

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

NAME : Koppineni Satya Sai

EMAIL satyasaikoppineni369@gmail.com

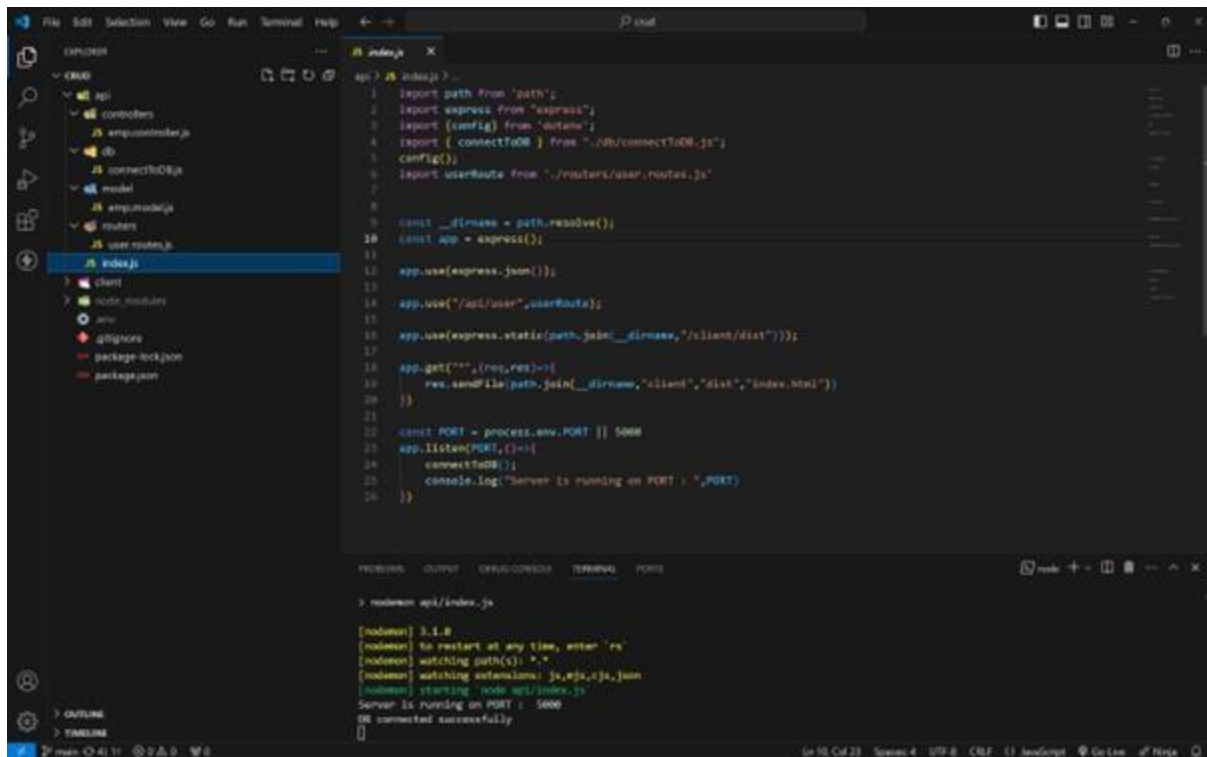
PH.NO : 9966223634

ROLL.NO : 20A71A0515

**COLLEGE : VARAPRASAD REDDY INSTITUTE OF
TECHNOLOGY, KANTEPUDI, GUNTUR.**

source code :

index.js file :



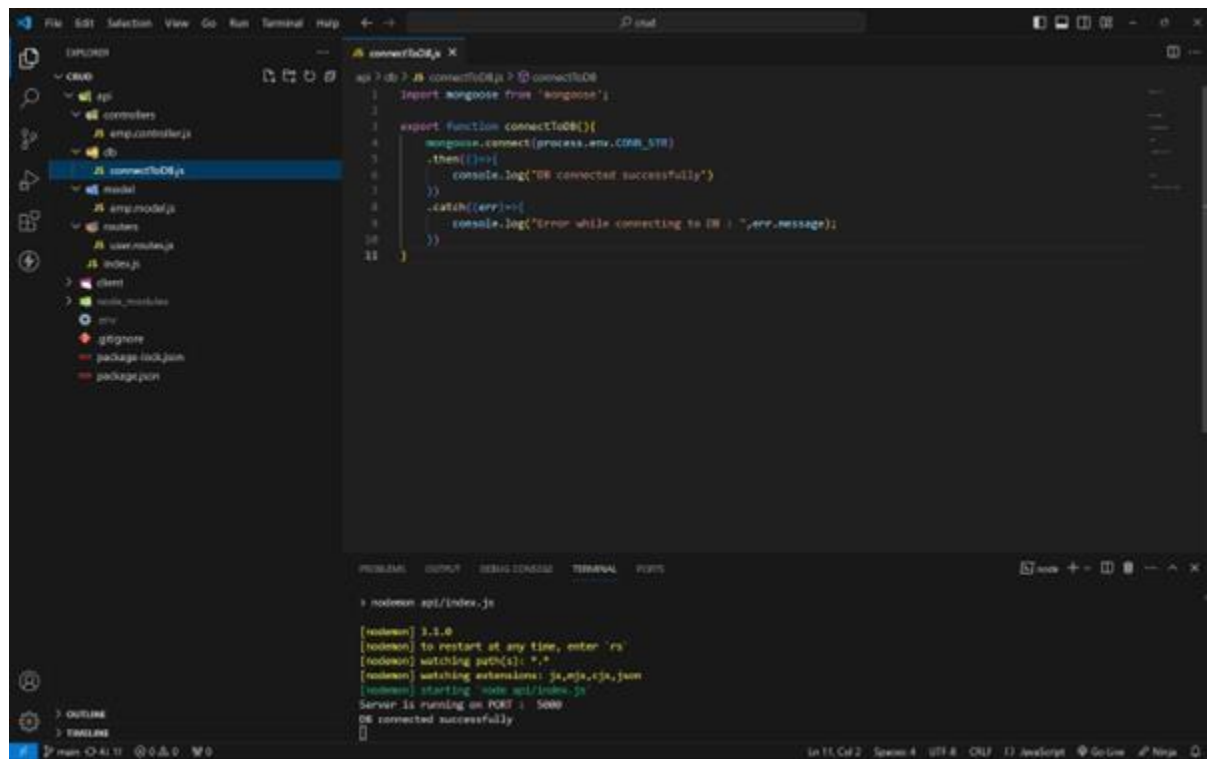
The screenshot shows a VS Code editor with a project named 'api'. The Explorer sidebar on the left shows the file structure, with 'index.js' selected under the 'api' folder. The main editor displays the content of 'index.js', which is a Node.js application using Express.js. The code imports 'path', 'express', 'config', 'connectToDB', and 'userRoutes'. It sets up an Express app, uses 'userRoutes', and serves static files from 'client/dist'. The app listens on port 5000. The terminal at the bottom shows the command 'node api/index.js' being executed, and the output indicates that the server is running successfully on port 5000.

```
api > index.js
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 userRoutes 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", userRoutes);
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 });
```

```
> node 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 :



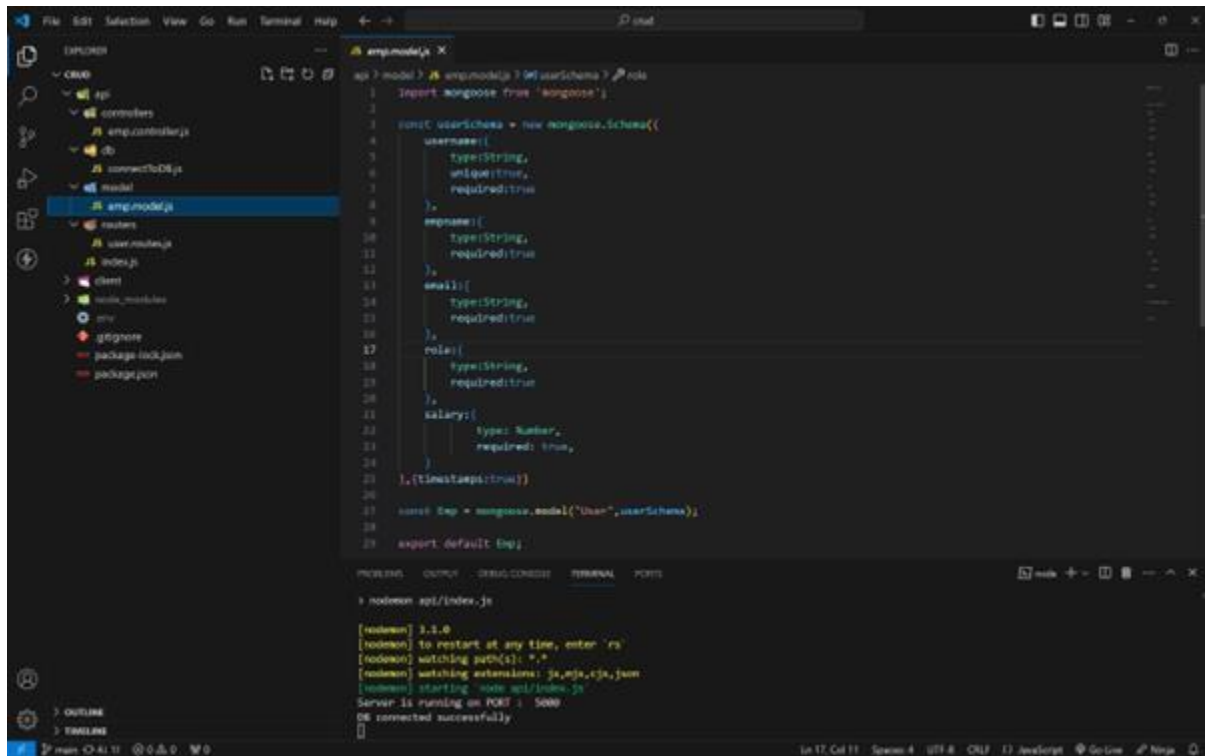
The screenshot shows a VS Code editor with a project named 'api'. The Explorer sidebar on the left shows the file structure, with 'connectToDB.js' selected under the 'api' folder. The main editor displays the content of 'connectToDB.js', which is a Node.js application using Mongoose.js. The code imports 'mongoose' and defines a 'connectToDB' function that attempts to connect to a MongoDB database using the 'process.env.DB_URI' environment variable. The function logs the connection status and handles any errors. The terminal at the bottom shows the command 'node api/index.js' being executed, and the output indicates that the server is running successfully on port 5000.

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

```
> node 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 :

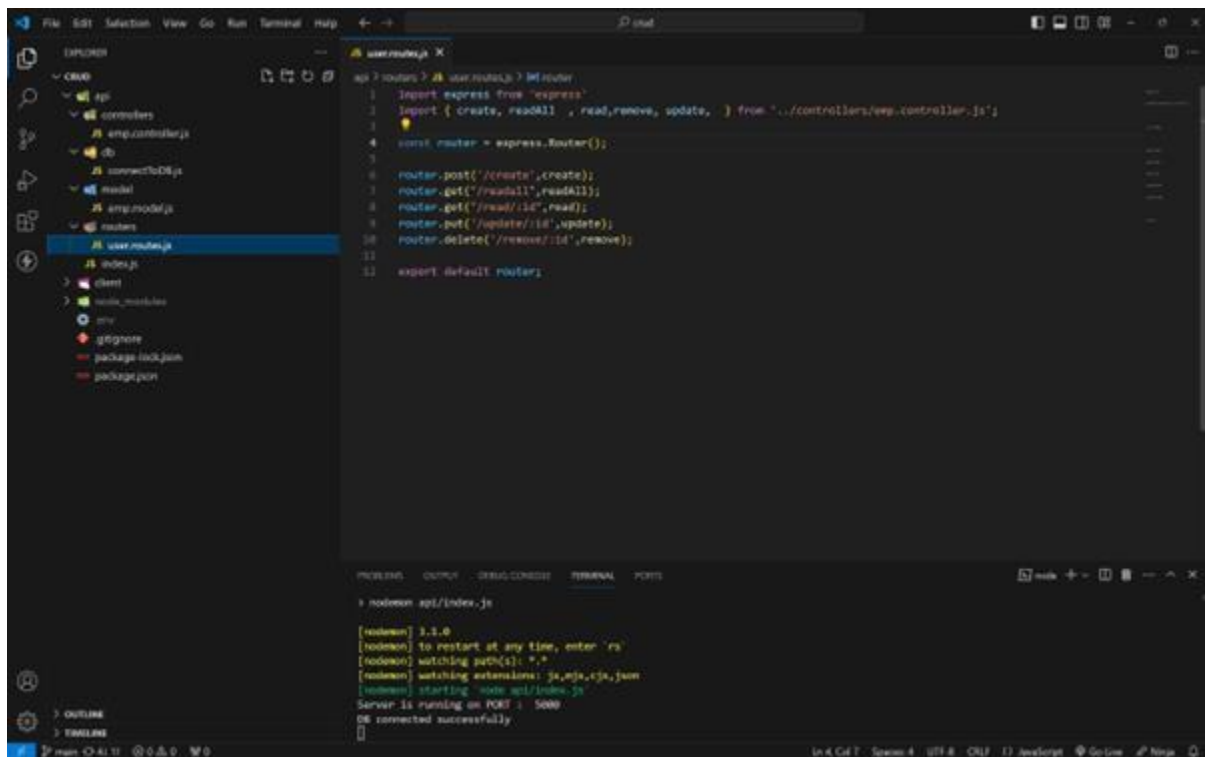


The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows a project structure with folders like 'api', 'controllers', 'db', 'model', 'routes', 'user.routes.js', 'index.js', 'client', 'node_modules', 'src', 'gitignore', 'package-lock.json', and 'package.json'. The 'model' folder is selected, and the file 'model.js' is open in the editor. The code in 'model.js' defines a Mongoose schema for a user and a corresponding model. The terminal at the bottom shows the command 'nodemon api/index.js' and the output, which includes the Node.js version (11.0), the command to restart (rs), the paths being watched, the extensions being loaded, and the server starting on port 5000.

```
api > model > # emp.model.js > # 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   email: {
10    type: String,
11    required: true
12   },
13   role: {
14    type: String,
15    required: true
16   },
17   salary: {
18    type: Number,
19    required: true
20   }
21 }, { timestamps: true })
22
23 const Emp = mongoose.model('User', userSchema);
24
25 export default Emp;
```

```
> nodemon api/index.js
[nodemon] 1.1.0
[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
DE connected successfully
```

ROUTES:



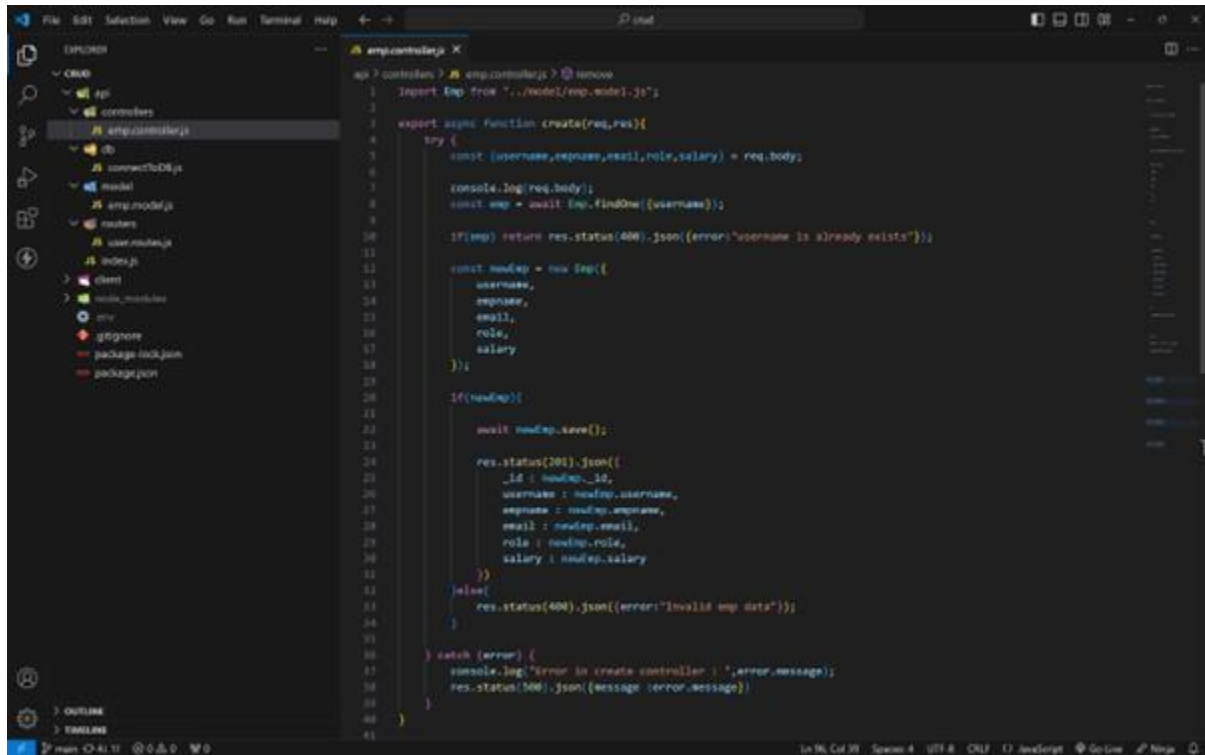
The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows a project structure with folders like 'api', 'controllers', 'db', 'model', 'routes', 'user.routes.js', 'index.js', 'client', 'node_modules', 'src', 'gitignore', 'package-lock.json', and 'package.json'. The 'routes' folder is selected, and the file 'user.routes.js' is open in the editor. The code in 'user.routes.js' defines the routes for the user model. The terminal at the bottom shows the command 'nodemon api/index.js' and the output, which includes the Node.js version (11.0), the command to restart (rs), the paths being watched, the extensions being loaded, and the server starting on port 5000.

```
api > routes > # 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;
```

```
> nodemon api/index.js
[nodemon] 1.1.0
[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
DE connected successfully
```

CONTROLLERS:

CREATE :



The screenshot shows a Visual Studio Code editor with a project structure on the left and a code editor on the right. The project structure includes folders for controllers, models, routes, and views. The code editor displays the `emp.controllers.js` file, which contains the `create` function. The function is an asynchronous function that takes `req` and `res` as arguments. It first checks if a user with the same username already exists in the database. If it does, it returns a 400 status with an error message. If not, it creates a new user object with the provided data and saves it to the database. Finally, it returns a 201 status with the created user object. Error handling is implemented using a `catch` block to log errors and return a 500 status.

```
1 import Emp from '../model/emp.model.js';
2
3 export async function create(req,res){
4   try {
5     const {username,password,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      password,
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        password : newEmp.password,
27        email : newEmp.email,
28        role : newEmp.role,
29        salary : newEmp.salary
30      });
31    }
32    else{
33      res.status(400).json({error:"Invalid emp data"});
34    }
35  }
36  catch (error) {
37    console.log("Error in create controller : ",error.message);
38    res.status(500).json({message :error.message});
39  }
40 }
```

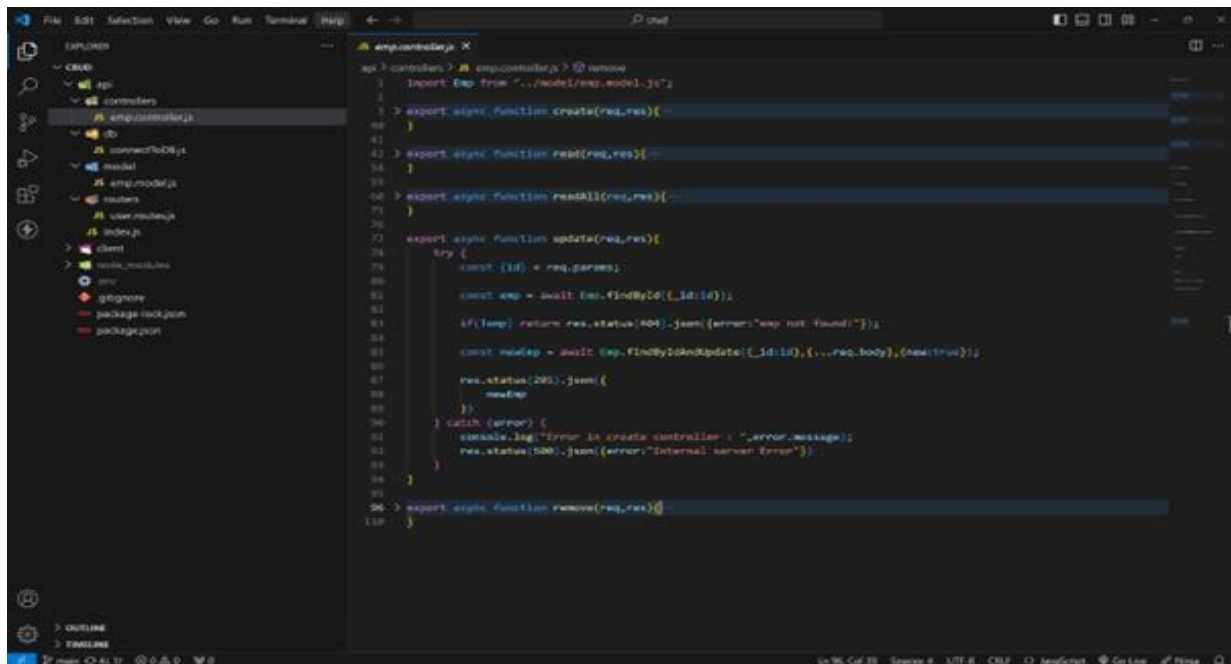
READ ALL:

```
empcontroller.js X
api > controllers > empcontroller.js > @ remove
1 import Emp from "...model/emp-model.js";
2
3 > export async function create(req,res){--
4 }
5
6 > export async function read(req,res){--
7 }
8
9 > export async function readAll(req,res){
10   try {
11     const emps = await Emp.find();
12
13     if(!emps || ! emps.length ) return res.status(404).json({error:" no emp data found!"});
14
15     res.status(201).json({
16       emps
17     })
18   } catch (error) {
19     console.log("Error in create controller : ",error.message);
20     res.status(500).json({error:"Internal server Error"});
21   }
22 }
23
24 > export async function update(req,res){--
25 }
26
27 > export async function remove(req,res){--
28 }
29 }
```

READ ONE :

```
empcontroller.js X
api > controllers > empcontroller.js > @ remove
1 import Emp from "...model/emp-model.js";
2
3 > export async function create(req,res){--
4 }
5
6 > export async function read(req,res){
7   try {
8     const {id} = req.params;
9
10    const emp = await Emp.findById({_id:id});
11
12    if(!emp) return res.status(404).json({error:"emp not found!"});
13
14    res.status(201).json({
15      emp
16    })
17   } catch (error) {
18     console.log("Error in create controller : ",error.message);
19     res.status(500).json({error:"Internal server Error"});
20   }
21 }
22
23 > export async function readAll(req,res){--
24 }
25
26 > export async function update(req,res){--
27 }
28
29 > export async function remove(req,res){--
30 }
31 }
```

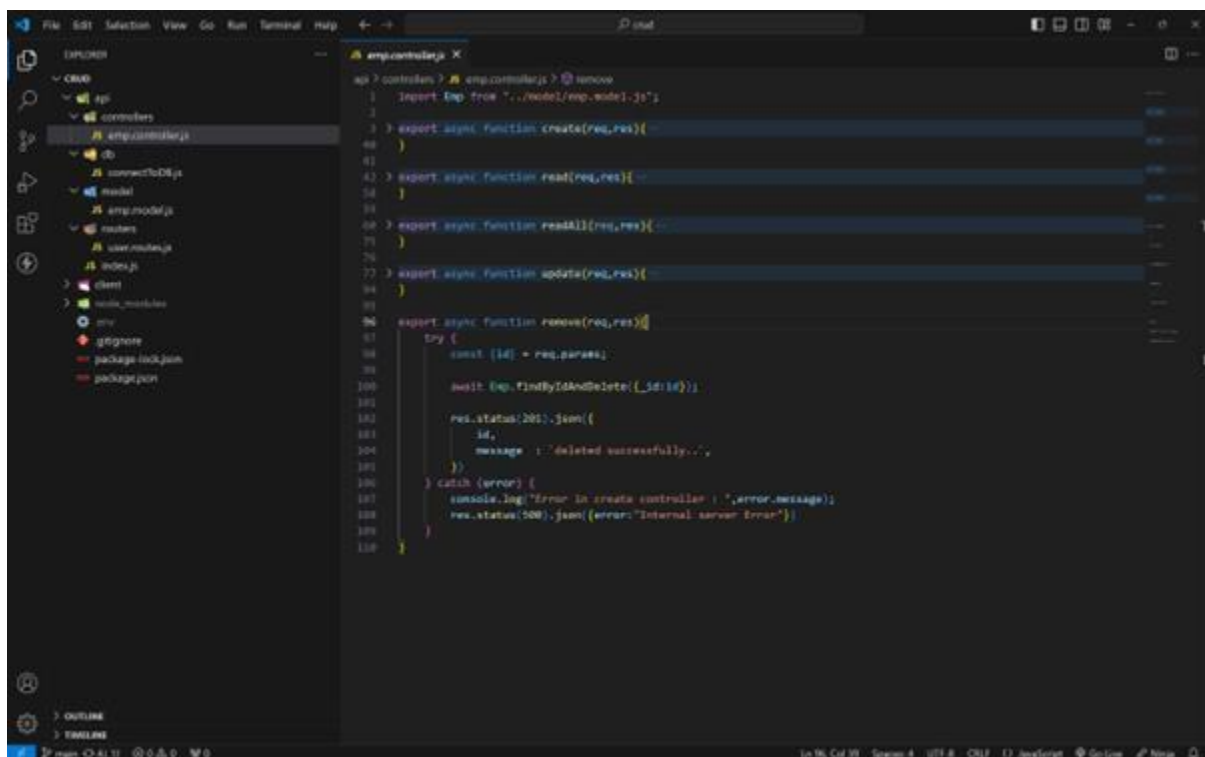
UPDATE :



The screenshot shows a VS Code editor with the file explorer on the left displaying a project structure for an Express.js application. The main editor window shows the `emp.controller.js` file. The code defines several asynchronous functions for handling employee data: `create`, `read`, `readAll`, `update`, and `remove`. The `remove` function is currently selected and highlighted in blue. The code uses `mongoose` for database operations and `res.json()` for sending responses.

```
ap > controller > emp.controller.js > @ remove
1  import Emp from '../model/emp-model.js';
2
3  > export async function create(req,res){
4    }
5
6  > export async function read(req,res){
7    }
8
9  > export async function readAll(req,res){
10   }
11
12  > export async function update(req,res){
13    try {
14      const {id} = req.params;
15      const emp = await Emp.findById(_id:id);
16      if(!emp) return res.status(404).json({error:"emp not found"});
17      const newEmp = await Emp.findByIdAndUpdate(_id:id,{...req.body},{new:true});
18      res.status(200).json({
19        newEmp
20      });
21    } catch (error) {
22      console.log("Error in create controller : ",error.message);
23      res.status(500).json({error:"Internal server Error"});
24    }
25  }
26  > export async function remove(req,res){
27  }
```

DELETE :



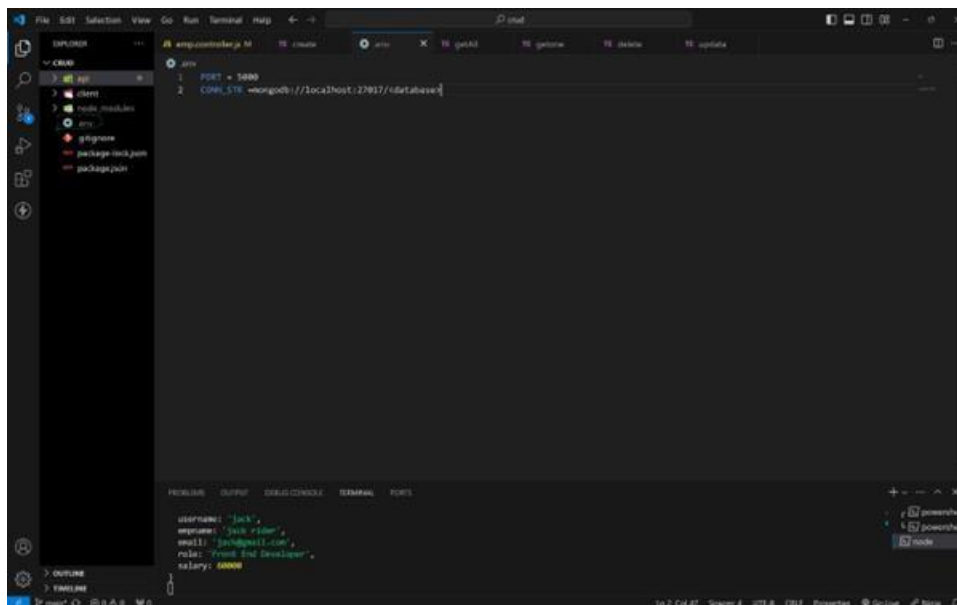
This screenshot shows the same VS Code editor, but the `remove` function in `emp.controller.js` is now fully implemented. The function uses `findByIdAndDelete` to remove an employee from the database and returns a 200 status with a success message. It also includes a catch block for handling errors.

```
ap > controller > emp.controller.js > @ remove
1  import Emp from '../model/emp-model.js';
2
3  > export async function create(req,res){
4    }
5
6  > export async function read(req,res){
7    }
8
9  > export async function readAll(req,res){
10   }
11
12  > export async function update(req,res){
13    try {
14      const {id} = req.params;
15      const emp = await Emp.findById(_id:id);
16      if(!emp) return res.status(404).json({error:"emp not found"});
17      const newEmp = await Emp.findByIdAndUpdate(_id:id,{...req.body},{new:true});
18      res.status(200).json({
19        newEmp
20      });
21    } catch (error) {
22      console.log("Error in create controller : ",error.message);
23      res.status(500).json({error:"Internal server Error"});
24    }
25  }
26  > export async function remove(req,res){
27    try {
28      const {id} = req.params;
29      await Emp.findByIdAndDelete(_id:id);
30      res.status(200).json({
31        id,
32        message : 'deleted successfully...',
33      });
34    } catch (error) {
35      console.log("Error in create controller : ",error.message);
36      res.status(500).json({error:"Internal server Error"});
37    }
38  }
39  }
```

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
- 9 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 .

10. 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)

11. 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...

READ ONE :

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.

READ ALL :

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**

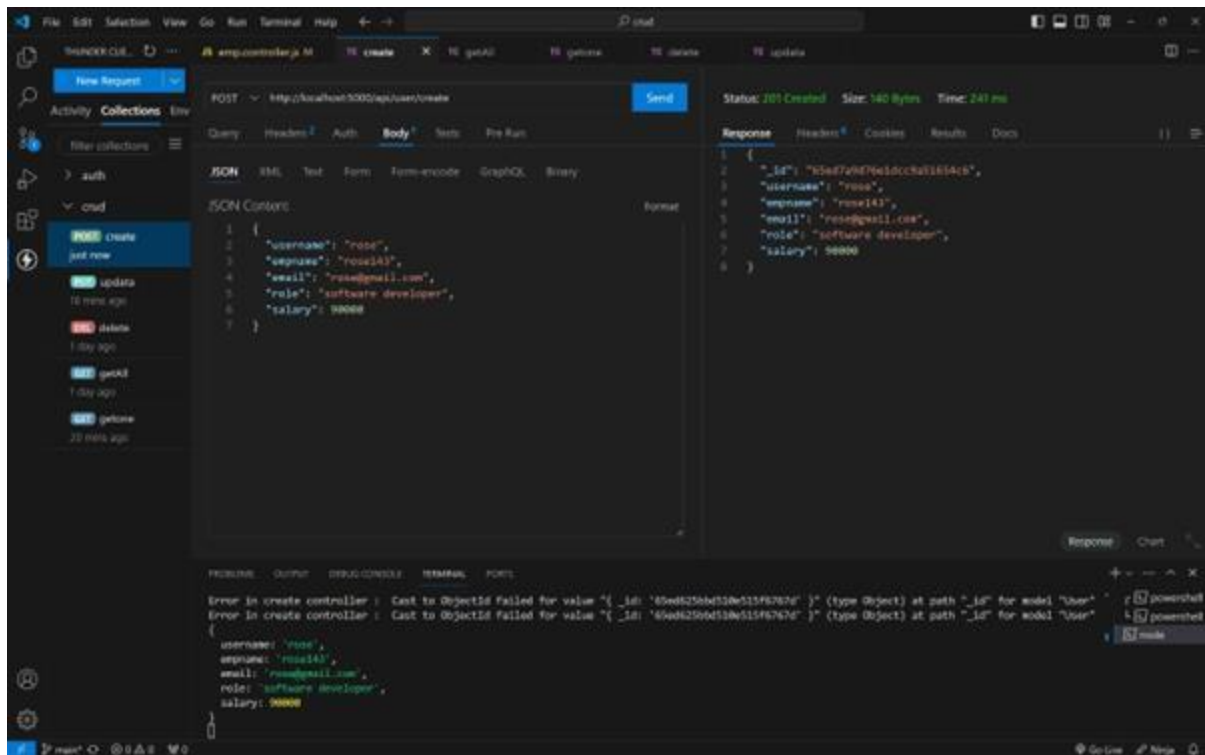
delete

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

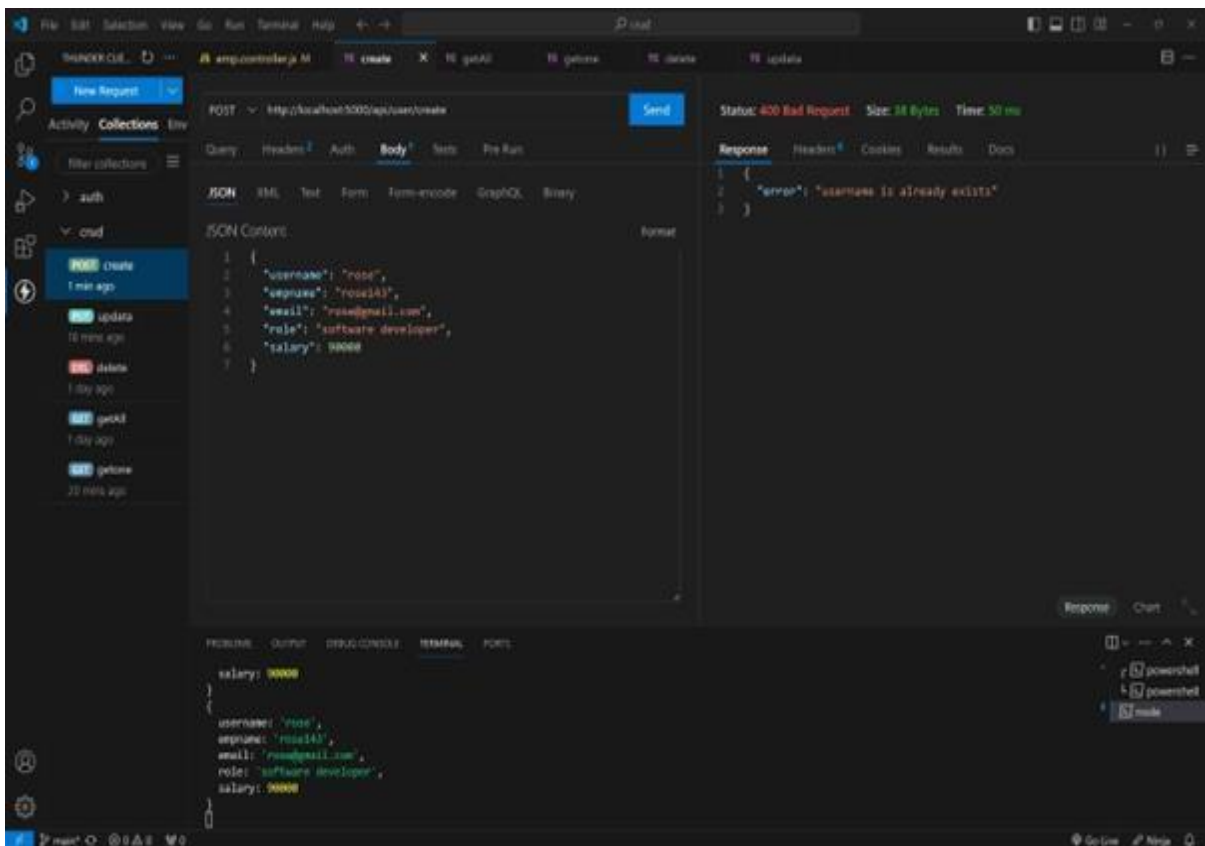
2 . After sending you will deleted successfully as response.

OUTPUT :

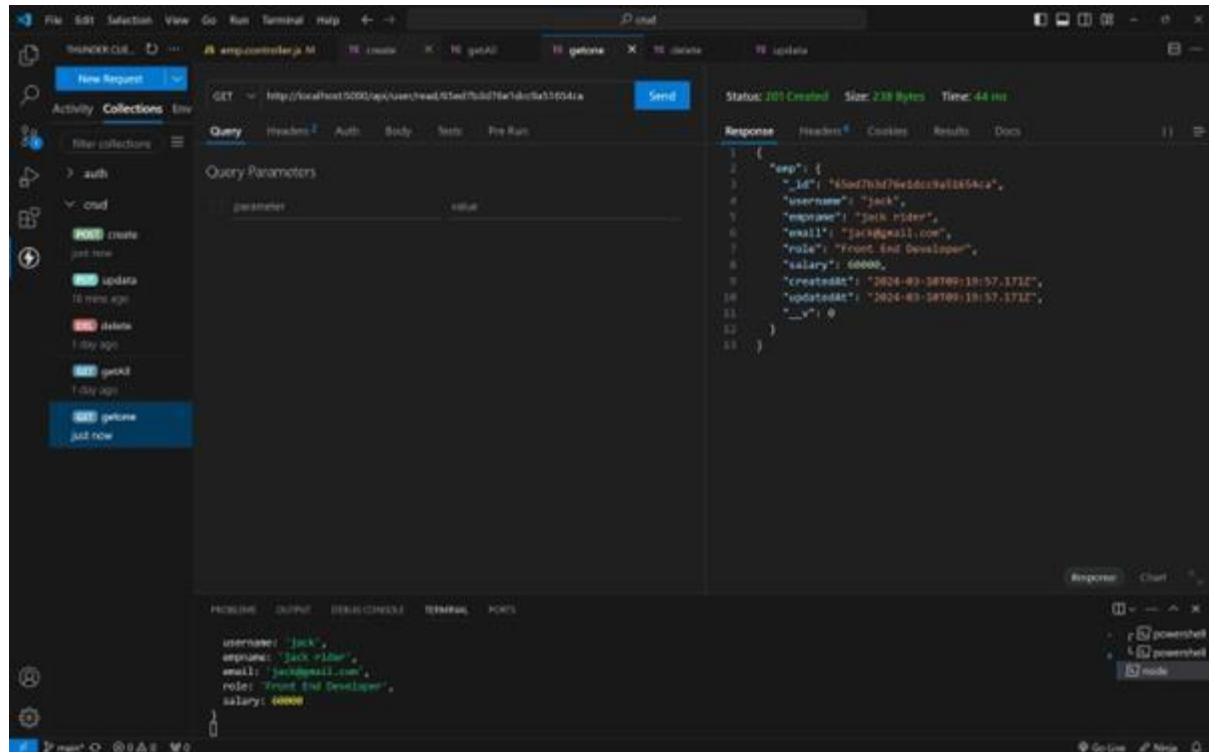
CREATE A NEW USER :



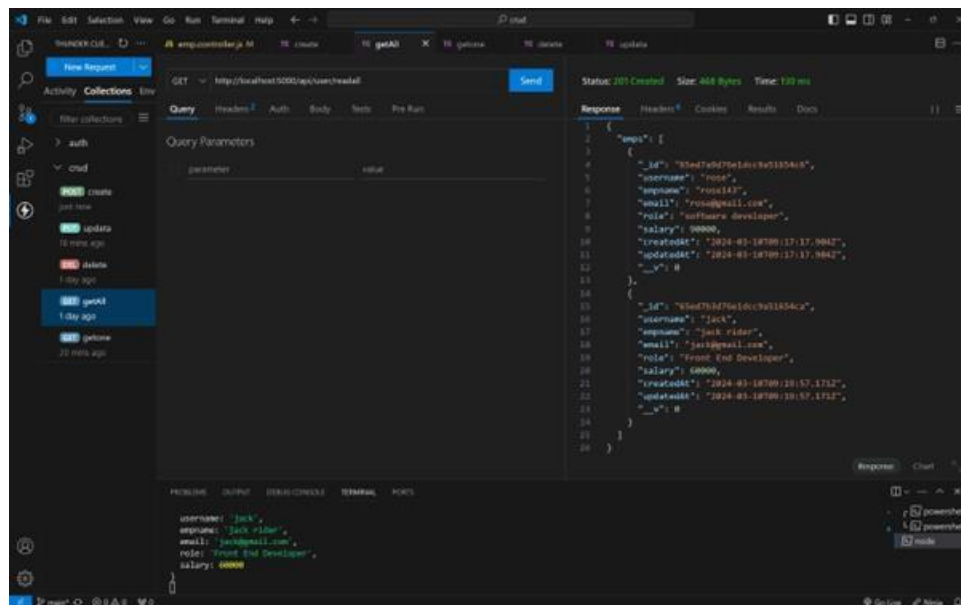
CREATING USER WITH EXISTING USERNAME :



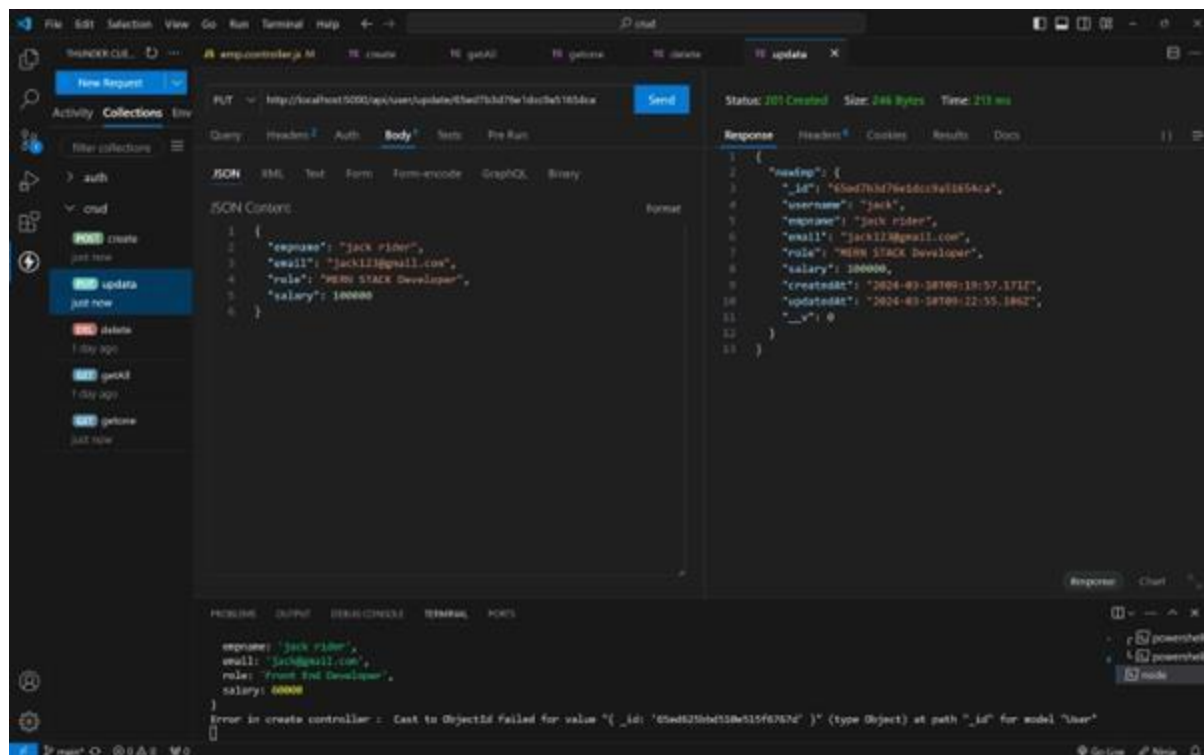
READ ONE :



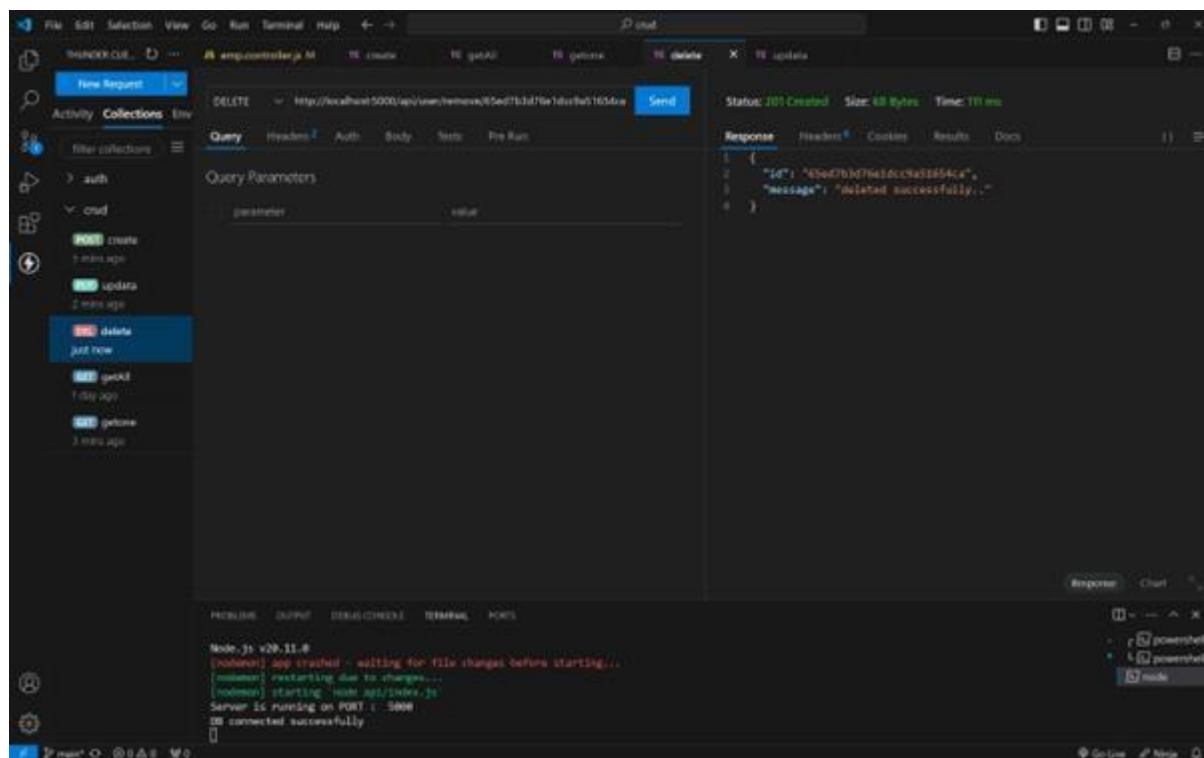
READ ALL :



UPDATE :



DELETE :



SOURCE CODE : <https://github.com/Satya0369>

LIVE DEMO : <https://yesu-crud.onrender.com>