

# **Creating a RESTful API using express.js and creating a database and index in MongoDB.**

**Name : Julakanti Harshavardhan Reddy**

**E-mail : 208x1a4250@khitguntur.ac.in**

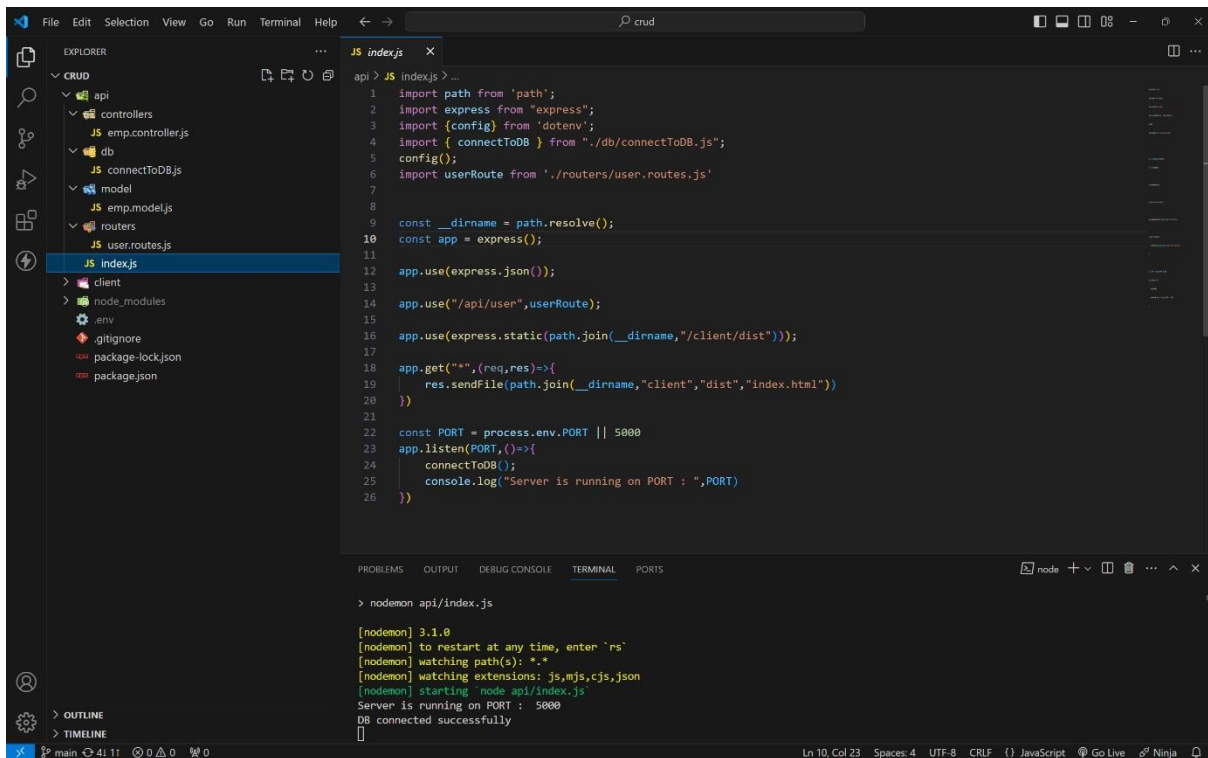
**Phone No : 9392683234**

**Roll No : 208x1a4250(AI&ML)**

**College Name: Kallam Haranadhareddy Institute Of Technology**

# source code :

## index.js file :

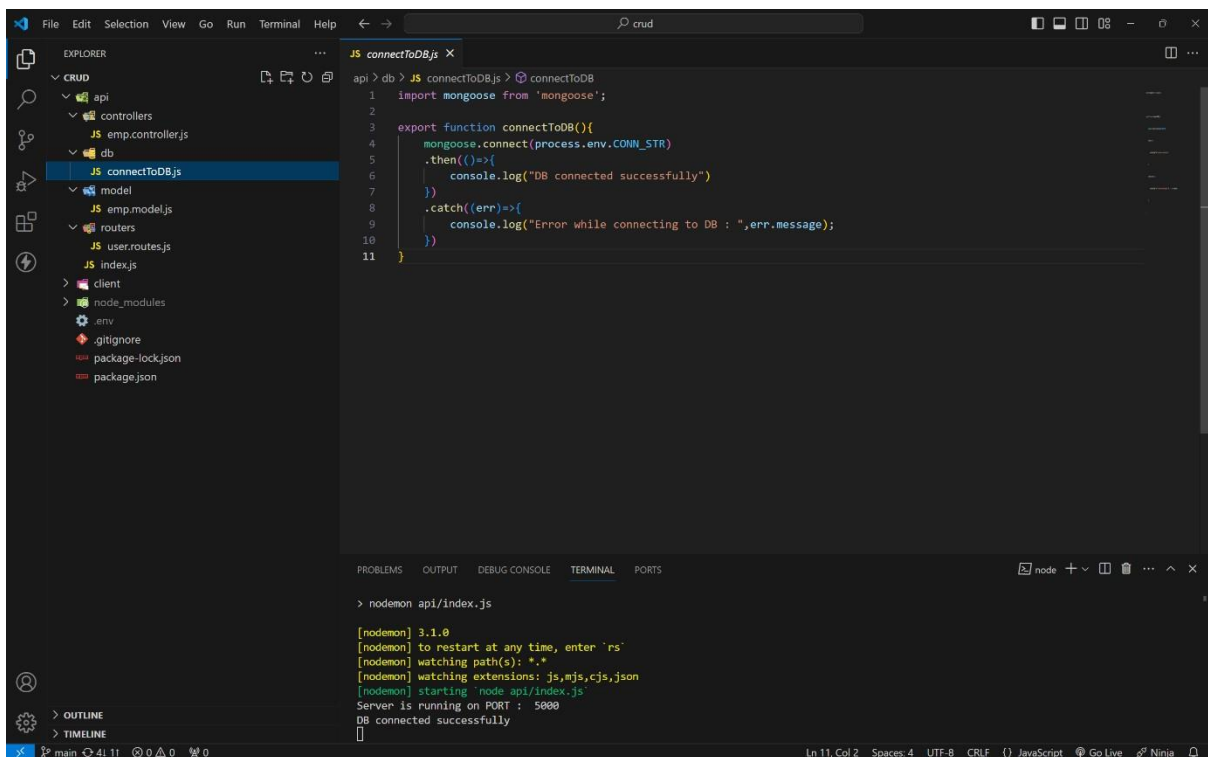


The screenshot shows the VS Code editor with the `index.js` file open. The file is located in the `api` directory. The code imports `path`, `express`, `config` (from `dotenv`), `connectToDB` (from `./db/connectToDB.js`), and `userRoute` (from `./routes/user.routes.js`). It sets up an Express app, uses `express.json()`, and serves static files from `./client/dist`. It also has a GET route for `/*` that serves `index.html` from the `client/dist` directory. The app listens on `PORT` (default 5000) and logs the server status.

```
1 import path from 'path';
2 import express from 'express';
3 import {config} from 'dotenv';
4 import { connectToDB } from './db/connectToDB.js';
5 config();
6 import userRoute from './routes/user.routes.js'
7
8
9 const __dirname = path.resolve();
10 const app = express();
11
12 app.use(express.json());
13
14 app.use("/api/user",userRoute);
15
16 app.use(express.static(path.join(__dirname,"/client/dist")));
17
18 app.get("/*",(req,res)=>{
19   res.sendFile(path.join(__dirname,"client","dist","index.html"))
20 })
21
22 const PORT = process.env.PORT || 5000
23 app.listen(PORT,()=>{
24   connectToDB();
25   console.log("Server is running on PORT : ",PORT)
26 })
```

The terminal output shows the command `nodemon api/index.js` being executed. The output includes the `nodemon` version (3.1.0), instructions to restart with `rs`, the paths being watched, the extensions being watched, and the message "Server is running on PORT : 5000".

## MONGODB CONNECTION :

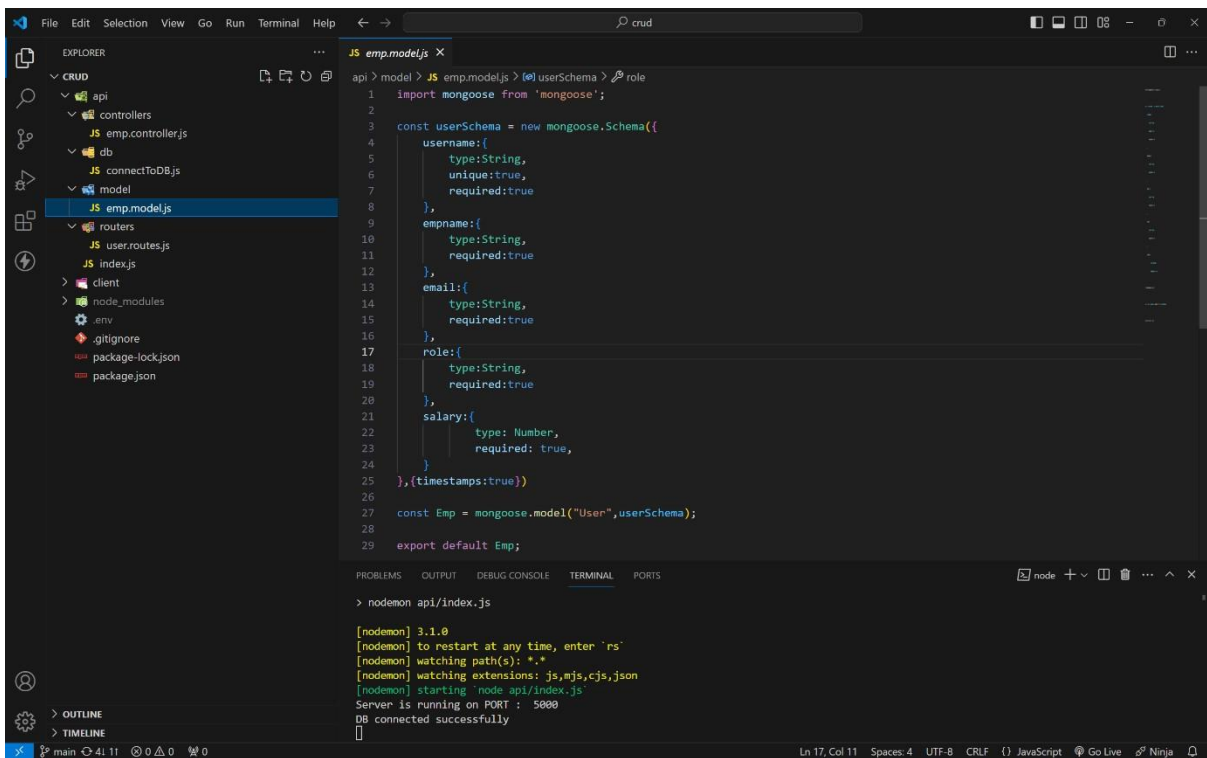


The screenshot shows the VS Code editor with the `connectToDB.js` file open. The file is located in the `db` directory. The code imports `mongoose` from `'mongoose'` and exports a `connectToDB` function. The function uses `mongoose.connect` with `process.env.CONN_STR` and logs the connection status. It also catches any errors and logs them.

```
1 import mongoose from 'mongoose';
2
3 export function connectToDB(){
4   mongoose.connect(process.env.CONN_STR)
5   .then(()=>{
6     console.log("DB connected successfully")
7   })
8   .catch((err)=>{
9     console.log("Error while connecting to DB : ",err.message);
10  })
11 }
```

The terminal output shows the command `nodemon api/index.js` being executed. The output includes the `nodemon` version (3.1.0), instructions to restart with `rs`, the paths being watched, the extensions being watched, and the message "Server is running on PORT : 5000".

## MODEL :



The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows a project structure with a 'crud' folder containing 'api', 'controllers', 'db', 'model', 'routers', and 'index.js'. The 'model' folder is expanded, and 'emp.model.js' is selected. The editor shows the following code:

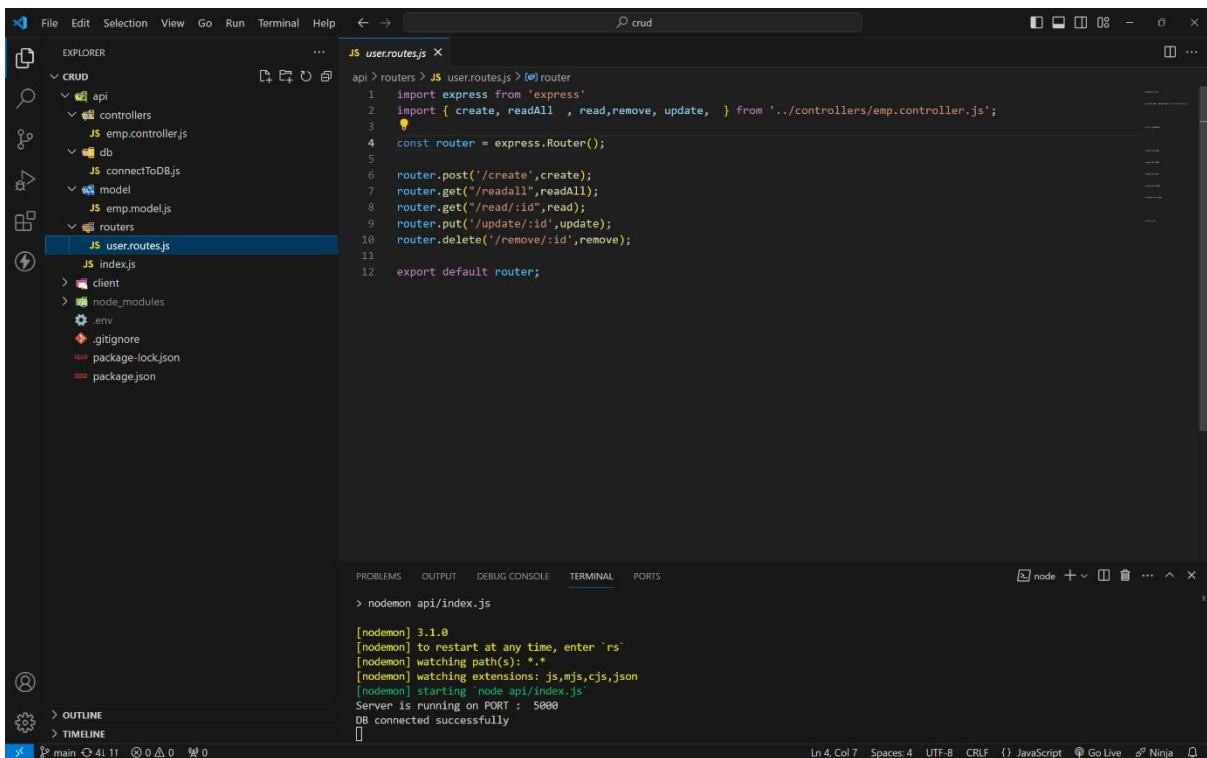
```
1  api > model > JS emp.model.js > (0) userSchema > / role
2  import mongoose from 'mongoose';
3
4  const userSchema = new mongoose.Schema({
5    username: {
6      type: String,
7      unique: true,
8      required: true
9    },
10    empname: {
11      type: String,
12      required: true
13    },
14    email: {
15      type: String,
16      required: true
17    },
18    role: {
19      type: String,
20      required: true
21    },
22    salary: {
23      type: Number,
24      required: true,
25    },
26    timestamps: true
27  });
28
29 const Emp = mongoose.model("User", userSchema);
30
31 export default Emp;
```

The terminal at the bottom shows the command 'nodemon api/index.js' and the output:

```
> nodemon api/index.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node api/index.js'
Server is running on PORT : 5000
DB connected successfully
```

## ROUTES:



The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows a project structure with a 'crud' folder containing 'api', 'controllers', 'db', 'model', 'routers', and 'index.js'. The 'routers' folder is expanded, and 'user.routes.js' is selected. The editor shows the following code:

```
1  api > routers > JS user.routes.js > (0) router
2  import express from 'express'
3  import { create, readAll, read, remove, update, } from '../controllers/emp.controller.js';
4
5  const router = express.Router();
6
7  router.post('/create', create);
8  router.get('/readall', readAll);
9  router.get('/read/:id', read);
10 router.put('/update/:id', update);
11 router.delete('/remove/:id', remove);
12
13 export default router;
```

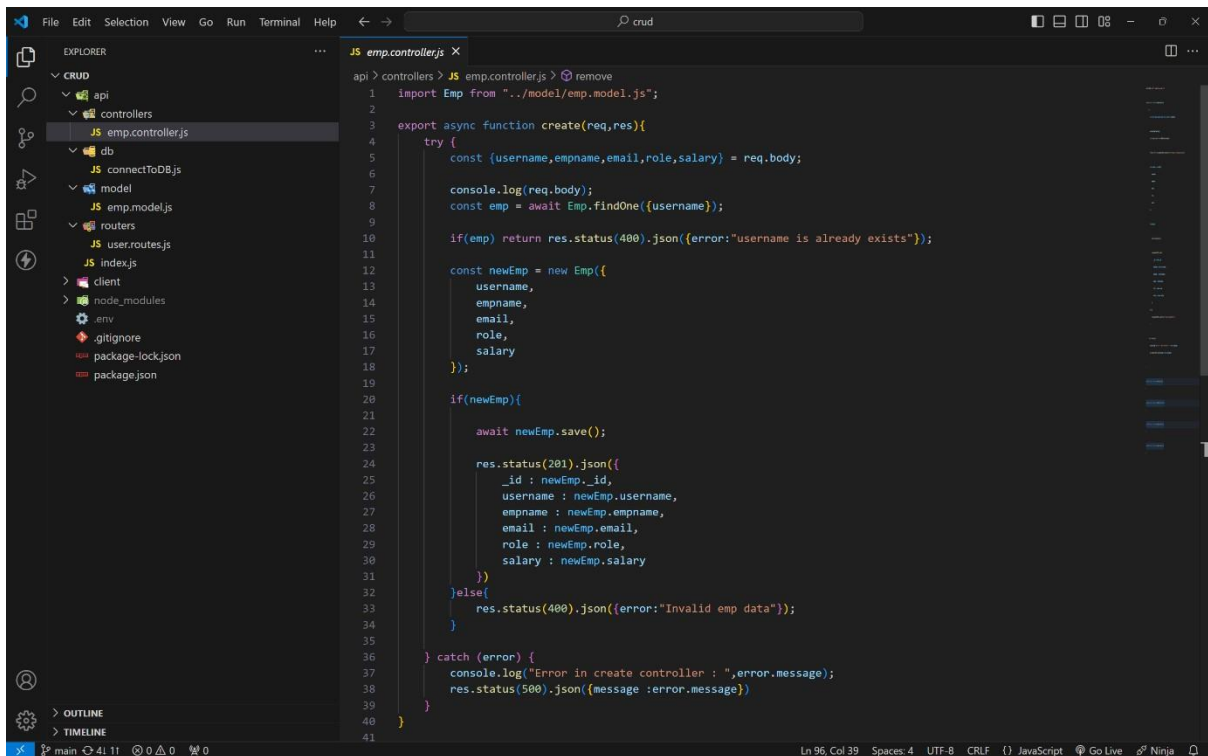
The terminal at the bottom shows the command 'nodemon api/index.js' and the output:

```
> nodemon api/index.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node api/index.js'
Server is running on PORT : 5000
DB connected successfully
```

# CONTROLLERS :

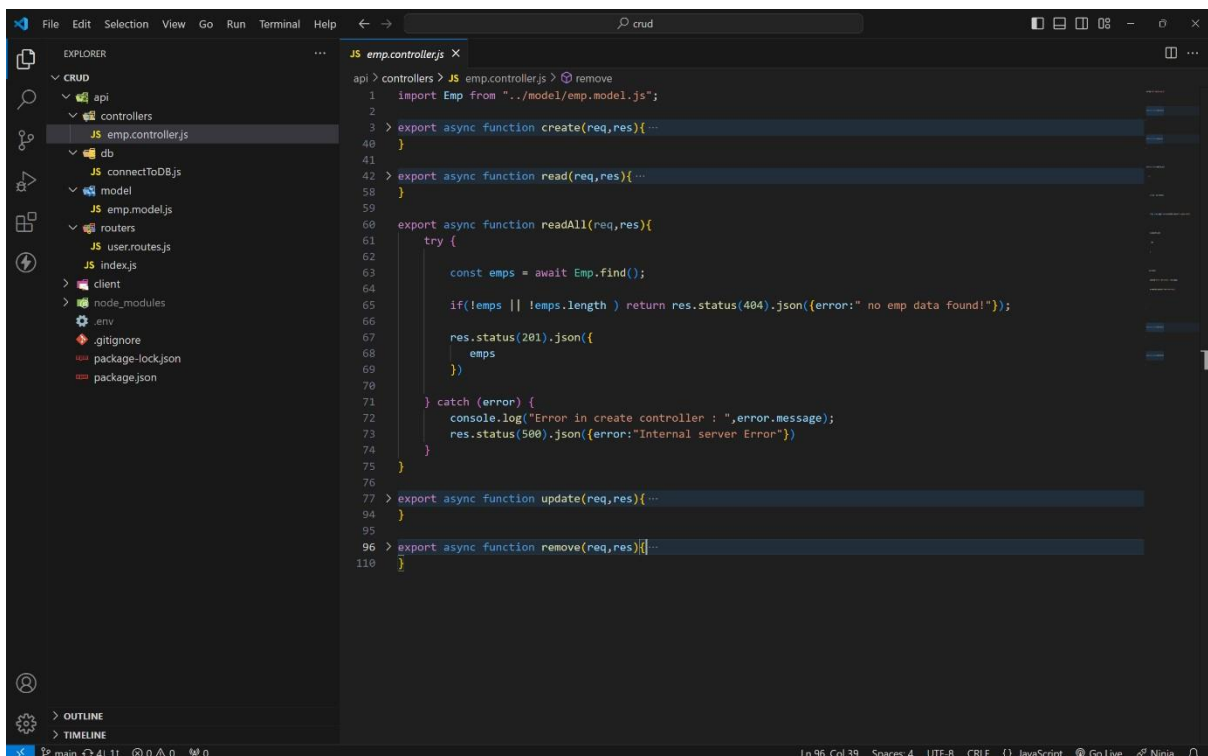
## CREATE :



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows a project structure with folders like 'api', 'controllers', 'db', 'model', 'routers', and 'client'. The code editor displays the 'emp.controller.js' file, which contains the 'create' function. The function imports 'Emp' from './model/emp.model.js' and defines an async function 'create(req, res)'. It uses 'req.body' to get employee data, checks if the employee already exists, and if not, creates a new employee and saves it. It returns the created employee as JSON with a 201 status code. Error handling is included for database errors and invalid data.

```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  export async function create(req, res){
4      try {
5          const {username, empname, email, role, salary} = req.body;
6
7          console.log(req.body);
8          const emp = await Emp.findOne({username});
9
10         if(emp) return res.status(400).json({error:"username is already exists"});
11
12         const newEmp = new Emp({
13             username,
14             empname,
15             email,
16             role,
17             salary
18         });
19
20         if(newEmp){
21             await newEmp.save();
22
23             res.status(201).json({
24                 _id : newEmp._id,
25                 username : newEmp.username,
26                 empname : newEmp.empname,
27                 email : newEmp.email,
28                 role : newEmp.role,
29                 salary : newEmp.salary
30             });
31         }else{
32             res.status(400).json({error:"Invalid emp data"});
33         }
34     } catch (error) {
35         console.log("Error in create controller : ", error.message);
36         res.status(500).json({message : error.message});
37     }
38 }
39
40
41
```

## READALL:

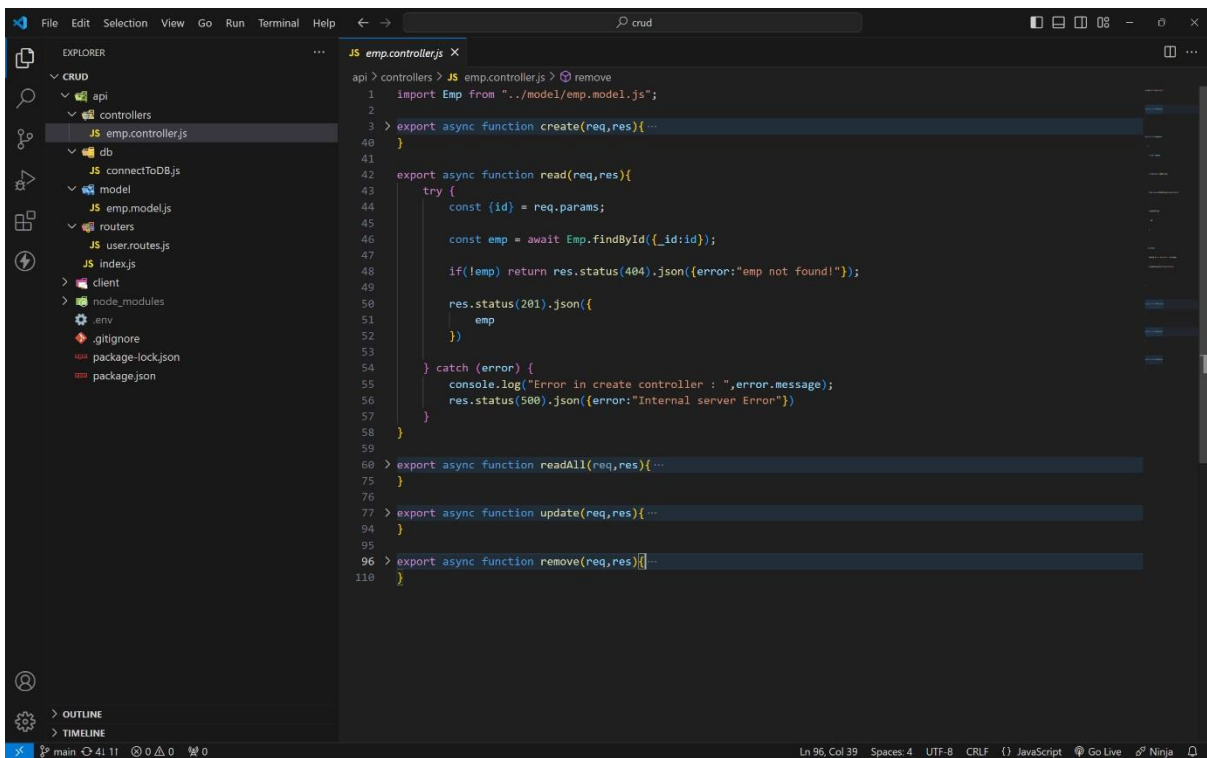


The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The code editor displays the 'emp.controller.js' file, which contains the 'readAll' function. The function imports 'Emp' from './model/emp.model.js' and defines an async function 'readAll(req, res)'. It uses 'Emp.find()' to get all employees. If no employees are found, it returns a 404 status code. Otherwise, it returns the list of employees as JSON with a 201 status code. Error handling is included for database errors and internal server errors.

```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  > export async function create(req, res){...
40 }
41
42 > export async function read(req, res){...
58 }
59
60 export async function readAll(req, res){
61     try {
62         const emps = await Emp.find();
63
64         if(!emps || !emps.length ) return res.status(404).json({error:" no emp data found!"});
65
66         res.status(201).json({
67             emps
68         });
69     } catch (error) {
70         console.log("Error in create controller : ", error.message);
71         res.status(500).json({error:"Internal server Error"});
72     }
73 }
74
75
76 > export async function update(req, res){...
94 }
95
96 > export async function remove(req, res){...
110 }

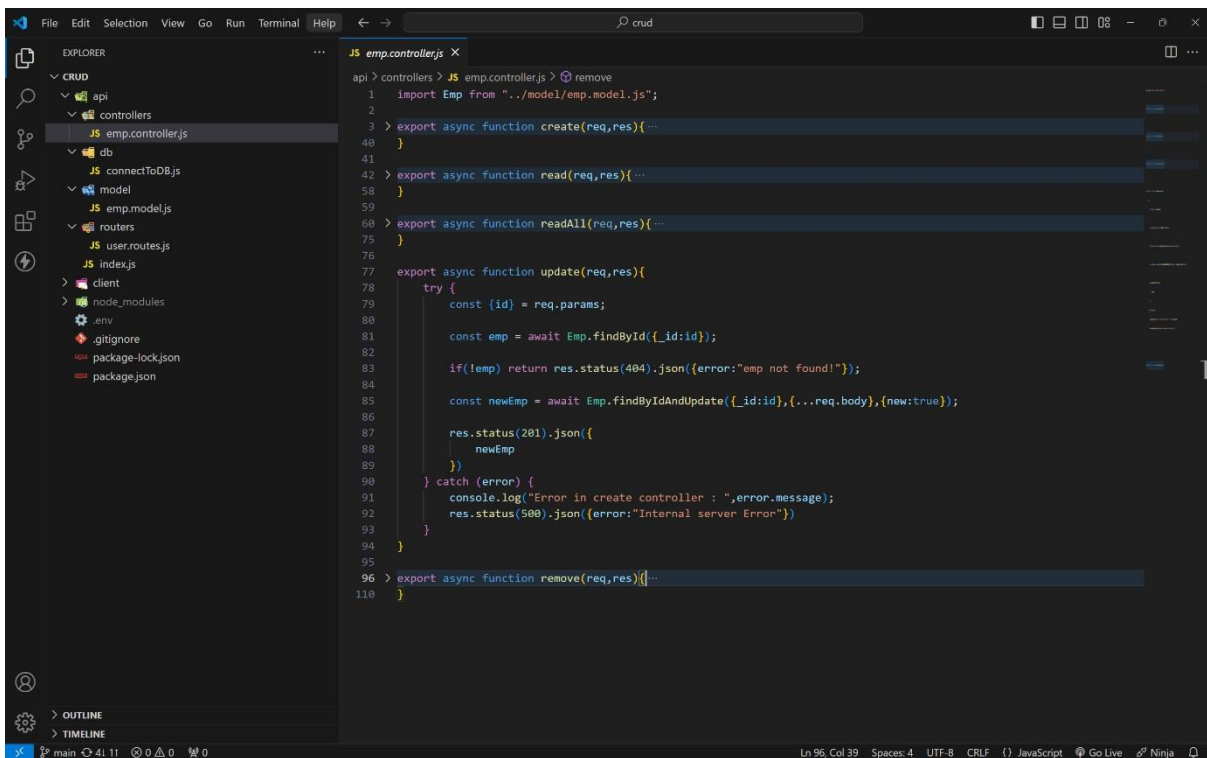
```

## READONE :



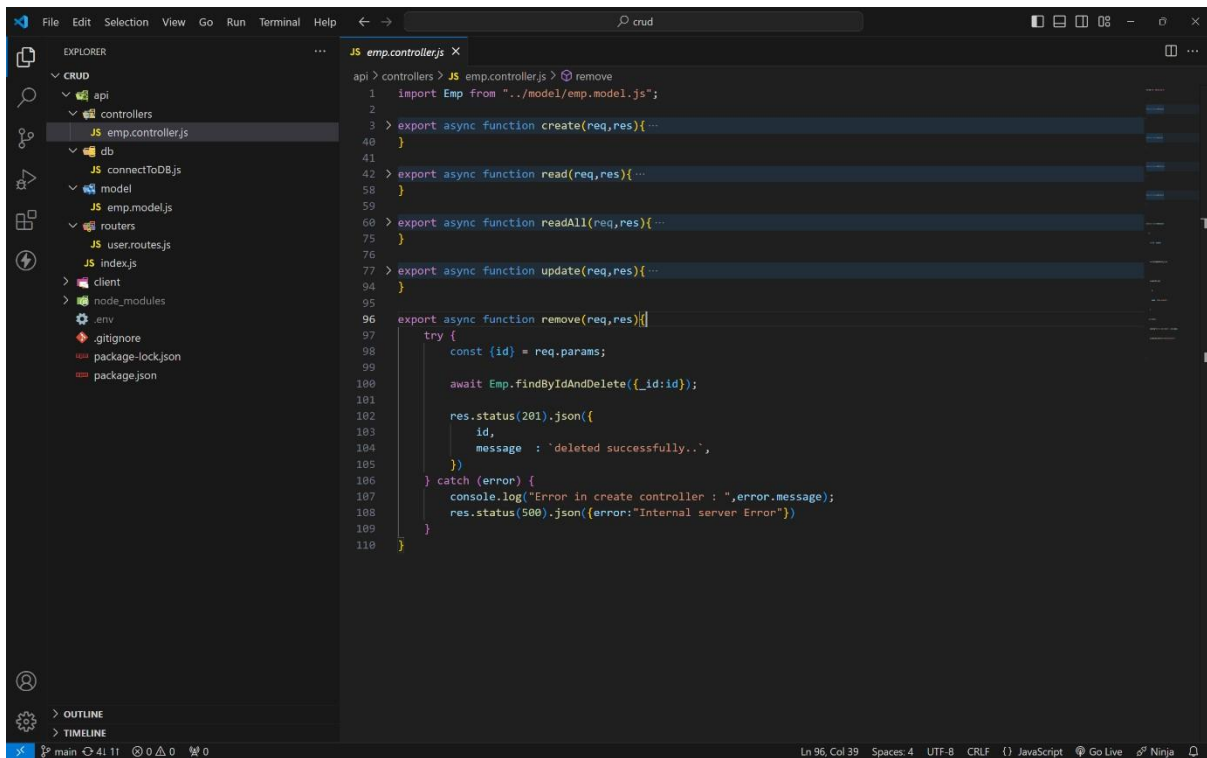
```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  > export async function create(req,res){...
40 }
41
42 export async function read(req,res){
43   try {
44     const {id} = req.params;
45
46     const emp = await Emp.findById({_id:id});
47
48     if(!emp) return res.status(404).json({error:"emp not found!"});
49
50     res.status(201).json({
51       emp
52     })
53   } catch (error) {
54     console.log("Error in create controller : ",error.message);
55     res.status(500).json({error:"Internal server Error"})
56   }
57 }
58
59 > export async function readAll(req,res){...
60 }
61
62 > export async function update(req,res){...
63 }
64
65 > export async function remove(req,res){...
66 }
```

## UPDATE :



```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  > export async function create(req,res){...
40 }
41
42 > export async function read(req,res){...
43 }
44
45 > export async function readAll(req,res){...
46 }
47
48 export async function update(req,res){
49   try {
50     const {id} = req.params;
51
52     const emp = await Emp.findById({_id:id});
53
54     if(!emp) return res.status(404).json({error:"emp not found!"});
55
56     const newEmp = await Emp.findByIdAndUpdate({_id:id},{...req.body},{new:true});
57
58     res.status(201).json({
59       newEmp
60     })
61   } catch (error) {
62     console.log("Error in create controller : ",error.message);
63     res.status(500).json({error:"Internal server Error"})
64   }
65 }
66
67 > export async function remove(req,res){...
68 }
```

## DELETE :



The screenshot shows the VS Code editor with the file explorer on the left displaying a project structure for a CRUD application. The main editor window shows the `emp.controller.js` file, where the `remove` function is implemented. The function uses `req.params` to get the `id`, calls `Emp.findByIdAndDelete` to remove the record, and returns a 201 status with a success message. Error handling is included with a 500 status for internal server errors.

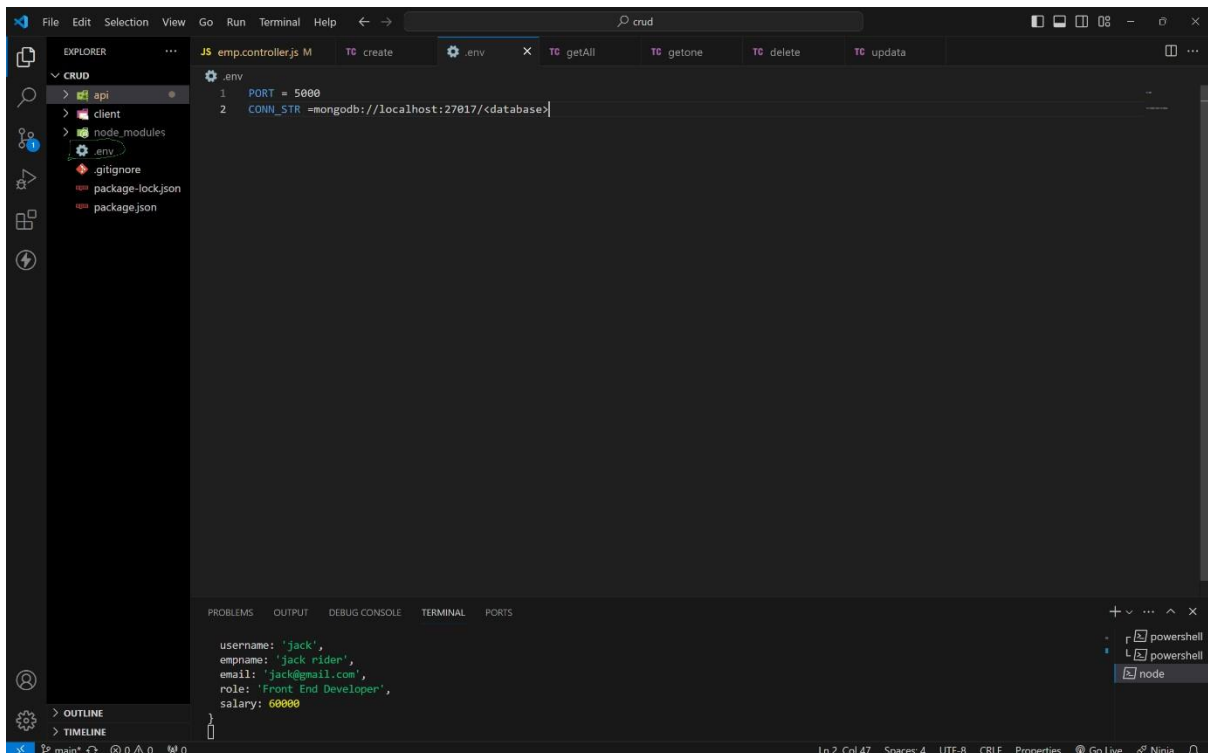
```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  > export async function create(req,res){...
48 }
41
42 > export async function read(req,res){...
58 }
59
60 > export async function readAll(req,res){...
75 }
76
77 > export async function update(req,res){...
84 }
85
96 export async function remove(req,res){
97   try {
98     const {id} = req.params;
99
100     await Emp.findByIdAndDelete({_id:id});
101
102     res.status(201).json({
103       id,
104       message : 'deleted successfully..',
105     });
106   } catch (error) {
107     console.log('Error in create controller : ',error.message);
108     res.status(500).json({error:'Internal server Error'});
109   }
110 }
```

## HOW TO RUN ON LOCALLY :

- 1 . Create a folder as any name.
- 2 . Open that folder in any code editor (vs code).
- 3 . Open terminal ( ctrl + ~ ) on code editor.
- 4 . Type this code to get code locally.      git clone  
<https://github.com/4727yesuraju/crud.git>
- 5 . Now move to crud folder (cd crud in terminal)
- 6 . Ignore client folder.
- 7 . Here crud is root folder.
- 8 . In root folder create a .env file and create a PORT and  
CONN\_STR variables and assign value.

**ex : PORT = 3000** (commonly any number between 3000 - 8080).

**CONN\_STR = your mongodb\_connection\_string.**



**--- trouble in above process ? :**

**simply paste this code in .env file .**

**PORT = 5000**

**CONN\_STR=mongodb+srv://4727yesuraju:rough@cluster0.wbclvtg.mongodb.net  
/?retryWrites=true&w=majority&appName=Cluster0**

**9 . After in terminal (in crud folder as root folder) type this command  
to run server.**

**npm i (installing all dependencies)**

**npm run dev (to run server)**

**10 . if you get below message in terminal then your server will  
running successfully.**

```
PS C:\Users\4727y\OneDrive\Desktop\internshala\crud> npm run dev

> crud@1.0.0 dev
> nodemon api/index.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node api/index.js`
Server is running on PORT : 5000
DB connected successfully
█
```

## **route and its functionality :**

**For this use any API using tools like Postman or Thunder Client.**

**i use THUNDER CLIENT.**

### **CREATE ROUTE :**

**1 . This route is used to create a new employee in database with a below fields.**

**username, empname, email, role, salary**

**2 . in thunder client click on new request and select this options**



**method as post url as** http://localhost:5000/api/user/create

**pass this json data as a body as your required value.**

```
{  
  "username": "jack",  
  "empname": "jack rider",  
  "email": "jack@gmail.com",  
  "role": "Front End Developer",  
  "salary": 60000  
}
```

**3 . finally press send to insert data in mongodb data base and get a  
inserted data  
as a response.**

**4 . If user is already in db it will return User is already exist as  
response. for more details visit below output images...**

## **READONE :**

**1 . This route is used to read specific user info by passing that user id  
as a param. method**

**as get**

**url as** http://localhost:5000/api/user/read/65ed7b3d76e1dcc9a51654ca

**2 . After sending you will get that specific user details as response.**

## **READALL :**

**1 . Read all route is used to get all the user data existing in the mongodb**

**data base .**

**method as get url as**

**http://localhost:5000/api/user/readall**

**2 . After sending you will get that all user details as response.**

## **UPDATE :**

**1 . This route is used to update specific user by passing that user id as**

**a param. method**

**as put**

**url as**

**http://localhost:5000/api/user/update/65ed7b3d76e1dcc9a51654ca**

**2 . After sending you will get updated user details as response.**

## **DELETE :**

**1 . This route is used to delete specific user by passing that user id as**

**a param. method**

**as delete**

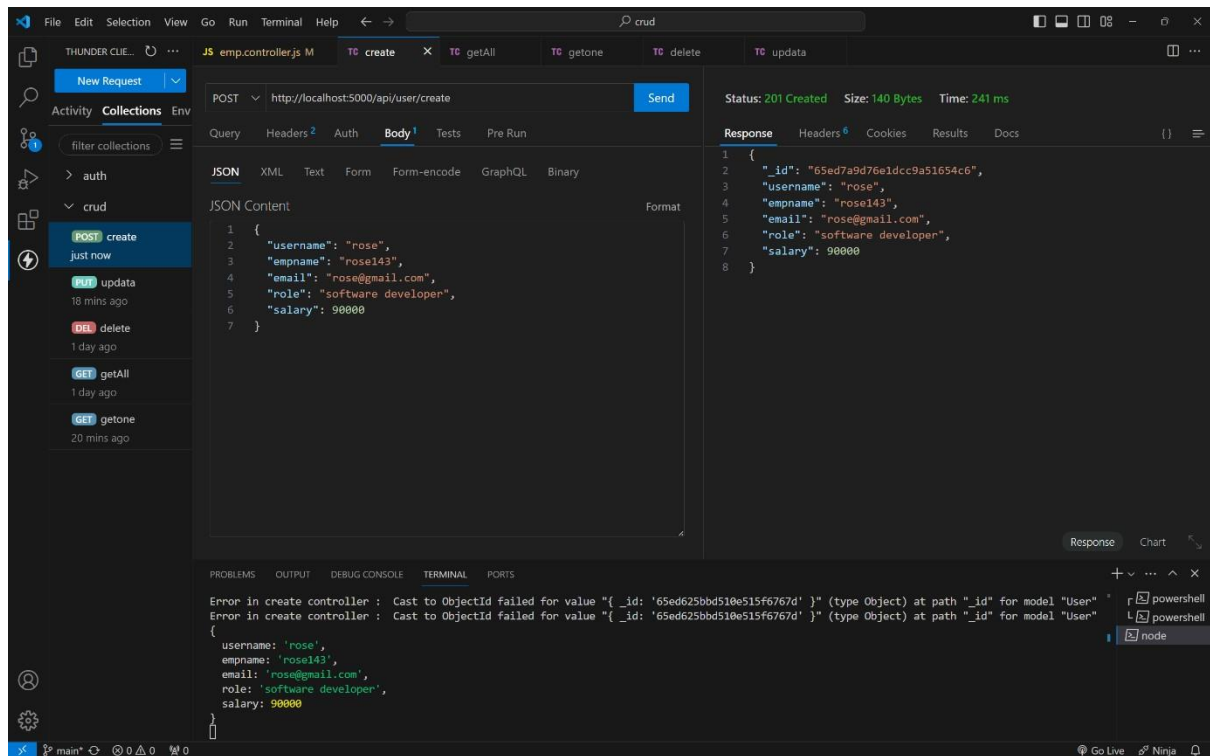
**url as**

**http://localhost:5000/api/user/delete/65ed7b3d76e1dcc9a51654ca**

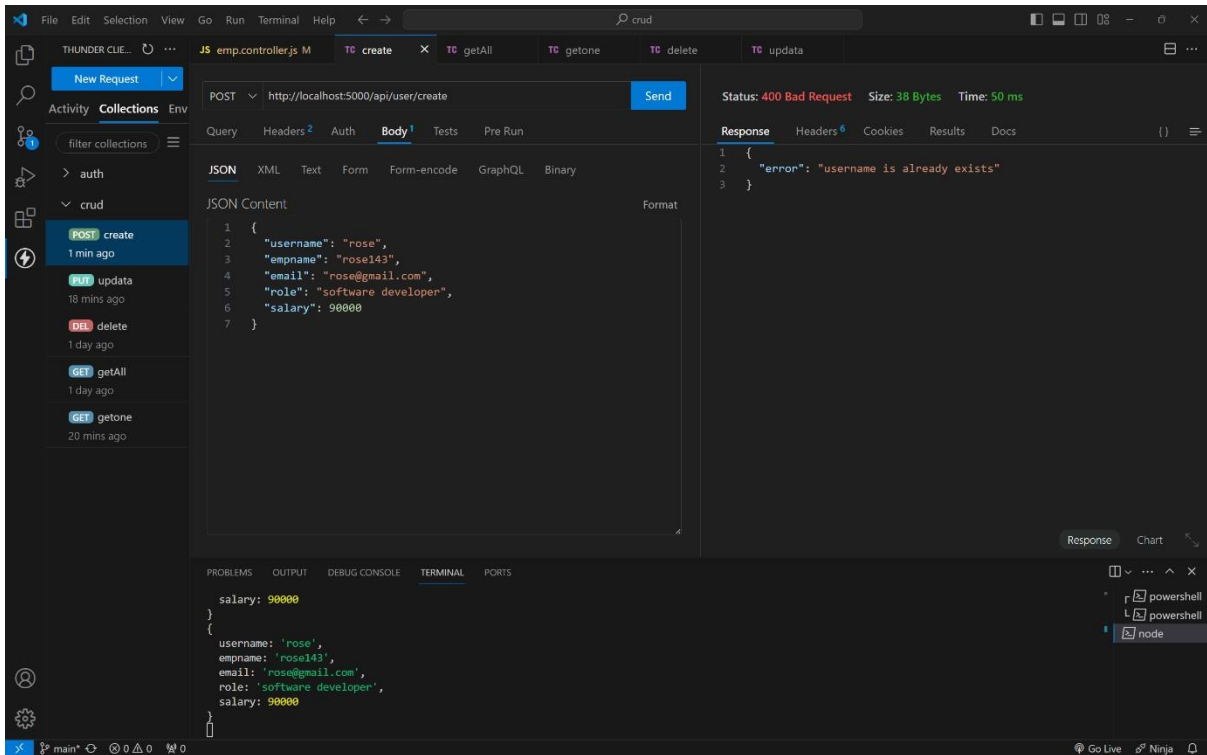
**2 . After sending you will deleted successfully as response.**

**OUTPUT :**

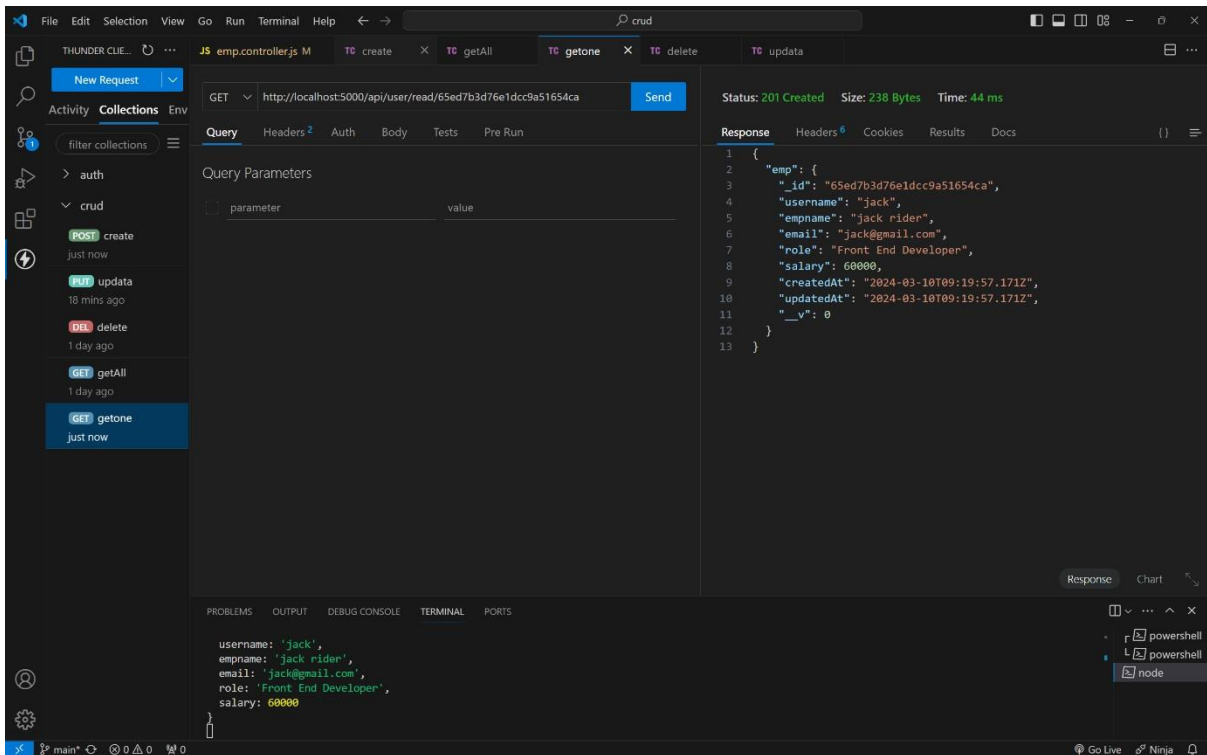
**CREATE A NEW USER :**



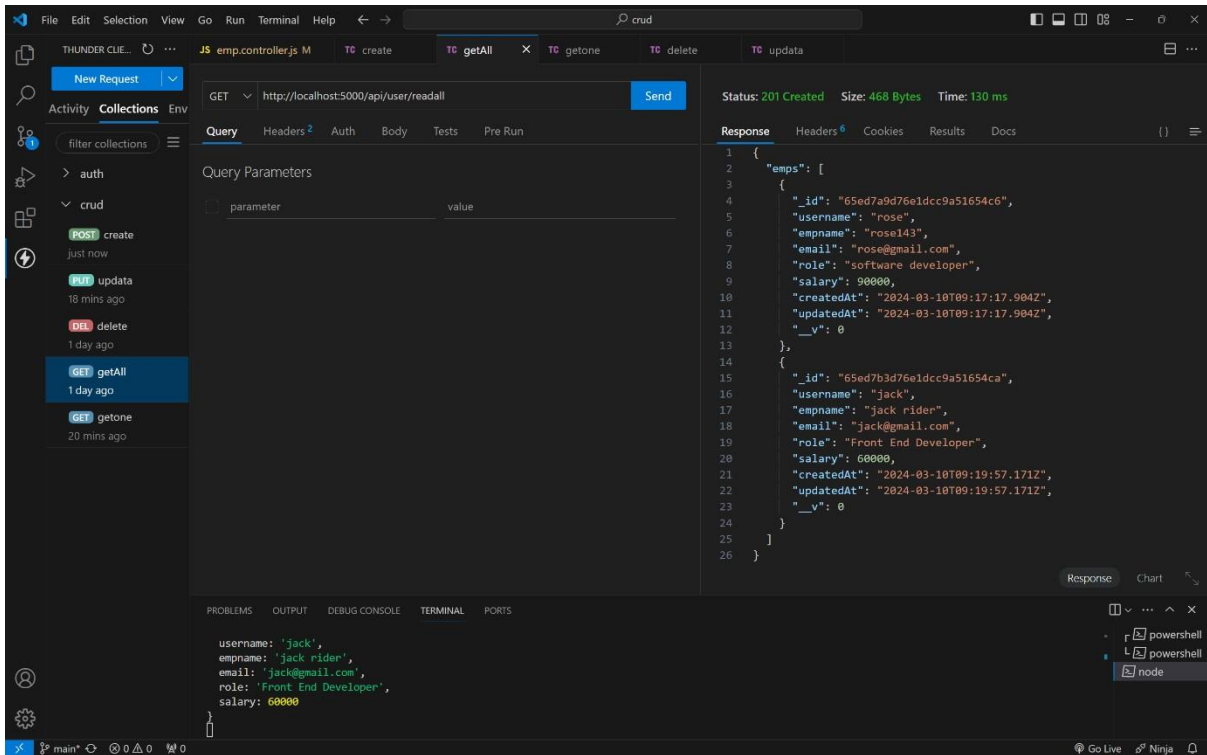
**CREATING USER WITH EXISTING USERNAEM :**



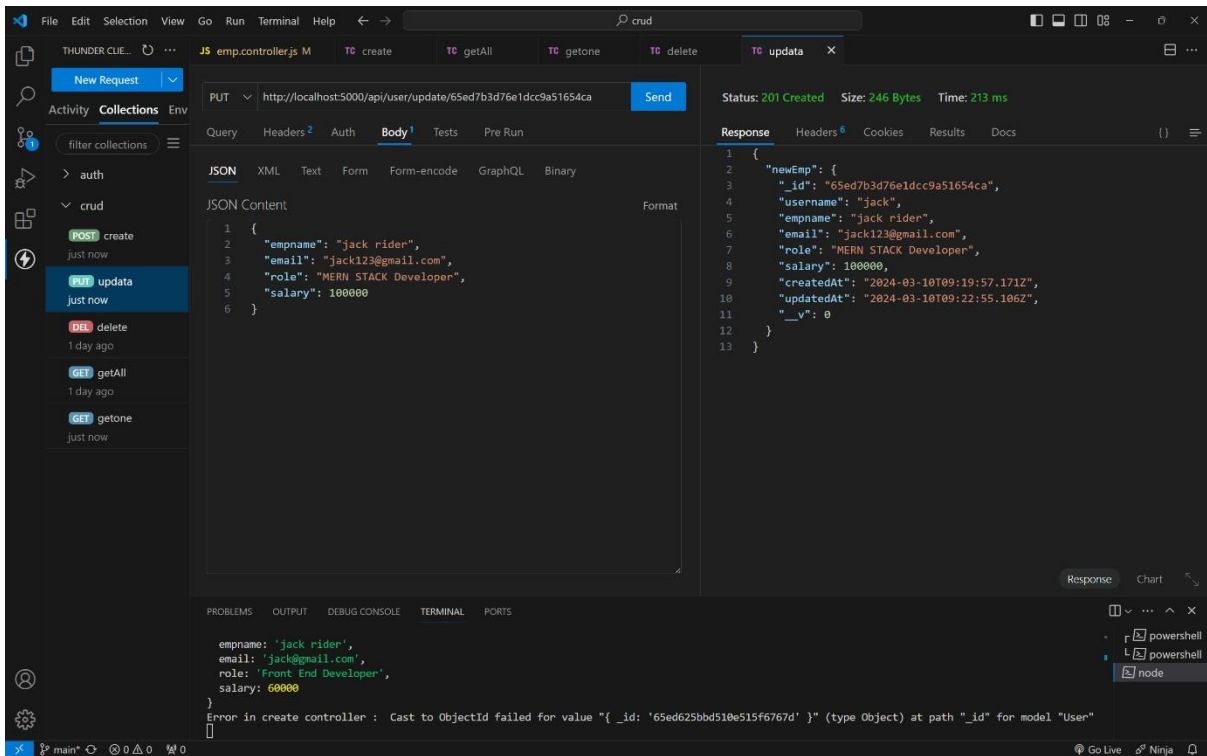
READONE :



READ ALL :



## UPDATE :



## DELETE :

THUNDER CLIENT

JS emp.controller.js M TC create TC getAll TC getone TC delete TC update

New Request

Activity Collections Env

filter collections

auth

crud

POST create 5 mins ago

PUT update 2 mins ago

DEL delete just now

GET getAll 1 day ago

GET getone 3 mins ago

DELETE http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca Send

Status: 201 Created Size: 68 Bytes Time: 111 ms

Query Headers Auth Body Tests Pre Run

Query Parameters

parameter value

Response Headers Cookies Results Docs

```
1 {
2   "id": "65ed7b3d76e1dcc9a51654ca",
3   "message": "deleted successfully.."
4 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Node.js v20.11.0
[nodemon] app crashed - waiting for file changes before starting...
[nodemon] restarting due to changes...
[nodemon] starting `node api/index.js`
Server is running on PORT : 5000
DB connected successfully
```

main\*

Go Live Ninja