**iNotebook Backend**

Building a Node js REST API is a four-step process. Follow the steps given below to build a secure Node js REST API:

* [Step 1: Create the Required Directories](https://hevodata.com/learn/building-a-secure-node-js-rest-api/#s1)
* [Step 2: Create your First App Express API](https://hevodata.com/learn/building-a-secure-node-js-rest-api/#s2)
* [Step 3: Creating the User Module](https://hevodata.com/learn/building-a-secure-node-js-rest-api/#s3)
* [Step 4: Creating the Auth Module](https://hevodata.com/learn/building-a-secure-node-js-rest-api/#s4)

**\*It is recommended that all the code of the backend is been written inside the try and catch block and if the error comes than the catch will handle it and the app will not be crashed**

**🡪Make sure to write the await before any statement that works with the database otherwise it will give an error at one or other point, i.e save, find, etc statements**

**🡪Make sure that you do not use the body in the GET request, Instead use the headers and can get the data of the header by the help of the following statement: req.header(“<HEADER\_NAME>”);**

**🡺Initial Setup**

1. For setup of the MERN we have to install the required things like mongoDB and node.js and than install the react(make app), express.js
2. Always keep the backend and frontend(react-app) folder different from each other and also keep it separate on the GIT also
3. First we will open the backend folder in the vs code
4. Make sure you had installed the Thunder client extension in the vs code
5. One can also use the postman instead of the Thunder client but the plus point of using the using thunder client is that we can use it directly in the vs code and we do not require to open the new tab
6. Then open the terminal in the vs code and write : **npm init**
7. And then we have to give the answers to some of the questions like name, description, etc and if you do not want to give answer than make the enter without typing anything till the question like y to write is come
8. And then write y and enter and then the package.json will be maded
9. Than write the **npm i express** and due to it the express server will be installed
10. Than write the npm i mongoose to install the mongoose, The mongoose is a framework of the node.js which is used makes us easy to use the mongoDB
11. And if vulnerabilities comes than do not take care about it they come and go
12. Moongose is an abstraction layer on addition to the mongoDB which will help us to connect to the node .js
13. Then make the index.js which is our entery point
14. And the thunder client that we installed that help us to test the api’s
15. Now, We have to open the project folder and then in that we have to open the .gitignore file and in that there will be #dependencies and in it :

🡪Change **/node\_modules** to the **node\_modules**

🡪And then only update the project on the git, As the setup files in the node modules will not get uploaded now, which are not necessary

1. In the package.json all the information is stored about that project and than we can get that information from it, Eg. we can get the list of the dependencies from the package.json
2. Now steps for configuring the mongoDB
3. Click on the new connection
4. Advance connection options
5. Do not make any changes in the default things
6. Click on connect
7. After doing it a dashboard screen will be come where we can see various things like My Queries, Databases, Performance
8. (**Not Compulsory**)And if you want to open the existing connection than make the double click on the server you want to see which is on the right side of the screen
9. And then go to the navbar and click on connect🡪disconnect🡪Then copy the URI link present there
10. And than in the **db.js** write the following code to connect to the mongoDB server:

const mongoose =require(‘mongoose’);

const mongoURI=”<Paste the uri that you copied(and make it /<new-databse-name>)>”

const connectToMongo=()=>{

mongoose.connect(mongoURI, ()=>{

console.log(“Connected to mongo Successfully”)

})

}

module.exports =connectToMongo;

🡪Here instead of the second function that we used we can also change the connectToMongo function to the async and it it we can use the await, But this method is also good but it will follow the JS’s asynchronized function.

1. Then write the following code in the **index.js**

const connectToMongo =require(‘./db’);

connectToMongo();

1. Then install the following things or statement in the terminal :

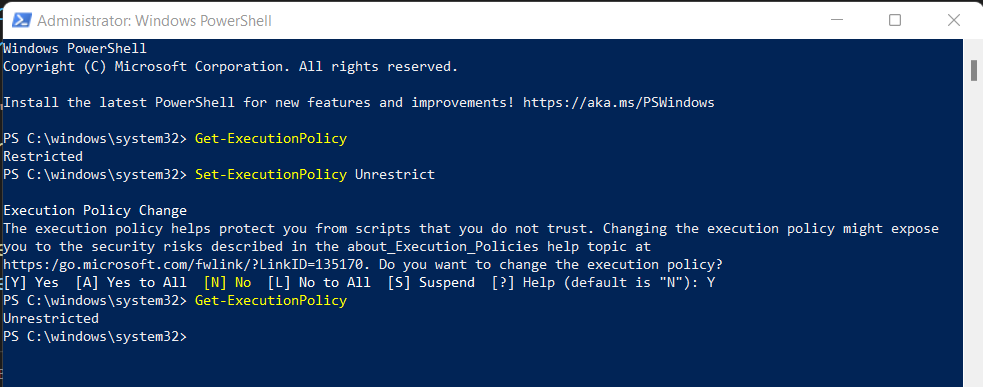
**npm install -D nodemon**  (see without writing if the output is coming or not)

🡪Wait till install and then write the following statement :

**nodemon .\index.js**

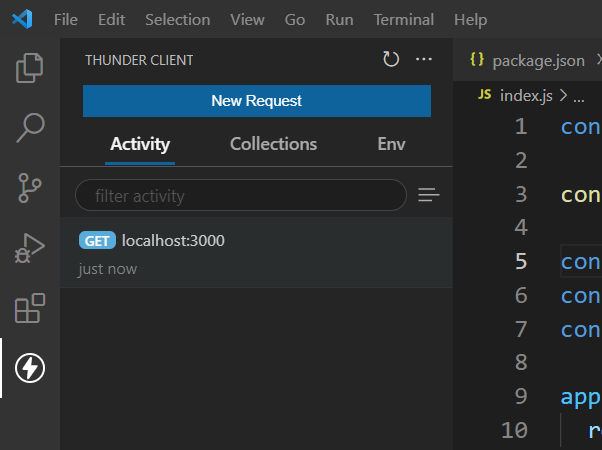
🡪And in the output we will see the **Connected to mongo Successfully** which we had written in the db.js

1. If after writing the nodemon .\index.js it gives error than follow the following instruction, **If not than bypass this step**
2. Open the powerShell as the administrator
3. Then follow the following steps :



🡪After doing this process the nodemon .\index.js will start working

1. And if in the terminal of the project it shows the error like **server crashed** than there would be any mistake in the program
2. Than go to the official website of the **express js**🡪getting started🡪Next: Hello
3. World(At the end of article)🡪Copy the code template which would be there🡪And paste it in the index.js after the things that we had written in it🡪and then save it and🡪go to terminal and write🡪**node index.js**🡪and then the output would come like 🡪Example app listening on port 3000, connected to mongo successful
4. Then go to the thunder extension that we have downloaded which is the last one in the image shown bellow :



This One

1. Click on the new request than add the localhost:<port-number-that-comed-in-terminal >, And then send and you will get the output results of the server
2. Now to keep the **mongoose models** in the project, We will make an folder named **models in the backend folder** as it is the part of the backend
3. And also make an another folder in the backend folder named routes
4. Now we will make the models of the mongoose in the models folder and while naming it take care while naming it, The name of the model should always start from the capital alphabet
5. We will make the file named Note.js, User.js
6. And on the starting of every models of mongoose we have to add the following lines on top:

**const mongoose = require(“mongoose”);**

**const {Schema} = mongoose;**

1. Now we want to make an schema(The term "schema" refers to the organization of data as a blueprint of how the database is constructed) in the mongoDB
2. Now we can see the steps for making schema and other thing from the documentation of the mongoose, For it search mongoose🡪read the docs🡪schema

### Step 3: Creating the User Module

You can use [Mongoose](https://mongoosejs.com/docs/), an **Object-Oriented Data Modelling(ODM)** library to create the User Guide within the User Schema that will help you in creating a Node js REST API. Follow the steps given below to create the user module for your project:

* Create a new schema by using the commands given below:

1. The format for making the schema’s is :

🡪The Schema is actually nothing but an object(relative to java) by the help of which we can take the data as per our requirement

const UserSchema =new Schema({

name: {

type: String,

required: true,

},

email: {

type: String,

required: true,

unique: true,

},

})

🡪Here, in the schema we can make the many columns or object by the help of the this method

🡪In above example the UserSchema is the name of the schema and the name, email are the columns or objects

1. And don’t forget to add the following line at the end of the model otherwise the server will crash

**module.exports = mongoose.model(“user”,<schema-name>);**

* **\***Now, simply connect the schema to the User Module by using the following command.
* Once you have connected the schema to the User Module, you can leverage this model to perform all of the**CRUD** operations that are required within the Express endpoints.

🡪Here we have to change the user instead of which we can write the thing by which we want to export

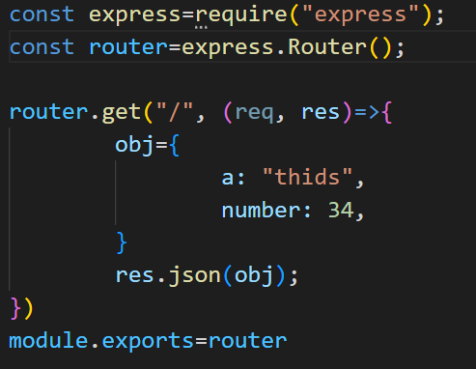
🡪And we also have to change the schema name also

1. **We put the schemas in the model folder as we make the model from the schemas**
2. We can link the routes to the app by the help of the app.use() method in **index.js**

Eg. **app.use(“/api/auth”, require(“./routes/auth”));**

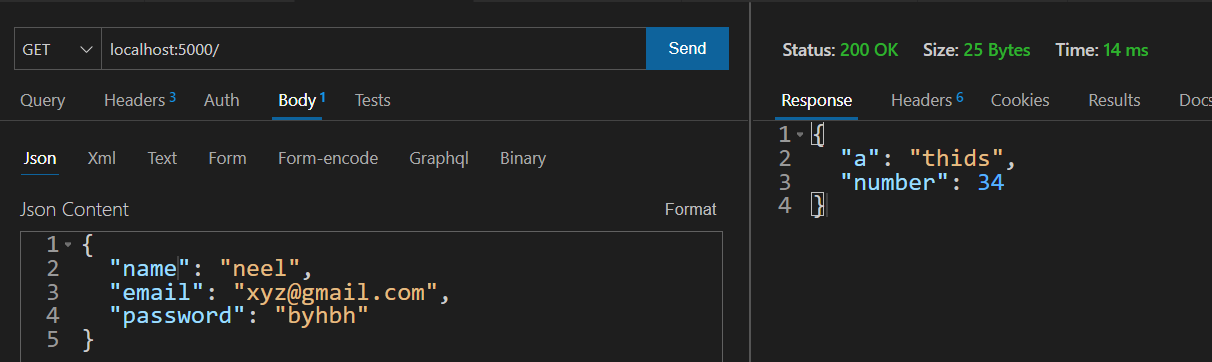
🡪In this example we have to change the name of the url which would be different for each thing and we can also change the file name which can also be different

1. Now we have to make the files in the routes folder, And in the routes, we do not have to compulsorily write the the first character capital
2. We will make the two files in the routes folder named **auth.js and the notes.js**
3. And in it we would write the things in the following format :

****

🡪Here, we can change the obj and instead of it we can put any data

🡪Now lets check that if our API is woking or not, By the help of the thunder client, The process to do it is metioned bellow after the some point(Here the things are done directly for the checking)



🡪The reply that we get is in the form of the json format and the body and headers are of the no use in the following examples

1. We can always start the api by the help of: **nodemon .\index.js**
2. **In the auth.js, we will connect it with the mongoDB database:**

🡪By removing all the old thing from the router.get and adding the following things in it:

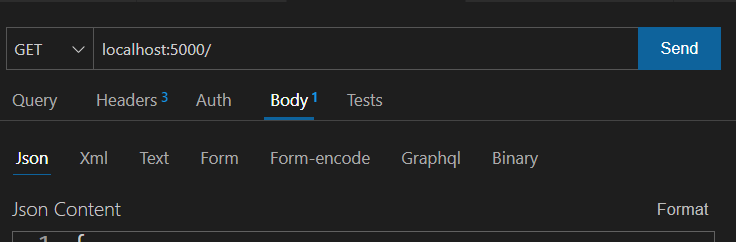
**console.log(req.body);**

**res.send(req.body);**

**🡪**The first sentence will make the data print on the console while the last will make the data print of the response

* You can now test the Mongoose model by starting the server (**npm init start**) and sending a POST request to /users with dummy JSON data:

**🡪Here the router.get is used but instead we can also use the router.post the only difference between them are they are different methods for operation of data in browser, If here we change the get to post or vice-versa than we have to also make change in the thunder client and there we have to also change the method which is aside the box to enter the url**

****

🡪Then add the following thing on the top of the auth.js

**const User =require(“../models/User”)**

\*If the model is made by any other name than instead of the User use that name by which it is exported and also change the address according to it

🡪This will **allow us to make the contact with databse via the user which is the schema and modules** and we have to enter the data as per the requirement of the user schema

🡪Now we have to make **an object of the user schema like bellow and add in the router.get or router.post method in the auth.js**

**const user=new User(req.body);**

**user.save();**

🡪This two statements are the main statements which will add the data to the MongoDB

🡪Each document can be saved to the database by calling its [save](https://mongoosejs.com/docs/api.html#model_Model-save) method. The first argument to the callback will be an error if any occurred.

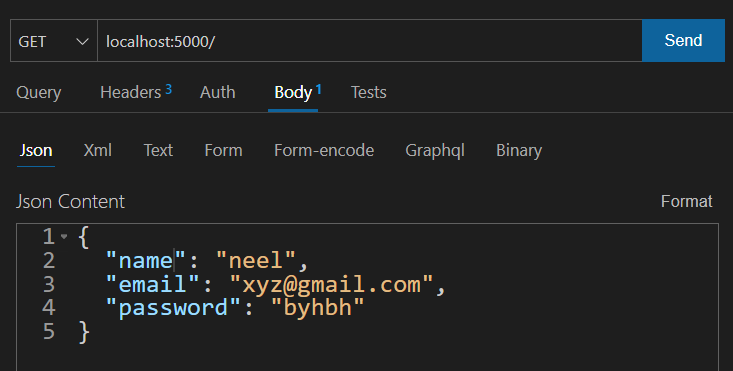
**🡪In the above first statement the user(with small u) is the object of the User which is an Schema**

**🡪Make sure you can make any name of the object of the Schema**

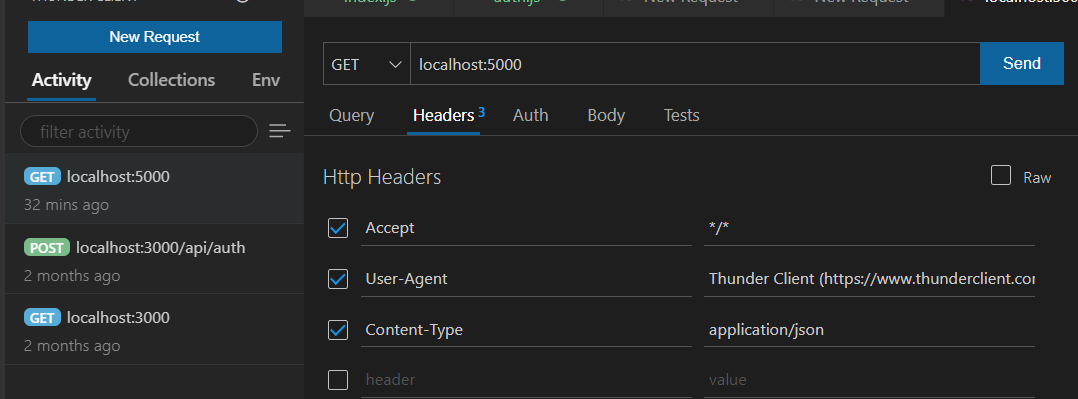
1. Now we have to add the data to the database from a .json file that is coming from the browser and for it, we have to add the following statement in the **index.js** before another app.use() statements, And this statement is called as the **middleware**

**app.use(express.json())**

****

1. And for sending the data from the .json from the browser for trail we will go to the thunder client init open an existing request or make an new request as and in it go to the body, And choose the json and write the data you want to send example :   
   **\*The images used contains localhost:5000 instead of the localhost:3000 as it would be changed afterwards**
2. And before running the program we have to make one another change in the thunder client which is,

🡪Go to the headers🡪type following things🡪Content-type🡪application\json



🡪run this program by writing the following url instead of the default one :

localhost:3000/api/auth/

🡪If the address of the file is to be changed than in the index.js add the another statement as that of the following one :

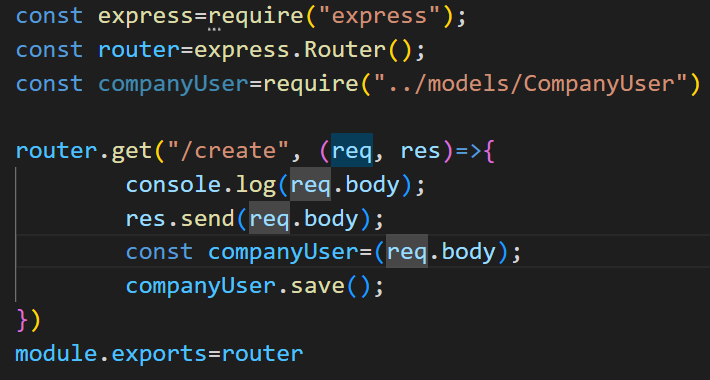


🡪If the api’s method is made in the new file other than that of the auth.js which is in the routes folder we have to first set an path at which the file is to be made called, It can be any address but it is defined to keep the address in the topic wise i.e. as we are now working with the api we will write the /api and than for what function we are using that thing i.e. /auth

🡪And than make the coma and write the require in which we have to write the address of the file which is to be called

🡪And hence while calling from the thunder client or any where we will be redirected to the specified location as per the url

🡪Also as per our requirement we have to also make some changes in the file which we are using i.e. in this case we are using the auth.js in the routes folder and in it we have to change the method as per which we are using and the url at which we want to use this method as there can be more than one method in the same file, And the other statements which are on the top of the file are also required



🡪In this block of the code we can change the method i.e. instead of the .get we can use the .post, .put or any other

🡪And instead of the /create which is an url, we can use any other url as per our requirement

🡪Also don’t forget to mention the (req, res) function

**🡺Adding the data validation using express-validator**

1. Express Validator is a set of Express. js middleware that wraps validator. js , a library that provides validator and sanitizer functions. Simply said, Express Validator is an Express middleware library that you can incorporate in your apps for **server-side data validation**, i.e. we can check the given input is of the specific length or the it is number or string and etc
2. First install the express-validator by the help of the following statement in the terminal of the backend of the project(Install it in new terminal not in which the nodemon is running as it will not install in it)

**npm install --save express-validator**

1. The steps done are as per the **express.js validator documentation** to get it express.js validator website🡪documentation🡪getting started🡪And for starting from now start from the second template code **Or** **otherwise the steps are mentioned below**
2. Add the following statement in the router files **i.e. auth.js** with other const’s in which we want to validate the data of the user

**const {body, validationResult} =require(‘express-validator’);**

1. Then as per the documentation in the router.post we will make an array and in which we will check the validation, and also change the logic as per the documentation(will copy from documentation in the **auth.js** or another file in the routes if working with other) the example is as follow :

router.post("/", [

    body('email').isEmail(),

    body('name').isLength({ min :5}),

    body('password').isLength({ min: 5 }),

], (req,res)=>{

    const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });

    }

    res.send(req.body);

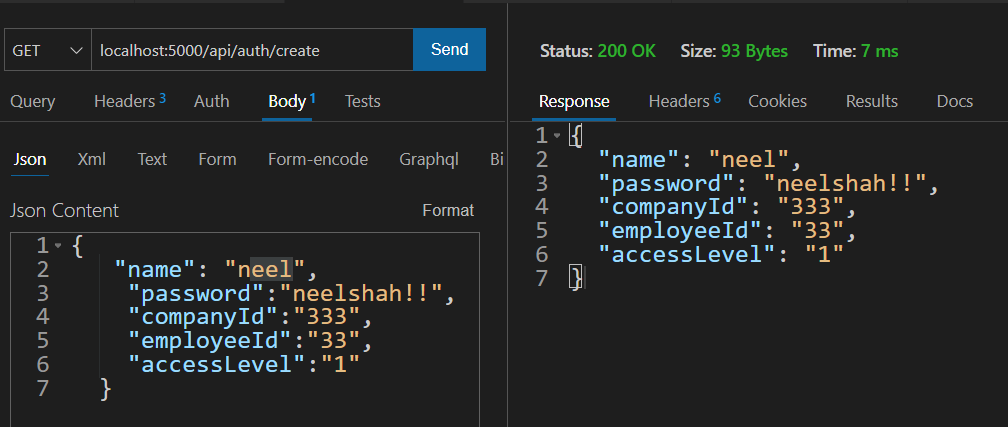
})

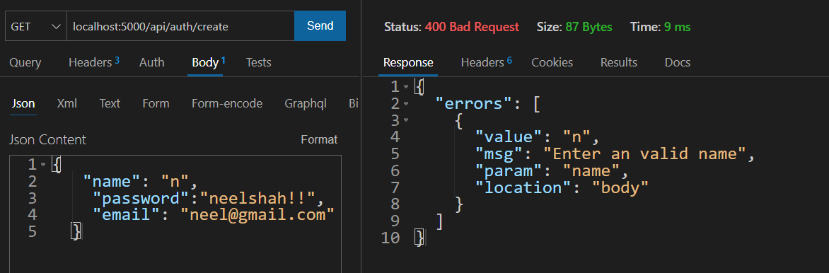
🡪**In this we done one thing different from the documentation that is we had maked an array inside the router.post/router.get and then in it we had written the validation logic**

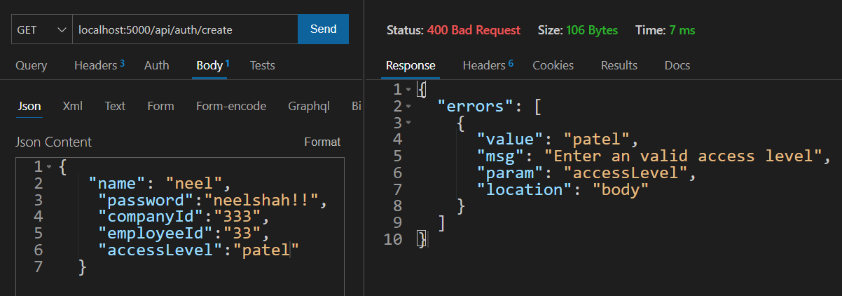
1. In the above part in the router.get/router.post in the body we had defined the name of the field, now we can also define the message to be printed if that thing is not entered according to the validator by : Example of the changes to be make are as follow :

**body(‘email’,’Enter a valid email’).isEmail()**

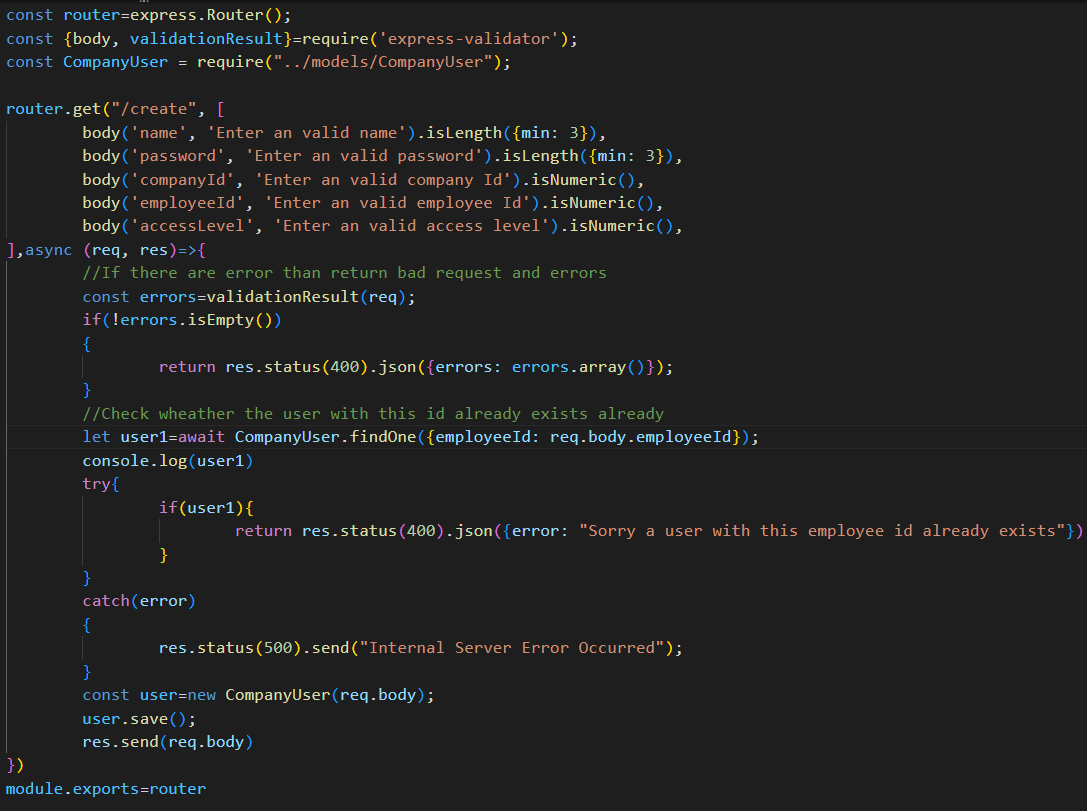
**🡪Now before going forward we will check if the program is running according to us or not by the help of the thunder client we will make an request(The given visuals are of the different program this are not exact the same but some what the same, Only for the explanation)**

****

****

****

1. **Now, we will check if the double entry of the email takes place than do not crash the server but instead give an error on the console as well as on the web** by the help of the if and try catch statement as shown below (The program shown bellow is of another project but some what same, just take the idea of the checking whether the user with this id already exists or not):



🡪We have made the following changes in the auth.js

* 1. In auth.js we changed the url in the router.post as follow :

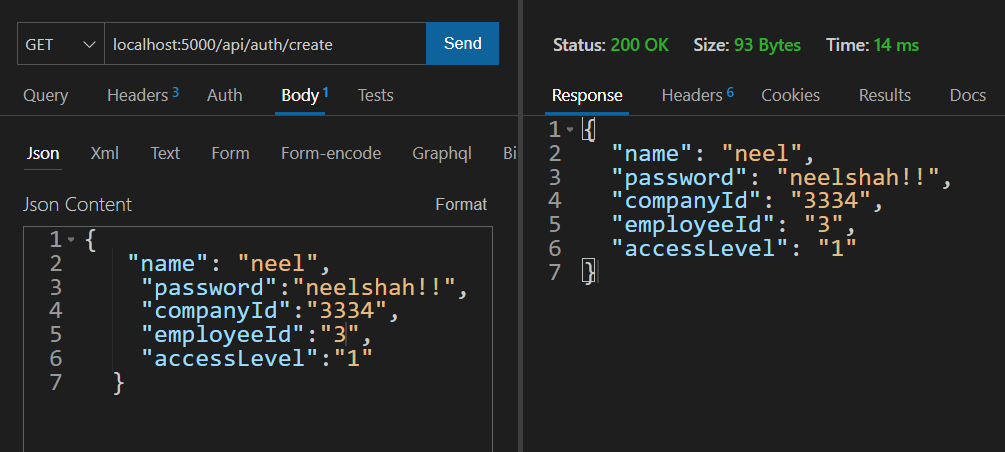
**router.post(“/”, [) 🡪 router.post(“/createuser”, [)**

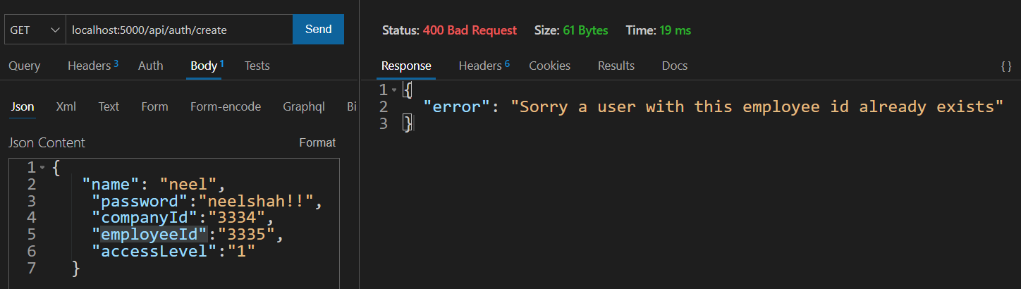
* 1. Now we will change the (req,res) method to the async method in auth.js by doing :

**async (req,res)=>{**

* 1. And we will also use the await method by the help of which we can wait until the particular statement is executed, we had used the await due to the asynchronized nature of the java script
  2. By doing this we will make sure that in our database the entry with the same email is not made(In this sample we would take care about employee id instead of the email)

🡺The sample run of the code in the thunder client is as follow :





1. Now we will make a new collection by opening the thunder client and going to **collection** and then clicking on the hamburger() and than new collection and enter the new collection name and then press enter(in this project we make the new collection by the name of the iNotebook)

1. \*Collections is generally the group of API request

🡪We make the collections as we want them all at the one place collected and saved in sequence instead of the different at the different places

1. And then **in the collection we make an folder named** Authentication and in it we will make the new request by clicking on the side button on it and will name it create new user
2. Then in it we will enter the url and will go to body and write the things in the json, Give the data like name, email, password
3. And we will make all the things in the try and catch, As if any error comes than it will show us on the console but due to the server will not crash
4. And ideally if the server crashes than we will send it to the **logger, SQS**

**🡺Doing configuration for the password safety**

1. Write **npm install bcryptjs** in the terminal of the vs code and then we are following the steps of the documentation of the bcrypt-npm in the nomjs.com and there go to the usage and then follow it (in this documentation we will follow it only)
2. The bcrypt js helps us to make the functions like salt, pepper and convet the code to hashcode and bcrypt js is an node js package
3. We will first import the bcrypt in the file in which we want to use the bcrypt and this time the file is auth.js and in it put this statement along with the other to import

**const bcrypt =require(“bcrypt”);**

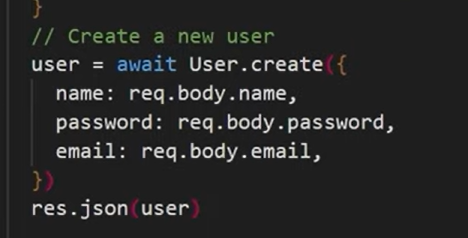
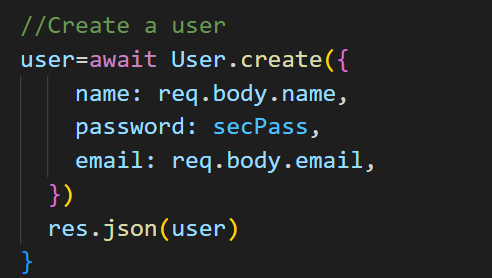
1. And then we have to make the given password by the user in the hashcode for it we have to write the following code :

const salt=await bcrypt.genSalt(10);

const secPass=await bcrypt.hash(req.body.password, salt);

🡪The above two functions or statement returns the promise that’s why we use the await but be sure the parent function used should be async

🡪And we have to also change the password written in the user while creating the user to secPass, Example

**TO**

1. And then we will check it, by going into the thunderstorm client and in it we will go to the collections and in it we will use the old request or make the new request and sent it to the database

**🡺JWT(Json Web Token)**

**Json web token are used to save the account from which user logged in last time, And by the help of it, We can make user logged in automatically without asking for the login again and again(Just as youtube and google do)**

1. When a user logins than **we will give him back a token for authentication for next login and the tokens are of the two types :**
   1. Session Token
   2. **Json Web Token**

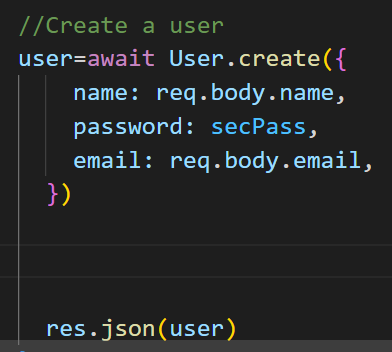
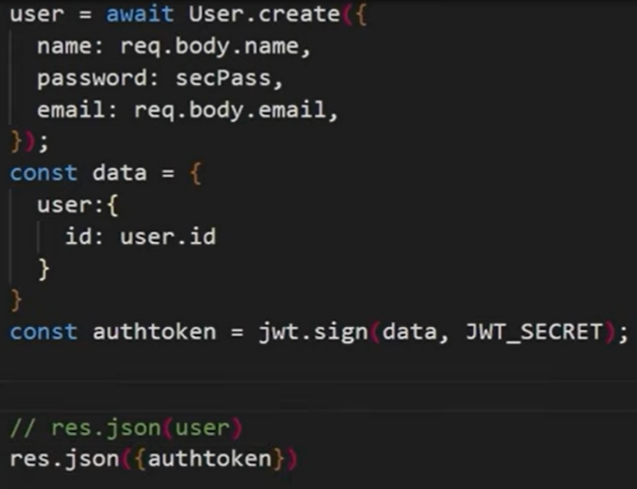
🡪And in this application we will work with the jwt(Json Web Token)

1. Now we will install the **JWT** by the help of the following code as we are following the steps of the documentation of the jwt to get it write jwt node.js🡪and then do the steps written in it or follow the bellow steps :
2. **npm install jsonwebtoken**
3. We will write the following on the starting of the file in which we have to make the token i.e. in this case auth.js
4. Now we will import the following statement on the top of the file where you want to use the application of the jasonWebToken(JWT)

**var jwt = require('jsonwebtoken');**

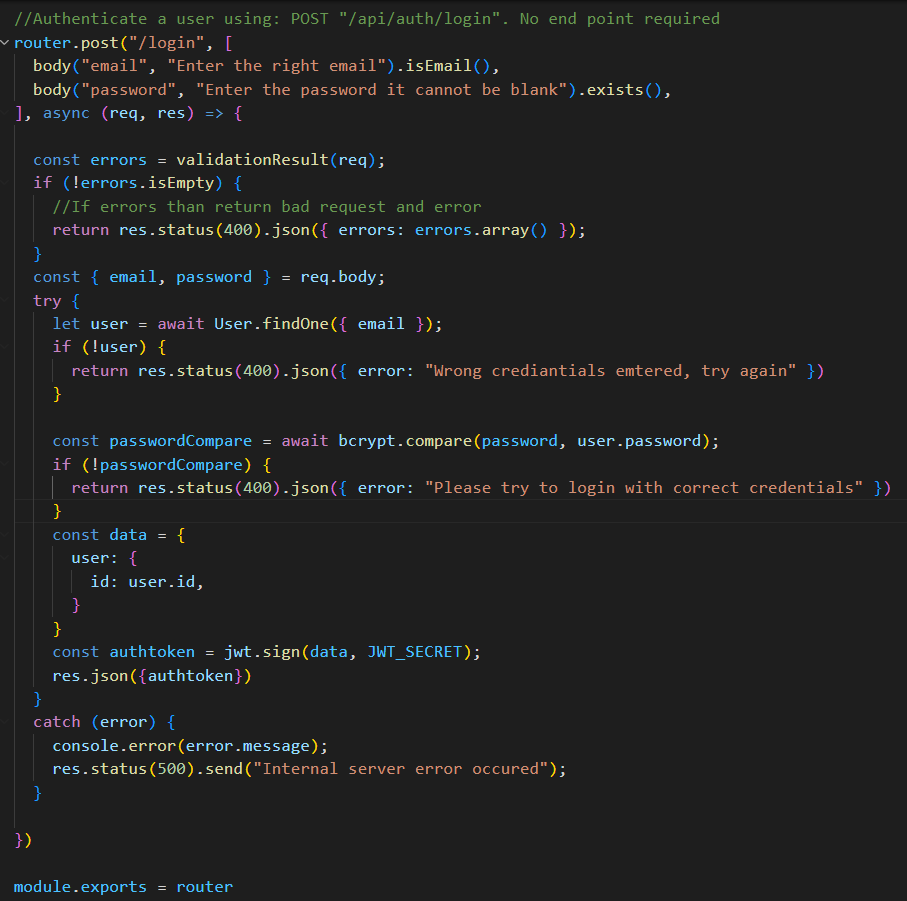
**const JWT\_SECRET=”<ANY-TEXT-HERE>”**

🡪Note that for this program we have used **“NeelIsGoodBoy”** as the jwt\_secret, But in any other program we can we any of the jwt

1. And we have to do the following changes 

TO

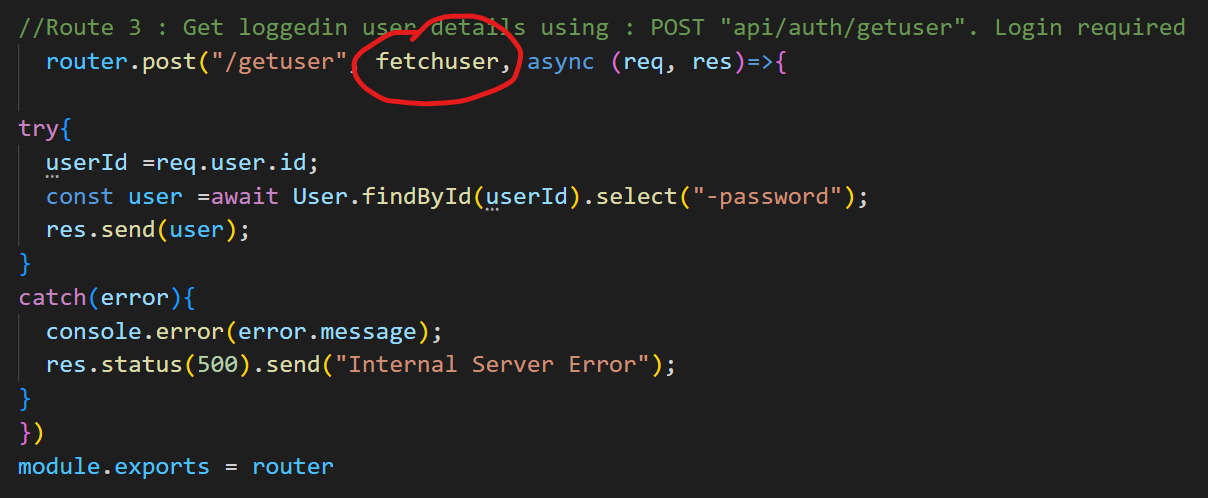
1. And then check the following code by going to the thunder client🡪collection🡪create new user
2. **Now we will make an another logic for login in the web-app in the auth.js** till now we had worked on the logic for the making of the user



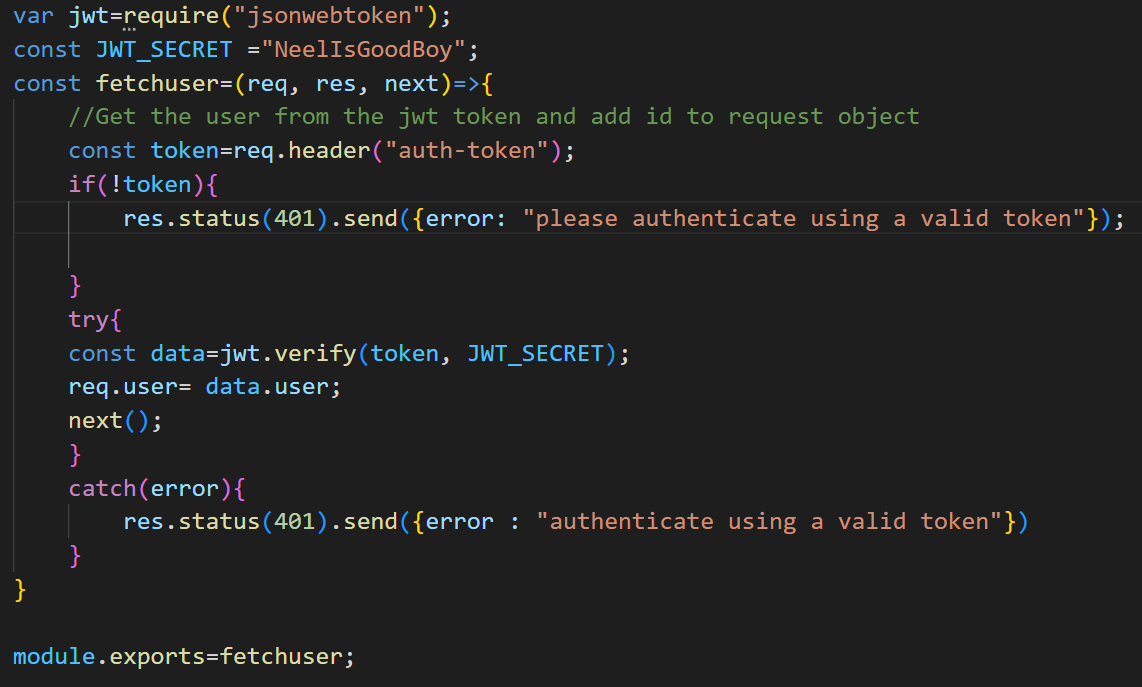
1. In node.js the **middle ware** is called when any request comes when the login is required, And for it we will store it in the new folder in the backend folder and in it we will make an file named as the fetchuser.js
2. And generally the middleware is a function which requires a request, a response and the next and the ending of the function in the middleware the **next();** should be called hence it will go to from where it had come and as in all the files we write the module.exports in this file we will also write :

**module.exports=fetchuser;**

1. And we will also add the middleware name at the place from where we want to call the middleware, example(example includes all the logic for the getuser) :



1. The example of the middle ware file is :

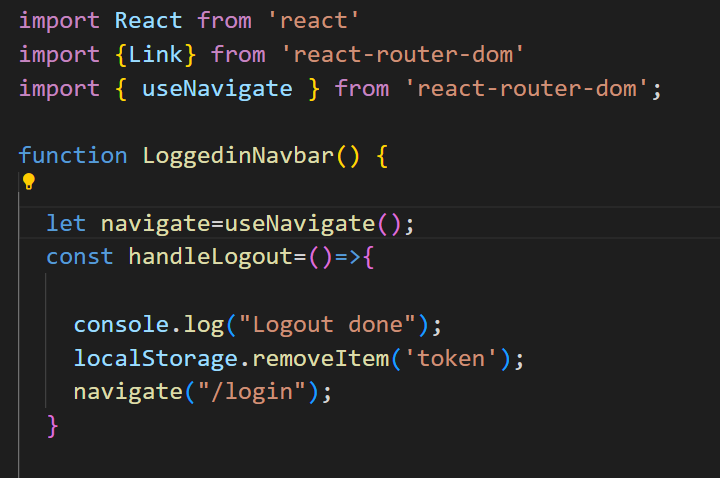


1. And then we have to send the token number to the function from the header as we had puted the req.header in the middle ware file
2. Now to give a token from the header and check it we will go to the thunder client and in it we will make a new request named get user data and init we will write the url of the function in the auth.js as the middleware is been called by an another function i.e. getuser
3. And in that request we will go to the headers and in it add the **auth-token** in the first box and in the second box we will give an **auth token** that had come from the login end-point of the API

🡪Don’t forget to add the import statement of the fetchUser where we want to use it on the top of the file with the other import statements, The sample statement is as follow but the location of the file changes as per where we have to call it

**const fetchuser=require(“../middleware/fetchuser”);**

**🡺System For logout while using the JWT in the frontend**

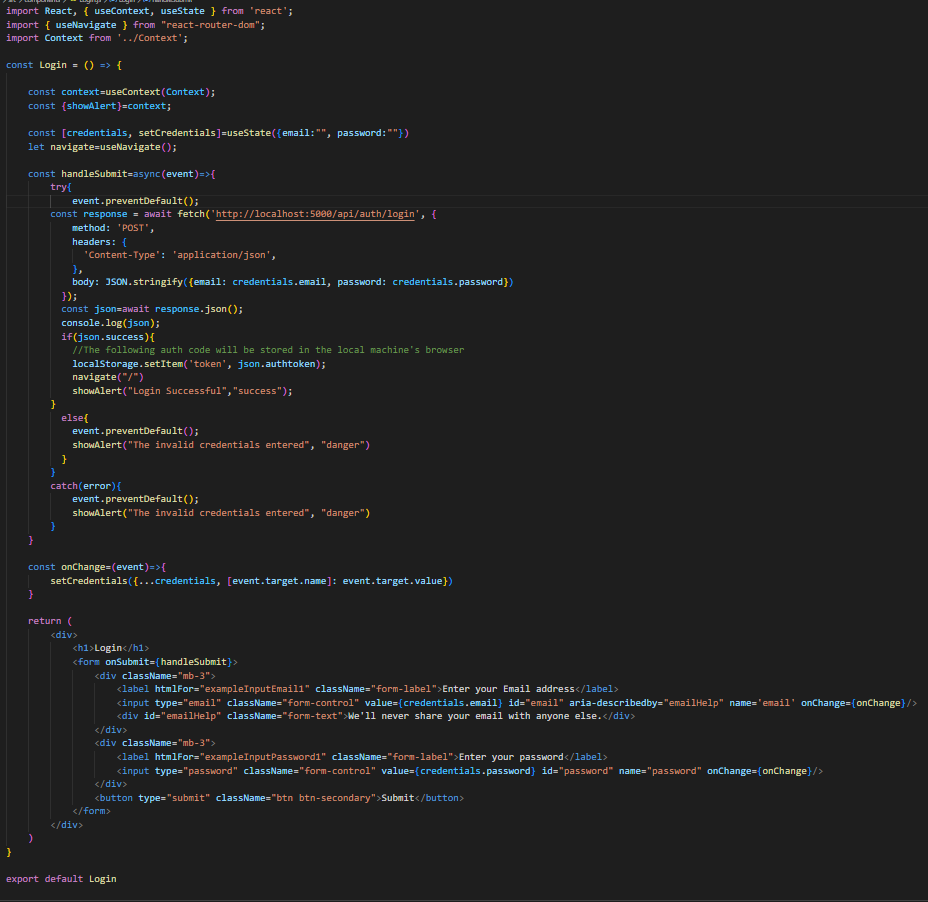
****

🡪Make sure you write the function inside the main function but before the retuen statement

🡪Add this function when the logout button is been made to be clicked and this will remove the quth-token from the local storage and the login page will prompt up

🡪And the navigate is not a function but it is a refrence of the useNavigate hook which can be called by the importing useNavigate hook as shown in the figure

🡺**System For Login in the frontend using JWT**

****

🡪Make this function work when the login’s submit button is been made to work

🡪The main part of the code is the storage of the auth token in the localStorage

🡪Make sure you change some part of the code according to your program, The address or etc can have to be made change

🡪and in the onChange function, we have used the spread operator which helps us to save the data every time we make change in them by the help of the useState hook’s credential method, where we have made the credential and setCredential method by the help of the useState hook also refer to that in the above code

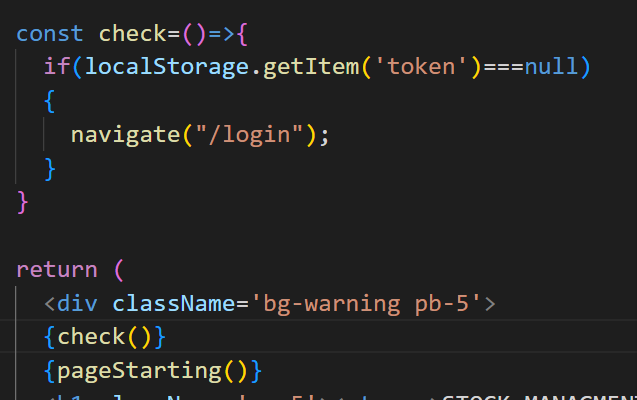
🡪Also notice the usage of the useNavigate hook, which is used to change the location of the page

**🡪In the frontend make sure that you add the onChange, value and name in the input field as shown in the above code**

🡪And if the one makes logout at any point than instead of the erorr the login page shoud be opened

🡺For checking at each page if the user is logged in or not

🡪We will use the following method as shown in the figure below to check, If user is not loggedin than transfer the user to /login page

­­

**🡺From now we will work on the notes adding, fetching, etc**

1. For working with the notes in the thunder client in the iNotebook we will make the new folder named **Notes** and in it we will make a new request named the **Fetch All Notes**
2. And this endpoint will be made in the notes.js which is in the routes and this is the first endpoint and it will help us to fetch all the notes of the user
3. And for adding the notes to the account we will need another end-point/route and it will be in the notes.js
4. And the endpoint to add the notes will be POST while the endpoint to see the notes will be GET
5. And we will also make some change in the module that we had made(i.e. Notes.js) in it we will add an object of the user which will act as the foreign key in the new object and it is defined as :

user:{

type: mongoose.Schema.Types.ObjectId,

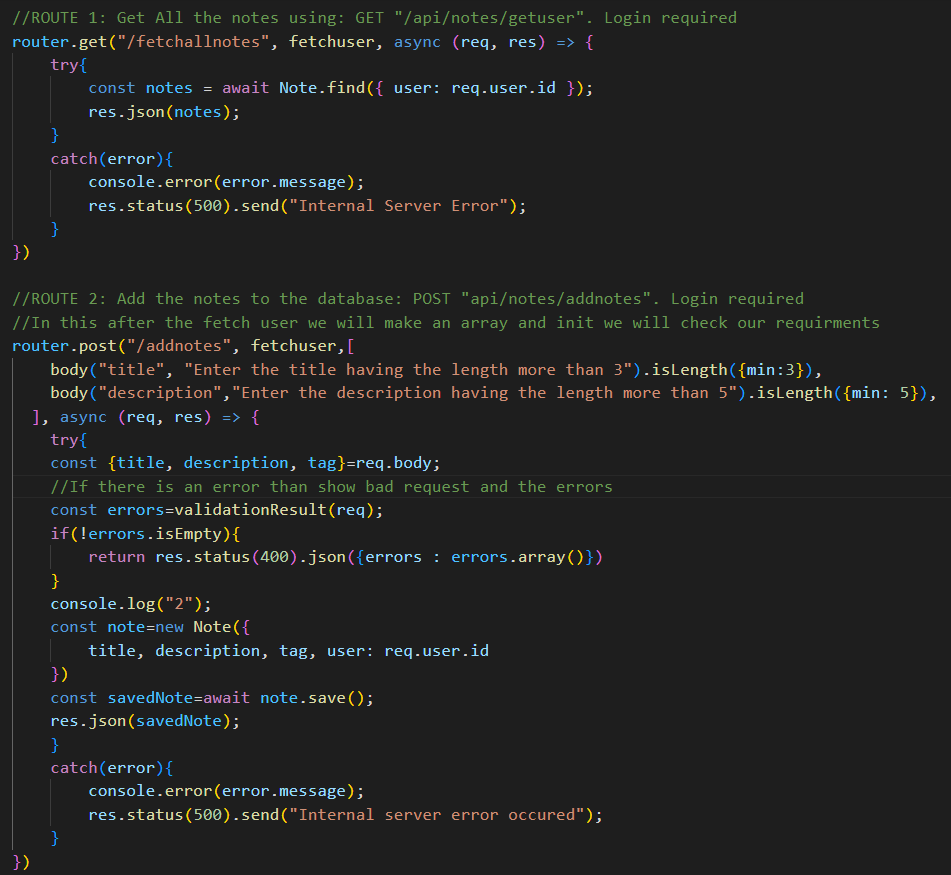
ref: “user”

},

1. And in the notes.js, we have to import the statement do not forget to import it as this statement is not automatically imported

const { body } = require("express-validator");

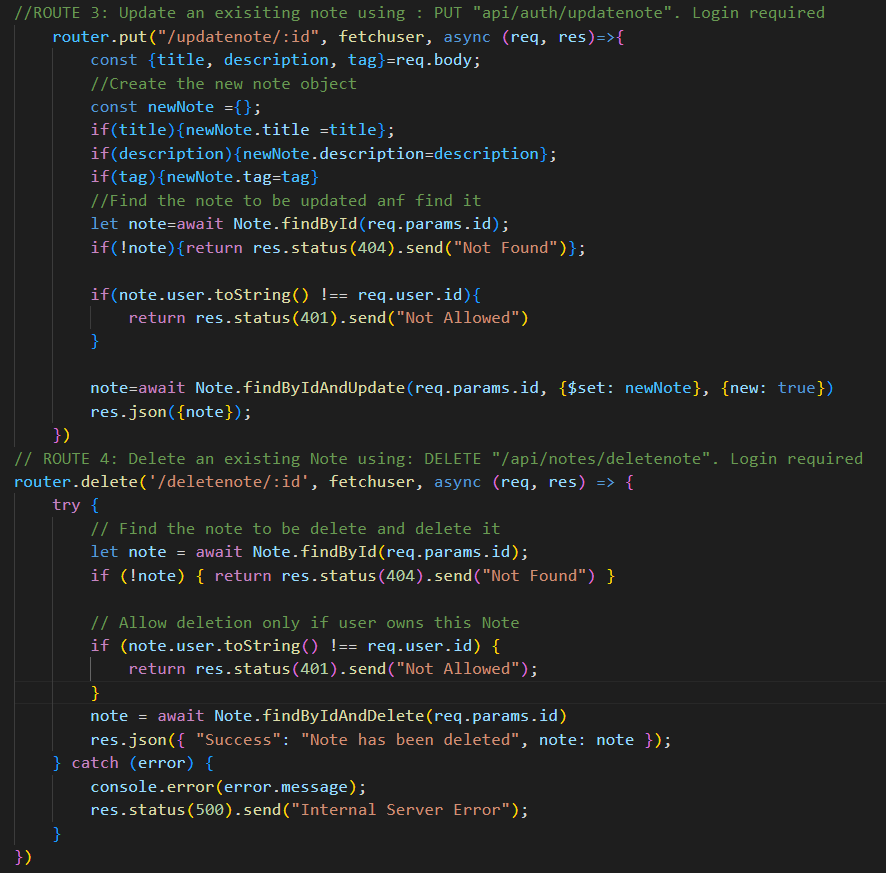
1. For using the endpoint of the add notes we will make a new request from the notes and name the new request as **Add notes,** In it’s header we will give the auth-token and also the content-type 🡪application/json, And in the json we will write the sample note and check it if it is working or not
2. And the code of the two routes are :



1. If any things comes like it is undefined than it means that we had forgotten to put the await or we had make some of the mistakes in the code
2. By the help of the different requests we can make the different works, Its ok to use the post for the all types of the request, but we can use the **get** to get data, **post** to send data, **put** to update data and **delete** to delete data, It is told to be the good practice to use specific keyword for specific thing

🡺Now we will make a new note to make the update to the notes

1. And the router three will be used for the updation and the router 4 for deletion



1. Now we have completed the work on the backend and **we will move toward the frontend and we will make it in the react app that we had maded**

**🡺Making the both server tobe start with one statment**

1. And in this react app we want to fetch the data from the api which we will deploy on the another server for it hence we will work with the multiple server and to use that we have to install the following statement in the react app(not in the backend folder)

**npm install react-router-dom concurrently**

\*\*(If the vulnerabilities come, than do not take care about it, They come and go)

🡪For it we also have to make the following changes in the outer **package.json** which is on the project folder not in the backend folder

🡪In it we have to add the following lines **under the scripts section**, originally we have to add the codes that we want to run but in this project we are running the two servers together and for it the command this is as follow :

"both": "concurrently \"npm run start\" \"nodemon backend/index.js\""

1. We can go back to the parent folder from the child folder by the use of the **../**
2. And then by the help of the **npm run both** we can run both the server’s together

**🡺CORS CONNECTION IN THE WEBSITE**

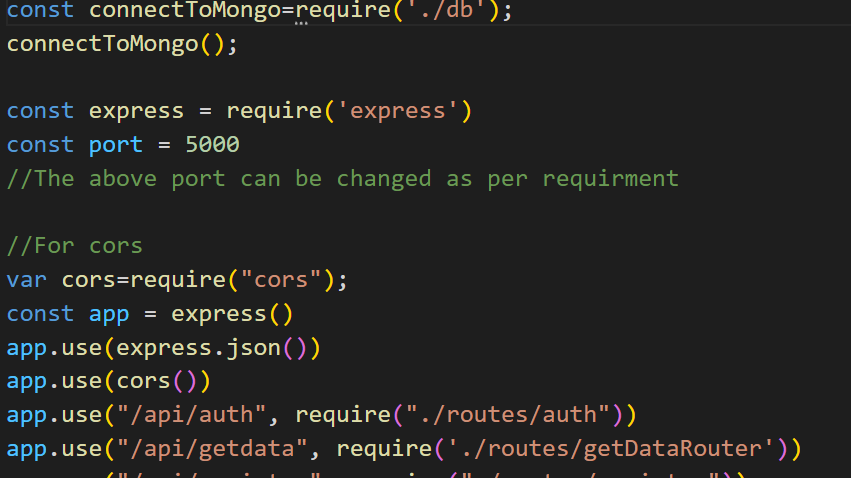
As we connect to the two servers in the website the exchange the data, So that there would be the error named the CORS and this can be solved in the node js by installing the cors in the terminal and to do this the following steps are tobe followed:

🡪For this website we will use an middle ware which is given by the express which is an framework and for more details visit the website whose link is as given :

<https://expressjs.com/en/resources/middleware/cors.html>

🡪We will follow the steps as follow :

1. Install the cors by writing the **npm install cors** in the backend side
2. Write the **var cors=require(‘cors’);** before the **const app=express();** statement as shown in the figure
3. And than before using the other app.use write the following statement first **app.use(cors());**



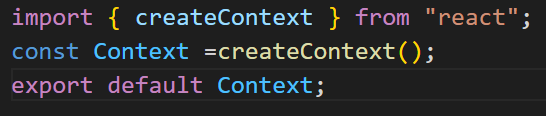
🡪Now the API is ready to be operated by the other website’s server

**🡺We can use the context for making the use of function or anything mobily in the another file than it is made in**

🡪By the help of the context we can use the function or any thing which is made in the one file into the other file

🡪For using the context, we first have to make the Context.js file in the src folder of the app in which we are working

🡪And in it we have to write the following code as shown in the image :



🡪Than we have to make the following import statement on the top of the App.js file

**import Context from ‘./Context’**

🡪Now all the setup required for the context is done, We can now use the context

🡪We have to wrap code of the router by the **Context.Provider** where we want to use the functions using the context as shown in the figue:



🡪Make sure you have change the value of the functions i.e notes, setNotes, addNotes etc and the element which you want to call as per the code you are writing, The code of the functions that you want to write is as simple as the other functions and the function is always before the return statement

🡪And the place where we want to use the functions of the context, there also make the import of the context by its respect of location, Note that the location for each program is different :

**import Context from ‘../../../../Context’**

**import {useContext} from ‘react’;**

🡪And than at the starting of the main function before the return statement add the following statement :

**const context=useContext(Context);**

**🡪**Ans than by help of this type of statement we can easily import the things from the Context and use in the our file, But make sure you import the same thing that you have added to Context, The statement for this is as follow:

****

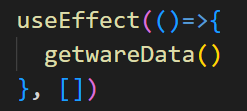
**🡺Usage Of the useEffect Hook**

🡪The useEffect hook can be used to render the anything first when the file is been loaded and this can also be done by the help of the time also for that refer the hook notes in the javascript folder, In this website we have used it to first load the data of the notes that are present, And this is to be made to use as we cannot use the function having the asynchronization directly in the function’s return method by writing it in the {}

🡪For using the useEffect hook first import it on the top of the file by writing the following statement

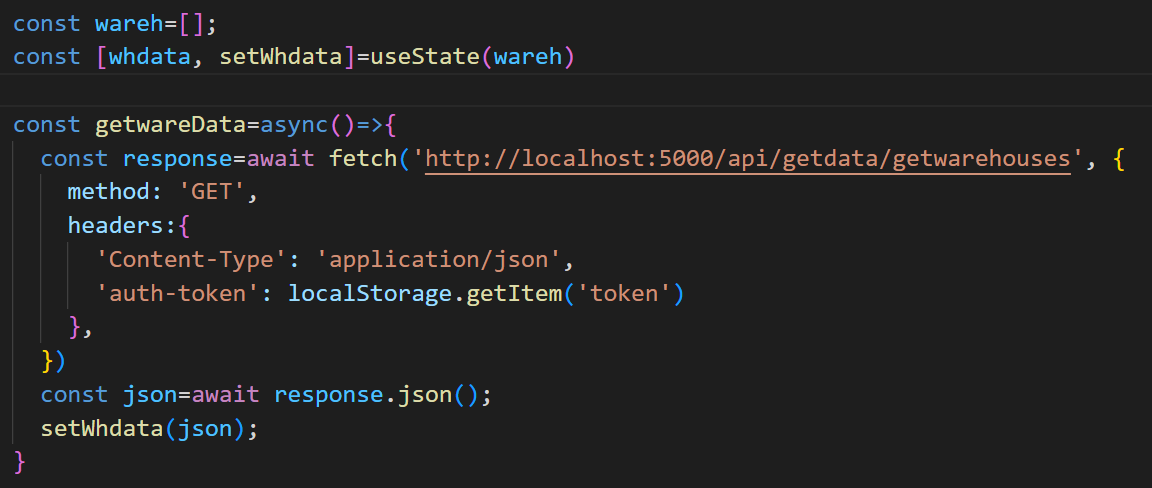
**import {useEffect} from ‘react’**

🡪And the sample usage of the useEffect hook is as follow:



🡪Here in this example getwareData() is an method for retriving the data of some thing

🡺**The Sample getData method to get the data from the API and make it to be looked in the frontend**



🡪In this method the fetch will help us to get the data from the API which is having the address as shown in the figure, And make sure you add the **http://** before copying and adding the address from the thunderClient as the thunderClient’s link does not have it and without it the module will not run and it will give us error

🡪Than we have to define the method as per the our need and that is set in the thunderClient

🡪Than comes the header which we have to also define as per the our need as that which is been added in the thunderClient

🡪And than we will save the response commed from the API in the form of json in an variable in this case the variable is json only

🡪And than comes the use of the useState hook, Which is very important hook if you don’t know about it than please refer it from the hook notes in the javaScript folder

🡪In this case by the help of the useState hook we have first make the method setWhData and by the help of which we can set the value of the whData, Initially the value of the whData is equal to the wareh which is an array and after the response comes it will be equal to the our response

🡪And than by the help of the **map** method of the js we can retrive the data from the json format till the data exists, this is the good method for the data retrivation, And inbetween we can use the html component in which format we want to add the our component, By seeing the code the concept will become more clear, the code for it is as follow :

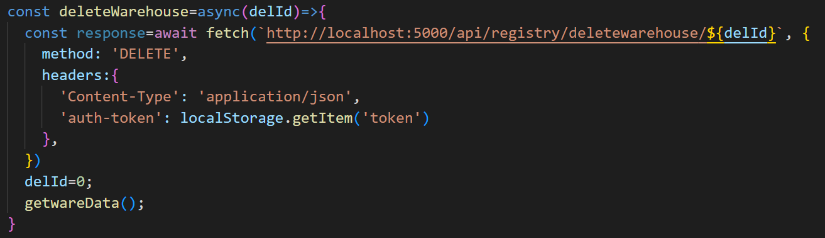


🡪Here we store the each array that comes line by line by the name of the wh and than by the help of that reference we retrieve the data stored in the json for example in this response of the json there were the wname, shopNum and etc which we can call as per our need

🡪And as the following block written in the map block is completed the same code runs for another object of the json that is stored in the line after the previous one

**🡺After changing the data, System for updation of data on the webpage automatically**

🡪We can make the update the data on the screen by adding the getData or any other method which gives the data on the page at the end of the method in which we have made change, Example code for this is as follow:



🡪And if the API call’s link want the Id of the object to do any of the work than it can be given while calling that function, Example:

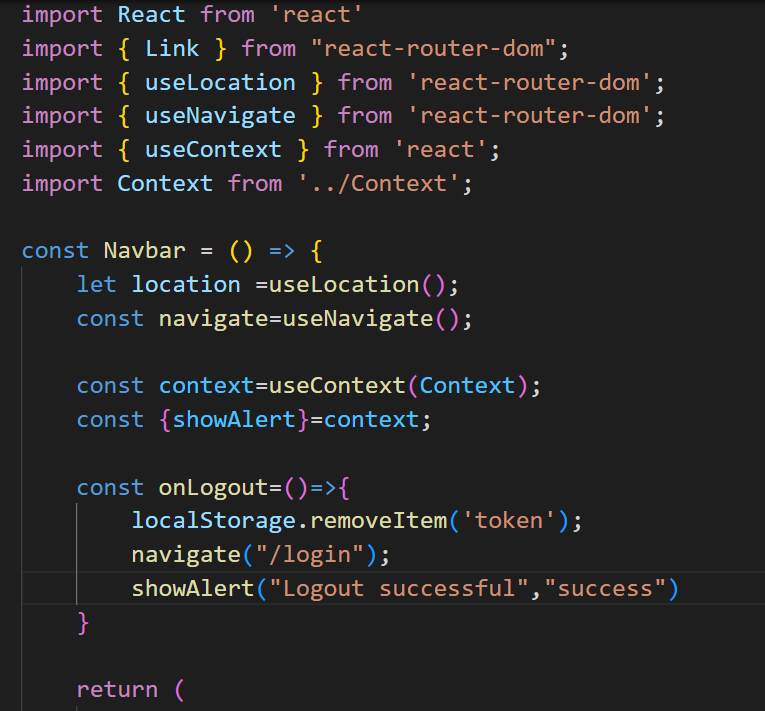
**onClick={()=>{deleteWarehouse(wh.\_Id)}} OR onClick={()=>{ delId=wh.\_Id}}**

🡪Here the wh is the refrence which we had created while mapping and this statement is been taken from the mapping code which have a button

🡪The difference between the first and second statement is that the first directly calls the delete function, While in the second statement we first store the id and than use a modal to conform and than from the conform button of that modal we call the delete function by passing the variable in the argument

**🡺We can make the link on the navbar to become highlighted** when the one is on the that portal by the help of the **active** keyword in the link, Which is as shown below :

🡪And for this we will use the useLocation hook which is to be first imported from the react-router-dom and than an refrence is been made of the it in the main function before the retuen statement which is as shown below in the image :



🡪Here we have used the refrence navigate as the refrence of the useNavigate() hook, and we can use it as shown in the figure, We can use any of the address to where we want to go instead of the /login

---TO be continued