

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

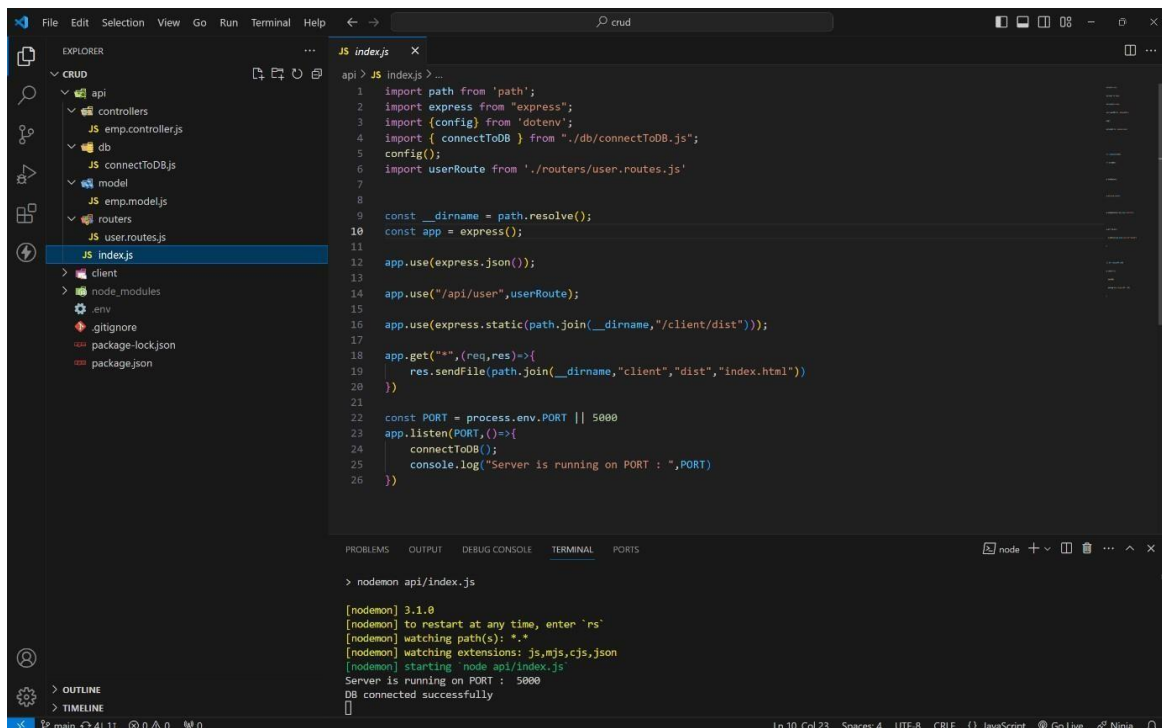
Name:Kakarla Asritha

Email:kakarlaasritha.222@gmail.com

College:Vignan's Nirula institute of technology and science for women

source code :

index.js file :

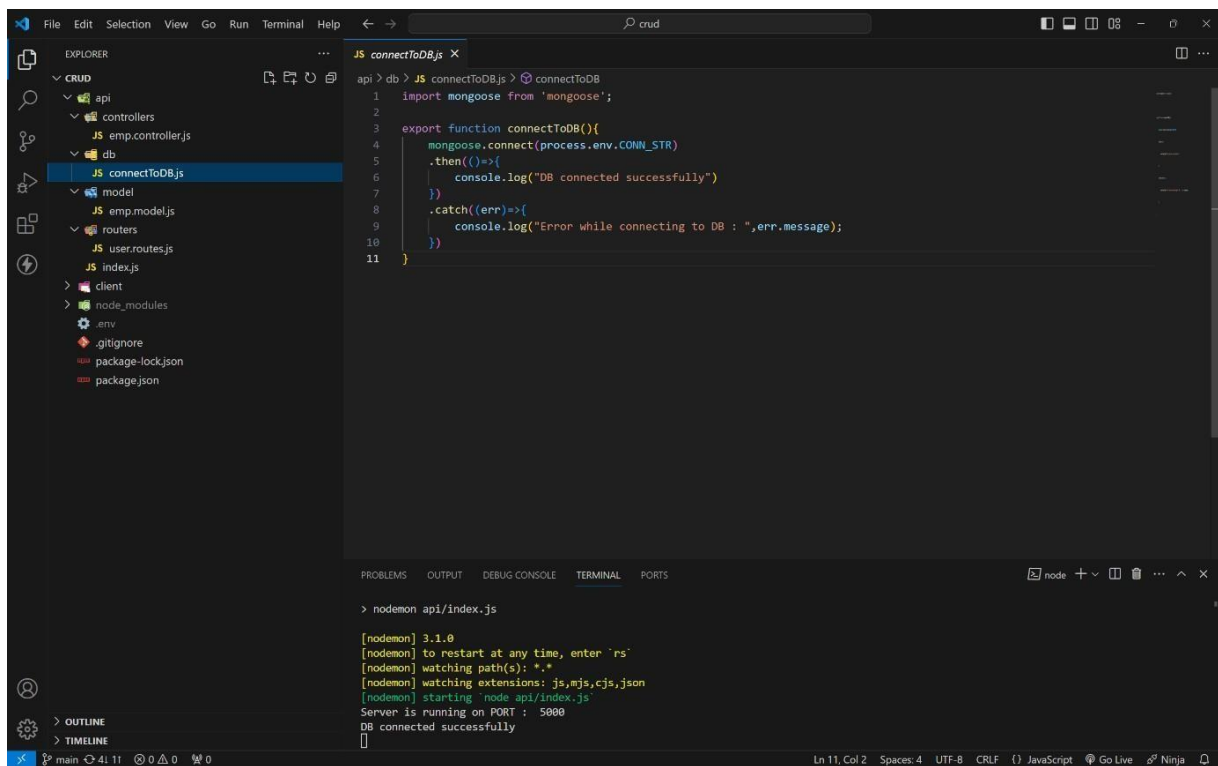


The screenshot shows a Visual Studio Code editor with a project named 'crud'. The Explorer sidebar on the left shows the file structure, with 'api/index.js' selected. The main editor window displays the code for 'index.js'. The code imports necessary modules, configures the database connection, sets up the Express app, and defines routes. The terminal at the bottom shows the command 'nodemon api/index.js' being executed, with output indicating that the server is running on port 5000 and the database is connected successfully.

```
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 })
```

```
> 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
```

MONGODB CONNECTION :



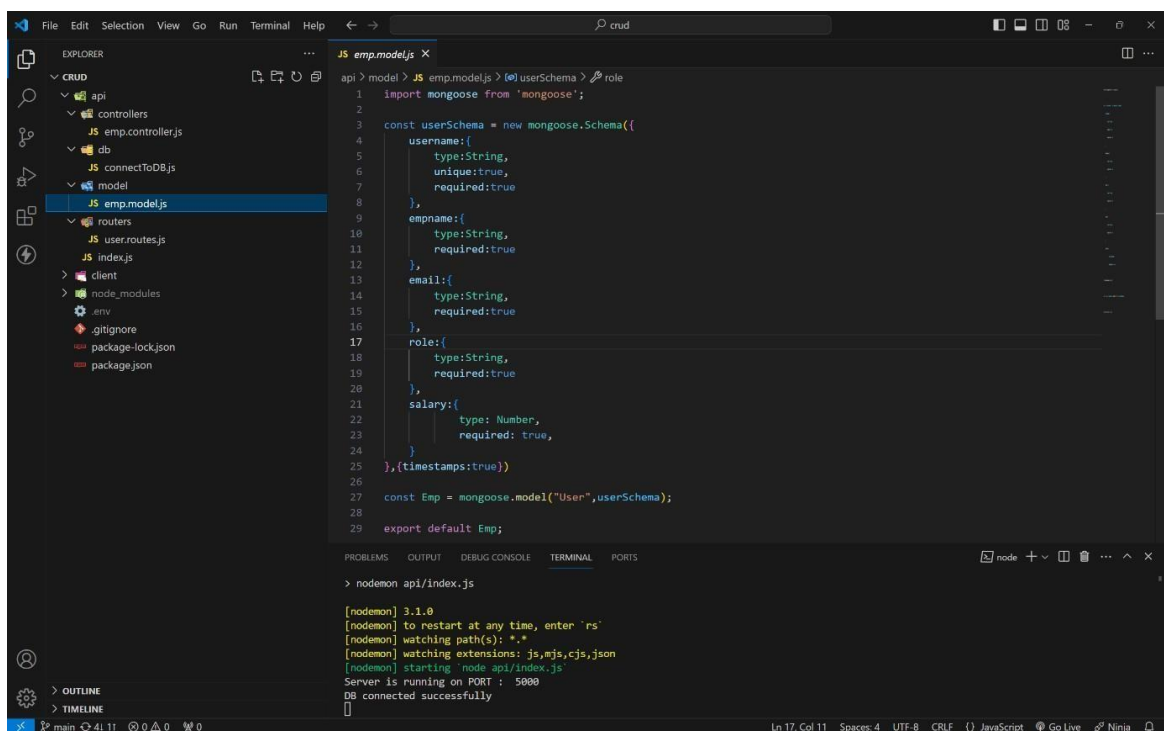
The screenshot shows the VS Code editor with the Explorer sidebar on the left. The file explorer shows a project structure with folders like 'api', 'controllers', 'db', 'model', 'routers', 'user.routes.js', 'index.js', 'client', 'node_modules', '.env', '.gitignore', 'package-lock.json', and 'package.json'. The file 'connectToDB.js' is selected in the Explorer. The main editor displays the code for 'connectToDB.js'.

```
api > db > JS connectToDB.js > connectToDB
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 at the bottom shows the output of running 'nodemon api/index.js'.

```
> 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
```

MODEL :



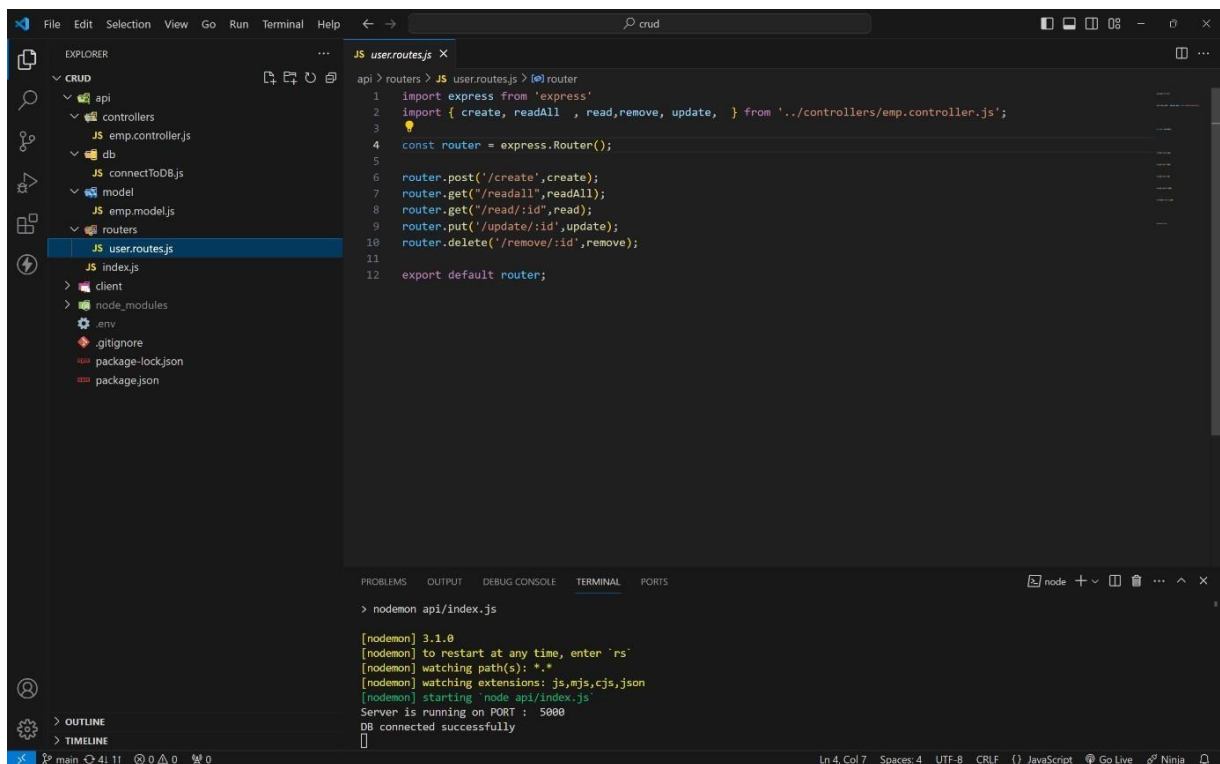
The screenshot shows the VS Code editor with the Explorer sidebar on the left. The file explorer shows the same project structure as the previous screenshot. The file 'emp.model.js' is selected in the Explorer. The main editor displays the code for 'emp.model.js'.

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

The terminal at the bottom shows the output of running 'nodemon api/index.js'.

```
> 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 folders like api, controllers, db, model, routers, and client. The file routers/user.routes.js is selected. The editor shows the following code:

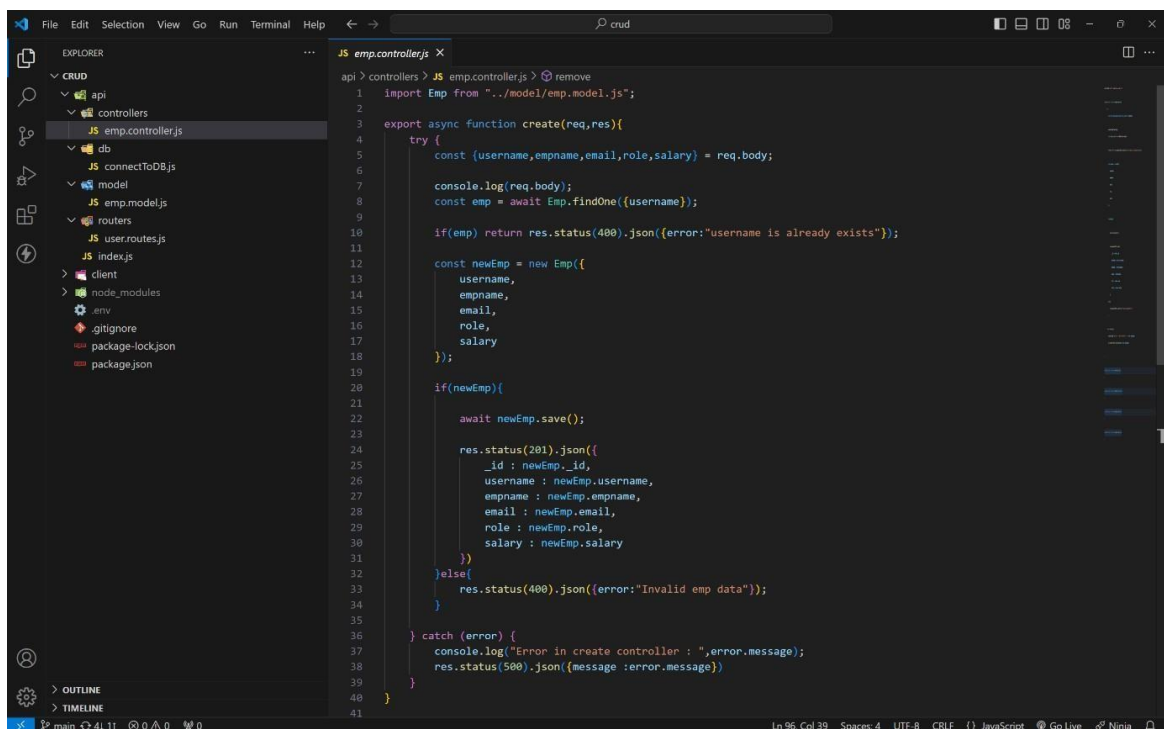
```
api > routers > JS user.routes.js > @router
1 import express from 'express'
2 import { create, readAll, read, remove, update, } from '../controllers/emp.controller.js';
3
4 const router = express.Router();
5
6 router.post('/create', create);
7 router.get('/readall', readAll);
8 router.get('/read/:id',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

The terminal at the bottom shows the output of running the application:

```
> 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 :

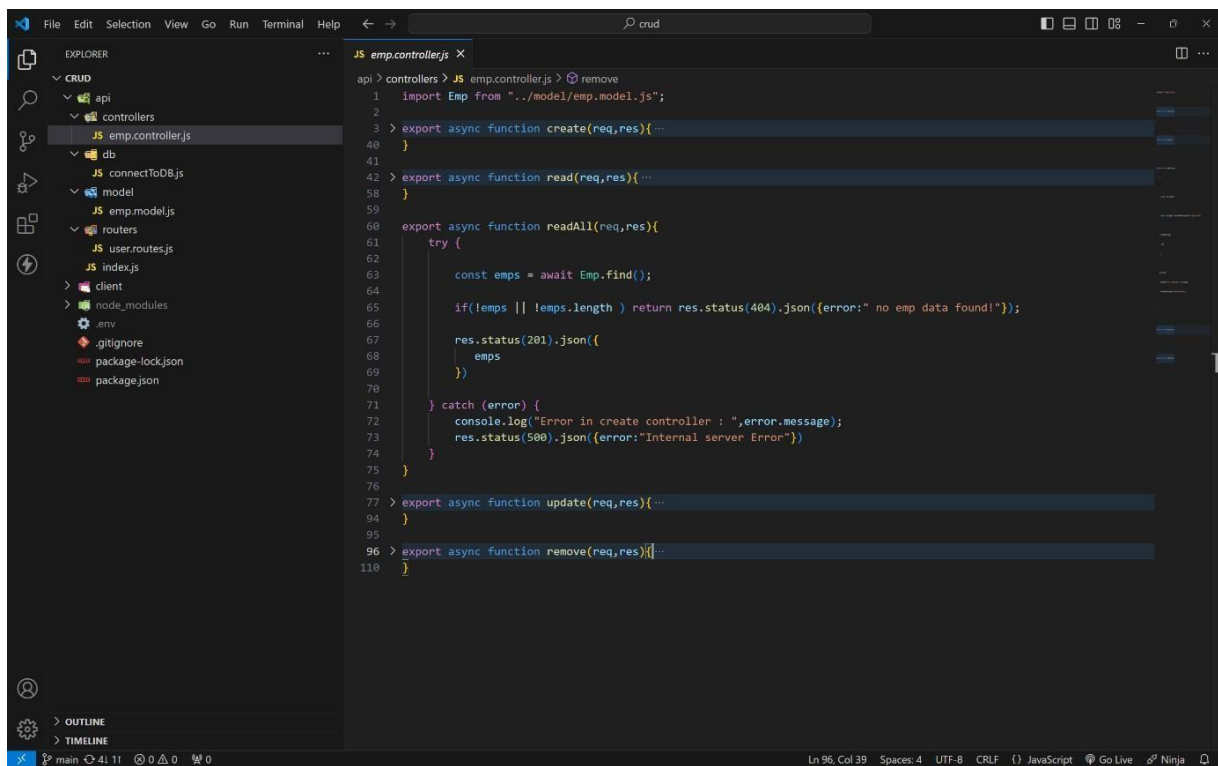
CREATE :



The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows a project structure with folders like api, controllers, db, model, routers, and client. The file controllers/emp.controller.js is selected. The editor shows the following code:

```
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 }
```

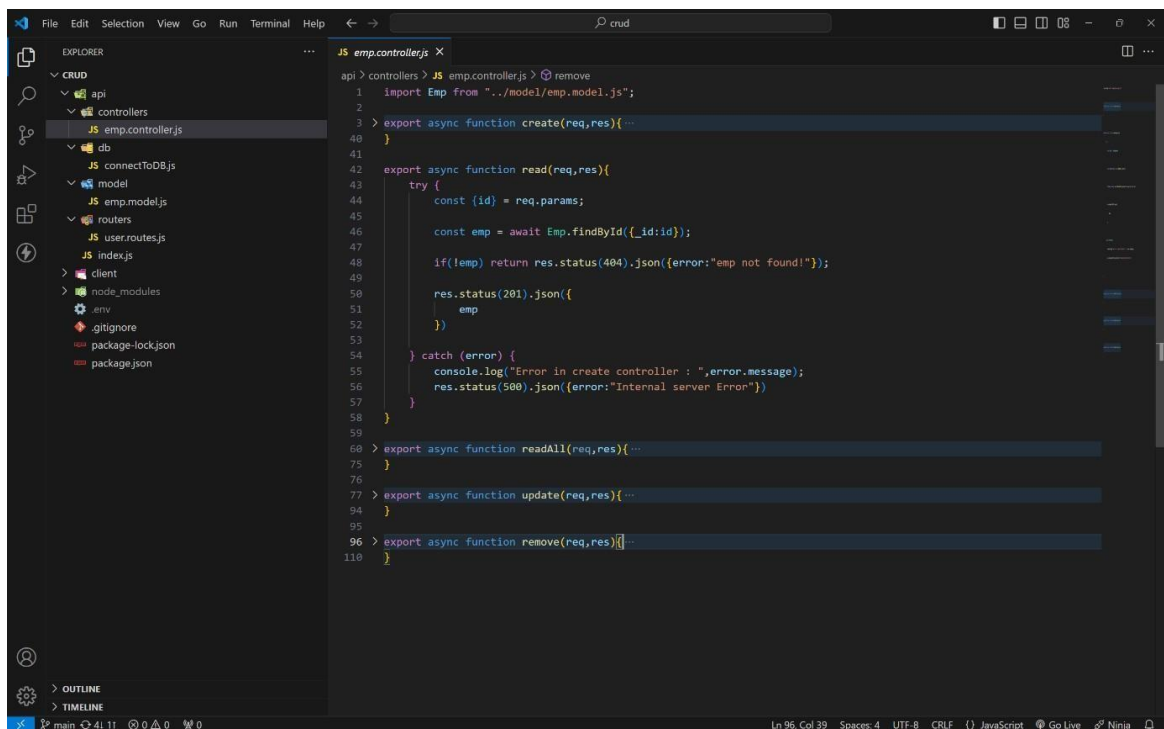
READALL:



The screenshot shows the VS Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure with folders like api, controllers, db, model, routers, and client. The code editor shows the file emp.controller.js with the following code:

```
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
77 > export async function update(req,res){ ...
94 }
95
96 > export async function remove(req,res){ ...
110 }
```

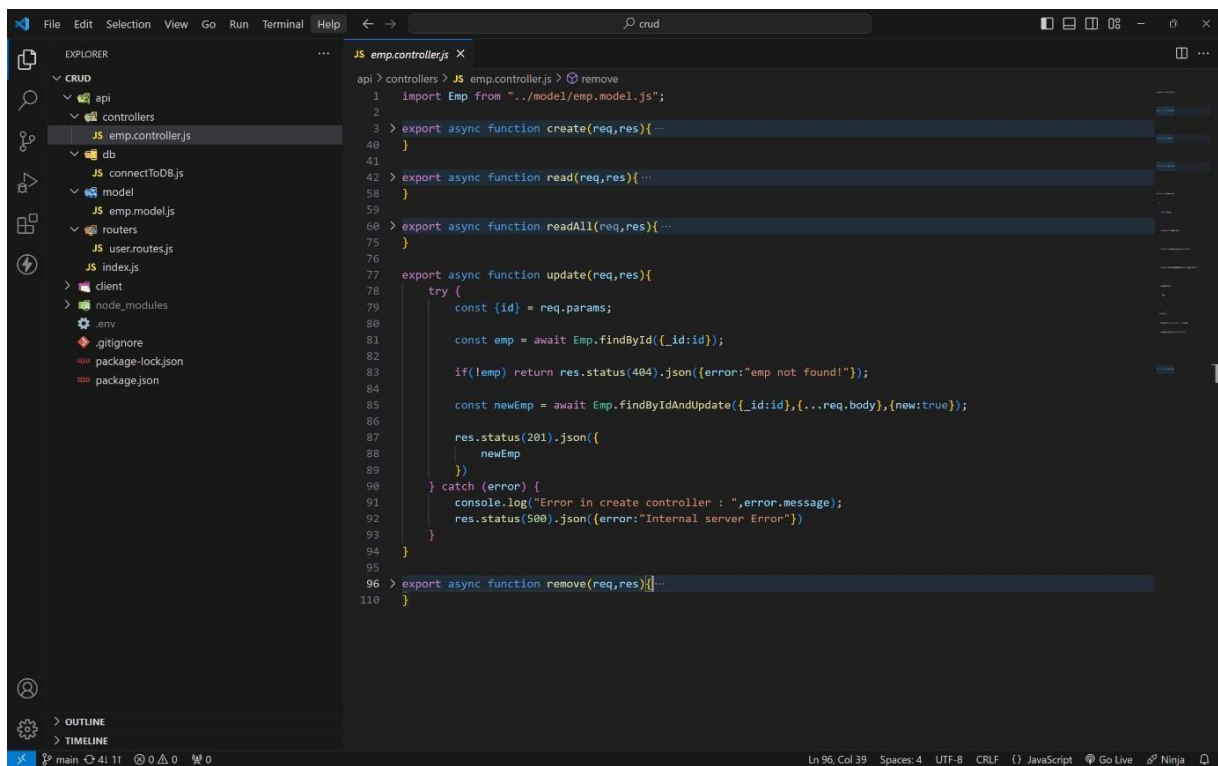
READONE :



The screenshot shows the VS Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure with folders like api, controllers, db, model, routers, and client. The code editor shows the file emp.controller.js with the following code:

```
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     const emp = await Emp.findById({_id:id});
46
47     if(!emp) return res.status(404).json({error:"emp not found!"});
48
49     res.status(201).json({
50       emp
51     })
52   } catch (error) {
53     console.log("Error in create controller : ",error.message);
54     res.status(500).json({error:"Internal server Error"})
55   }
56 }
57
58
59
60 > export async function readAll(req,res){ ...
75 }
76
77 > export async function update(req,res){ ...
94 }
95
96 > export async function remove(req,res){ ...
110 }
```

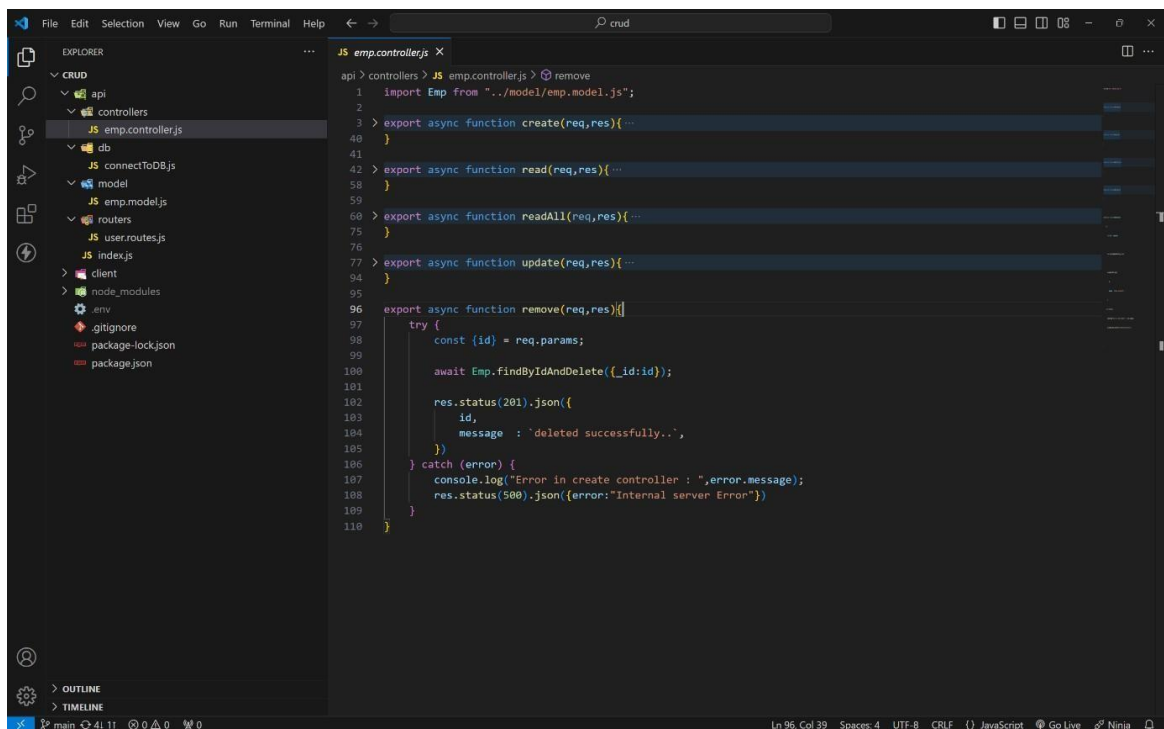
UPDATE :



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor in the center. The file explorer shows a project structure with folders like api, controllers, db, model, routers, and client. The file emp.controller.js is selected. The code editor shows the following JavaScript code:

```
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){...
75 }
76
77 export async function update(req,res){
78   try {
79     const {id} = req.params;
80
81     const emp = await Emp.findById({_id:id});
82
83     if(!emp) return res.status(404).json({error:"emp not found!"});
84
85     const newEmp = await Emp.findByIdAndUpdate({_id:id},{...req.body},{new:true});
86
87     res.status(201).json({
88       newEmp
89     });
90   } catch (error) {
91     console.log("Error in create controller : ",error.message);
92     res.status(500).json({error:"Internal server Error"});
93   }
94 }
95
96 > export async function remove(req,res){...
110 }
```

DELETE :



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor in the center. The file emp.controller.js is selected. The code editor shows the following JavaScript code:

```
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){...
75 }
76
77 > export async function update(req,res){...
94 }
95
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 }
```

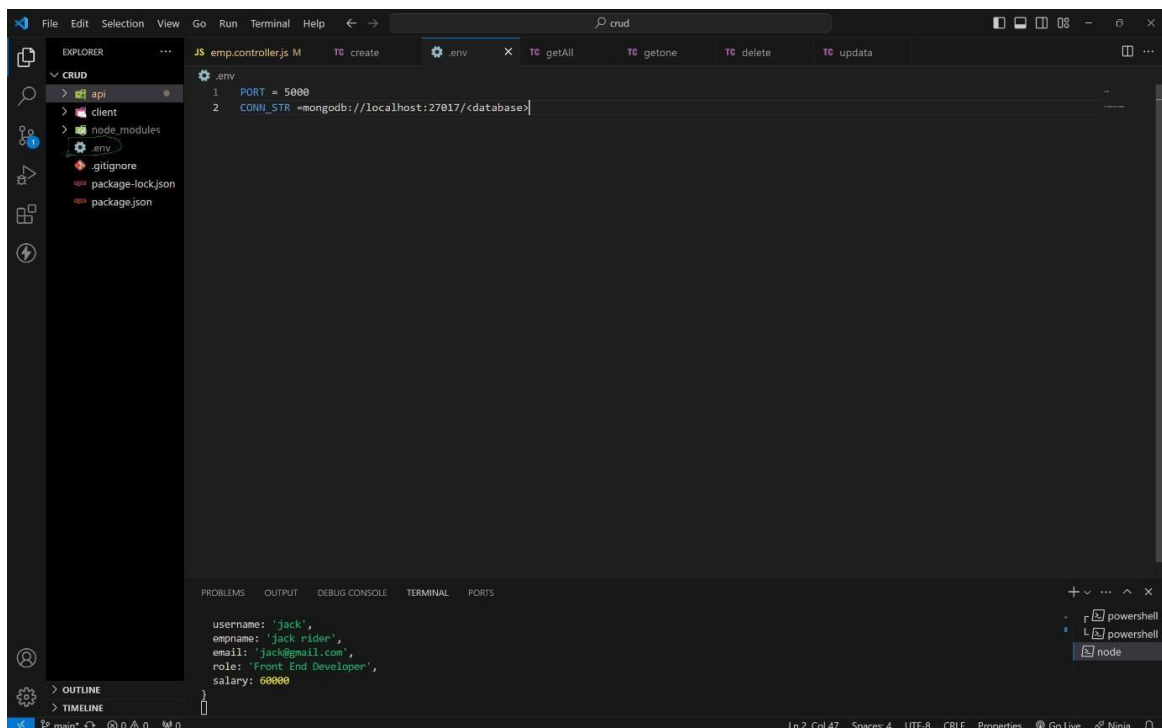
HOW TO RUN ON LOCALLY :

- 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.
- 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

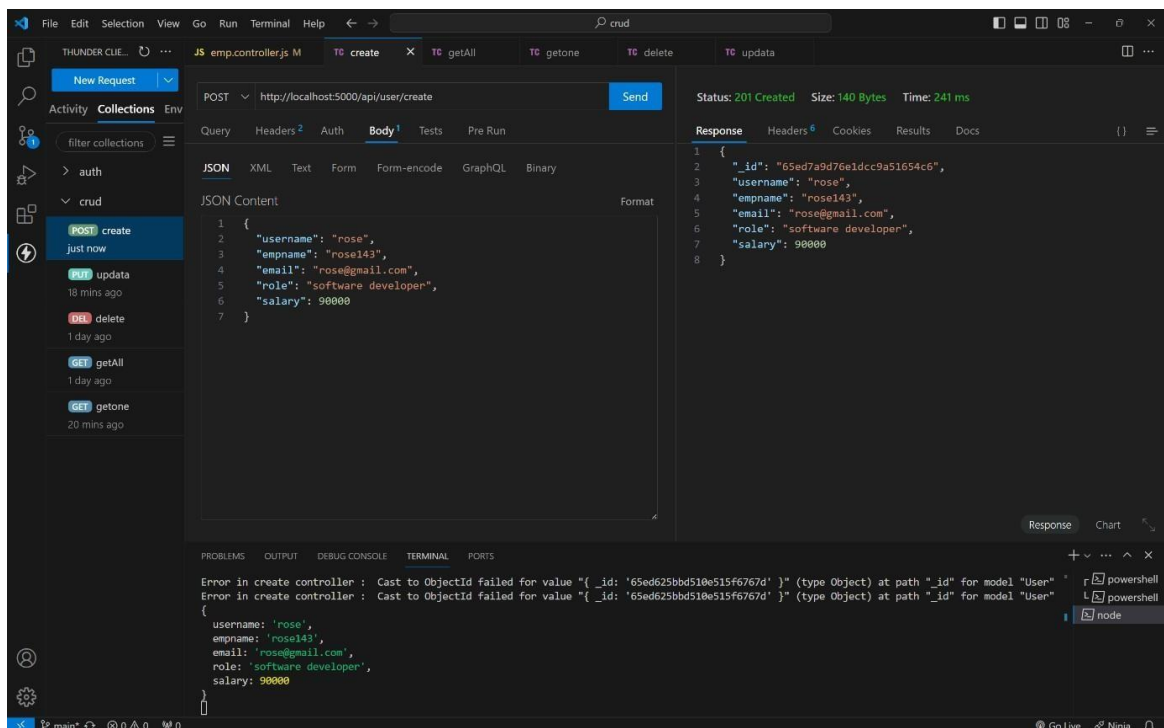
url as

<http://localhost:5000/api/user/delete/65ed7b3d76e1dcc9a51654ca>

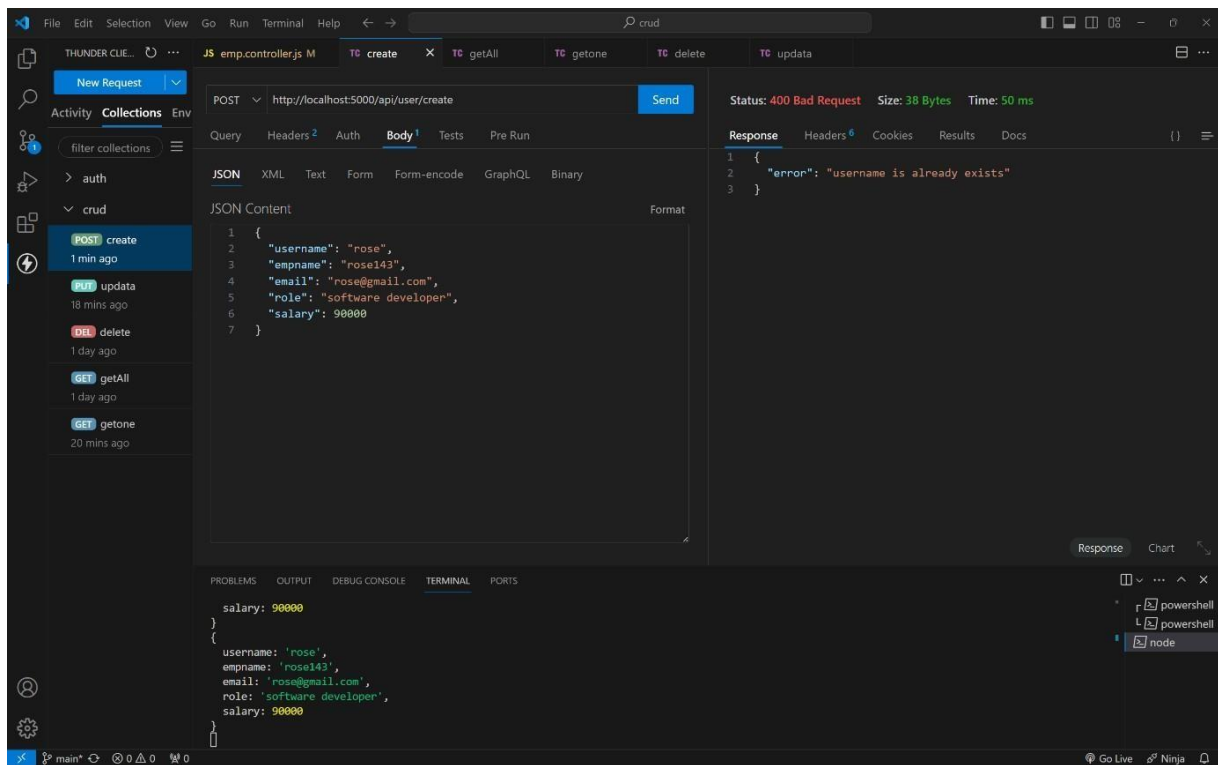
- 2 . After sending you will deleted successfully as response.

OUTPUT :

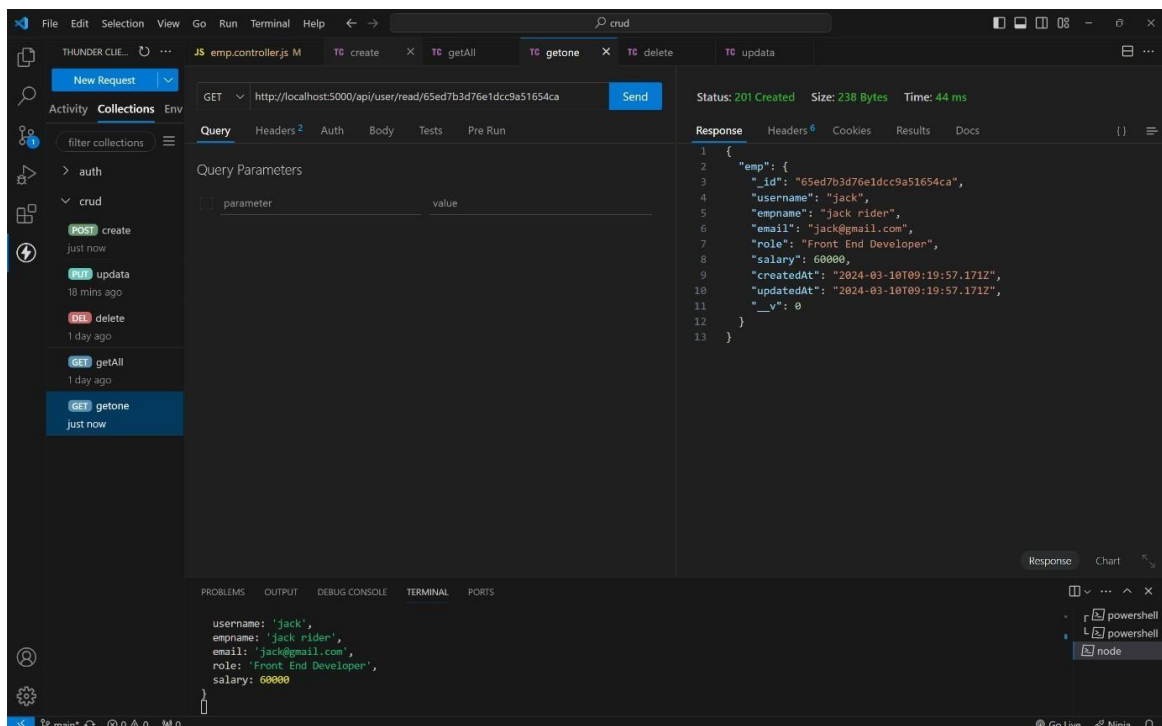
CREATE A NEW USER :



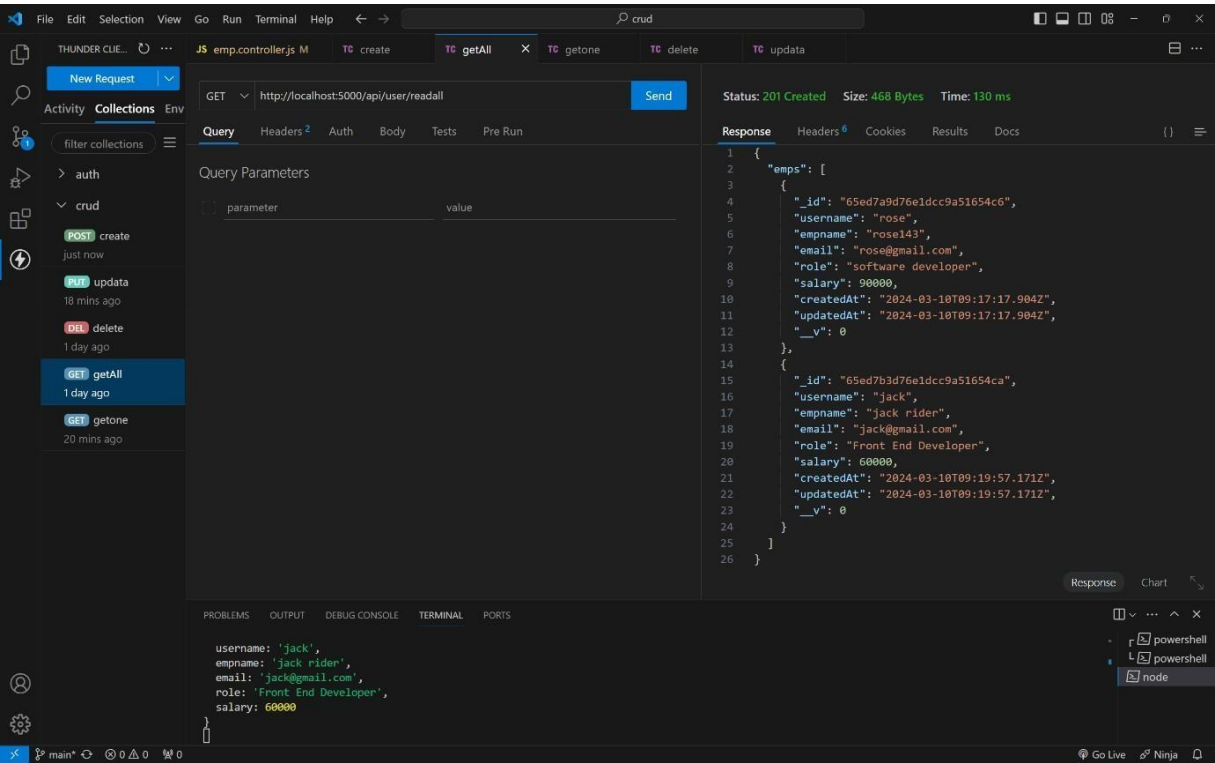
CREATING USER WITH EXISTING USERNAEM :



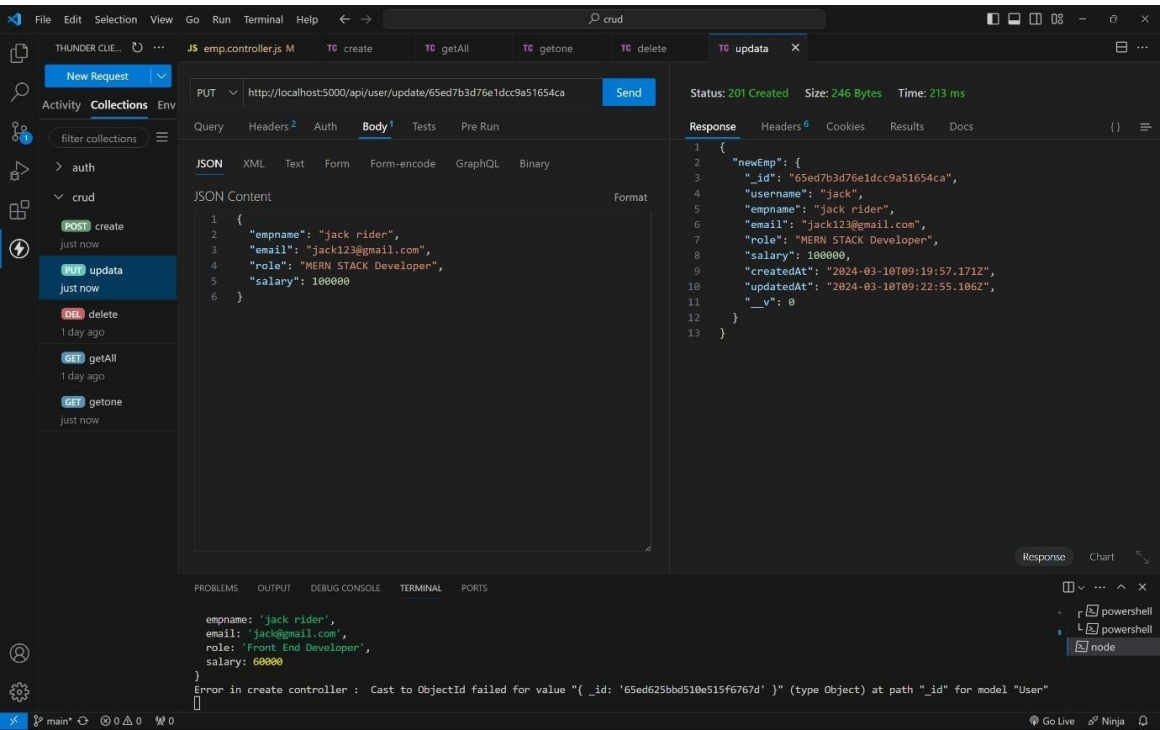
READONE :



READ ALL :



UPDATE :



DELETE :

The screenshot displays the Thunder Client interface with a REST client request configured for a DELETE method. The URL is `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`. The response status is 201 Created, with a size of 68 Bytes and a time of 111 ms. The response body is a JSON object: `{ "id": "65ed7b3d76e1dcc9a51654ca", "message": "deleted successfully.." }`. The terminal at the bottom shows the server running on port 5000 and the database connected successfully.

Thunder Client interface showing a DELETE request to `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`. The request is successful, returning a 201 Created status with a response body: `{ "id": "65ed7b3d76e1dcc9a51654ca", "message": "deleted successfully.." }`.

The terminal output shows the server running on port 5000 and the database connected successfully.