**iNotebook Backend**

**🡺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 used as it 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 :

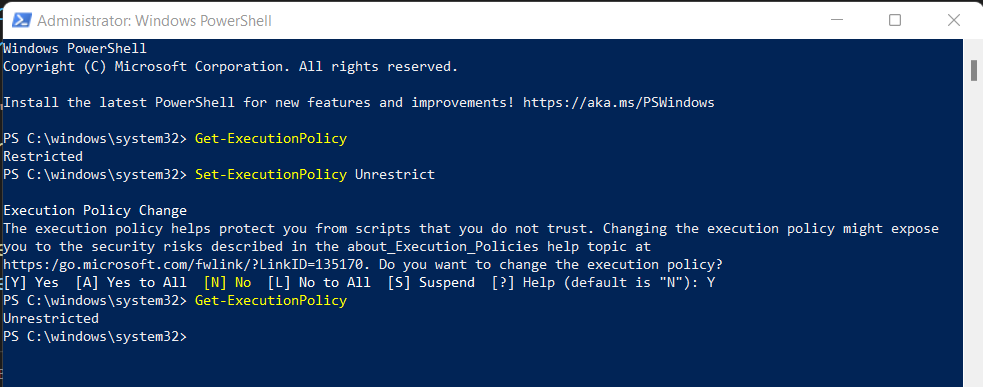
**npm install -D nodemon**  (see without writing if the output is coming or not)

🡪Wait till install and then write the following statement :

**nodemon .\index.js**

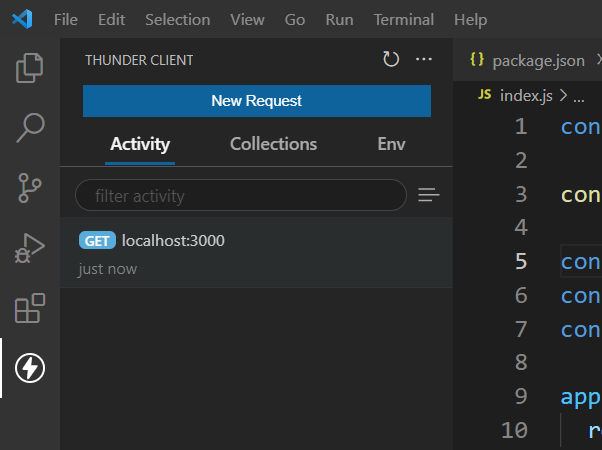
🡪And in the output we will see the **Connected to mongo Successfully** which we had written in the db.js

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

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>);**

🡪Here we have to change the user instead of whih 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 :

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

**const router =express.Router();**

**router.get(“/”, (req,res)=>{**

**obj={**

**a: “thios”,**

**number: 34,**

**}**

**res.json(obj);**

**})**

**module.exports=router**

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

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

**🡪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 make the following thing on the top of the auth.js

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

🡪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

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

**user.save();**

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

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

**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 localhost:3000/api/auth and in it go to the body, And choose the json and write the data you want to send example :

{

“name”: “neel”

“email”: [xyz@ggg.com](mailto:xyz@ggg.com)

“password”: “byhbh”

}

1. 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

🡺Adding the data validation using express-validator

1. 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) 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()**

1. Now we will also change the method of creating the user and we will copy it from the documentation and the example of it is as follow :

User.create({

        name: req.body.name,

        password: req.body.password,

        email: req.body.email,

      }).then(user => res.json(user))

      .catch(err => {console.log(err)

      res.json({error: "please enter a unique valuefor email", message: err.message})});

🡪The following hightlited part is being add extra other than the documentation and this part makes sure that 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

🡪And some required changes that were done were name, password, email were added intead of the initial values

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

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

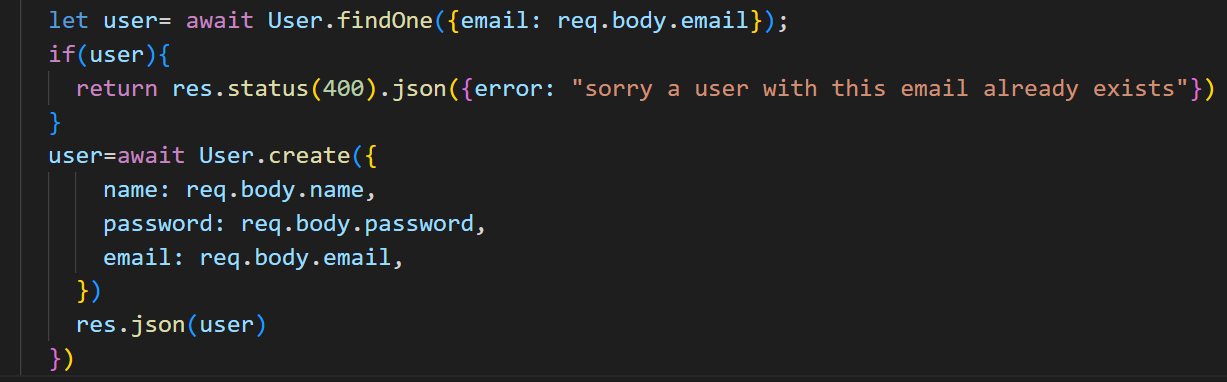
1. Now we are making an logic by which no duplicate email can be excepted from the user that already exists
2. Now we will change the (req,res) method to the async method in auth.js by doing :

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

1. Now we will change the following in the auth.js



**TO**

****

🡪By doing this we will make sure that in our databse the entry with the same email is not made

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 enter(we entered the inotebook)
2. Collections is generally the group of API request
3. 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
4. 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
5. 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
6. 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”);**

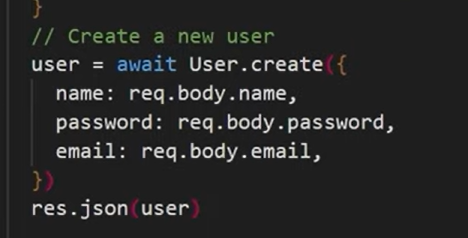
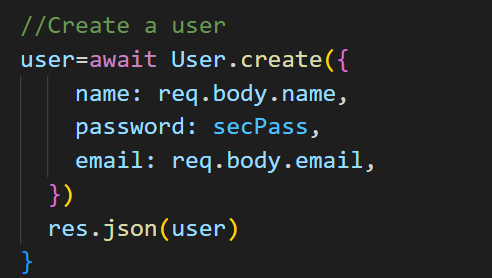
1. And then we have to make the given password by the user in the hashcode for it we have to write the following code :

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

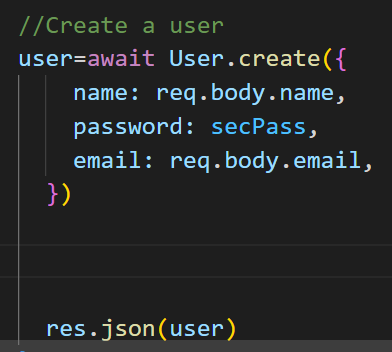
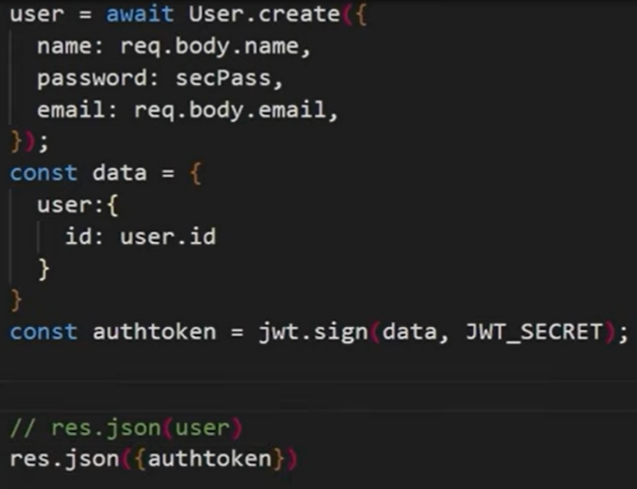
**TO**

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
2. 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)

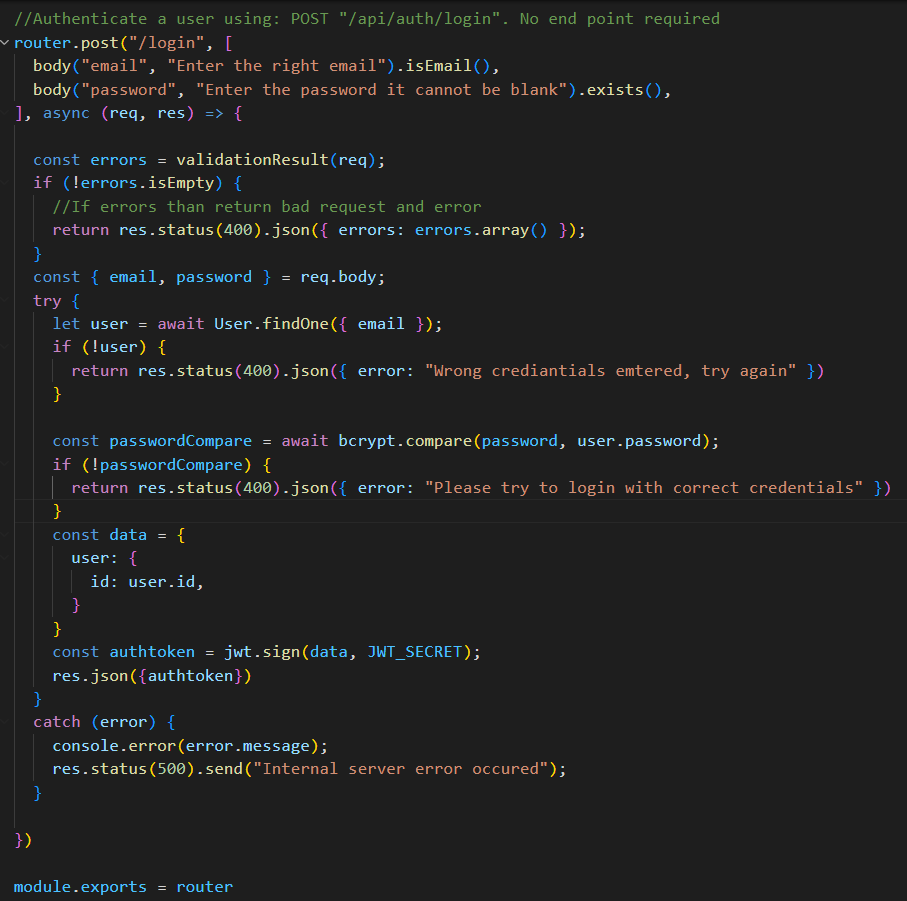
1. Now we will install the jwt by the help of the following code ans we are following the steps of the documentation of the jwt to get it write jwt node.js🡪and then do the steps written in it or follow the bellow steps :
2. **npm install jsonwebtoken**
3. We will write the following on the starting of 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

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

1. And we have to do the following changes 

TO

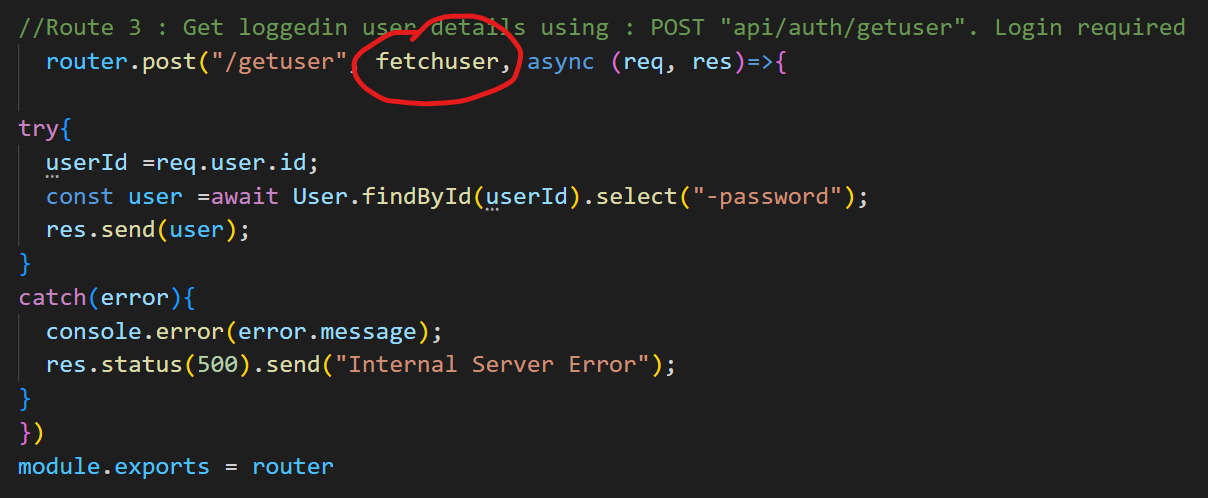
1. And then check the following code by going to the thunder client🡪collection🡪create new user
2. Now we will make an another logic for login in the web-app in the auth.js till now we had worked on the logic for the making of the user



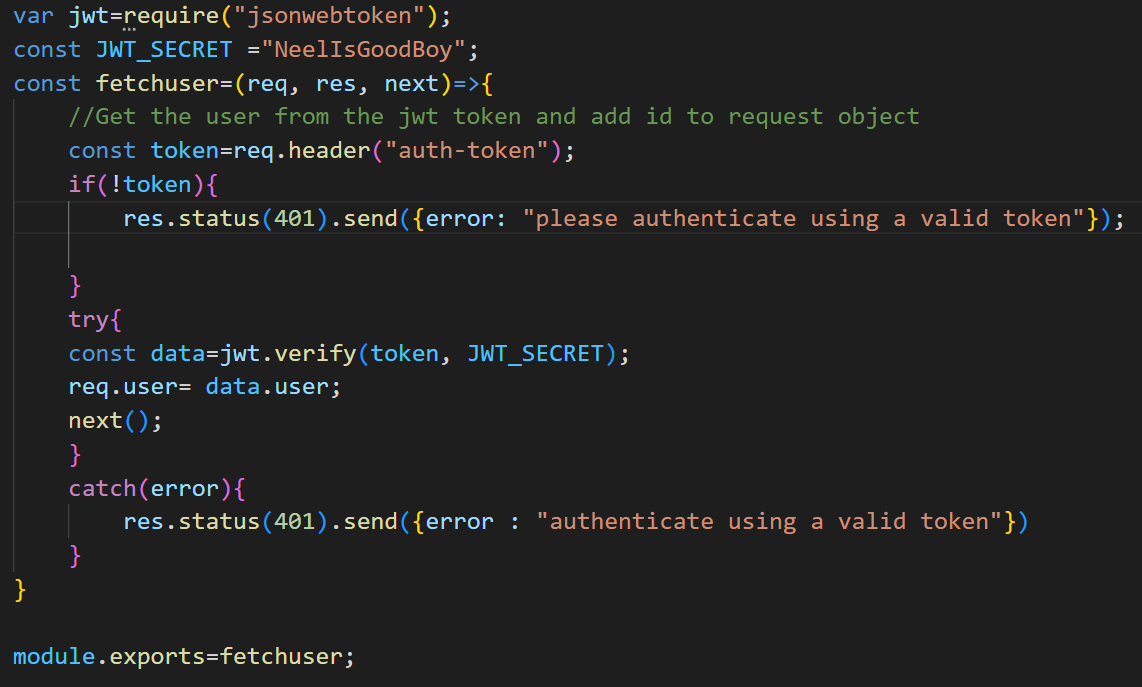
1. In node.js the **middle ware** is called when any request comes when the login is required, And for it we will store it in the new folder in the backend folder and in it we will make an file named as the fetchuser.js
2. And generally the middle ware is an function which requires an request, response and the next and the ending of the function in the middle ware the **next();** should be called hence it will go to from where it had comed 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 place from where we want to call the middle ware, 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 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 second box we will give an **auth token** that had comed from the login end-point of the api

🡺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,

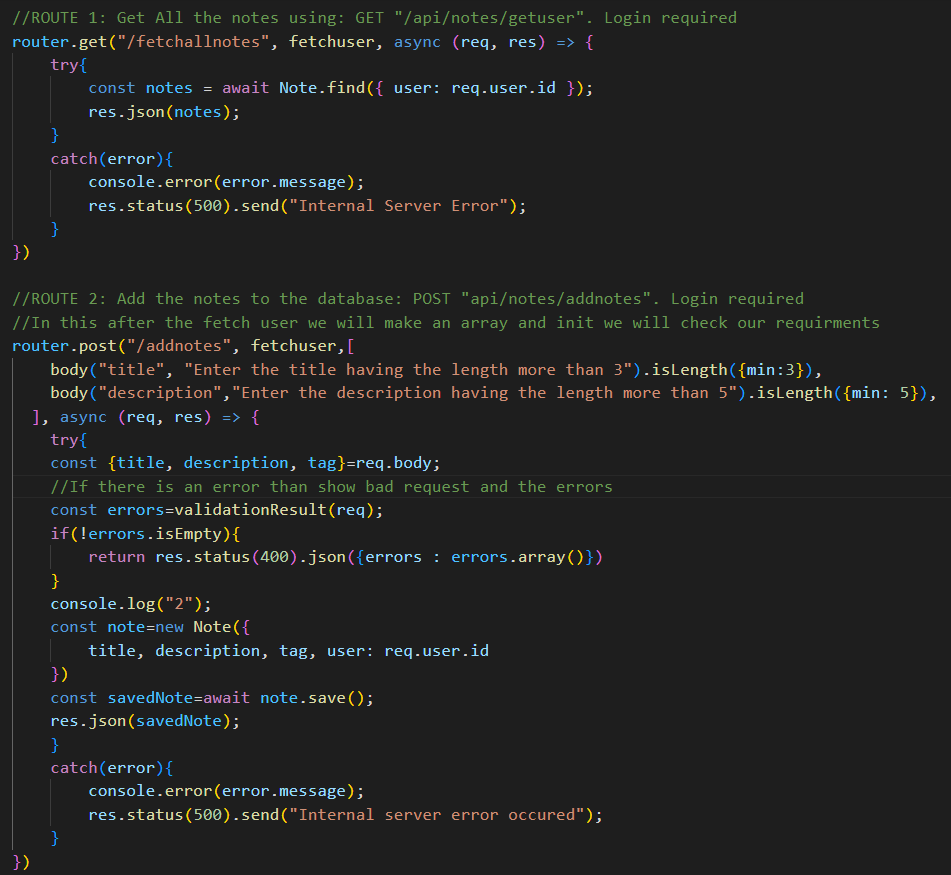
ref: “user”

},

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");

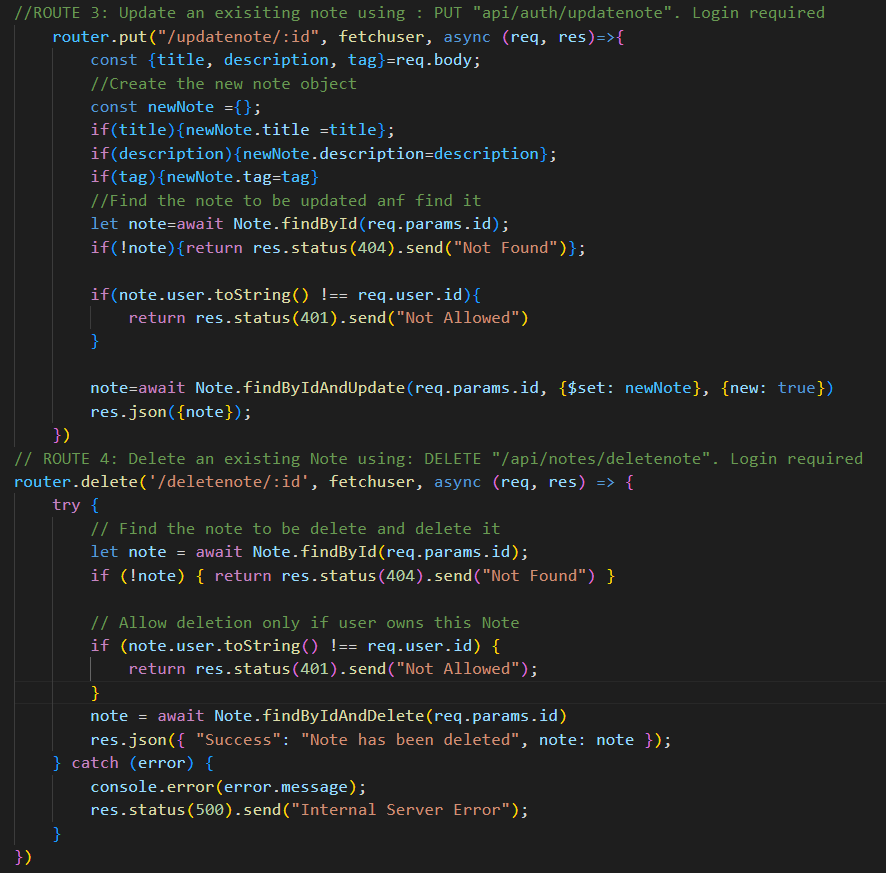
1. For using the endpoint of the add notes we will make a new request from the notes and name the new request as **Add notes,** In it’s header we will give the auth-token and also the content-type 🡪application/json, And in the json we will write the sample note and check it if it is working or not
2.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

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

🡪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 have to run but in this project we are running the two servers together and for it the command is

"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