara Uchiha"
email: "kerenhackill@gmail.com"
```

- Here, you can see that we are getting an output in the format of the JSON data.

- Now, we are using a POSTMAN.

- First install the POSTMAN, if you don't have
 - First download the POSTMAN
 - Setup the POSTMAN

- Now, create a separate collection after that put the URL (<http://localhost:5000/>), in the URL bar, with value of the GET.



A screenshot of the Postman application interface. The title bar says "New Request - My Workspace". The left sidebar shows "Collections", "Environments", "Flows", and "History". The main area shows a "GET New Request" with the URL "http://localhost:5000/". The "Body" tab is selected, showing the JSON response:

```
{}
{
  "name": "Madara Uchiha",
  "email": "kerenhackill@gmail.com"
}
```

- Here, you can see that we are getting the values. That we already got in the Browsers.

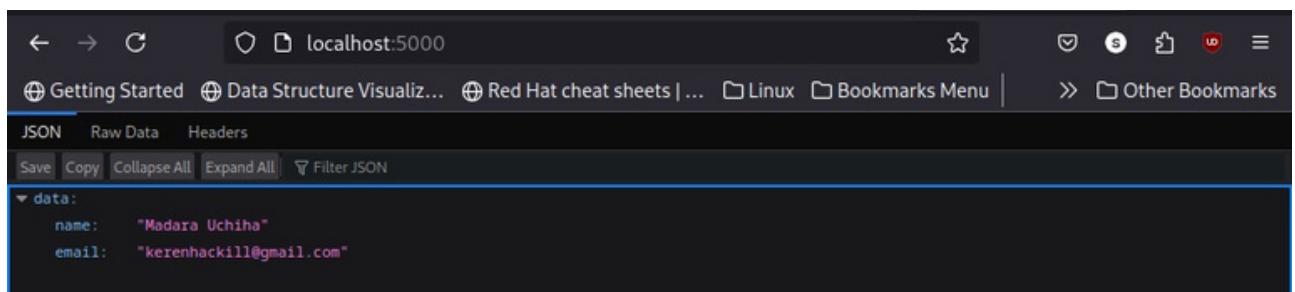
- And even "200", value is getting OK. After fetching the data.

- Now, we are connecting the two files:

Code:

```
const http = require('http')
// Now, here we are importing that data
const data = require('./data')
http.createServer((req, resp) => {
  resp.writeHead(200, { 'Content-Type': 'application/json' })
  resp.write(JSON.stringify({ data }))
  resp.end() // This is one necessary to execute a particular API.
}).listen(5000, () => {
  console.log("Server is running on the localhost: http://localhost:5000")
})
```

- Output:



- Here, is the new one output.

Now, we are changing only in the database, where we are taking the data in an Array format.

The screenshot shows a Postman collection interface. A GET request is made to 'http://localhost:5000/'. The response body is a JSON array:

```
[ { "data": { "name": "Madara Uchiha", "email": "kerenhackill@gmail.com" } } ]
```

Postman also displays the status code '200 OK' and response time '4 ms'.

- Here, is the updated

database:

Code:

```
const data = [ { name: 'Madara Uchiha', email: 'kerenhackill@gmail.com' }, ]
```

```

{ name: 'Madara ', email: 'hackill@gmail.com', },
{ name: 'Uchiha', email: 'kerenh@gmail.com', },
{ name: 'Sakamoto Days', email: 'sakamotodays@gmail.com', },
{ name: 'Sunny Leone', email: 'sunnyleone@gmail.com', },
]

module.exports = data;

```

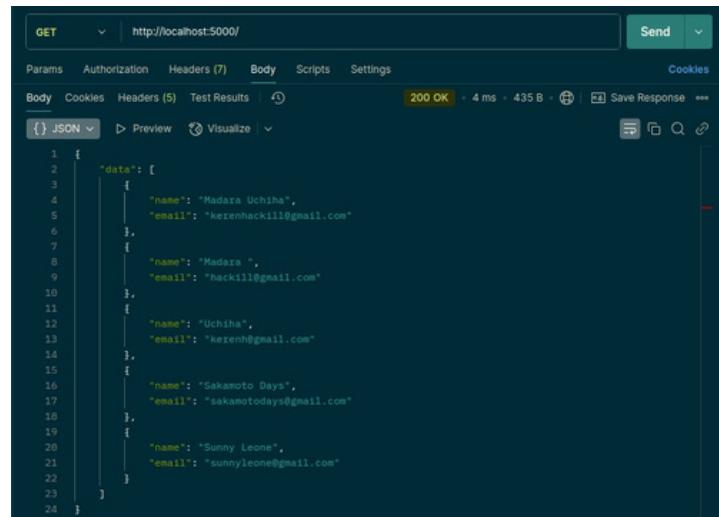
- Now, output of the system is the -

```

{
  "data": [
    {
      "name": "Madara Uchiha",
      "email": "kerenhhackill@gmail.com"
    },
    {
      "name": "Madara ",
      "email": "hackill@gmail.com"
    },
    {
      "name": "Uchiha",
      "email": "kerenh@gmail.com"
    },
    {
      "name": "Sakamoto Days",
      "email": "sakamotodays@gmail.com"
    },
    {
      "name": "Sunny Leone",
      "email": "sunnyleone@gmail.com"
    }
  ]
}

```

- And the output in the POSTMAN API is the:



- Now, Questions:

Why only we use the 200? Can we use other numbers?

-> Yes, we can use other HTTP status codes besides 200. The HTTP status code 200 indicates that the request was successful.

- 2XX Success:

- - **200** OK: The request was successful
- - **201** Created: The request was successful and a **new** resource was created
- - **204** No Content: The request was successful, but there is no content to send back.

- 3xx Redirection:

- - **301** Moved Permanently: The resource has been moved to a **new** URL permanently.
- - **302** Found: The resource has been moved temporarily to a different URL.
- - **304** Not Modified: The resource has not been modified since the last request.

- 4xx Client Error:

- - **400** Bad Request: The request could not be understood or was missing required parameters.
- - **401** Unauthorized: The client must authenticate itself to get the requested response.
- - **403** Forbidden: The client does not have access rights to the content.
- - **404** Not Found: The server can not find the requested resource.
- - **409** Conflict: The request could not be completed due to a conflict **with** the current state **of** the resource.

- 5xx Server Error:

- - **500** Internal Server **Error**: The server has encountered a situation it doesn't know how to handle.
- - **502** Bad Gateway: The server received an invalid response **from** the upstream server.
- - **503** Service Unavailable: The server is not ready to handle the request.

- Here, is proper numbers of the which indicates some errors, redirection and success.

Input from Command Line:

Code:

```
console.log("Code"); //It only run in the console  
// Now, we are just want some information about files of this program,  
// First is NodeJS's file's location and where the File of the Node JS is getting saved.  
  
console.log(process.argv); //argv = It means "argument vector"
```

- Output:

```
[root@parrot]~[/home/keren/Code]  
└─#node "/home/keren/Code/Node-3.0/index.js"  
Code  
[  
  '/root/.nvm/versions/node/v22.13.0/bin/node',  
  '/home/keren/Code/Node-3.0/index.js'  
]
```

Now, we are taking an input from the user and getting an output:

- Code:

```
// Now, we can take the particular input as an input  
// By running the "node file_name.js argument"  
// E.g.: node index.js Hello -> It will give an input.  
// Try this one command also:  
// node index.js Hello Hi  
console.log(process.argv[2]);
```

- Output:

```
[root@parrot]~[/home/keren/Code]  
└─#node "/home/keren/Code/Node-3.0/index.js" Hello Hi  
Code  
[  
  '/root/.nvm/versions/node/v22.13.0/bin/node',  
  '/home/keren/Code/Node-3.0/index.js',  
  'Hello',  
  'Hi'  
]  
Hello
```

- Now, we are using this in one of the usage.

Here, we are creating that one file for the creating a file's and giving an input:

Code:

```
const fs = require('fs')
const input = process.argv;
fs.writeFileSync(input[2], input[3]);
```

Output:

```
[root@parrot]~[/home/keren/Code/Node-3.0]
● └─ #node "/home/keren/Code/Node-3.0/just.js" madara.txt "This is a fruit"
```

- Now, you can see that **madara.txt** file has been generated with input text like – **This is a fruit**.

Now,

We are making a file operation where we are creating a file, delete or give an output.

Code:

```
const fs = require('fs')
const input = process.argv;
if (input[2] == 'add') {
    fs.writeFileSync(input[3], input[4]);
} else if (input[2] == 'remove') {
    fs.unlinkSync(input[3]);
} else {
    console.log("Invalid input")
}
```

- Here, you can see that we are creating a file with particular values and remove it.

```
[root@parrot]~[/home/keren/Code/Node-3.0]
● └─ #node "/home/keren/Code/Node-3.0/just.js" add orange.txt "This is a color and fruit"
```

- Now, we are removing:

```
[root@parrot]~[/home/keren/Code/Node-3.0]
● └─ #node "/home/keren/Code/Node-3.0/just.js" remove orange.txt
```

- You can change the file name or type and then it will get removed.

Show file List:

- Now, we are the different directories.

Code:

```
const fs = require('fs') // fs.writeFileSync("apple.txt","This is an apple.txt file") //-> It creates a file with given text // Now, we are creating a 10 - 15 file at a time: const path = require('path') // Now, we are taking the path // const dirPath = path.join(__dirname) //This gives me the path of the directorie's where code is running. // -> Here, we are getting a Path of the current directory. // console.warn(dirPath) const dirPath = path.join(__dirname, 'files') //Here, we are getting the path of // console.log(dirPath) for (i = 0; i < 5; i++) {
```

// fs.writeFileSync("apple.txt","A simple text file through the loop ")

// The above is executed at least 5 times and it overwrite again and again same files

// Now, we are creating a Dynamic file.

// fs.writeFileSync("apple" + i + ".txt", "A simple text file through the loop ")

// Or use the given below syntax:

// fs.writeFileSync(` apple\${i}.txt` , "A simple text file through the loop ")

// Here, we are running the code in the same directory that's why we didn't give the directorie's path.

// fs.unlinkSync(` apple\${i}.txt` , "A simple text file through the loop ") //Here, we are deleting those files.

// Now, we are creating files in the Particular directorie's path:

// fs.writeFileSync(dirPath, "apple" + i + ".txt", "A simple text file through the loop ")

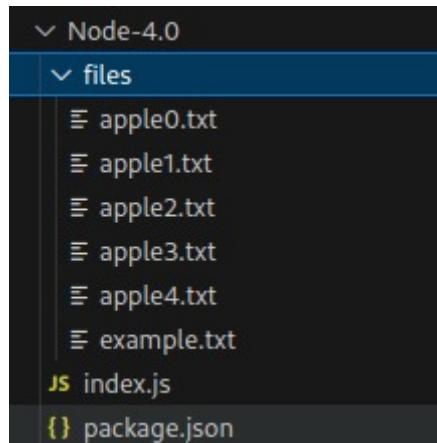
// fs.writeFileSync(dirPath + ` apple\${i}.txt` , "A simple text file through the loop ")

```
const filePath = path.join(dirPath, ` apple${i}.txt`)
```

```
fs.writeFileSync(filePath, "A simple text file through the loop")
```

```
}
```

- Output:



Code:

```
// Now, we are reading the file's content:  
fs.readdir(dirPath, (err, files) => {  
    // console.warn(files); //Here, we are getting a files in the format of an array. What if you just  
    // want a single file.\  
    files.forEach((item) => {  
        console.log("File name is:",item);  
    })  
})
```

- Output:

```
[root@parrot] - [/home/kezen/Code/Node-4.0]  
• └─ #node "/home/kezen/Code/Node-4.0/index.js"  
[  
    'apple0.txt',  
    'apple1.txt',  
    'apple2.txt',  
    'apple3.txt',  
    'apple4.txt',  
    'example.txt'  
]  
• └─ #node "/home/kezen/Code/Node-4.0"  
apple0.txt  
apple1.txt  
apple2.txt  
apple3.txt  
apple4.txt  
example.txt  
[root@parrot] - [/home/kezen/Code/Node-4.0]  
• └─ #node "/home/kezen/Code/Node-4.0/index.js"  
File name is: apple0.txt  
File name is: apple1.txt  
File name is: apple2.txt  
File name is: apple3.txt  
File name is: apple4.txt  
File name is: example.txt
```

Question:

- Can we access the files from the PC's other drives?

- > No, because whenever you are working in particular directory then it will create a web – server, and that's why it can't access out of the file's content.
- This is because security concern.

Asynchronous Programming Languages:

CRUD with FILE Systems (Create, Read, Update and Delete):

Code:

```
const fs = require('fs')

const path = require('path')

const dirPath = path.join(__dirname, "crud"); //It gives a path of the crud's

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

// fs.writeFileSync(filePath, "This is just") //This is for creating a file.

// Here, we have created a file inside the directory.

// Now, we want to read the code:

// fs.readFile(filePath, 'utf-8', (err, item) => {

    // // if(!err) // Now, we want to update a particular file, then here
    // is the condition if as per the file path the file name is apple.txt,
    // (err) => {

        console.log(item)

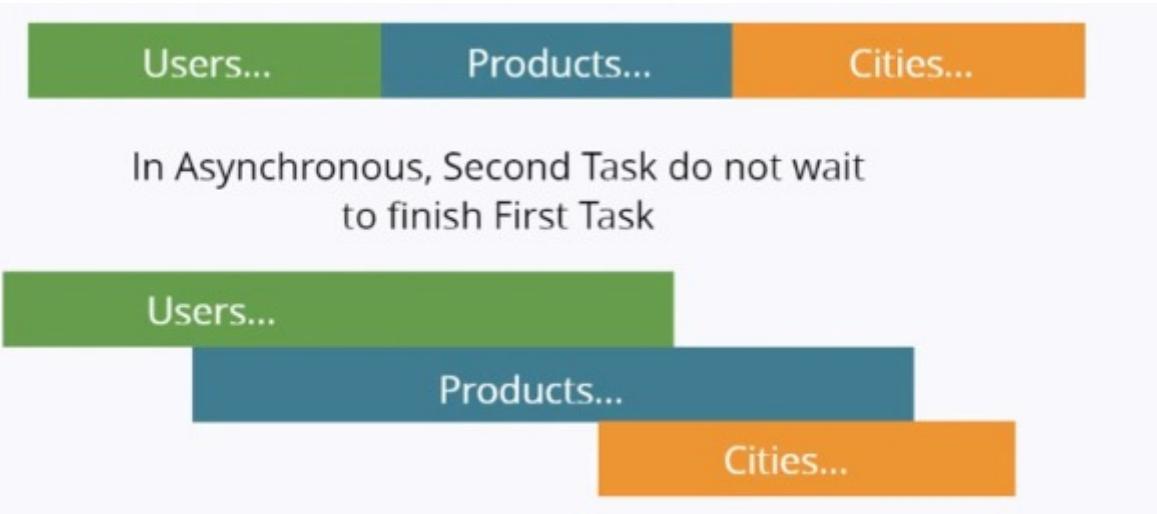
    // if (!err) {
    //     console.log("File is updated")
    // }
    // );
    // Now, we are re - naming the file's name:
    // fs.rename(filePath, `${dirPath}/fruit.txt`, (err) => {
        // if (!err) {
        //     console.log("Renamed file successfully")
    
```

```
// //} else { // Now, we are deleting a particular file: // We have changed the file's name from  
apple.txt to fruit.txt. This line is the new code of the  
filePath  
}  
const newPath = `${dirPath}/fruit.txt`;  
  
fs.unlinkSync(newPath)  
  
// This is how we perform all the basic operations on the directories.
```

- What is Buffer?

-> It is a temporary memory to perform an operations

Basic of Asynchronous



- Now, we are doing some program:

Code:

```
console.log("Start time...")  
// console.log("Logic time...")  
  
// Now, we are using to execute our program delay  
  
// Here, we are using a setTimeout();  
  
setTimeout(() => {  
  
  console.log("logic time...")  
  
}, 2000) //Here, 2000 is a mili second.  
  
console.log("Complete time...")  
  
//Here, we are using this for better implementations.
```

- Output:

```
[root@parrot]~[/home/keren/Code/Node-6.0]  
● └── #node "/home/keren/Code/Node-6.0/index.js"  
  Start time...  
  Complete time...  
  logic time...
```

Drawback's of Asynchronous:

- *// Now, for the Drawback, we are working on that thing*

```
let a = 10
```

```
let b = 0
```

```
setTimeout(() => {
```

```
    b = 20;
```

```
}, 2000)
```

```
console.log(a + b)
```

- Here, actually we want an output as a 30. But, we are getting an output as 10. To solve these we are using an exception handling. And, one more thing is that This is one of the drawback. Because, if we want a particular answer and that is being late then it gave the bad impact on the people. That's why to solve this we are using some exception handling cases.

Handle Asynchronous Data:

Code:

```
let a = 10

let b = 0

// Now, we are handling the exceptions to get an exact output that we want. So, we are using Promise
// to solve this..

// This Promise wait for a particular result and then it executes an output.

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

setTimeout(() => {

    b = 30;

    resolve(30); //Here, we can pass other data - types also like Array, Variables, and Strings
}, 2000)

})

waitingData.then((data) => {

    b = data;

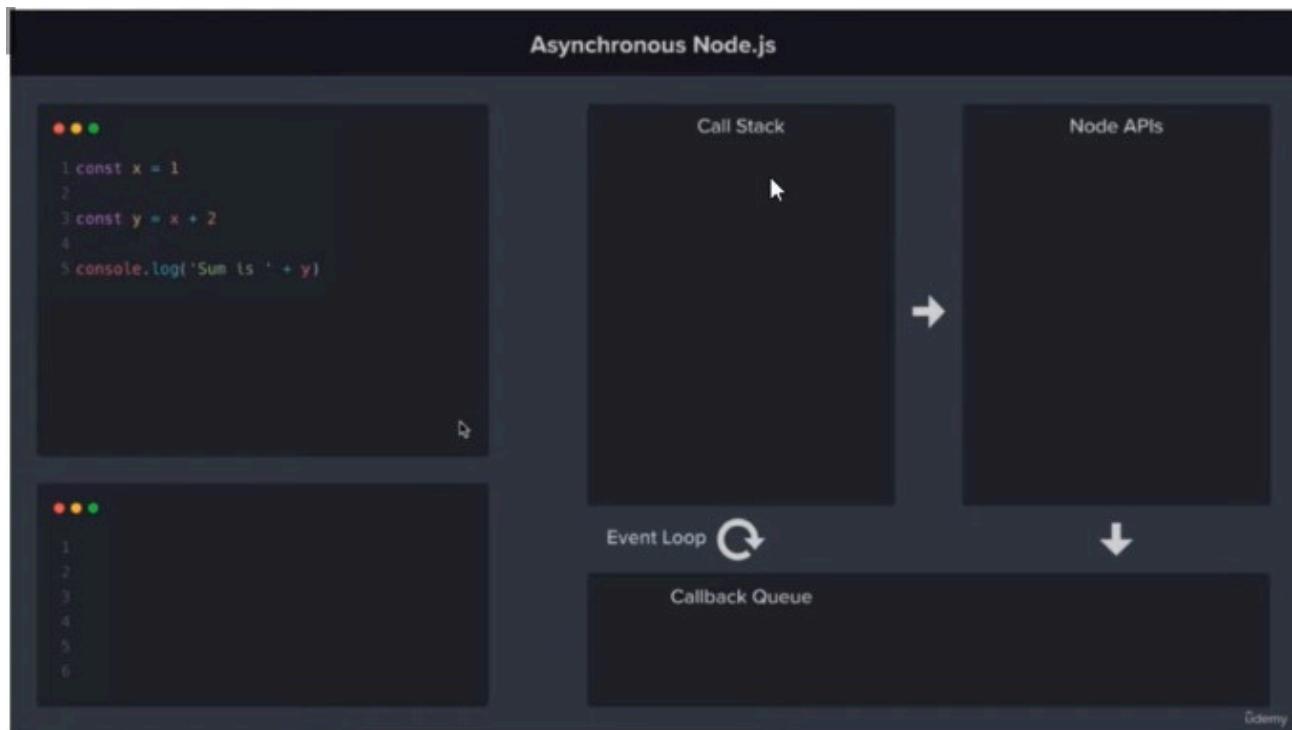
    console.log(a + b)
})
```

- Output:

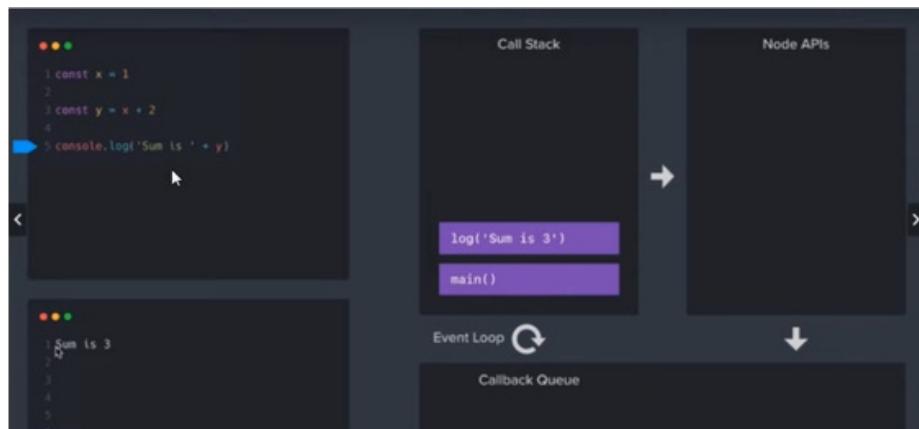
```
[root@parrot]~[/home/keren/Code]
● └─#node "/home/keren/Code/Node-6.0/index.js"
    40
```

How Node JS Works:

Architecture of the Node JS program's run:



- Call Stack
 - First it goes to the Call Stack
 - Call Stack is mandatory
 - - `main()` *function* is by default added



- Here, first is `main()` and then `log()`, after that it checks other required things and then executes it.

- It checks all the Syntax error and checks the functions and register it in the Series.

Now, here is one example to know that how Call Stack is working:

```
console.log("Starting app....")
setTimeout(() => {
  console.log("2 Second log....");
}, 2000)
setTimeout(() => {
  console.log("0 Seconds delay")
}, 0.19)
```

```
console.log("fininshing up....")
```

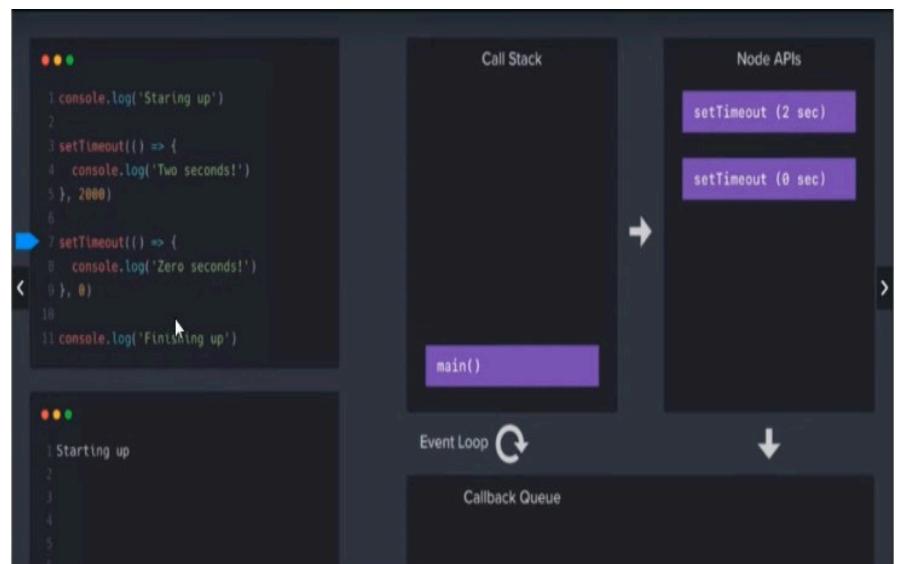
- Output:

```
[root@parrot]~[~/home/keren/Code]
└─#node "/home/keren/Code/Node-7.0/index.js"
  Starting app.....
  fininshing up.....
  0 Seconds delay
  2 Second log....
```

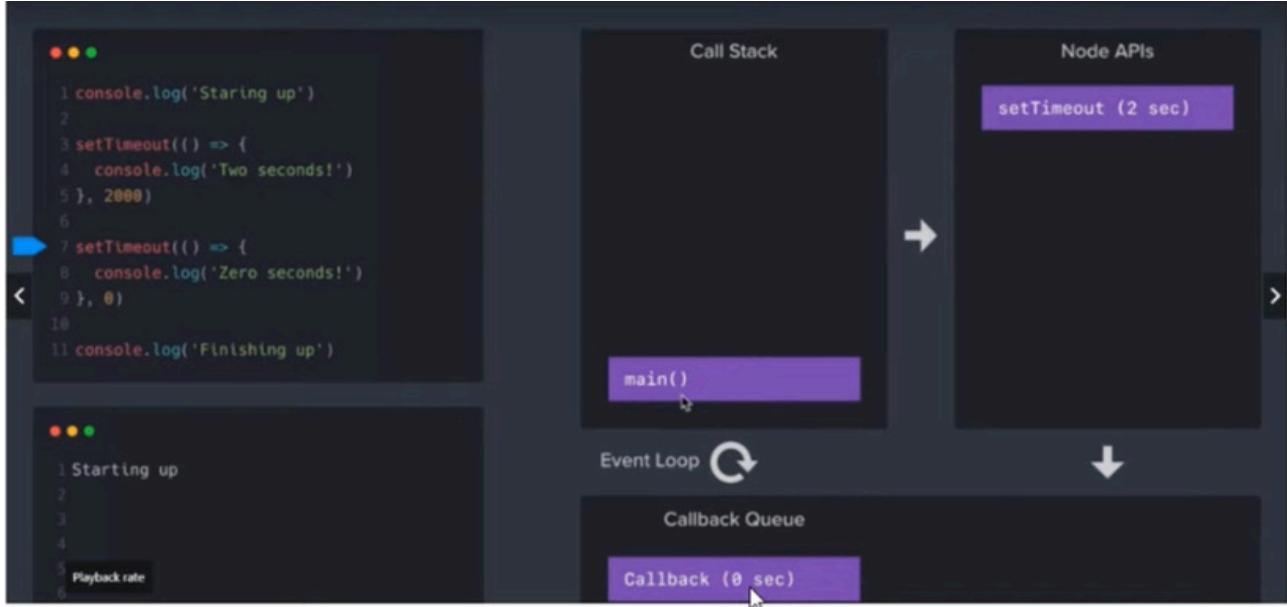
But, you can see that First output is Starting app, Second is finishing up, third is 0 seconds delay and last one is 2 Second log

- Now, after executing all the functions, this where the Node API's are getting stored and then after particular time it got to executed.

- `setTimeout()` is the function of the C++



- Now, the last one is the Event loop



- It compares the timeout and passes it to the Event Loop and then it gets to be executed.

Express JS

- It is a framework of the Node JS
- It is used in the 90% of the program

Install:

```
- yarn global add express
```

Now, create a new directory and initialize the yarn:

```
- yarn init
```

Now, create a new file:

```
- touch index.js
```

Now, here is the code:

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

const app = express();

app.get('/', (req, resp) => { //"" " This Means Home page and here we are adding a CallBack
function (req, resp)=>{}

resp.send('Hello, this is Home Page');

}

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

    resp.send('Hello, this is About Page');

}

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

    resp.send('Hello, this is Help Page');

}

app.listen(4500, () => {

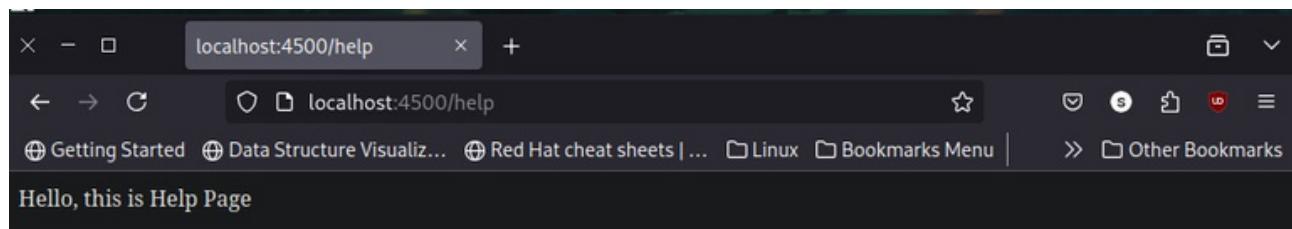
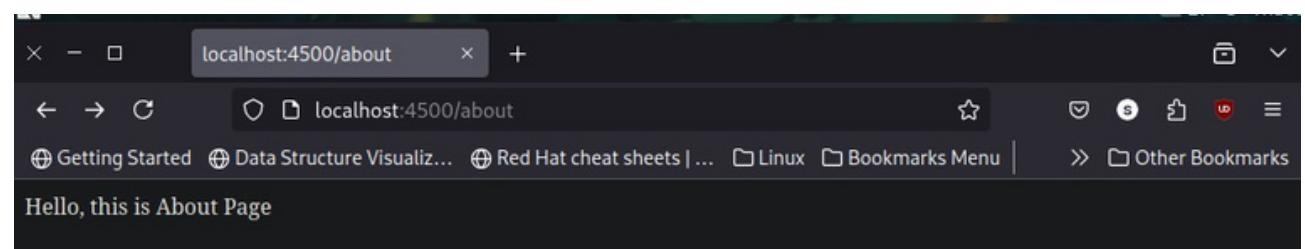
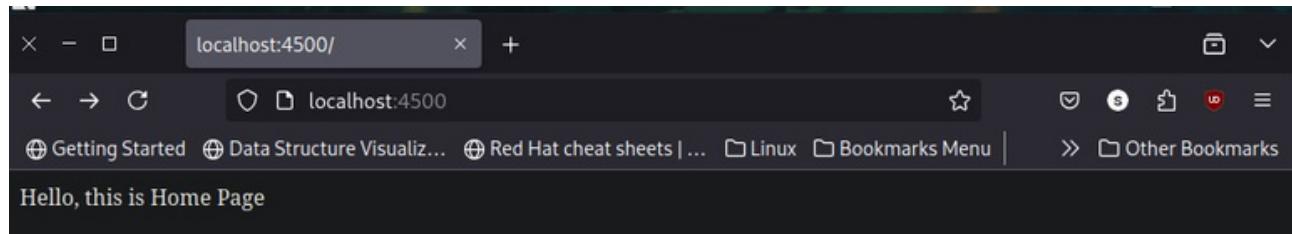
    console.log('Server is running on port 4500')

})
```

- Now, run this with the nodemon or node. But, for live changes try to run with nodemon

```
- nodemon index.js
```

Output:



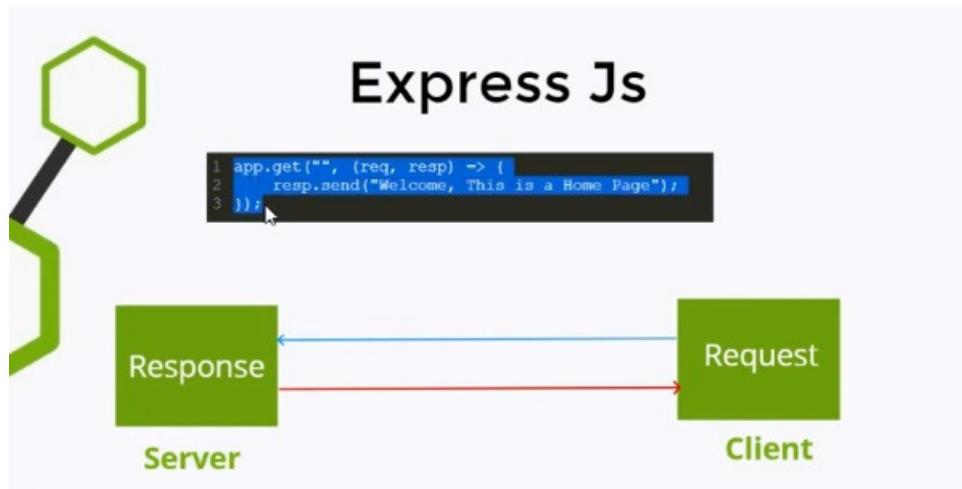
- Here, you can see that we have even changed the URL's route (Like, '/', '/about', '/help').

Routing Parameters – Request and Response:

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

```
resp.send('Hello, this is Home Page');
})
```

- Here, we are learning about the request and response's section.



- We can even put any name instead of the `req` and `resp` in that section.

Like:

```
app.get('/about', (a, b) => {
  b.send('Hello, this is About Page');
})
```

- To just for understanding neat clean code. We are using the `req`, and `resp`.
- Generally,
- Request is done from the `Client side` and Response is done from the `Server side`.
- `Req` is generally used to send the data from the client side to the server side.

Now, we are taking the input from the Browser's URL's section:

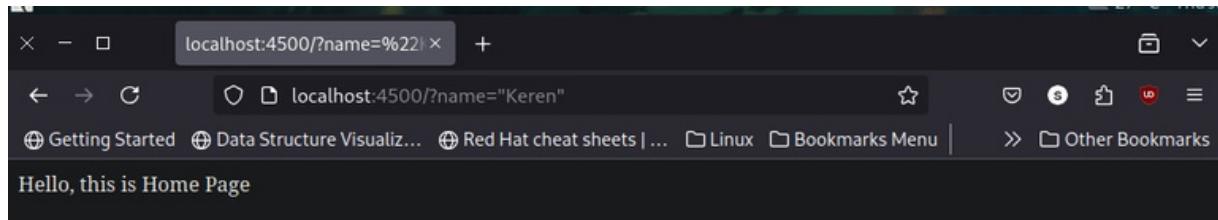
Code:

```
app.get('/?', (req, resp) => {
  console.log("The data from the user is the: ", req.query.name);
  resp.send('Hello, this is Home Page');
})
```

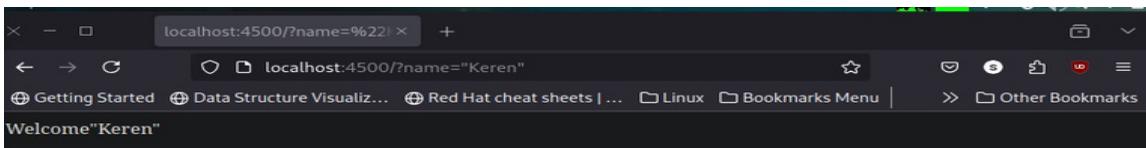
- Here, is the output in the console of the terminals:

```
^C^X [x] -[root@parrot] -[ /home/keren/Code/Express ]
└─ #nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server is running on port 4500
The data from the user is the: "Keren"
□
```

- Here, you can see that, the data from the URL's which is Keren (This name was added manually).



Now, we are making some changes by putting the Static data:



Code:

```
app.get('/', (req, resp) => {
  console.log("The data from the user is the: ", req.query.name);
  resp.send('Welcome' + req.query.name);
})
```

Render HTML and JSON:

Code:

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

```
const app = express();

app.get('/', (req, resp) => {
    console.log("The data from the user is the: ", req.query.name);
    resp.send(` <h1>Welcome</h1> ${req.query.name} <a href="/about"> About </a> <br> <a href="/help"> About </a> `)
})

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

    // Now, we are routing the pages from one page to another:

    resp.send(` <input type="text" placeholder="Username"/> <button> Click me </button> <a href="/">Click me </a> `);
}

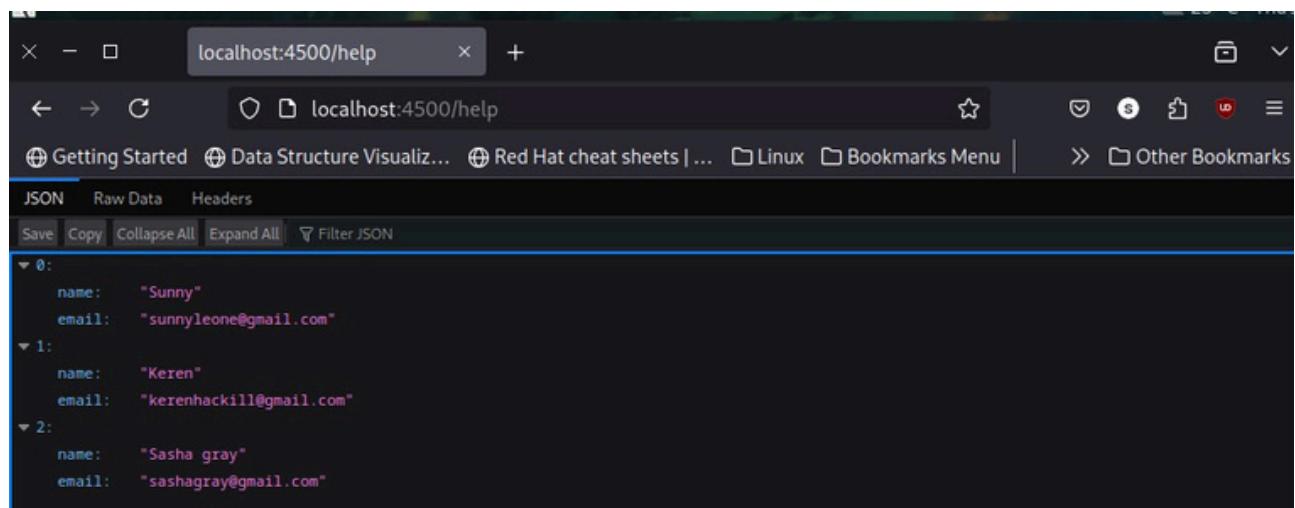
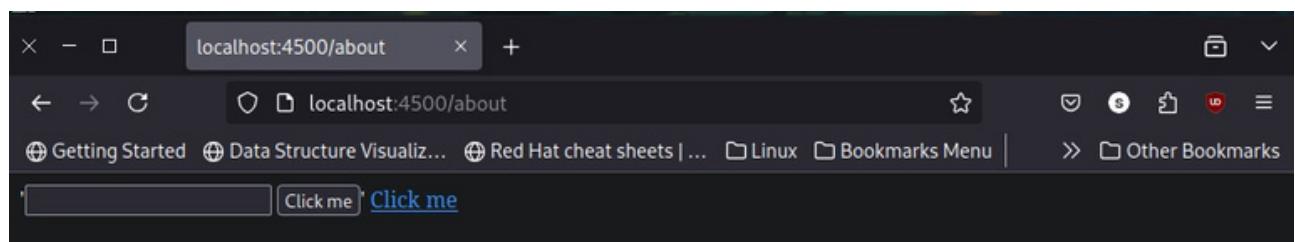
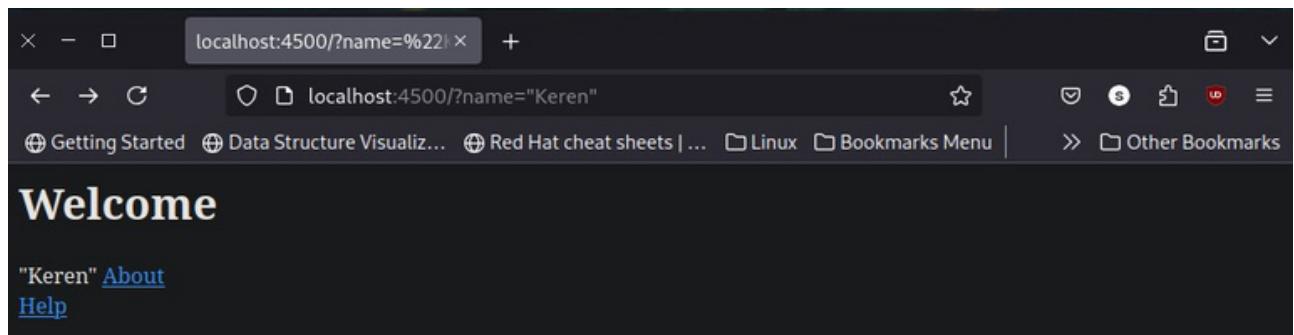
app.get('/help', (req, resp) => {

    // resp.send('Hello, this is Help Page');

    // Here, we are rendering the JSON data:

    resp.send( // Now, for multiple user:
        [
            { name: "Sunny", email: "sunny.leone@gmail.com" },
            { name: "Keren", email: "keren.hackill@gmail.com" },
            { name: "Sasha gray", email: "sasha.gray@gmail.com" },
        ],
    )
})

app.listen(4500, () => {
    console.log('Server is running on port 4500')
})
```



Make HTML Pages:

- Here, we are accessing the files from the directory.

Code:

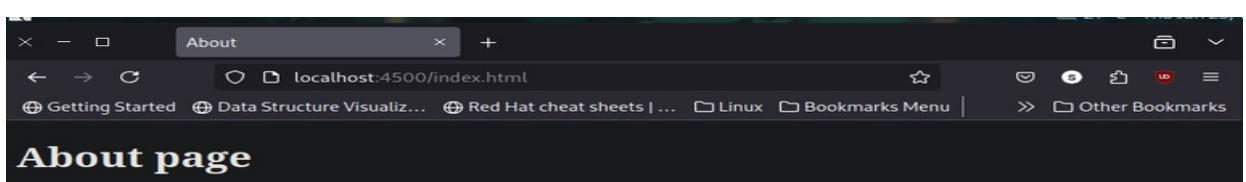
```
const express = require('express') //the const "express" is known as a different function
const path = require('path')
const app = express() //This one is known as the executed function which is a "express()"
// console.log(__dirname); //First we need the path
// Here, we are adding the two path
const publicPath = path.join(__dirname, 'Compone')
// Here, we are using "use", and "use" is a middleware
app.use(express.static(publicPath)); //static() method loads the static pages
app.listen(4500)
```

HTML Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>About</title>
</head>
<body>
<h1> About page</h1>
</body>
</html>
```

Output:



Remove extension from URL:

- In the above pages there is a one picture, where there is a "html" in the URL. And, now we are removing that.

- This is for the security purpose.

- And we are even making the '404' error page.

Code:

```
const express = require('express') //the const "express" is known as a different function
```

```
const path = require('path')
```

```
const app = express() //This one is known as the executed funcntion which is a "express()"
```

```
// Here, we are adding the two path
```

```
const publicPath = path.join(__dirname, 'Compone')
```

```
// Here, we are using "use", and "use" is a middleware
```

```
app.get('/', (_, resp) => {
  resp.sendFile(` ${publicPath}/index.html`);
})
```

```
app.get('/about', (_, resp) => {
```

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

```
app.get('/help', (_, resp) => {
  resp.sendFile(` ${publicPath}/help.html`)
})
```

```
// If the particular page's URL is removed or not available:
```

```
app.get('*', (_, resp) => {
  resp.sendFile(` ${publicPath}/404.html`)
})
app.listen(4500)
```

- HTML's code:

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About</title>
</head>

<body>
    <h1> About page</h1>
</body>
</html>
```

about.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About</title>
</head>
<body>
    <h1>This a page of the About</h1>
</body>
</html>
```

help.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Help</title>
```

```
</head>
<body>
  <h1>This is a help page</h1>
</body>

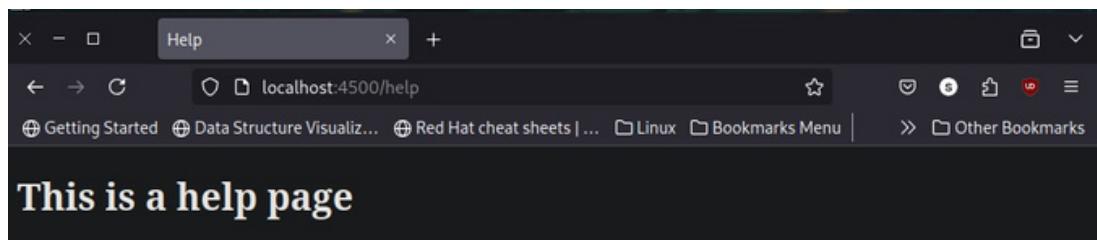
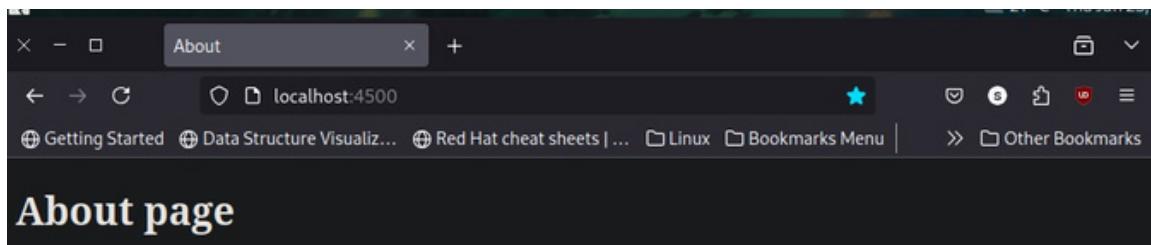
</html>
```

404.html:

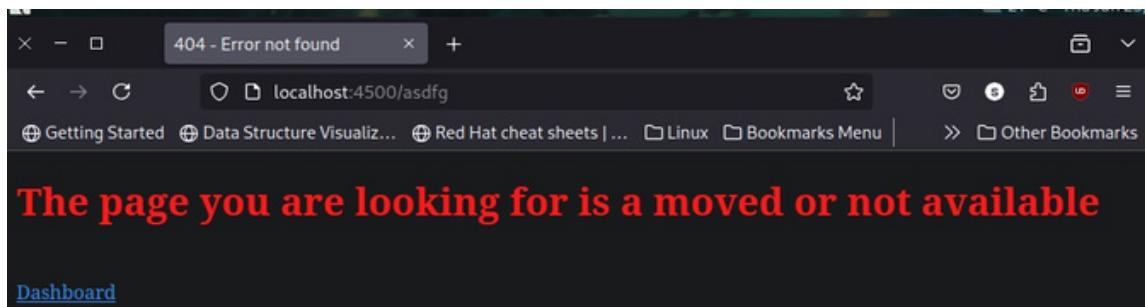
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>404 - Error not found</title>
  <style>
    h1 {
      color: red;
    }
  </style>
</head>
<body>
  <h1>The page you are looking for is a moved or not available</h1> <br>
  <a href="/">Dashboard</a>
</body>

</html>
```

Output:



Now, for an Error page:



- Question:
- How to remove the extension from the URL?
- > By using the “get” method

- How to handle the different URL's?
- > By creating a separate file and routing them.

Template Engine:

- Here, we are adding the “ejs” – Embedded JavaScript Template

```
- yarn add ejs || yarn global add ejs
```

- We have to create a directory named as the “views”, this is because ejs by default uses the views.

- And even we are using the - .ejs as an extension for a file.

Code:

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

```
const app = express()
```

```
// Now, we are adding a EJS
```

```
app.set('view engine', 'ejs');
```

```
app.get('/', (_, resp) => {
```

```
    const user = {
```

```
        name: "Murlidhar",
```

```
        email: "murlidharkanha@gmail.com",
```

```
        city: "Vaikunth",
```

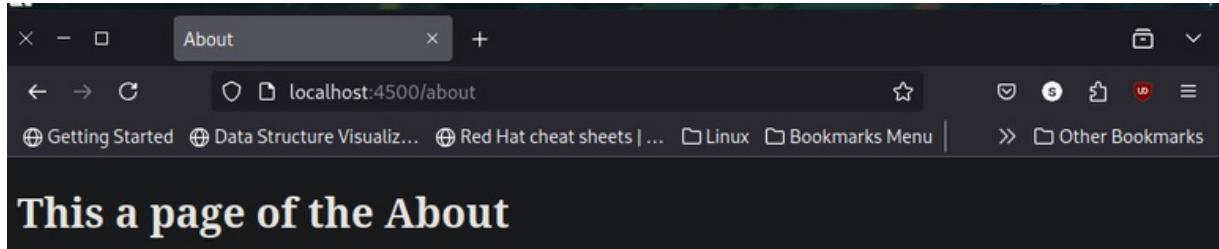
```
}
```

```
    resp.render('index', { user })
```

```
)
```

```
app.listen(4500)
```

```
)
```

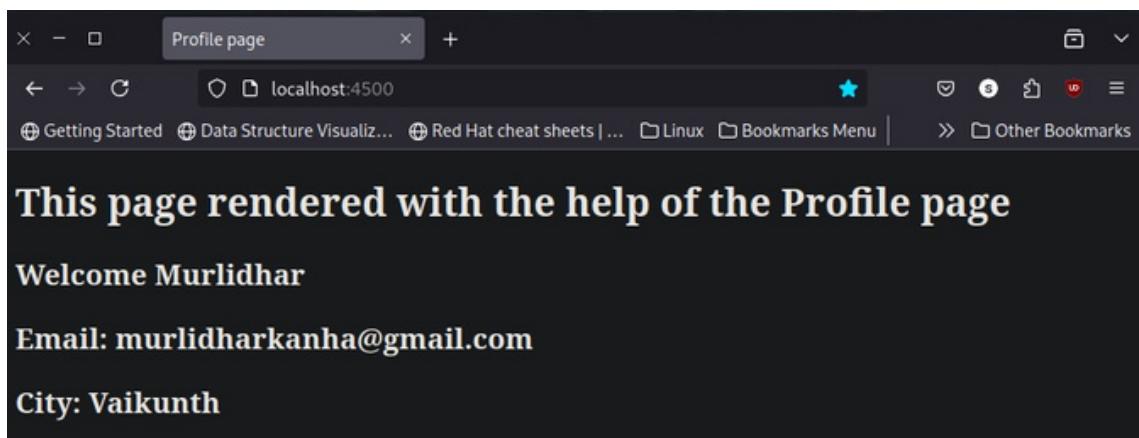


- **index.ejs:**

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Profile page</title>
  </head>
  <body>
    <h1>This page rendered with the help of the Profile page</h1>
    <!-- Here, we are taking the dynamic data -->
    <h2>Welcome <%= user.name %></h2>
    <h2>Email: <%= user.email %></h2>
    <h2>City: <%= user.city%></h2>
  </body>
</html>
```

- Now, the output:



Dynamic Page with ejs:

- Here, we are taking the things in the loops:

Code:

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

```
const app = express()
```

```
// Now, we are adding a EJS
```

```
app.set('view engine', 'ejs');
```

```
app.get('/', (_, resp) => {
```

```
    const user = {
```

```
        name: "Murlidhar",
```

```
        email: "murlidharkanha@gmail.com",
```

```
        city: "Vaikunth",
```

```
        skills: ['Python Django', 'CS', 'C++', 'Java'],
```

```
// Now, we are iterating the things on our website. In the file of the index.ejs
```

```
}
```

```
    resp.render('index', { user })
```

```
)
```

```
app.listen(4500)
```

index.ejs Code:

```
<body>
```

```
    <h1>This page rendered with the help of the Profile page</h1>
```

```
    <!-- Here, we are taking the dynamic data -->
```

```
    <h2>Welcome <%= user.name %>
```

```
    </h2>
```

```
    <h2>Email: <%= user.email %>
```

```
    </h2>
```

```

<h2>City: <%= user.city%>

</h2>

<ul>

<% user.skills.forEach((item)=>{ %> <!--Here, <% %> is used for just definit the value-->

<li><%= item %></li> <!--Here, <%= %> is used to print out the value-->

<% }) %>

</ul>

</body>

```

- Here, you can see that we have used the **forEach** loop:

```

<% variableName.forEach((item)=>{ %>

<%= item %>

<% }) %>

```

- This above one is a syntax for an iterating **forEach** loop:

Now, we are creating a one more directory inside the views with the name of the “common”,

- “common” name because it is used everywhere of the code. – navbar, footer, header and other things

index.js's Code:

```

const express = require('express')

const app = express()

app.set('view engine', 'ejs');

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

  const user = {

    name: "Murlidhar",

    email: "murlidharkanha@gmail.com",

    city: "Vaikunth",

    skills: ['Python Django', 'CS', 'C++', 'Java'],
  }

```

```

    }

    res.p.render('index', { user })

}) app.get('/login', (_, resp) =>

{
  resp.render('login')
}

app.listen(4500)

```

- Now, the index.js Code:

```

<body>

<%- include('common/header')%>

<h1>This page rendered with the help of the Profile page</h1>

<!-- Here, we are taking the dynamic data -->

<h2>Welcome <%= user.name %>

</h2>

<h2>Email: <%= user.email %>

</h2>

<h2>City: <%= user.city%>

</h2>

<ul>

<% user.skills.forEach((item)=>{ %> <!--Here, <% %> is used for just definint the value-->

<li><%= item %></li> <!--Here, <%=%> is used to print out the value-->

<% }) %>

</ul>

</body>

```

- Here, we have included the file from another destination and it is going to be pasted here:

```
<%- include('dir_path'/ 'file_name') %>
```

- And login.ejs file's code:

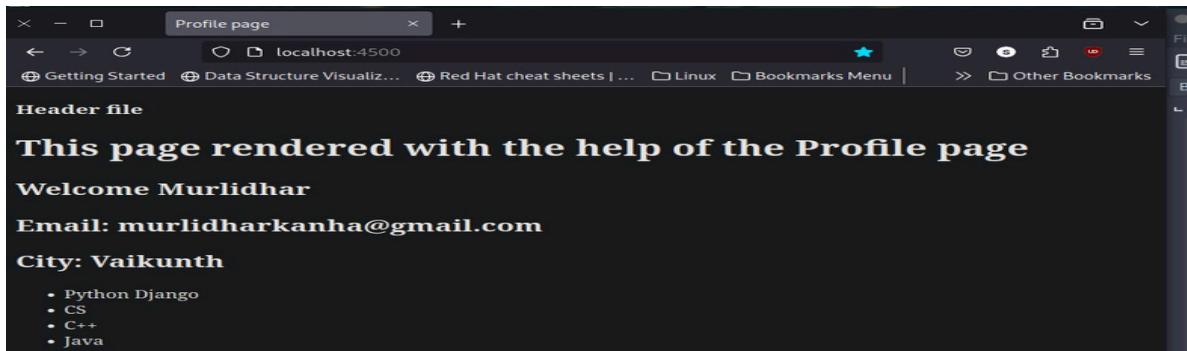
```
<body>  
<%- include('common/header')%> <!--Here, we are using <%- %> for importing the files -->  
    <h1>This is a login page</h1>  
</body>
```

- And the code from the file named as the header from other directory:

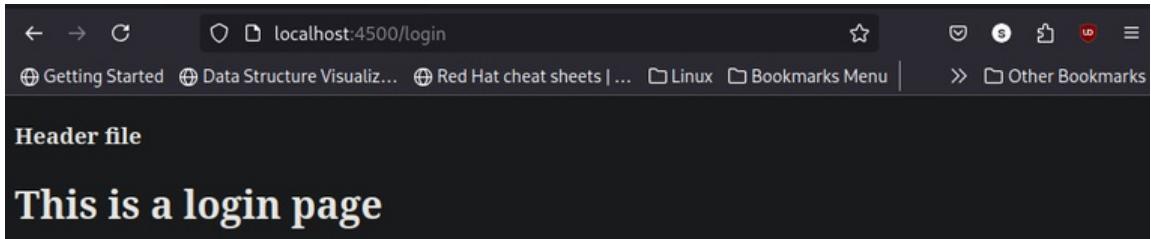
header.ejs:

```
<nav>  
    <h3>Header file</h3>  
</nav>
```

- Output:



- And,



- This is how we are importing one code to another in the format of other files.

Middleware:

- These are some function which are widely used with routes, and it access the request and response and modify it.

- Used for the user – authentication, is user TRUE or FALSE
- Blocking website for the specific country

Code:

```

const express = require('express')

const app = express()

app.set('view engine', 'ejs')

// Now, we are adding a middleware here:

// Here, we are getting 3 input fields

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

  // Now, we are applying some conditions:

  if (!req.query.age) {

    resp.send('Please provide age');

  }else{

    next();

  }

}

a p p . u s e ( r
e q F i l t e r )
a p p o s t g e t ( ' 
resp.send('index')
() ' ,       ( r e q ,
r e s t )        = >
app.listen(4500)
}

```

{ Here, if we are getting an error like this –

```
[root@parrot]~[/home/keren/Code/Express/E-0.4]
└─# nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
    at ServerResponse.setHeader (node:_http_outgoing:699:11)
    at ServerResponse.header (/home/keren/Code/Express/E-0.4/node_modules/express/lib/response.js:794:10)
    at ServerResponse.send (/home/keren/Code/Express/E-0.4/node_modules/express/lib/response.js:174:12)
    at /home/keren/Code/Express/E-0.4/index.js:19:10
    at Layer.handle [as handle_request] (/home/keren/Code/Express/E-0.4/node_modules/express/lib/router/layer.js:95:5)
    at next (/home/keren/Code/Express/E-0.4/node_modules/express/lib/router/route.js:149:13)
    at Route.dispatch (/home/keren/Code/Express/E-0.4/node_modules/express/lib/router/route.js:119:3)
    at Layer.handle [as handle_request] (/home/keren/Code/Express/E-0.4/node_modules/express/lib/router/layer.js:95:5)
    at /home/keren/Code/Express/E-0.4/node_modules/express/lib/router/index.js:284:15
    at Function.process_params (/home/keren/Code/Express/E-0.4/node_modules/express/lib/router/index.js:346:1
2)
```

- So, to solve this by using the if – else condition:

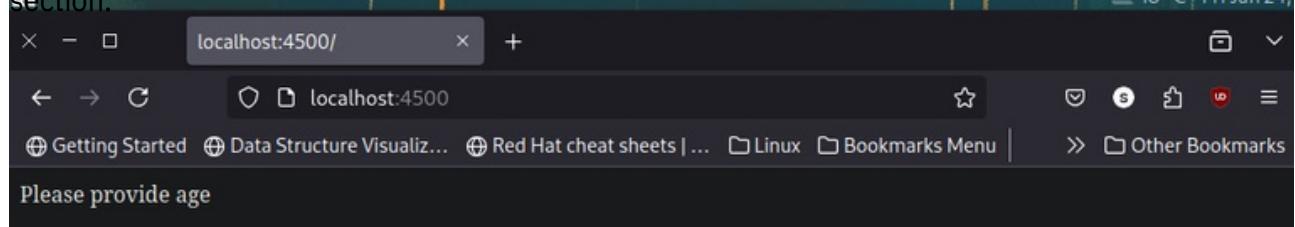
Code:

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

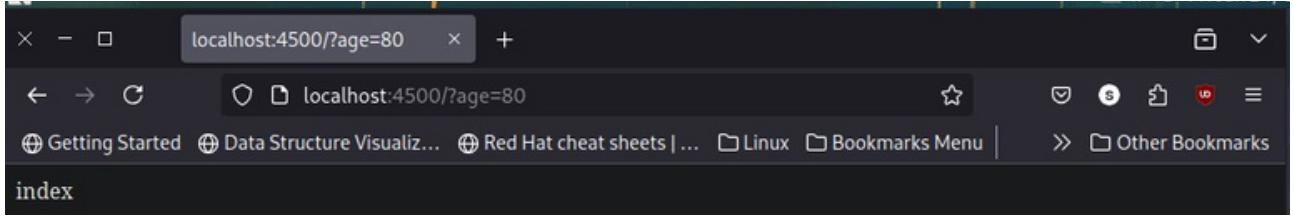
// Now, we are applying some conditions:

```
if (!req.query.age) {
  resp.send('Please provide age');
} else {
  next();
}
```

} - Here, we are even using the URL base condition to apply this on the condition in the URL's section.



- Now, after giving some particular values in the URL to



- Here, you can see that our after giving the input, it opens up the next page.

Route Level Middleware:

- Now, we are applying the Middleware on multiple routes

Single Middleware's Code:

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

const app = express()

// This is our MiddleWare Condition

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

    // Now, we are applying some conditions:

    if (!req.query.age) {

        resp.send('Please provide age');

    } else {

        next();

    }

}    // Now, we are applying the middleware at the multiple points
// Now, we are applying that condition on separate one's -> This is also known as the single router.

app.get('/', reqFilter, (req, resp) => {
    resp.send("Welcome");
})

app.get('/home', reqFilter, (req, resp) => {
    resp.send('Welcome to home')
})

app.listen(4500)
```

- Now, we can create a separate directory for this one:

middleware.js:

```
// This is our MiddleWare Condition
module.exports = reqFilter = (req, resp, next) => {
    // Now, we are applying some conditions:
    if (!req.query.age) {
        resp.send('Please provide age');
    } else {
        next();
    }
}
```

- index.js:

```
const express = require('express') const app = express() // Now, we are importing the
middleware.js file: const reqFilter = require('./middleware') // Now, we are applying the
middleware at the multiple points // Now, we are applying that condition on separte one's -> This is
also known as the signle router. // We can even add this on other one also. app.get('/', reqFilter,
(req, resp) => { resp.send("Welcome"); }) app.get('/home', reqFilter, (req, resp) => {
resp.send('Welcome to home') }) app.listen(4500)
```

- The output is same as the others. Now, we want to add the route for multiple routes without adding the **reqFilter** inside.

Here, we have applied the route for particular route:

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

const app = express()

// Now, we are importing the middleware.js file:

const reqFilter = require('./middleware')

// Now, we are creating an instance for the route:

const route = express.Router()

route.use(reqFilter)

app.get('/', (req, resp) => {
  resp.send("Welcome");
})

app.get('/home', (req, resp) => {
  resp.send('Welcome to home')
})

route.get('/contact', reqFilter, (req, resp) => {
  resp.send("Welcome tp contact");
})

// If we want to apply the middle-ware then we are using the - "route" instead of the app.

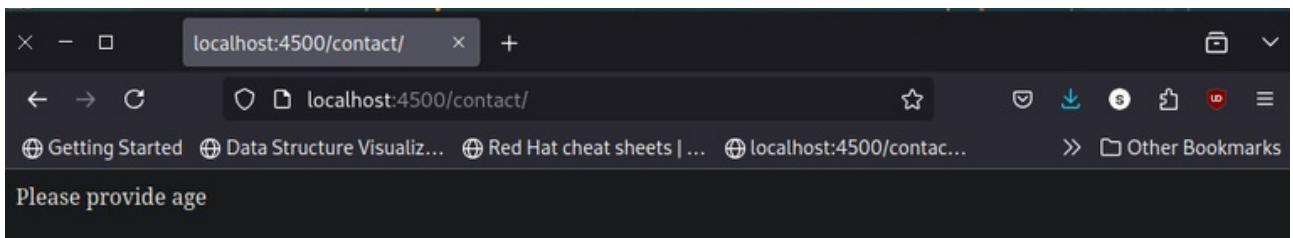
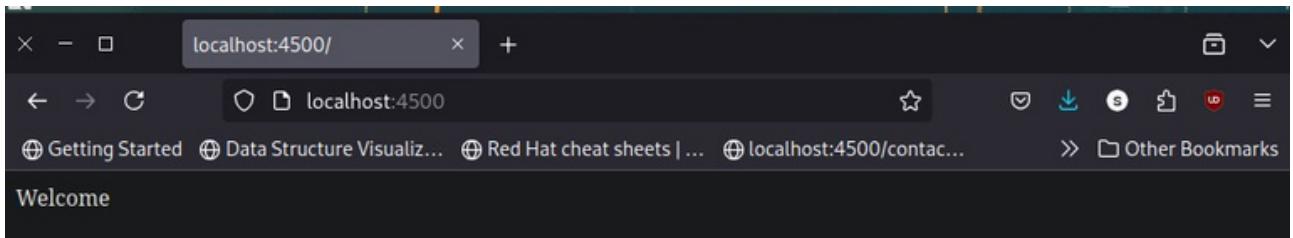
route.get('/contact', reqFilter, (req, resp) => {
  resp.send("Welcome tp contact");
})

// after adding the "route" instead of "app", we have to add the given belowo line:

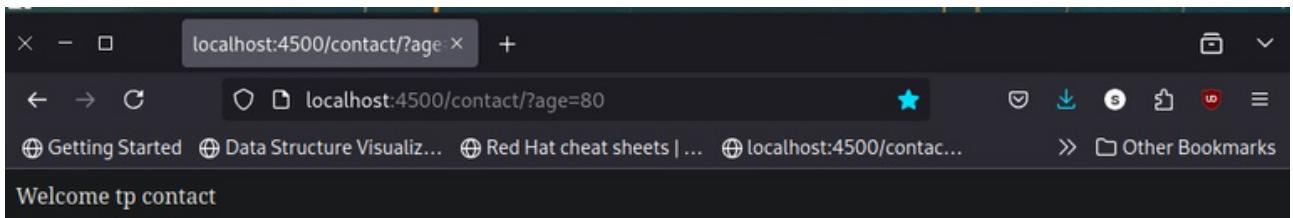
app.use('/', route);

app.listen(4500)
```

- Output:



- After providing the age:



- Question difference between Global route and Local route, and how to apply the middleware.

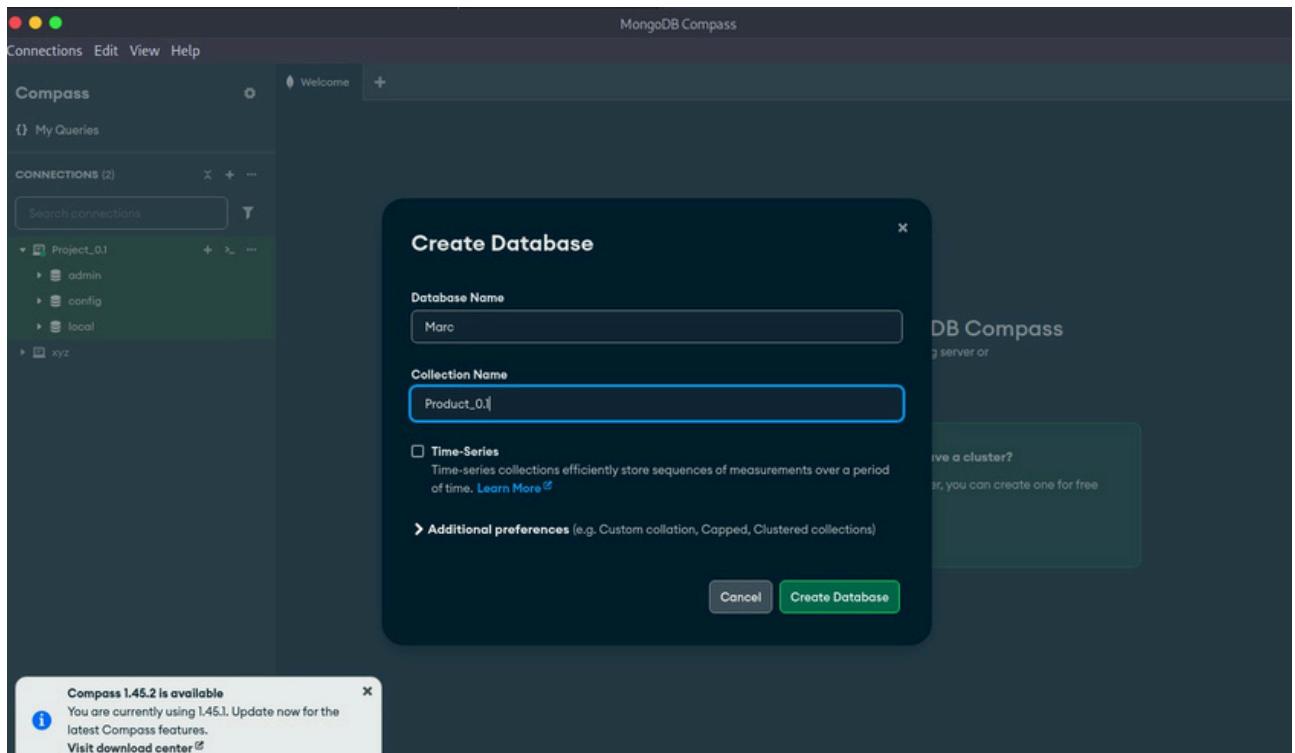
- -> Global Middleware route:
 - Applies to all the routes
 - Executed for every incoming request
- -> Local Middleware route:
 - Only executed for specific route handlers
 - More targeted application

MongoDB:

- Download MongoDB
 - [Click here](#), to check more about the MongoDB.
- Now,
- By reading the documents regarding about the MongoDB.

CRUD Operations in MongoDB:

- Before that create a new Database:



- Now, create a new collection as per your choices:

- And, **remember** that on which “collection” you are working just click and connect that collection after that you are able to perform some actions inside that.

- Questions:

- Difference between MongoDB vs MySQL?
- Why we use the MongoDB?

Connect Node with MongoDB:

- Install the yarn package of the MongoDB's:

- [Click here](#) to download the MongoDB:

```
yarn global add mongodb | yarn add mongodb #(For locally)
```

Make sure of installing the:

```
yarn add nodemon
```

Code:

```
const { MongoClient } = require('mongodb')

// Now, we are taking the path of the Path.

const url = "mongodb://localhost:27017"

// Now, we are passing the url to the MongoClient

const client = new MongoClient(url)

const database = "Marc"

// Function for GET data:

// function getData() {

// let result = client.connect();

// }

// The above one is a normal function. Even we have to handle an error.

// That's why we are using an error handle with an "async" keyword

async function getData() {

  try {

    let result = await client.connect();

    // Now, we are giving exact path of the Database in the MongoDB's. So, that it connects with it.

    db = result.db(database)

    // Now, we have connected the database. Now, we are connecting the "collections"

    let collection = db.collection('Products') // "Marc" is name of the collection in the database of the "Project_0.1"

    let response = await collection.find({}).toArray()

    console.log(response)
```

```

} catch (error) {
    console.error("An error occurred", error)
}

} finally {
    await client.close()
}
}

get Data ( ) .

```

c a t c h (c o n s

o l e e r r o r)

Output

```

^C-[X]-[root@parrot]-[/home/kezen/Code3/MongoDB/Mongo_0.1]
└── #nodemon
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node index.js'
[
  [
    {
      _id: new ObjectId('67a7a5710314ccdc1ae43269'),
      name: 'Madara Uchiha',
      age: 28,
      place_of_birth: 'Hidden Leaf'
    }
  ]
[nodemon] clean exit - waiting for changes before restart

```

- Can we connect the two databases at a time?

-> Yes, you can connect the two databases.

Read Data from MongoDB:

- Now, we are creating a separate file for the connection between the Node and MongoDB.
- By making this we can handle an error smoothly.

Code:

```

const { MongoClient } = require('mongodb')
const url = "mongodb://localhost:27017"
const client = new MongoClient(url)
const database = "Marc"
async function dbConnection() {
  try {

```

```

let result = await client.connect();
db = result.db(database)

return db.collection('Products')

} catch (error) {
  console.error("An error occurred", error)
} finally {
  await client.close()
}
}

dbConnection().catch(console.error)

console.warn(dbConnection()) //This line is for know which types of an error we should have to handle here.

```

Output:

```

^C-[x]-[root@parrot]-[/home/keren/Code3/MongoDB/MongoDB/Mongo_0.2]
└─#nodemon

[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Promise { <pending> }
[nodemon] clean exit - waiting for changes before restart

```

- Here, you can see the "Promise <pending>". Now, we are handling this promise

- What is Promise?

-> Whenever we try to do the heavy operations with the data, then to handle an **async** and **await**.

- Here, first we are handling the first promise which is generated for the handling the

```
return db.collection('products');
```

Code:

```

const { MongoClient } = require('mongodb')
const url = "mongodb://localhost:27017"
const client = new MongoClient(url)

```

```

const database = "Marc"

async function dbConnection() {

    let result = await client.connect();

    db = result.db(database)

    return db.collection('Products')

} const main = async () => {

// console.log("Main")

}

let data = await dbConnection()

data = await data.find({}).toArray()

console.warn(data)

} main(); - Here, you can see that we are creating an async function. - Now, we are creating another file with named as the – mongodb.js and paste it all the connection code.

```

Mongodb.js:

Code:

```

const { MongoClient } = require('mongodb')

const url = "mongodb://localhost:27017"

const client = new MongoClient(url)

const database = "Marc"

async function dbConnection() {

    let result = await client.connect();

    db = result.db(database)

    return db.collection('Products')
}

```

```

}

// module.exports = dbConnection() //Don't call here the function, just directly pass it.

module.exports = dbConnection

```

- Now, we are importing this file another one.

Code:

```

const dbConnection = require('./mongodb')

const main = async () => {

// console.log("Main")

let data = await dbConnection()

data = await data.find({}).toArray()

console.warn(data)

} main();

```

Output:

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
[nodemon] restarting due to changes...
[nodemon] starting `node index.js'
[
  {
    _id: new ObjectId('67a7a5710314ccdc1ae43269'),
    name: 'Madara Uchiha',
    age: 28,
    place_of_birth: 'Hidden Leaf'
  },
  {
    _id: new ObjectId('67a8468e3619a84e01e43269'),
    name: 'Keshav',
    age: 20,
    work: 'Maintainer'
  }
]

```

- In the upcoming session, we are keeping the same file which is of the – [mongodb.js](#)'s **insert.js**:

```

const dbConnection = require('./mongodb')

const insert = async () => {

  const db = await dbConnection()

```

```

// Now, this one is for the single.

// const result = await db.insertOne({ name: "Vallabh", age: "21", work: "Swami", place:
"Vaikunth" })

//Now, for the multiple data inserting by using an Array

const result = await db.insertMany([{ name: "Vallabh", age: "21", work: "Swami", place:
"Vaikunth" },
{ name: "Kanha", age: 4, work: "Fun", place: "Vrindavan" },
])

// Whenever I tried to change somethings then it will automatically saves the changes.

if (result.acknowledged) {
    console.log("Data inserted")
}
}

insert()

```

- By doing this we can solve most of the inserting problems

Update Data in MongoDB:

- Create a new file with named as the: - **update.js**

Code:

```

const dbConnection = require('./mongodb')

const update = async () => {
    let data = await dbConnection();

    let result = await data.updateMany({ name:"Vallabh" }, { $set: { name: "Nityananda" } })

    console.warn(result)
}

update()

```

- Question:
- Is update and updateOne are same?

-> No, they are different, `updateOne` is used for updating only single value, and `update` is used for updating the multiple values. It means that `update` and `updateMany` are the same

Delete Data in MongoDB:

- Create a new file with named as the `delete.js`:

Code:

```
const dbConnection = require('./mongodb')

const deleteData = async () => {
    // console.log("function coal")
    let data = await dbConnection()
    // console.warn(data)

    let result = await data.deleteMany({ name: "Nityananda" })

    console.warn(result)

    if(result.acknowledged){
        console.log("Done")
    }
}

d e l e
t e D a
t a ( )
```

```
[nodemon] restarting due to changes...
[nodemon] starting `node delete.js`
{ acknowledged: true, deletedCount: 0 }
Done
[nodemon] restarting due to changes...
[nodemon] starting `node delete.js`
{ acknowledged: true, deletedCount: 0 }
Done
[nodemon] restarting due to changes...
[nodemon] starting `node delete.js`
{ acknowledged: true, deletedCount: 0 }
Done
[nodemon] restarting due to changes...
[nodemon] starting `node delete.js`
{ acknowledged: true, deletedCount: 18 }
Done
□
```

Basic GET API with MongoDB:

- GET – Fetch data from the Database POST
- – Upload data in the Database PUT –
- Update data from the Database DELETE –
- Delete data from the Database

- In this we are using our previous code of the MongoDB for configuring the code space:

mongodbConnection.js:

```
const { MongoClient } = require('mongodb')
const url = "mongodb://localhost:27017"

const client = new MongoClient(url)
const database = "Marc"

async function dbConnection() {
  let result = await client.connect();
```

```
db = result.db(database)

return db.collection('Products')

}

// module.exports = dbConnection() //Don't call here the functino, just directly pass it.
module.exports = dbConnection
```

- Install the given below dependencies:

- - express = yarn add express
- - mongodb = yarn add mongodb
- - nodemon = yarn add nodemon

Now, the code for GET means fetching the data from the Database:

Code:

```
const exp = require('express')
const dbConnection = require('./mongodb')

const app = exp()

// 'get' means we are making a route for the GET API
// Here, we are combining an Express and MongoDB.

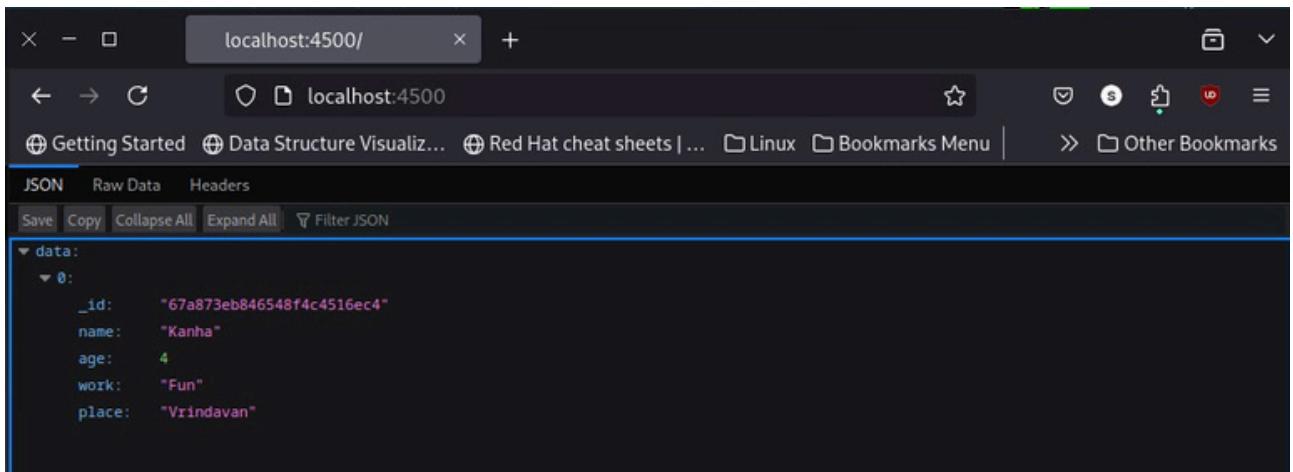
app.get('/', async (req, resp) => {
  let data = await dbConnection()
  data = await data.find().toArray()
  console.log(data)
  // You'll get a data in the terminal. Now, you can put that data in the Browser
  resp.send({ data })
}).listen(4500, (err) => {
  console.error("Error found", err)
})
```

- Now, run with the – nodemon

Output:

```
[root@parrot]~[/home/keren/Code3/MongoDB/MongoDB/Mongo_0.3]
└─# nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Error found undefined
[
  {
    _id: new ObjectId('67a873eb846548f4c4516ec4'),
    name: 'Kanha',
    age: 4,
    work: 'Fun',
    place: 'Vrindavan'
  }
]
```

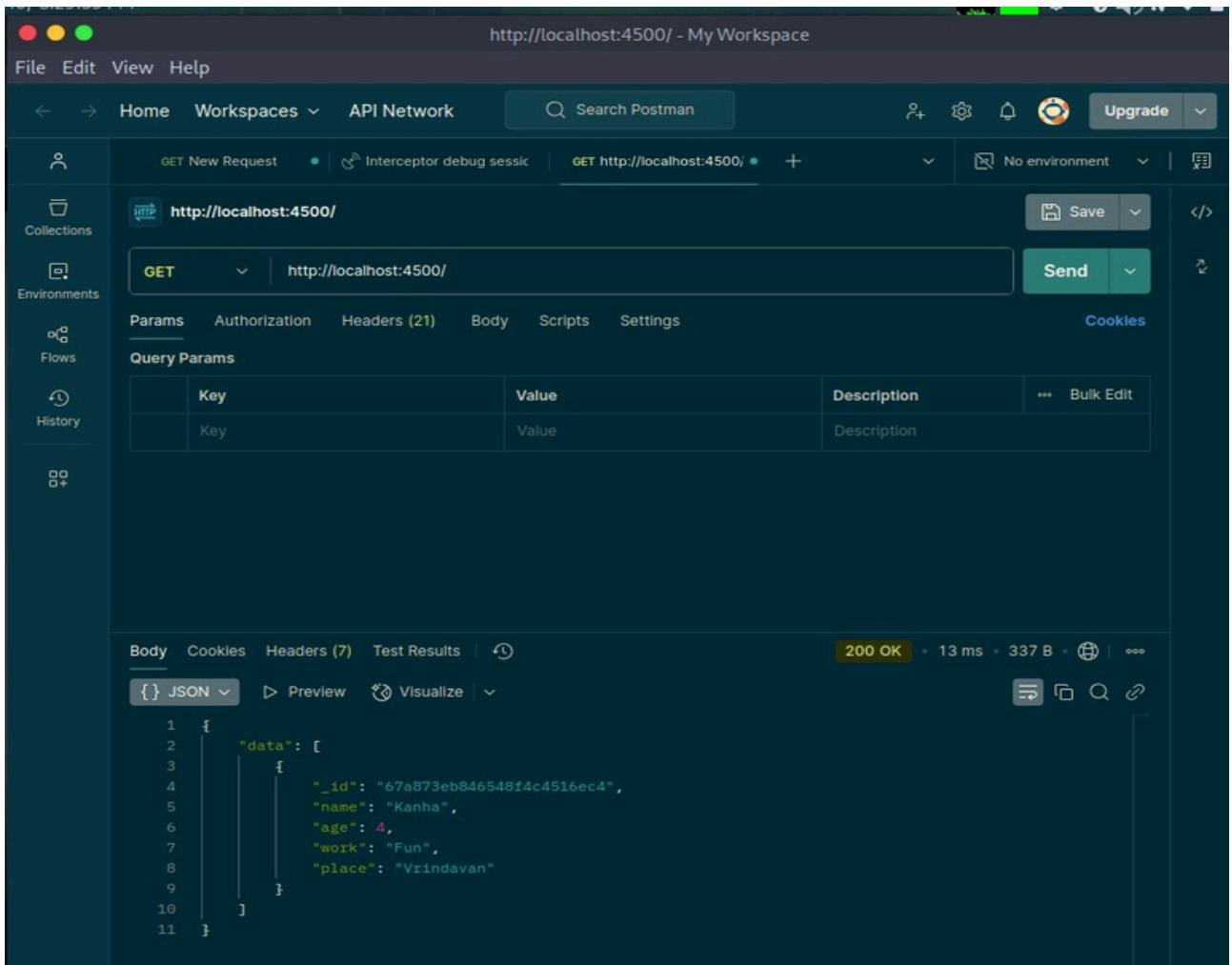
- In the Browser's page:



A screenshot of a web browser window titled "localhost:4500/" showing a JSON response. The JSON object has a single key "data" which contains an array of one element. This element is an object with properties: _id, name, age, work, and place. The values are: _id: "67a873eb846548f4c4516ec4", name: "Kanha", age: 4, work: "Fun", and place: "Vrindavan".

```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
▼ data:
  ▼ 0:
    _id: "67a873eb846548f4c4516ec4"
    name: "Kanha"
    age: 4
    work: "Fun"
    place: "Vrindavan"
```

- In the Postman:



A screenshot of the Postman application interface. The top bar shows the URL "http://localhost:4500/- My Workspace". The main area shows a GET request to "http://localhost:4500/". The "Body" tab is selected, displaying the JSON response from the previous screenshot. The response is a 200 OK status with 13 ms duration and 337 B size. The JSON body is identical to the one shown in the browser.

```
GET New Request • Interceptor debug sessic | GET http://localhost:4500/ + No environment | Save </>
GET http://localhost:4500/
Params Authorization Headers (21) Body Scripts Settings Cookies
Query Params
Key Value Description
Key Value Description
Body Cookies Headers (7) Test Results 40
200 OK 13 ms 337 B
[{"data": [{"_id": "67a873eb846548f4c4516ec4", "name": "Kanha", "age": 4, "work": "Fun", "place": "Vrindavan"}]}
```

- Question:

- Can we pass the **body** in the **GET** method?

-> No.

Now, the different file's code looks a like:

GET.js:

```
const dbConnection = require('./mongodb');

async function getData(resp) {

  try {
    const client = await dbConnection();
    const data = await client.find().toArray();
    console.log(data);
    resp.send({ data });
  } catch (error) {
    console.error('Error fetching data:', error);
    resp.status(500).send('Internal Server Error');
  }
}

module.exports = {
  index: () => {
    const app = express();
    app.get('/', async (req, res) => {
      res.json(await getData(res));
    });
    app.listen(4500, () => {
      console.log('Server is running on port 4500');
    });
  }
};
```

index.js:

```
const exp = require('express')
const GET = require('./GET')

const app = exp()
app.get('/', async (req, res) => {
  res.json(await GET(res))
})

app.listen(4500, () => {
  console.log("It's running on the localhost 4500")
})
```

POST API method:

```
const exp = require('express')
const GET = require('./GET')
```

```

const app = exp()

// Now, here data is getting converted into the JSON format:
app.use(exp.json())

app.get('/data', async (req, resp) => {
    await GET(resp)
})

app.post('/', async (req, resp) => {
    // We can give "/" this to the GET and POST at a time, it doesn't contradict. If the Operation of
    GET is happening theh GET's "/" is working otherwise POST's
    // resp.send({ name: "Murlidhar" })

    // Remember that, POST sends one request at a time.

    // Now. we are taking a input from the Postman and giving an output.
    console.log(req.body)
    resp.send(req.body)
})

app.listen(4500, (err) => {
    console.log("It's running on the localhost 4500")
})

```

- Here, we are using the POSTMAN for performing all an operations:

Open the POSTMAN and put the listening link and then choose an operation.

- Here we are choosing the POST operation for performing the actions.

- Then, we go to the Body's section and then choose the JSON data structure. And putting all the data inside that.

The screenshot shows the Postman application interface. At the top, the URL is http://localhost:4500 - My Workspace. The main area displays a POST request to http://localhost:4500. The request body is set to raw JSON, containing the following data:

```

1 {
2   "name": "Madara Uchiha"
3 }

```

The response section shows a successful 200 OK status with a response time of 16 ms and a response size of 259 B. The response body is identical to the request body.

- Now, we are taking an input from the Postman and putting it in the MongoDB's collections

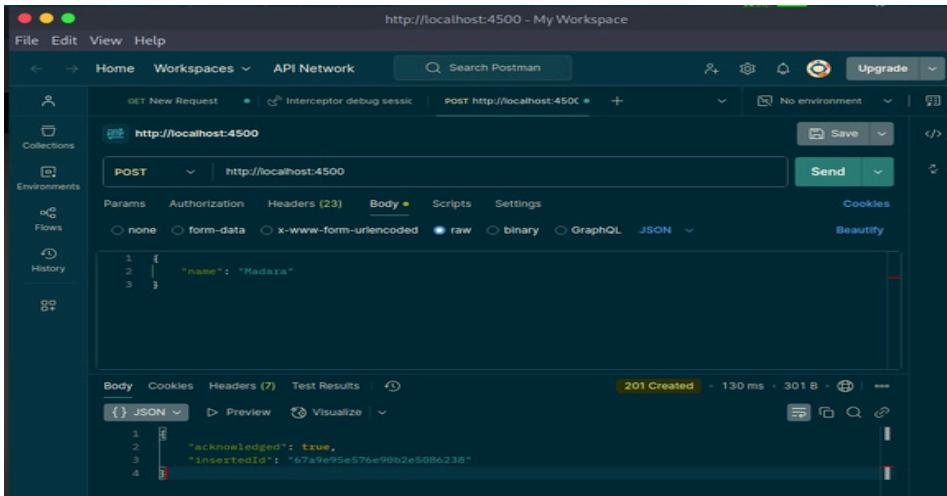
Now, we have to make some changes for the Data usages:

Post API's code:

```

app.post('/products', async (req, resp) => {
  try {
    const collection = await dbConnection();
    const result = await collection.insertOne(req.body);
    resp.status(201).send(result);
  } catch (error) {
    console.error('Error inserting data:', error);
    resp.status(500).send('Internal Server Error: ' + error.message);
  }
});

```



- Here, is how we can communicate with the data.

PUT API

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

    // console.log(req.body)

    let data = await dbConnection()

    let result = data.updateOne(
        // {name: "Madara"}, {$set:{Age:20}} //This method is for updating manually
        { name: req.body.name }, //This one is for the dynamic data.
        { $set: req.body }

    )
    resp.send("Result updated")
})
```

The screenshot shows the MongoDB Compass interface with the 'Products' collection selected. It lists four documents:

- `name: "Karna"`, `age: 4`, `work: "Fun"`, `place: "Vrindavan"`
- `name: "Madara"`, `Age: 20`, `Sex: "Male"`
- `name: "Madara"`, `Age: 34`, `Sex: "Male"`
- `name: "Radhusudan"`

- In the above one you can see that the data is not updated. After putting the PUT request from the POSTMAN it's getting updated.

The screenshot shows the Postman interface. On the left, there's a sidebar with 'Collections', 'Environments', 'Flows', and 'History'. The main area has a header 'Overview' and a search bar 'Search Postman'. Below that, a request card is shown: 'PUT http://localhost:4500/'. The 'Body' tab is selected, showing the following JSON:

```

1  {
2    "name": "RadhaMadhav",
3    "age": 2345
4  }

```

Below the body, the response is displayed: '200 OK' with a duration of '5 ms' and a size of '241 B'. The response body is 'Result updated'.

- Here, you can see that we are using the PUT request to update the data **dynamically**.

The screenshot shows the MongoDB Compass interface. The top navigation bar includes 'Project > Marc > Products', 'Open MongoDB shell', and tabs for 'Documents' (with 1 document), 'Aggregations', 'Schema', 'Indexes' (with 1 index), and 'Validation'. A search bar at the top says 'Type a query: { field: 'value' } or [Generate query +](#)'. Below it are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The results section shows five documents:

- `_id: ObjectId('67a9f729f1c6735d8ecbbc10')`
`name : "Madara"`
`Age : 20`
`Sex : "Male"`
- `_id: ObjectId('67a9f748932248d5ee57998c')`
`name : "Madara"`
`Age : 34`
`Sex : "Male"`
- `_id: ObjectId('67a9ff997fcc35e39ea3f64c0')`
`name : "RadhaMadhav"`
`age : 2345`
- `_id: ObjectId('67a9fb3e0e62cf39b0f6d8c8')`
`name : "RadhaMadhav"`
`age : 34342`
- `_id: ObjectId('67a9fb96363180ab603a171d')`
`name : "RadhaMadhav"`
`age : 2345`

- Here, is the updated data.

- We can even send the data in the parameter format:

```

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

  let data = await dbConnection()

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

  resp.send("Result updated")
})

```

Delete API:

The screenshot shows the MongoDB Compass interface. The top navigation bar includes 'Welcome', 'xyz', 'Products', 'Marc', and a '+' button. Below the navigation is a breadcrumb path 'Project > Marc > Products'. On the left, there's a sidebar with tabs for 'Documents' (selected), 'Aggregations', 'Schema', 'Indexes', and 'Validation'. The main area has a search bar with placeholder text 'Type a query: { field: "value" } or [Generate query](#)' and buttons for 'Explain', 'Reset', 'Find', 'Options', 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. A status bar at the bottom indicates '25 1 - 6 of 6'. The list of documents contains the following data:

- `_id: ObjectId('67a873eb846548f4c4516ec4')`
`name : "Kanha"`
`age : 4`
`work : "Fun"`
`place : "Vrindavan"`
- `_id: ObjectId('67a9f729f1c6735d8ecbbc10')`
`name : "Madara"`
`Age : 20`
`Sex : "Male"`
- `_id: ObjectId('67a9f748932248d5ee57998c')`
`name : "Madara"`
`Age : 34`
`Sex : "Male"`
- `_id: ObjectId('67a9f997fcc35e39ea3f64c0')`
`name : "RadhaKadhai"`
`age : 2345`
- `_id: ObjectId('67a9fb3e0e62cf39b0f6d8c8')`
`name : "RadhaKadhai"`

- We are deleting the Highlighted one ID:

id: `67a9f729f1c6735d8ecbbc10`

Now, Code:

```

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

  let result = await
  console.log(req.params.id)
}

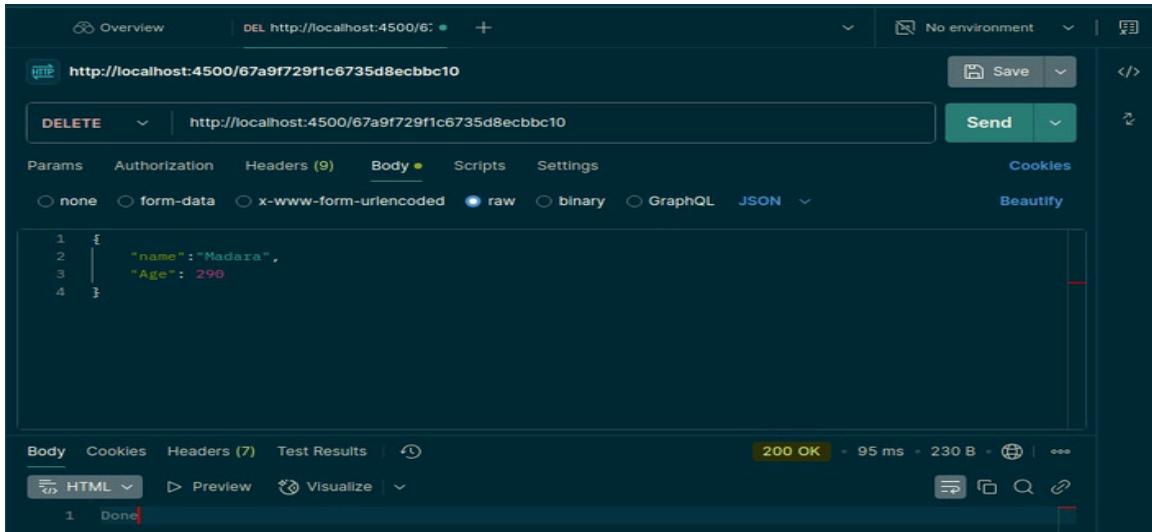
```

```

    resp.send("Done")
}

```

- After performing the operation. It got successful.



- Now, the result is as follows:

The screenshot shows the MongoDB Compass interface for the `Products` collection. The documents listed are:

- `_id: ObjectId('67a873eb846548f4c4516ec4')`, `name: "Kanha"`, `age: 4`, `work: "Fun"`, `place: "Vrindavan"`
- `_id: ObjectId('67a9f729f1c6735d8ecbbc10')`, `name: "Madara"`, `Age: 20`, `Sex: "Male"`
- `_id: ObjectId('67a9f748932248d5ee57998c')`, `name: "RadhaMadhav"`, `Age: 34`, `Sex: "Male"`
- `_id: ObjectId('67a9f997fcc35e39ea3f64c8')`, `name: "RadhaMadhav"`, `age: 2345`

- Here, you can see that in the result that Data is got deleted.

Code:

```
const exp = require('express')
```

```
const dbConnection = require('./mongodb')
```

```
const app = exp()
const mongodb = require('mongodb')

app.use(exp.json())

app.get('/', async (req, resp) => {
  let data = await dbConnection()
  data = await data.find().toArray()
  console.log(data)
  resp.send({ data })
})

app.post('/', async (req, resp) => {
  try {
    const collection = await dbConnection();
    const result = await collection.insertOne(req.body);
    resp.status(201).send(result);
  } catch (error) {
    console.error('Error inserting data:', error);
    resp.status(500).send('Internal Server Error: ' + error.message);
  }
});

app.put("/:id", async (req, resp) => {
  let data = await dbConnection()
  let result = await data.updateOne(
    { name: req.params.name },
    { $set: req.body }
  )
  resp.send("Result updated")
})
```

```

app.delete("/:id", async (req, resp) => {
  let data = await dbConnection()
  let result = await data.deleteOne({ _id: new mongodb.ObjectId(req.params.id) })
  console.log(req.params.id)
  resp.send(result)
})

app.listen(4500, (err) => {
  console.log("It's running on the localhost 4500")
})

```

- Here, we are creating a separate object for performing the id operation.
- And here we are not directly using the - "id", instead of that we are using the - "_id"

- Here, how it shows an output in console:

```

[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
It's running on the localhost 4500
67a9f729f1c6735d8ecbbc10
67a9f729f1c6735d8ecbbc10
67a9f729f1c6735d8ecbbc10

```

Mongoose

In the MongoDB there are some limitations which full-fill by the Mongoose

- In the MongoDB's we were not able to use the **Model** and **Schema**.
 - E.g.: While entering the data there is some specific data that you want but user put some extra data. That data is taken by the MongoDB.

So, to make this proper thing we can make the **schemas** in the **Mongoose** and stop it.

- Here, we can identify the specific values according to our need. Like, specific data type input in the data.

[Click here](#) to go the Mongoose package manager at the “yarn”

```
yarn add mongoose | npm i mongoose
```

-Required packages:

- nodemon
- mongodb

Now, what is mean by Schemas?

-> It validates the field in the Database.

What is model?

-> It uses the Schema and then make a connection between the NodeJS and MongoDB

Code for inserting with Particular conditions:

```
const mongoose = require('mongoose')

// Importing the "moongose"

const url = "mongodb://localhost:27017/Marc"

// Here, we are providing the databases's name.

const main = async () => {

  await mongoose.connect(url)

  // Here, we are making an Schema:

  const ProductSchema = new mongoose.Schema( //Here, we are making an object

  {

    name: String

  }

  )

  const ProductsModel = mongoose.model('Products', ProductSchema) //Here, we are passing the collection name and rules

  let data = new ProductsModel({ name: "m19" })
```

```
let result = await data.save()  
  
console.log(result)  
  
}  
  
main()
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS node - Mongo_0.4 + × [root@parrot]~[~/home/keren/Code3/MongoDB/MongoDB/Mongo_0.4]
o └── #nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
{ name: 'm19', _id: new ObjectId('67ab8126ba3fe57eaf714a20'), __v: 0 }
```

Data in the MongoDB:

```
[Marc] Marc> db.products.find()
[{"_id": ObjectId('67ab7dd0b6de9d3f3d7d1366'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab7dd98e65adb8ba471d6'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab7eb5539d5f02a525c827'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab7ec5be0c60954b81a9da'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab7eca672afe982b0e9929'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab7f09f2d89896264210cf'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab7f3fe48b1bdd9a113b6f'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab7fc727017c2719f0ad8a'), "name": "m19", "__v": 0},
 {"_id": ObjectId('67ab7fff6b87206e545307ce'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab8084d1a65ae734e6da2b'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab810d88c9b349b2d65722'), "name": "m8", "__v": 0},
 {"_id": ObjectId('67ab811a71aed44cbda02010'), "name": "m19", "__v": 0},
 {"_id": ObjectId('67ab8126ba3fe57eaf714a20')}, "name": "m19", "__v": 0}]
```

CRUD with

Mongoose:

- Here, we are performing the CRUD operation on the Data.

Code:

```
const mongoose = require('mongoose')

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

mongoose.connect(url);
```

```
const userSchema = new mongoose.Schema({  
  name: String, age: Number, email: String,  
  
  brand: String,  
});  
  
const User = mongoose.model('Products', userSchema);  
  
const SaveInDB = async () => {  
  let data = new User({  
    name: "ShriHari",  
    age: 1,  
    email: "shrihari@gmail.com",  
    brand: "Universe Builder"  
  })  
  
  let result = await data.save()  
  
  console.log(result)  
}  
  
const updateInDB = async () => {  
  // Here, we are upadting the data for the "m8"  
  
  let data = await User.updateOne({ name: "m8" }, { $set: { brand: "Samsung" } })  
  
  // Here, we don't need to save the data.
```

```
// let result = await data.save()

console.log(data)

}

const deleteInDB = async () => {

let data = await User.deleteOne({ name: "m19" })

// let result = await data.save()

console.log(data)

} const find = async () => {

let data = await User.find({})

console.log(data)

} SaveInDB() updateInDB() deleteInDB() find() - You can changes the MongoDB's query to get your data according to your condition.
```

```
[  
  {  
    _id: new ObjectId('67ab7dd0b6de9d3f3d7d1366'),  
    name: 'm8',  
    __v: 0,  
    brand: 'Samsung'  
  },  
  { _id: new ObjectId('67ab7dd98e65dad8ba471d6'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab7eb5539d5f02a525c827'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab7ec5be0c60954b81a9da'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab7eca672afe982b0e9929'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab7f09f2d89896264210cf'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab7f3fe48b1bdd9a113b6f'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab7fff6b87206e545307ce'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab8084d1a65ae734e6da2b'), name: 'm8', __v: 0 },  
  { _id: new ObjectId('67ab810d88c9b349b2d65722'), name: 'm8', __v: 0 },  
  {  
    _id: new ObjectId('67ab8126ba3fe57eaf714a20'),  
    name: 'm19',  
    __v: 0  
  },  
  {  
    _id: new ObjectId('67ab8af942c20292551e624b'),  
    name: 'ShriHari',  
    email: 'shrihari@gmail.com',  
    brand: 'Universe Builder',  
    __v: 0  
  },  
  {  
    _id: new ObjectId('67ab8b2c25c5cce28fa5fc09'),  
    name: 'ShriHari',  
    email: 'shrihari@gmail.com',  
    brand: 'Universe Builder',  
    __v: 0  
  }  
]
```

- There are different types of an Output. But, most of them are vanished after using the Delete function.
- But, you can check all the files according to your needs.

POST API with Mongoose:

- Require packages:

- nodemon
- mongoose
- mongodb
- express

- Make three files with name of the:

- - config.js
- - Product.js
- - index.js

Code of config.js:

```
//Here, we are making the separate connection.

const mongoose = require('mongoose') const uri =
"mongodb://localhost:27017/Marc"

mongoose.connect(uri)
```

Code of Product.js:

```
const mongoose = require('mongoose') const
productSchemas = new mongoose.Schema({
name: String, price: Number,
brand: String,
category: String
}) module.exports = mongoose.model('products',
productSchemas)
```

code of index.js:

```
const express = require('express')
require('./config')
```

```

const Product = require('./Product')

const app = express()

app.use(express.json()) //This is used for taking the data from the Body and it will automatically
convert it into the JSON file.

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

    let data = new Product(req.body)

    //Here, we are getting that data from the ""

    resp.send("Done")

    console.log(req.body)

    //Now, saving the data inside the MongoDB database

    let result = await data.save()

    console.log(result)

})

app.listen(4500)

```

- Output:



```

[root@parrot]~[/home/keren/Code3/MongoDB/MongoDB/Mongo_0.5]
└── #nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
{
  name: 'm 40',
  price: 4000,
  brand: 'Samsung',
  category: 'mobile'
}
{
  name: 'm 40',
  price: 4000,
  brand: 'Samsung',
  category: 'mobile',
  _id: new ObjectId('67ab95c8635f9fb1abdfe21'),
  __v: 0
}

```

And, input from the POSTMAN's API:

The screenshot shows the Postman interface. At the top, it says "HTTP" and "http://localhost:4500/create". Below that, there's a "POST" method and the URL "http://localhost:4500/create". On the right, there are "Save" and "Send" buttons. Under the "Body" tab, the content type is set to "raw" and the JSON body is:

```
1 {  
2   "name": "m 40",  
3   "price": 4000,  
4   "brand": "Samsung",  
5   "category": "mobile"  
6 }
```

At the bottom, the response status is "200 OK" with a latency of "21 ms" and a size of "230 B". The "Body" tab is selected, showing the response content:

```
_id: ObjectId('67ab95c8635f9fb1abfe21')  
name : "m 40"  
price : 4000  
brand : "Samsung"  
category : "mobile"  
__v : 0
```

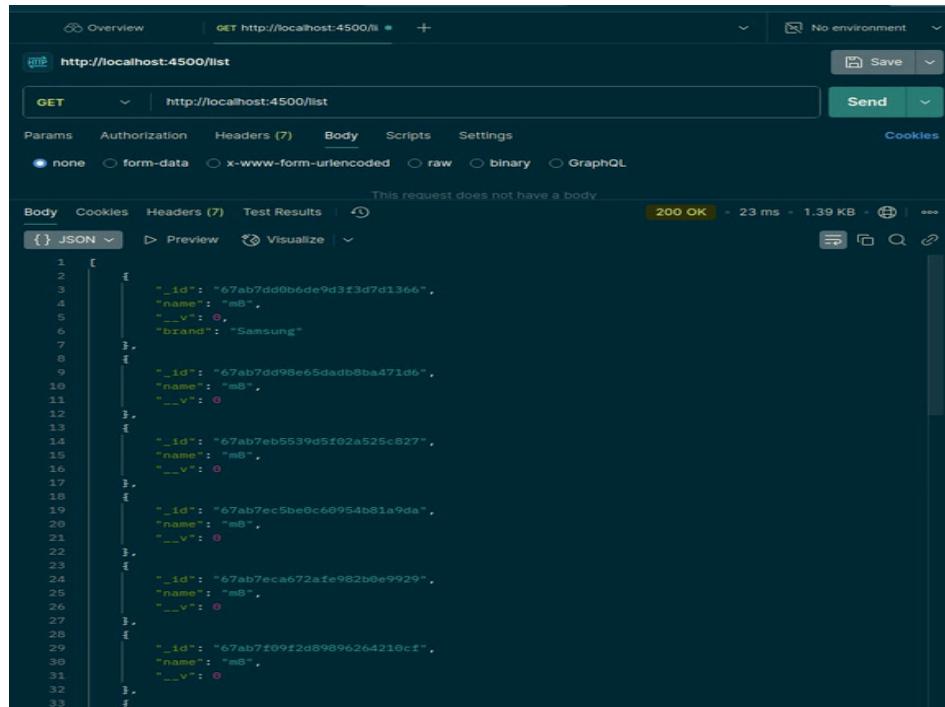
Now, the data got in the MongoDB's database:

```
_id: ObjectId('67ab95c8635f9fb1abfe21')  
name : "m 40"  
price : 4000  
brand : "Samsung"  
category : "mobile"  
__v : 0
```

GET API's:

```
app.get("/list", async (req, resp) => {  
  
  let data = await Product.find()  
  
  resp.send(data)  
  
  console.log(data)  
  
})
```

Output:



The screenshot shows a Postman interface with a successful API call to `http://localhost:4500/list`. The response status is `200 OK` with a response time of 23 ms and a size of 1.39 KB. The JSON response body contains a list of mobile phone documents, each with fields: `_id`, `name`, and `brand`. The list includes documents for 'm 40' (Samsung) and 'mB' (Samsung).

```
[{"_id": "67ab7dd0b6de9d3f3d7d1366", "name": "mB", "__v": 0, "brand": "Samsung"}, {"_id": "67ab7dd98e65dadba471d6", "name": "mB", "__v": 0}, {"_id": "67ab7eb5539d5f02a525c827", "name": "mB", "__v": 0}, {"_id": "67ab7ec5be0c60954b81a9da", "name": "mB", "__v": 0}, {"_id": "67ab7eca672afe982b0e9929", "name": "mB", "__v": 0}, {"_id": "67ab7f09f2d89896264210cf", "name": "mB", "__v": 0}]
```

And output from the console:

```
name: 'ShriHari',
email: 'shrihari@gmail.com',
brand: 'Universe Builder',
__v: 0
},
{
  _id: new ObjectId('67ab8b2c25c5cce28fa5fc09'),
  name: 'ShriHari',
  email: 'shrihari@gmail.com',
  brand: 'Universe Builder',
  __v: 0
},
{
  _id: new ObjectId('67ab95790fb0aaf1da8b65da'),
  name: 'm 40',
  price: 4000,
  brand: 'Samsung',
  category: 'mobile',
  __v: 0
},
{
  _id: new ObjectId('67ab958e52182982defc2d24'),
  name: 'm 40',
  price: 4000,
  brand: 'Samsung',
  category: 'mobile',
  __v: 0
},
{
  _id: new ObjectId('67ab95c8635f9fb1abdfe21'),
  name: 'm 40',
  price: 4000,
  brand: 'Samsung',
  category: 'mobile',
  __v: 0
}
```

DELETE API's Code:

```
app.delete("/delete/:_id", async (req, resp) => {
  console.log(req.params)
  let data = await Product.deleteOne(req.params)
  resp.send("done")
})
```

- Here, we are passing an Object's ID.

Before deleting the Data:

```
_id: ObjectId('67ab95790fb0aa1da8b65da')
name : "ShriHari"
email : "shrihari@gmail.com"
brand : "Universe Builder"
__v : 0

_id: ObjectId('67ab958e52182982defc2d24')
name : "# 40"
price : 4000
brand : "Samsung"
category : "mobile"
__v : 0

_id: ObjectId('67ab95c8635f9fb1abdfe21')
name : "# 40"
price : 4000
brand : "Samsung"
category : "mobile"
__v : 0
```

Now, we are taking the last one's object ID:

67ab95c8635f9fb1abdfe21

The screenshot shows the Postman application interface. At the top, there is an 'Overview' tab, a URL field containing 'http://localhost:4500/delete/67ab95c8635f9fb1abdfe21', and a 'Send' button. Below the URL field, the method is set to 'DELETE'. The 'Headers' tab is selected, showing '(7)' entries. Under the 'Body' tab, it says 'This request does not have a body'. The 'Test Results' tab shows a successful response: '200 OK', '12 ms', '230 B', and a small globe icon. At the bottom, there are buttons for 'HTML', 'Preview', 'Visualize', and other interface controls.

- After performing this operation, we will get an output in console also.

```
[root@parrot]~[/home/keren/Code3/MongoDB/MongoDB/Mongo_0.5]
o └─ #nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Port is running on the localhost 4500
{ id: 'max' }
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Port is running on the localhost 4500
{ id: '67ab95c8635f9fb1abdfe21' }
```

Now, an output in the MongoDB's database is like:

```
_id: ObjectId('67ab8b2c25c5cce28fa5fc09')
name: "ShriHari"
email: "shrihari@gmail.com"
brand: "Universe Builder"
__v: 0

_id: ObjectId('67ab95790fb0aaaf1da8b65da')
name: "m 40"
price: 4000
brand: "Samsung"
category: "mobile"
__v: 0

_id: ObjectId('67ab958e52182982defc2d24')
name: "m 40"
price: 4000
brand: "Samsung"
category: "mobile"
__v: 0
```

PUT's Update's API's Code:

```
app.put("/update/:_id", async (req, resp) => {
  try {
    const productId = req.params._id
    const updateData = req.body

    let data = await Product.findByIdAndUpdate(productId, updateData, { new: true })
    resp.json(data)
  } catch (err) {
    resp.status(500).json({ message: err.message })
  }
})
```

```

if (data) {

    let result = await data.save()

    console.log(result)

    resp.status(200).send("Product update successfully")

}

else {

    res.status(404).send("Product not found")

}

} catch (error) {

    console.error("Error updating products:", error)

    res.status(500).send("Internal server error")

}

}

```

```

1 _id: ObjectId('67ab958e52182982defc2d24')
2 name : "m 40"
3 price : 4000
4 brand : "Samsung"
5 category : "mobile"
6 __v : 0

```

ObjectId	String
String	Int32
String	String
Int32	Int32

- Here, the price is 4000, and now we are updating that price with the Object ID:

```

_id: ObjectId('67ab958e52182982defc2d24')
name : "m 40"
price : 80000
brand : "Samsung"
category : "mobile"
__v : 0

```

Postman's data:

The screenshot shows the Postman application interface. At the top, the URL is set to `http://localhost:4500/update/67ab958e52182982defc2d24`. Below the URL, the method is selected as `PUT`, and the target URL is `http://localhost:4500/update/67ab958e52182982defc2d24`. A green "Send" button is visible on the right. The "Body" tab is selected, showing a raw JSON payload:

```
1  {
2    |   "price":80000
3  }
```

Below the body, the response status is `200 OK` with a response time of `155 ms` and a size of `255 B`. The response content is: "Product update successfully".

- Terminal's output.

The terminal window shows the execution of a `nodemon index.js` command. The output indicates that nodemon version 3.1.9 is running, watching files in the current directory, and starting a node process for `index.js`. It also mentions that the port is running on localhost 4500. The final part of the output shows a MongoDB document being inserted or updated, with fields like `_id`, `name`, `price`, `brand`, `category`, and `__v`:

```
[root@parrot]~[/home/keren/Code3/MongoDB/MongoDB/Mongo_0.5]
o └─#nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Port is running on the localhost 4500
{
  _id: new ObjectId('67ab958e52182982defc2d24'),
  name: 'm 40',
  price: 80000,
  brand: 'Samsung',
  category: 'mobile',
  __v: 0
}
```

Final Code of the Mongoose CRUD API's:

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

```
require('./config')
const Product = require('./Product')
const app = express()

app.use(express.json()) //This is used for taking the data from the Body and it will automatically
convert it into the JSON file.
app.post("/create", async (req, resp) => {
    let data = new Product(req.body)
    // Here, we are getting that data from the ""
    resp.send("Done")
    console.log(req.body)
    // Now, saving the data inside the MongoDB database
    let result = await data.save()
    console.log(result)
})

app.get("/list", async (req, resp) => {
    let data = await Product.find()
    resp.send(data)
    console.log(data)
})

app.delete("/delete/:_id", async (req, resp) => {
    console.log(req.params)
    let data = await Product.deleteOne(req.params)
    resp.send("done")
})

app.put("/update/:_id", async (req, resp) => {
    try {
        const productId = req.params._id
```

```

const updateData = req.body

let data = await Product.findByIdAndUpdate(productId, updateData, { new: true })

if (data) {
  let result = await data.save()
  console.log(result)
  res.status(200).send("Product updated successfully")
}
else {
  res.status(404).send("Product not found")
}

} catch (error) {
  console.error("Error updating products:", error)
  res.status(500).send("Internal server error")
}

}

app.listen(4500, () => {
  console.log("Port is running on the localhost 4500")
})

```

Search API in Mongoose:

Code:

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

```
const key = req.params.key;

// let data = await Product.find({ brand: key }) //This is normal method to find the data from the database

// Now, we are working with the "regex"

let data = await Product.find({

  "$or": [
    { "brand": { $regex: key } }

  ]
}

resp.send(data)

})
```

```
[root@parrot]# nodemon
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node index.js'
localhost is running on the 4500 port
[nodemon] restarting due to changes...
[nodemon] starting 'node index.js'
localhost is running on the 4500 port
█
```

- The key should be **case – sensitive**.

The screenshot shows a Postman interface with the following details:

- Request URL:** `http://localhost:4500/find/Samsung`
- Method:** GET
- Headers:** (7) - Includes `Content-Type: application/json`, `Accept: application/json`, and `Cache-Control: no-cache`.
- Body:** (Raw JSON) - Displays the response body as an array of three documents. Each document has fields: `_id`, `name`, `price`, `brand`, and `category`. The first document is for a `m8` phone, the second for a `m 40` phone, and the third for a `m 40` phone.
- Test Results:** (1) - Shows a success message: `Test passed: Response status was 200 OK`.
- Response Headers:** (7) - Includes `Content-Type: application/json`, `Content-Length: 528`, `Content-Encoding: gzip`, `Date: Mon, 10 Oct 2022 10:45:20 GMT`, `Connection: keep-alive`, `Set-Cookie: session=67ab95790fb0aaaf1da8b65da; expires=Mon, 10-Oct-2022 10:45:20 UTC; path=/; secure; HttpOnly`, and `X-Powered-By: Express`.
- Body Content (JSON View):**

```

1  [
2    {
3      "_id": "67ab95790fb0aaaf1da8b65da",
4      "name": "m8",
5      "__v": 0,
6      "brand": "Samsung"
7    },
8    {
9      "_id": "67ab95790fb0aaaf1da8b65da",
10     "name": "m 40",
11     "price": 4000,
12     "brand": "Samsung",
13     "category": "mobile",
14     "__v": 0
15   },
16   {
17     "_id": "67ab958e52182982defc2d24",
18     "name": "m 40",
19     "price": 80000,
20     "brand": "Samsung",
21     "category": "mobile",
22     "__v": 0
23   }
24 ]

```

- Now, we want to search different types of data in the field then we are using the:

`$or:[]`

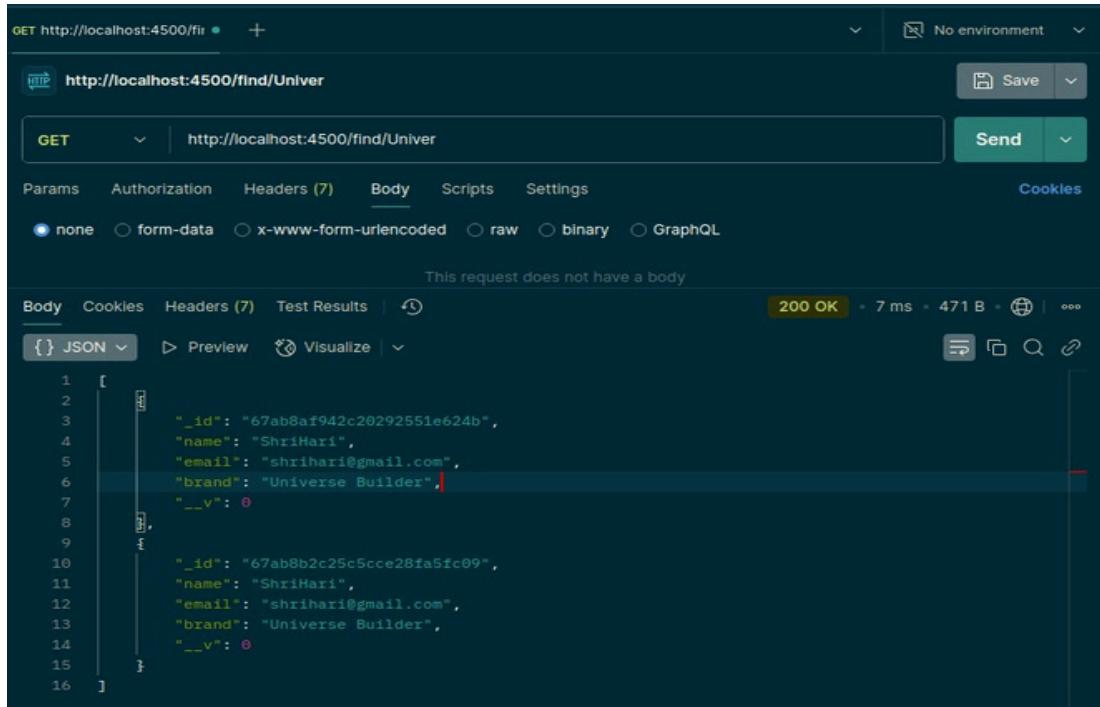
- You can add as much as field that you want to get the result.

```
app.get("/find/:key", async (req, resp) => {
  const key = req.params.key;

  let data = await Product.find({
    //Here, "$or":[] - It is used for the finding multiple data field at once, without changing the data
    - types
    "$or": [
      { "name": { $regex: key } },
      { "brand": { $regex: key } }
    ]
  })
  resp.send(data)
})
```

- Output:

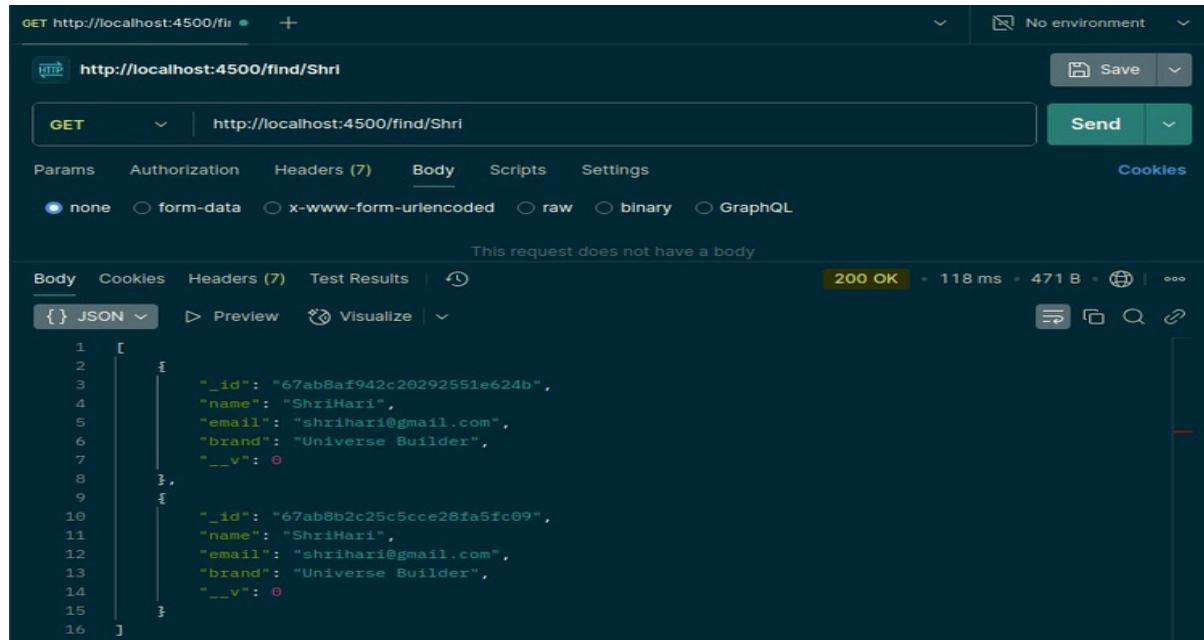
Here, you can see we are just searching the – “Univer”, and it shows me the data.



The screenshot shows a Postman request for `http://localhost:4500/find/Univer`. The response is a `200 OK` with a body containing two JSON documents:

```
[{"_id": "67ab8af942c20292551e624b", "name": "ShriHari", "email": "shrihari@gmail.com", "brand": "Universe Builder", "__v": 0}, {"_id": "67ab8b2c25c5cce28fa5fc09", "name": "ShriHari", "email": "shrihari@gmail.com", "brand": "Universe Builder", "__v": 0}]
```

And, one more thing that. Now, we are trying to search the data with name:



The screenshot shows a Postman request for `http://localhost:4500/find/Shri`. The response is a `200 OK` with a body containing two JSON documents, identical to the previous search:

```
[{"_id": "67ab8af942c20292551e624b", "name": "ShriHari", "email": "shrihari@gmail.com", "brand": "Universe Builder", "__v": 0}, {"_id": "67ab8b2c25c5cce28fa5fc09", "name": "ShriHari", "email": "shrihari@gmail.com", "brand": "Universe Builder", "__v": 0}]
```

Question: Can we search using the DELETE, POST, and PUT method?

-> Yes, we can do that. But, as per Coding standard it's not recommended to do that.

Upload File:

- Here, we are installing the – **multer** package

```
yarn add multer
```

- Code will be given later:

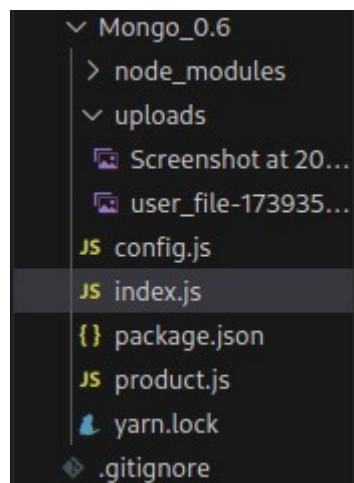
Steps from the POSTMAN API's application:

- Choose the **POST** method, and then choose the **form-data** after that put the some value in the **key**

The screenshot shows the POSTMAN interface with a POST request to `http://localhost:4500/upload`. The 'Body' tab is selected, and the 'form-data' option is chosen. A table under 'Body' shows a single entry with 'Key' as 'user_file' and 'Value' as 'Text'. There are other tabs like 'Params', 'Headers', 'Scripts', and 'Settings'.

- After that choose the **files** instead of the texts, and upload it.

Now, directories order:



Code:

```
const express = require('express')
const multer = require('multer')
require('./config')
const Product = require('./product')

const app = express()

const upload = multer({
  storage: multer.diskStorage({
    destination: function (req, files, cb) {
      cb(null, "uploads") // "uploads" -> It is a destination directory
    },
    filename: function (req, file, cb) {
      cb(null, file.fieldname + "-" + Date.now() + ".jpg")
    }
  })
  // Here, we are defining the condition we have to upload the - "single" file or "multiple" file at a time
}).single("user_file") // Here, we are defining the parameter of the function

app.post("/upload", upload, async (req, resp) => {
  resp.send("File uploaded").status(200)
})
```

```

app.listen(4500, (err) => {
  if (err) {
    console.error("Error occurred", err)
  } else {
    console.log("Site is running on the 4500 port")
  }
})

```

- Here, you can see that we are uploading the file from the different directories.

And stores data inside the – “upload directories”

The screenshot shows a Postman interface for a POST request to `http://localhost:4500/upload`. The 'Body' tab is selected, showing a form-data key `user_file` with a value of `1358140.png`. The response tab shows a successful `200 OK` status with the message `1 File uploaded`.

- Here, you can see that the file has been uploaded inside the upload directory.

OS Module:

It is for the Operating System's information

```
const os = require('os')

// console.log(os)

console.log(os.freemem()/(1024*1024*1024)) //Just some memory

console.log(os.totalmem()/(1024*1024*1024)) //Shows the totla memory in the system.

// Now, checking the HOSTName

console.log(os.hostname())

console.log(os.platform())

console.log(os.userInfo())
```

Output:

```
[root@parrot]~[/home/keren/Code3/MongoDB/MongoDB/LastOne]
● └─#node index.js

1.1868705749511719
7.12322998046875
parrot
linux
[Object: null prototype] {
  uid: 0,
  gid: 0,
  username: 'root',
  homedir: '/root',
  shell: '/bin/bash'
}
```

Events and Event Emitter in NodeJS:

- In simple language, **event** means the signal. It's like the ReactJS's **onClick** event
- Buttons are made up in the HTML, CSS and JS but and it is called through the API of the NodeJS's
- Event is also inbuilt module inside the NodeJS.

Code:

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

// Here, we are taking some capital letter's "EventEmitter", this is because it's an object and it's a function

```
const EventEmitter = require('events')
```

```
const app = express()
```

```
let count = 0;
```

// Now, we want to count that how many API's are inside in my project. So, that's why we are making an event

```
const event = new EventEmitter()
```

```
event.on("CountAPI", () => {
```

```
    count++;
```

```
    console.log(` Event called ${count}`)
```

```
)}
```

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

```
    resp.send("Main API")
```

```
event.emit("CountAPI")  
}  
  
app.get("/search", async (req, resp) => {  
    resp.send("Search API")  
    event.emit("CountAPI")  
}  
  
}  
  
app.get("/module", async (req, resp) => {  
    resp.send("Module's API")  
    event.emit("CountAPI")  
}  
  
}  
  
app.listen(4500, (err) => {  
    if (err) {  
        console.error("Error occurred", err)  
    }  
    e l s e {  
        console.log("Site runningg on the PORT 4500")  
    }  
})
```

- Output:

```
[root@parrot]~[/home/keren/Code3/MongoDB/MongoDB/LastOne]
o #nodemon Event.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node Event.js`
Site runningg on the PORT 4500
Event called 1
Event called 2
Event called 3
Event called 4
Event called 5
Event called 6
```

The screenshot shows the Postman application interface. At the top, there are navigation links for Home, Workspaces, API Network, and a search bar. On the right side, there are buttons for upgrading, saving, sharing, and closing. Below the header, a list of requests is shown with one selected: "GET http://localhost:4500/module". The main area contains a form for making a request. It has dropdown menus for "Method" (set to "GET") and "URL" (set to "http://localhost:4500/module"). To the right of the URL field is a "Send" button. Below the form, there are tabs for "Params", "Authorization", "Headers (7)", "Body", "Scripts", and "Settings". Under "Headers (7)", there is a table with one row. The table has columns for "Key" and "Value". There is one entry: "Key" is empty and "Value" is also empty. To the right of the table is a "Description" column with the text "Description". At the bottom of the interface, there are tabs for "Body", "Cookies", "Headers (7)", and "Test Results". The "Test Results" tab is active, showing a status of "200 OK" with a response time of "3 ms" and a size of "239 B". Below the status, there are buttons for "HTML", "Preview", and "Visualize". The "HTML" button is highlighted.

- By changing the different API's and performing an Event operation. It will get completed.

REPL (Read – Eval (Evaluation) – Print – Loop)

- It is a command line tool of the NodeJS's where you can run the NodeJS's and JS's code on that.

```
[keren@parrot]~$node
Welcome to Node.js v18.19.0.
Type ".help" for more information.
> 2+2
4
> a = 3; b = 5; a+b;
8
> function test(){
... console.log("Hello world");
...
}
undefined
> test()
Hello world
undefined
> █
```

- This is just for the very simple operations. But, we need the editor of the NodeJS's then we use given below command:

```
.editor
```

```
> .editor
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
function apple(){
console.log("Just an Apple")
}
apple()

Just an Apple
undefined
█
```

To know more about this: .help

```
Just an Apple
undefined
> .help
.break Sometimes you get stuck, this gets you out
.clear Alias for .break
.editor Enter editor mode
.exit Exit the REPL
.help Print this help message
.load Load JS from a file into the REPL session
.save Save all evaluated commands in this REPL session to a file

Press Ctrl+C to abort current expression, Ctrl+D to exit the REPL
> █
```

Question:

```
const x = 10  
console.log(x++)
```

-> We have taken the **const** that's why **x** will not be change, and gives an error

```
[root@parrot]~[ /home/keren/Code3/MongoDB]  
└─#node "/home/keren/Code3/MongoDB/MongoDB/LastOne/repl.js"  
/home/keren/Code3/MongoDB/MongoDB/LastOne/repl.js:2  
  console.log(x++)  
          ^  
  
TypeError: Assignment to constant variable.  
    at Object.<anonymous> (/home/keren/Code3/MongoDB/MongoDB/LastOne/repl.js:2:14)  
    at Module._compile (node:internal/modules/cjs/loader:1562:14)  
    at Object..js (node:internal/modules/cjs/loader:1699:10)  
    at Module.load (node:internal/modules/cjs/loader:1313:32)  
    at Function._load (node:internal/modules/cjs/loader:1123:12)  
    at TracingChannel.traceSync (node:diagnostics_channel:322:14)  
    at wrapModuleLoad (node:internal/modules/cjs/loader:217:24)  
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:170:5)  
    at node:internal/main/run_main_module:36:49  
  
Node.js v22.13.0
```

[Linkedin](#)

[ConnectWithMe](#)

[GitHub](#)

Libraries():

- - `filter()`: //Used to filter out the particular things from the set
- - `console.warn()`: //It gives a particular warning and warning symbol.
- - `setTimeOut()`: //It used to delay the output and apply a particular time boundation on the output

