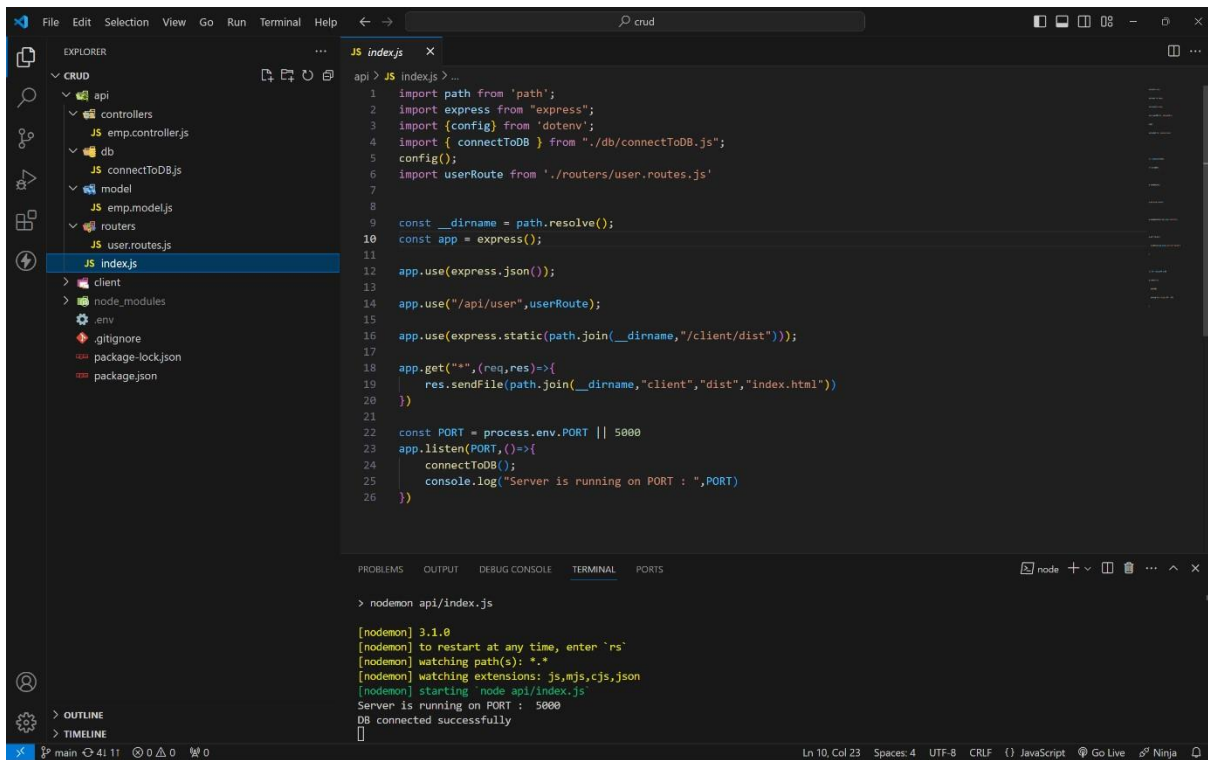


# **Creating a RESTful API using express.js and creating a database and index in MongoDB.**

**Name** : Vutukuri Vinay Sai  
**Email** : [vinaysaivutukuri@gmail.com](mailto:vinaysaivutukuri@gmail.com)  
**Roll no** : 20T91A05A0  
**College Name** : Giet Engineering College

source code :

index.js file :



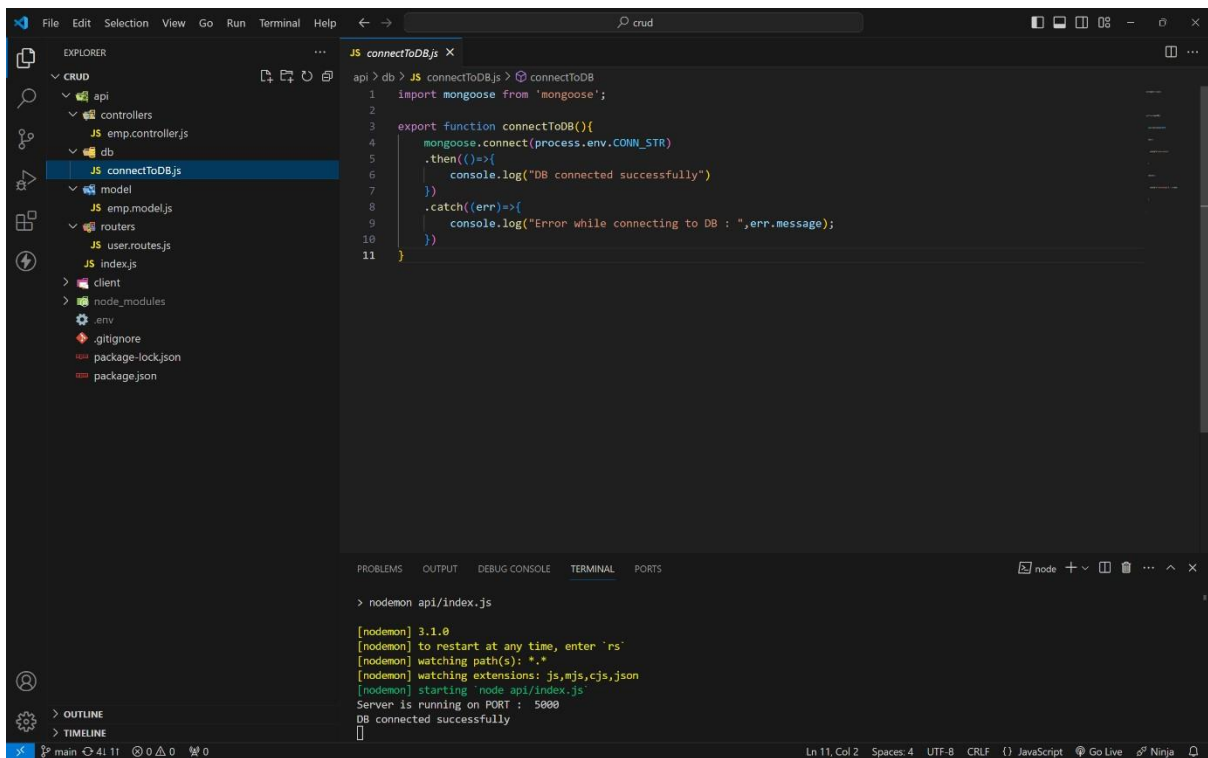
The image shows a screenshot of the Visual Studio Code editor. The Explorer panel on the left shows a project structure with folders like 'api', 'controllers', 'db', 'model', 'routers', and 'client'. The 'api' folder is expanded, showing files like 'index.js', 'connectToDB.js', 'emp.controller.js', 'emp.model.js', 'user.routes.js', and 'client'. The 'index.js' file is selected and its content is displayed in the main editor area. The code is a Node.js Express application that connects to a database and serves static files. The terminal at the bottom shows the command 'nodemon api/index.js' and the output, which includes the message 'Server is running on PORT : 5000' and 'DB 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 './routers/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 :



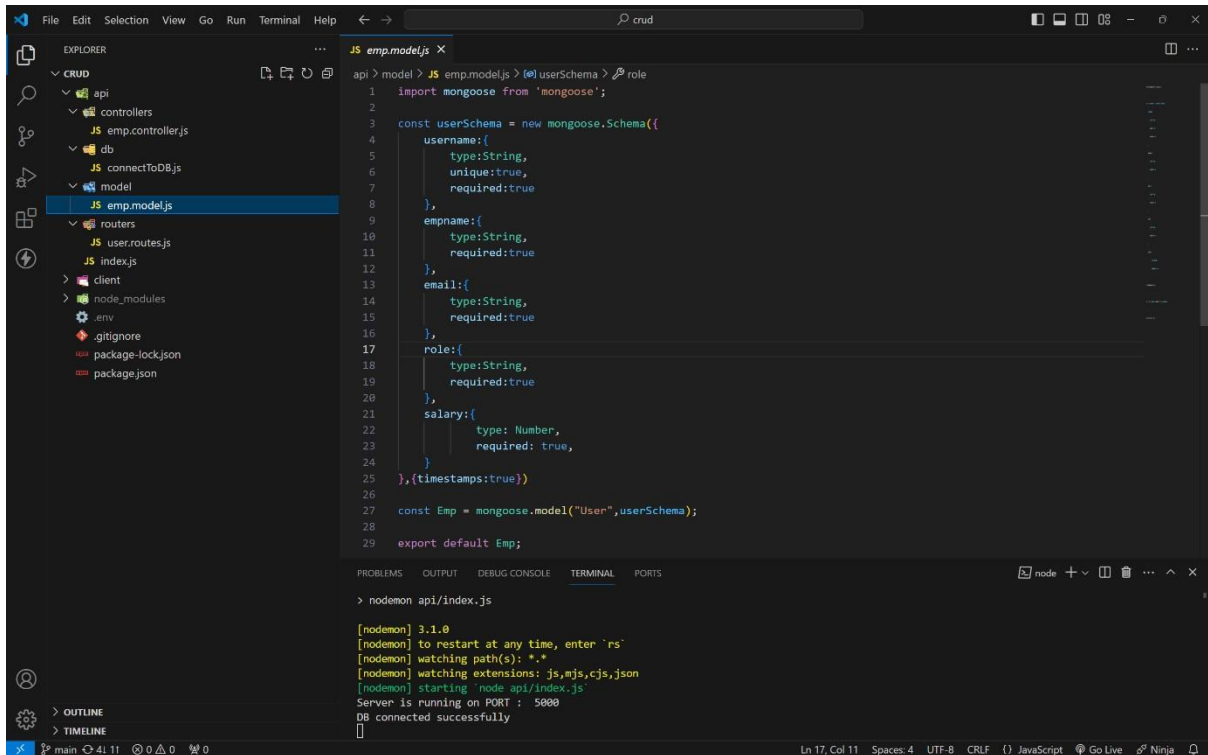
The screenshot shows the VS Code interface with the Explorer panel on the left displaying the project structure. The file `connectToDB.js` is selected under the `api/db` directory. The main editor displays the code for `connectToDB.js`, which imports `mongoose` and defines a `connectToDB` function that connects to a MongoDB database using `process.env.CONN_STR`. The function logs a success message or an error message. The bottom panel shows the terminal with the command `nodemon api/index.js` and its output, which includes the message `DB connected successfully`.

```
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 }
```

```
> 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 :



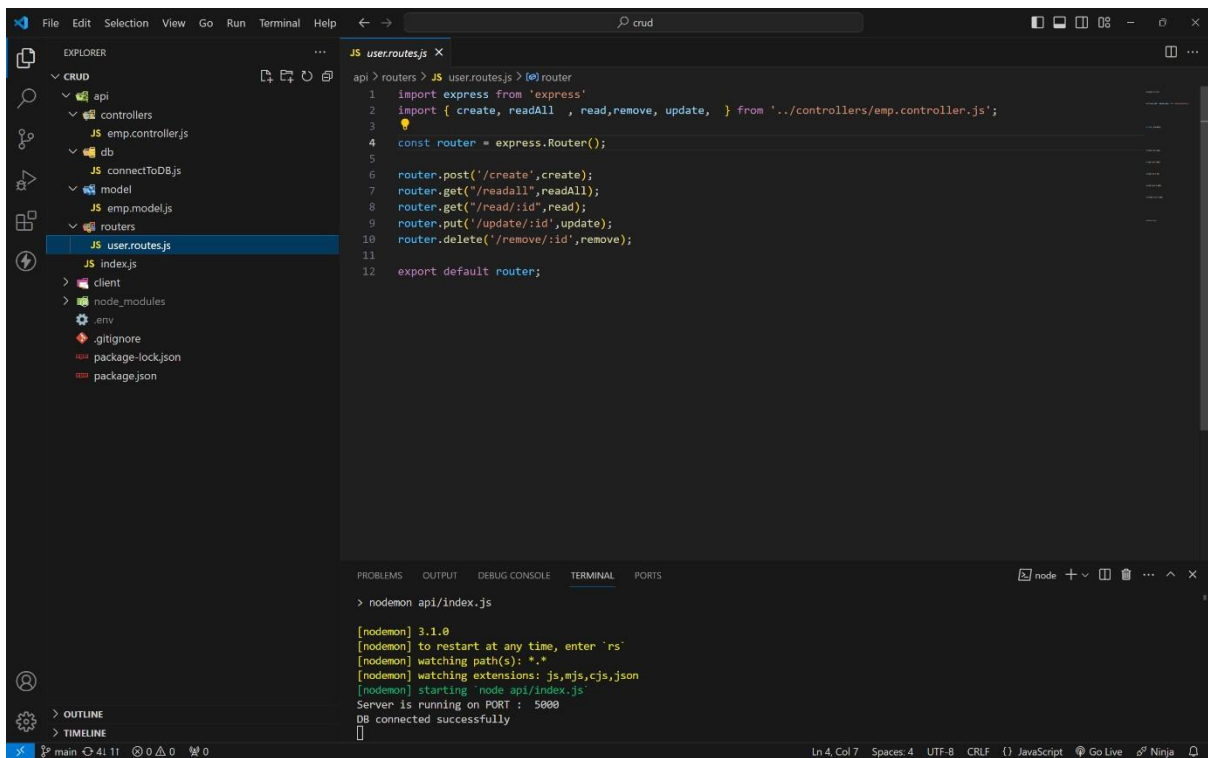
The screenshot shows the VS Code interface with the Explorer panel on the left displaying the project structure. The file `emp.model.js` is selected under the `api/model` directory. The main editor displays the code for `emp.model.js`, which imports `mongoose` and defines a `userSchema` for a user model. The schema includes fields for `username`, `empname`, `email`, `role`, and `salary`. The model is then created using `mongoose.model` and exported as `Emp`. The bottom panel shows the terminal with the command `nodemon api/index.js` and its output, which includes the message `DB connected successfully`.

```
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;
```

```
> 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:

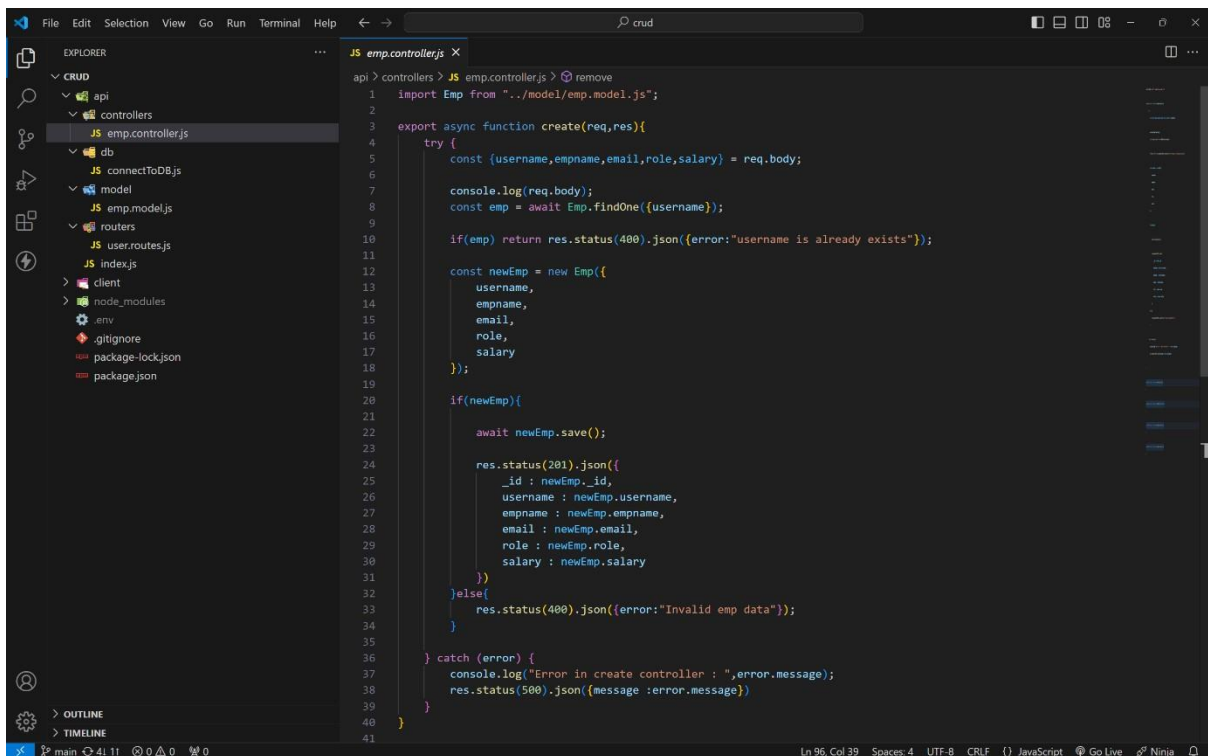
```
1 import express from 'express'
2 import { create, readAll, read, remove, update, } from '../controllers/emp.controller.js';
3
4 const router = express.Router();
5
6 router.post('/create', create);
7 router.get("/readall", readAll);
8 router.get("/read/:id", read);
9 router.put("/update/:id", update);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

The terminal at the bottom shows the output of nodemon running 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
```

## CONTROLLERS :

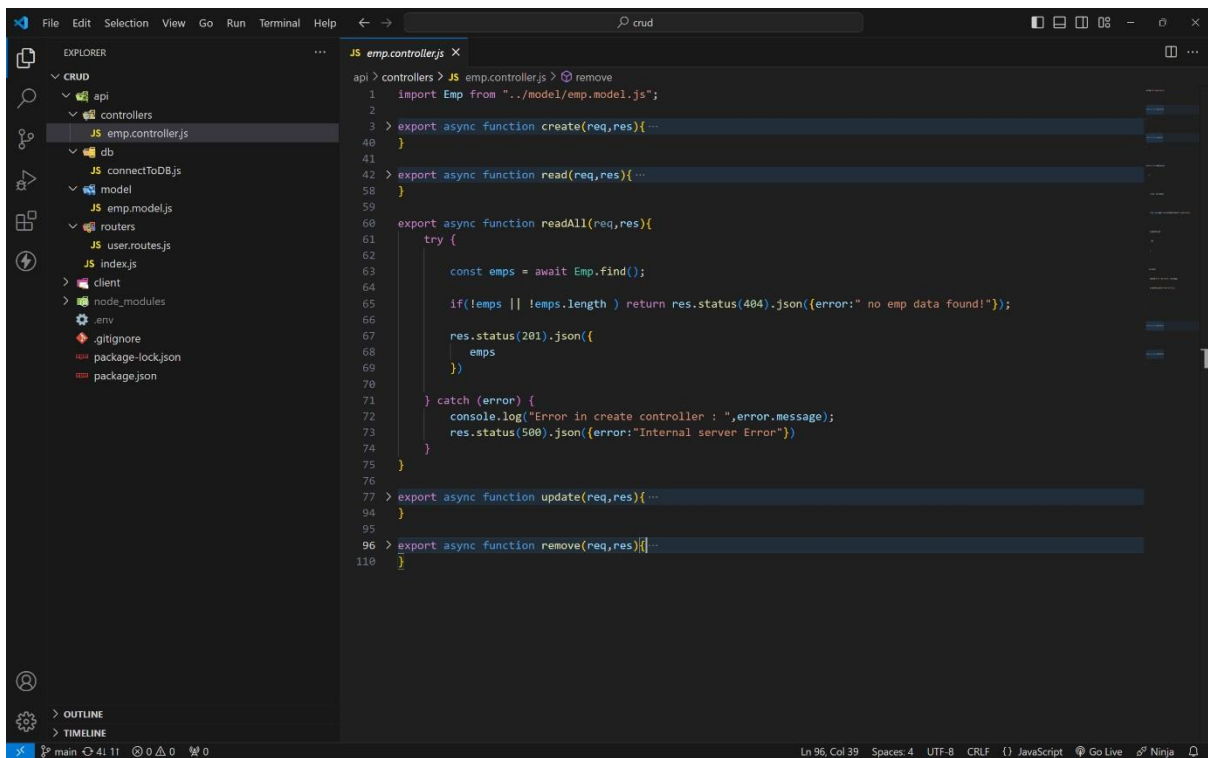
### CREATE :



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:

```
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
```

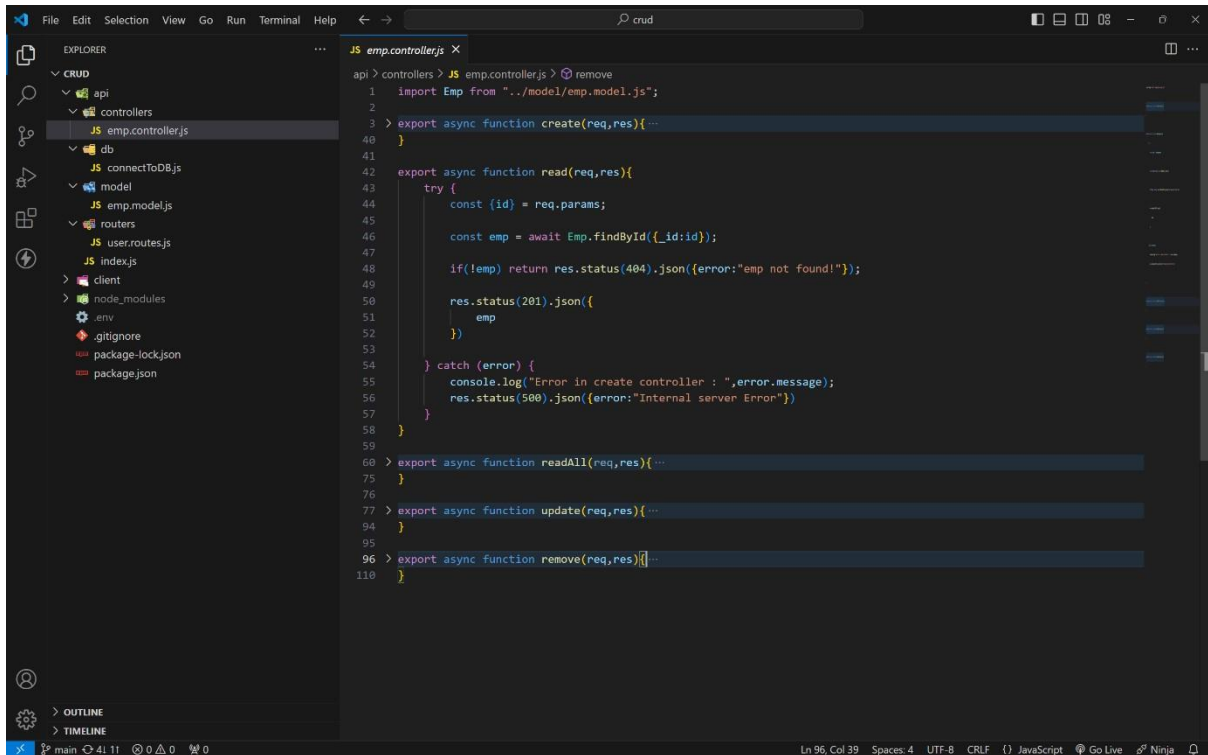
## READALL:



The screenshot shows the VS Code editor with the file explorer on the left and the code editor on the right. The file explorer shows a project structure with a 'crud' folder containing 'api', 'controllers', 'db', 'connectToDB.js', 'model', 'emp.model.js', 'routers', 'user.routes.js', 'index.js', 'client', 'node\_modules', '.env', '.gitignore', 'package-lock.json', and 'package.json'. 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 > export async function update(req,res){...
84 }
85
96 > export async function remove(req,res){...
110 }
```

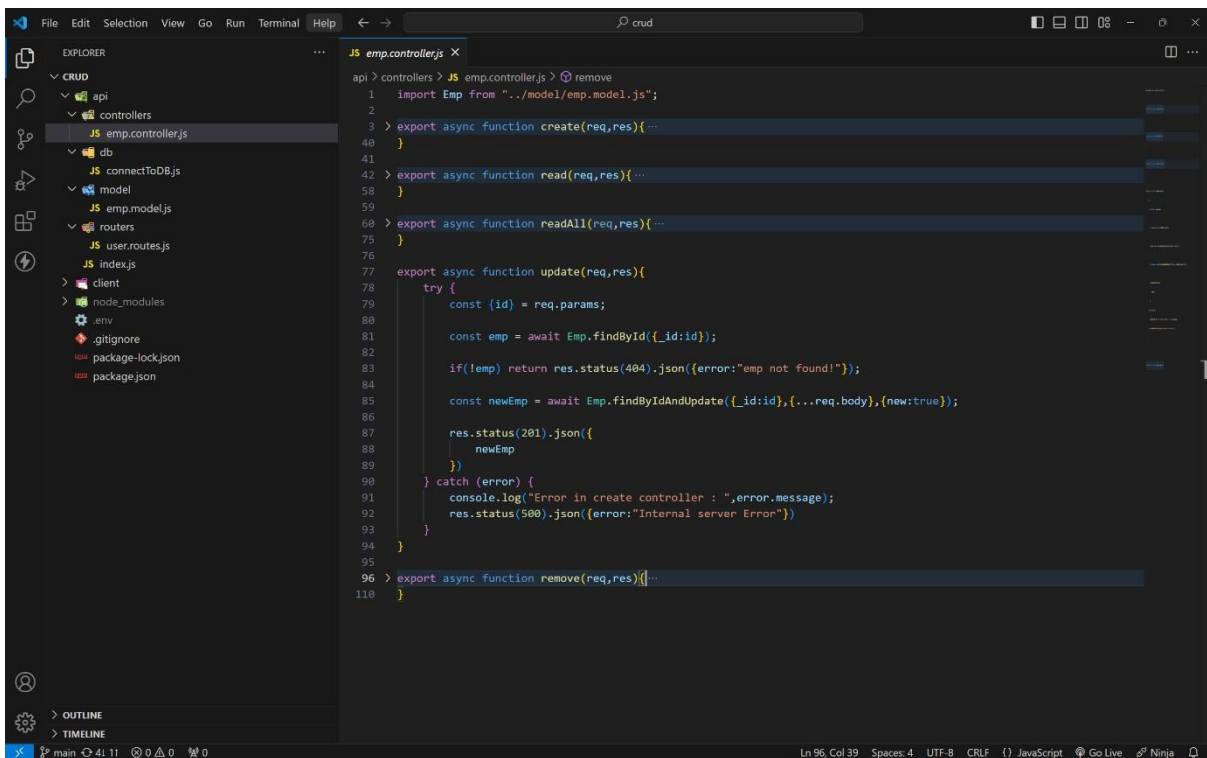
## READONE :



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 a project structure with a 'crud' folder containing 'api', 'controllers', 'db', 'connectToDB.js', 'model', 'emp.model.js', 'routers', 'user.routes.js', 'index.js', 'client', 'node\_modules', '.env', '.gitignore', 'package-lock.json', and 'package.json'. 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
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
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){...
110 }
```

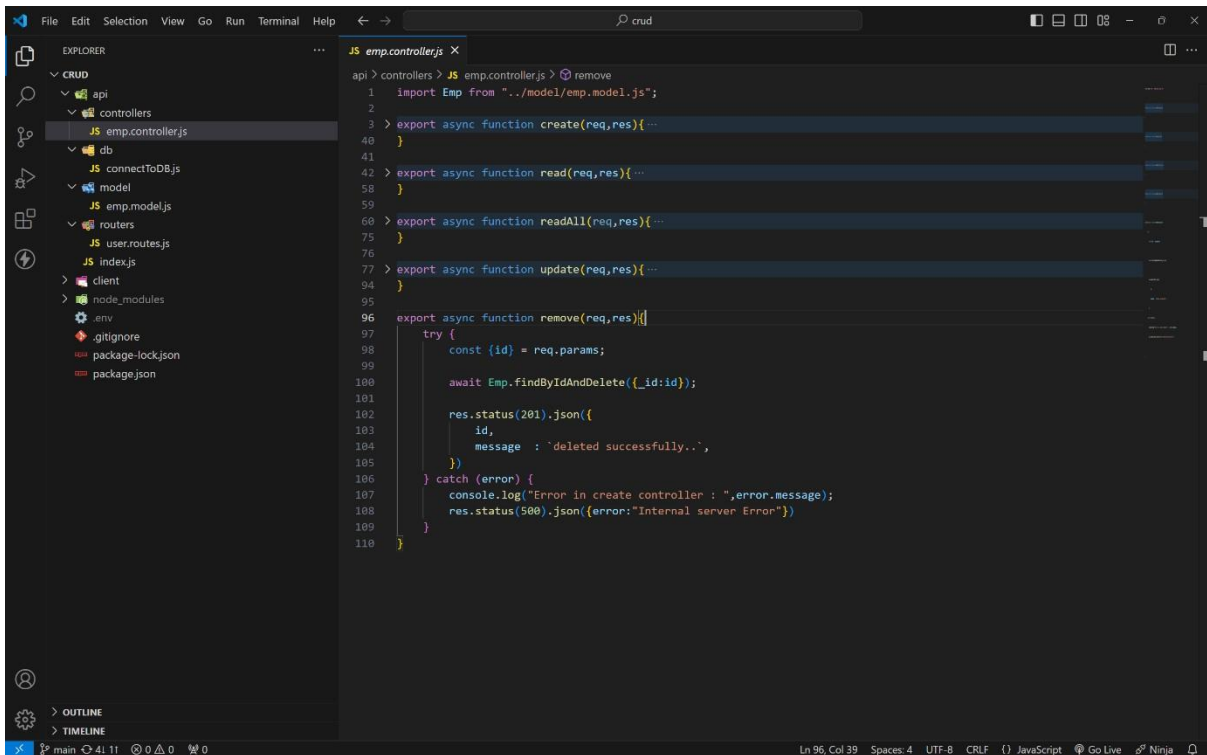
## UPDATE :



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', 'user.routes.js', 'index.js', 'client', 'node\_modules', '.env', '.gitignore', 'package-lock.json', and 'package.json'. The code editor displays the 'emp.controller.js' file, which contains several async functions: 'create', 'read', 'readAll', 'update', and 'remove'. The 'update' function is highlighted, showing its implementation. It uses 'req.params' to get the 'id', finds the employee by ID, and updates it with the provided body. It returns a 201 status code with the updated employee object. Error handling is included for cases where the employee is not found (404) or an internal server error occurs (500).

```
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 :

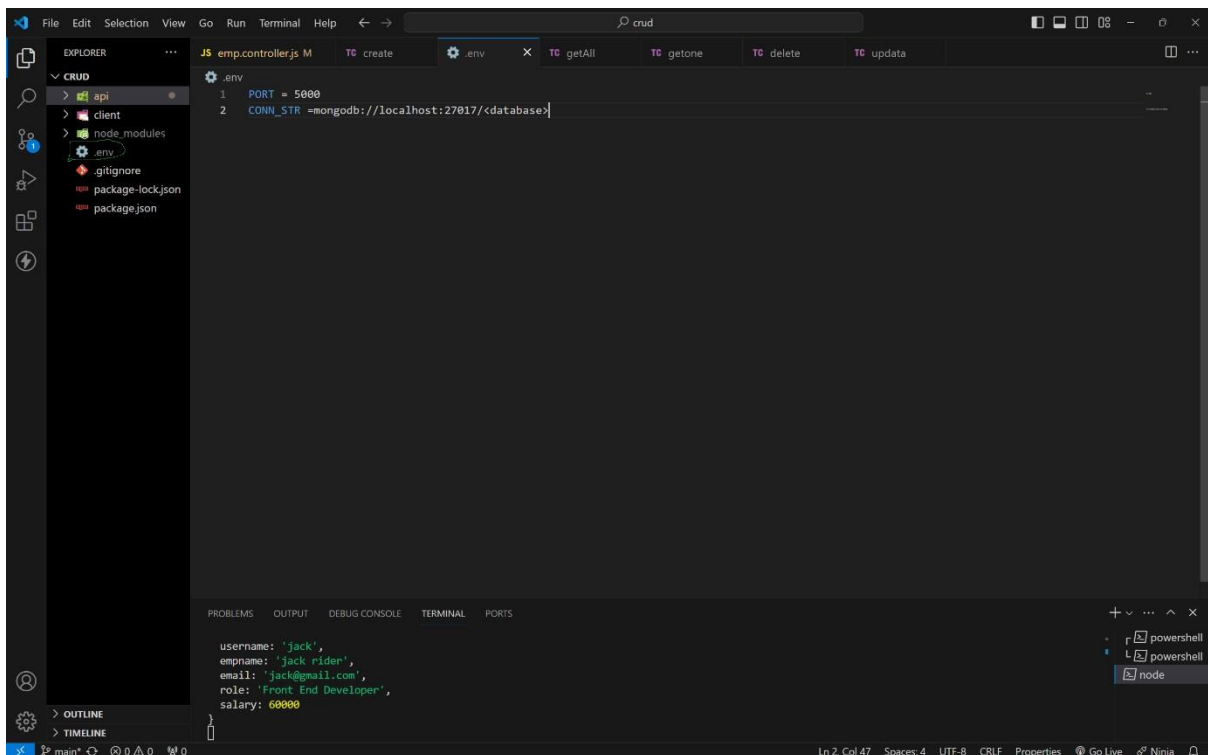


The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the same project structure as the previous screenshot. The code editor displays the 'emp.controller.js' file, which contains several async functions: 'create', 'read', 'readAll', 'update', and 'remove'. The 'remove' function is highlighted, showing its implementation. It uses 'req.params' to get the 'id', finds the employee by ID, and deletes it. It returns a 201 status code with a success message. Error handling is included for cases where the employee is not found (404) or an internal server error occurs (500).

```
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.      `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

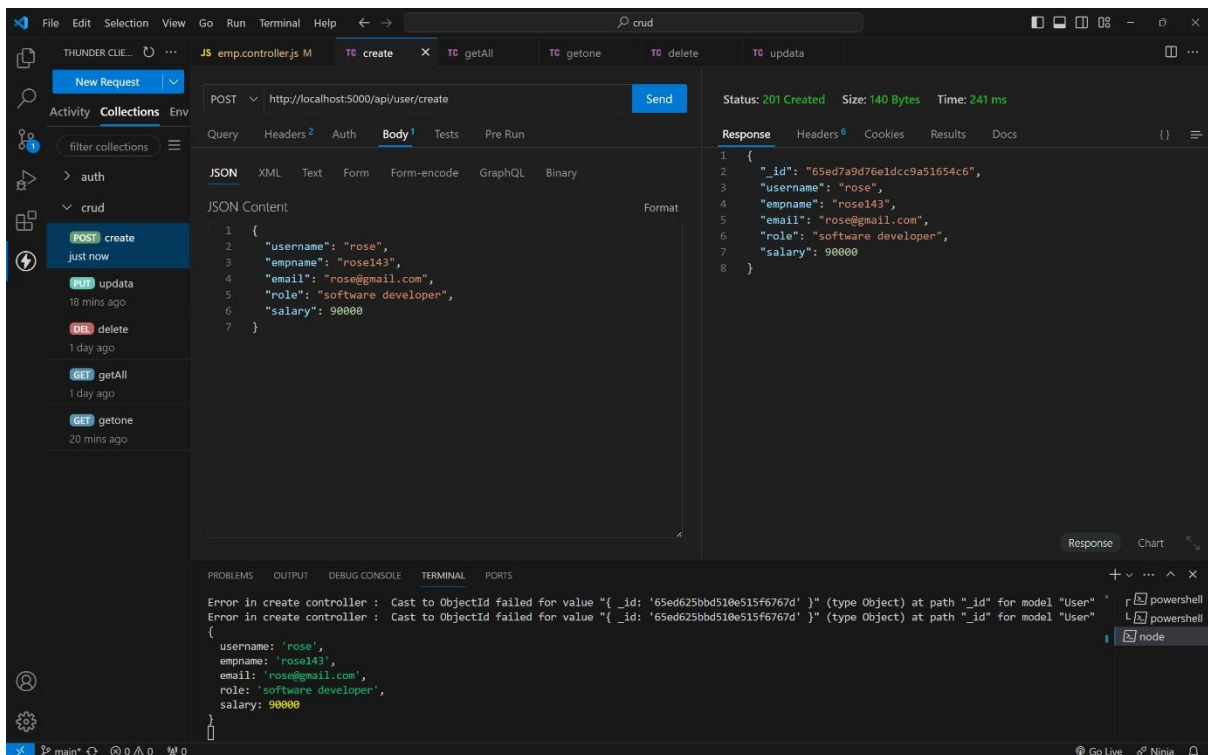
url as

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

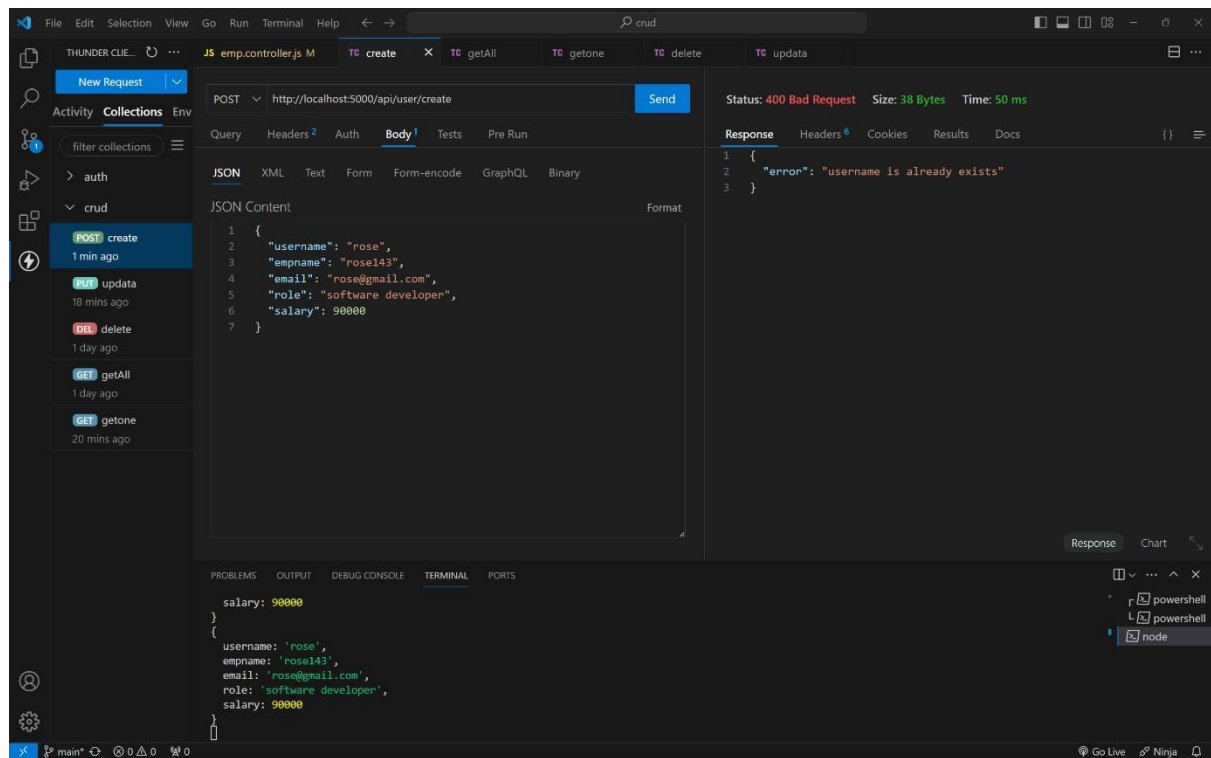
2 . After sending you will deleted successfully as response.

OUTPUT :

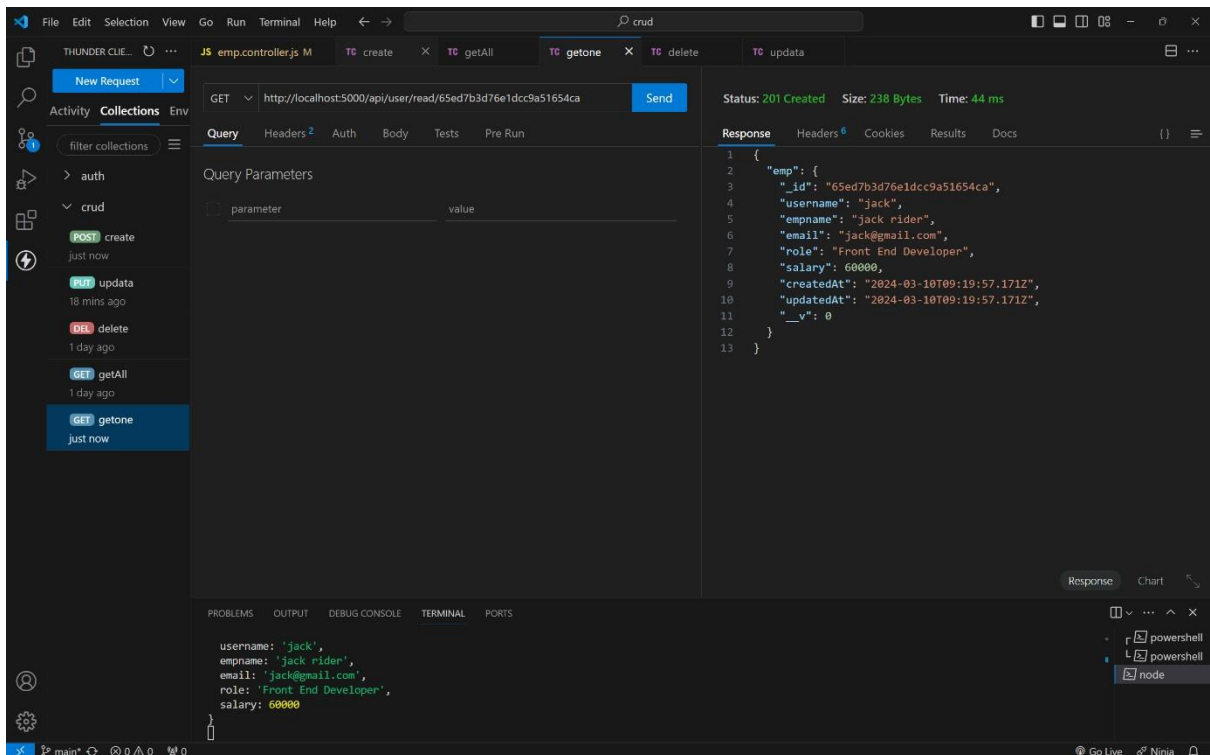
CREATE A NEW USER :



## CREATING USER WITH EXISTING USERNAEM :



## READONE :



## READ ALL :

Thunder Client interface showing a GET request to `http://localhost:5000/api/user/readall`. The response is a JSON array of two employee objects.

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

**Response:**

```
1 {
2   "emps": [
3     {
4       "id": "65ed7a9d76e1dcc9a51654c6",
5       "username": "rose",
6       "empname": "rose143",
7       "email": "rose@gmail.com",
8       "role": "software developer",
9       "salary": 90000,
10      "createdAt": "2024-03-10T09:17:17.984Z",
11      "updatedAt": "2024-03-10T09:17:17.984Z",
12      "__v": 0
13    },
14    {
15      "id": "65ed7b3d76e1dcc9a51654ca",
16      "username": "jack",
17      "empname": "jack rider",
18      "email": "jack@gmail.com",
19      "role": "Front End Developer",
20      "salary": 60000,
21      "createdAt": "2024-03-10T09:19:57.171Z",
22      "updatedAt": "2024-03-10T09:19:57.171Z",
23      "__v": 0
24    }
25  ]
26 }
```

**Terminal Output:**

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

## UPDATE :

Thunder Client interface showing a PUT request to `http://localhost:5000/api/user/update/65ed7b3d76e1dcc9a51654ca`. The response is a JSON object representing the updated employee.

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

**Body:**

```
1 {
2   "empname": "jack rider",
3   "email": "jack123@gmail.com",
4   "role": "MERN STACK Developer",
5   "salary": 100000
6 }
```

**Response:**

```
1 {
2   "newEmp": {
3     "id": "65ed7b3d76e1dcc9a51654ca",
4     "username": "jack",
5     "empname": "jack rider",
6     "email": "jack123@gmail.com",
7     "role": "MERN STACK Developer",
8     "salary": 100000,
9     "createdAt": "2024-03-10T09:19:57.171Z",
10    "updatedAt": "2024-03-10T09:22:55.106Z",
11    "__v": 0
12  }
13 }
```

**Terminal Output:**

```
empname: 'jack rider',
email: 'jack@gmail.com',
role: 'Front End Developer',
salary: 60000
}
Error in create controller : Cast to ObjectId failed for value "{ '_id': '65ed625bbd510e515f6767d' }" (type Object) at path "_id" for model "User"
```

## DELETE :

The screenshot displays the Thunder Client interface with a REST client tab titled 'crud'. The active request is a DELETE method to the URL `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`. The 'Send' button is visible. 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 left sidebar shows a collection of requests, including 'create', 'update', 'delete', 'getAll', and 'getone'. The bottom panel shows the terminal output of a Node.js application running on port 5000, indicating a successful database connection.

Thunder Client interface showing a REST client tab titled "crud". The active request is a DELETE method to the URL `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 left sidebar shows a collection of requests, including "create", "update", "delete", "getAll", and "getone". The bottom panel shows the terminal output of a Node.js application running on port 5000, indicating a successful database connection.

## OUTPUT : CREAT

createupdatereaddelete

VINAYSAL

VINAYSAL VUT

vinaysaivutukuri

FRONT-END-DE

45000

submit

operation

get all data

clear data

```
[
  {
    "_id": "65edc3a4df619a9d61c72b2b",
    "username": "VINAYSAL",
    "empname": "VINAYSAL VUTUKURI",
    "email": "vinaysaivutukuri@gmail.com",
    "role": "FRONT-END-DEVELOPER",
    "salary": 45000
  }
]
```

## UPDATE

createupdatereaddelete

65eddc7cdfc19a9

VINAYSAL VUT

vinaysaivutukuri

FRONT-END-DE

45000

submit

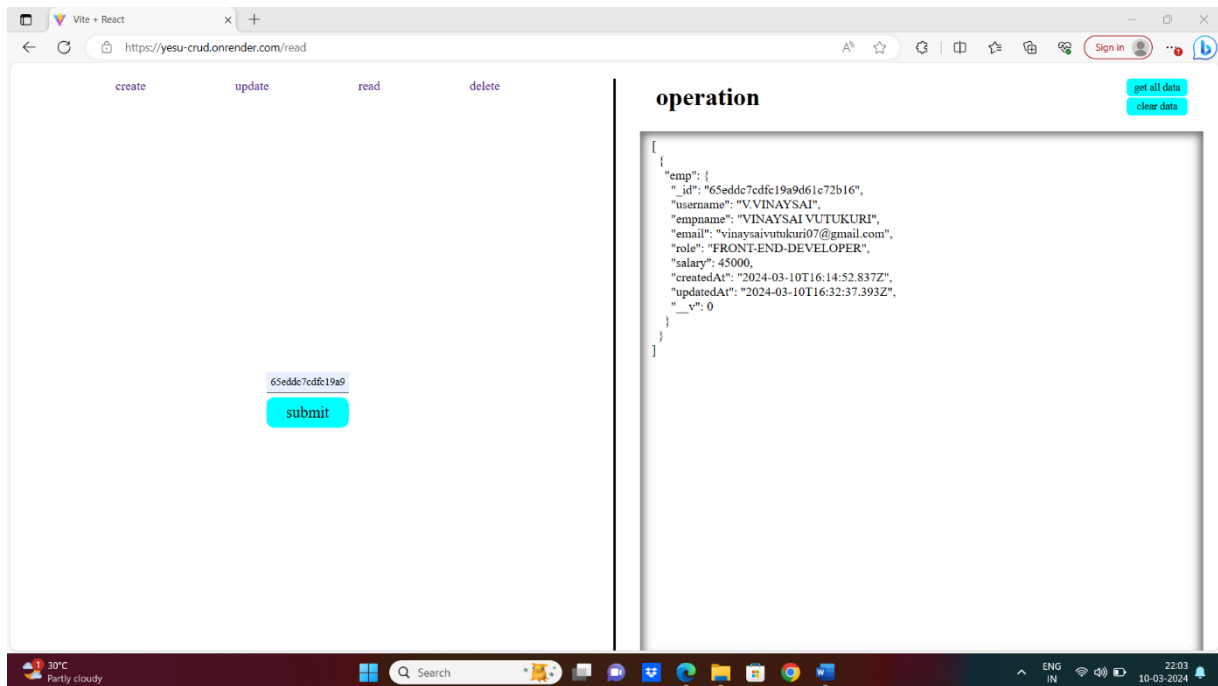
operation

get all data

clear data

```
{
  "newEmp": {
    "_id": "65eddc7cdfc19a9d61c72b16",
    "username": "VINAYSAL",
    "empname": "VINAYSAL VUTUKURI",
    "email": "vinaysaivutukuri07@gmail.com",
    "role": "FRONT-END-DEVELOPER",
    "salary": 45000,
    "createdAt": "2024-03-10T16:14:52.837Z",
    "updatedAt": "2024-03-10T16:32:37.393Z",
    "__v": 0
  }
}
```

# READ



# DELETE

