

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

Name : Repalle Akshaya

Email Id : akshaya.repalle@gmail.com

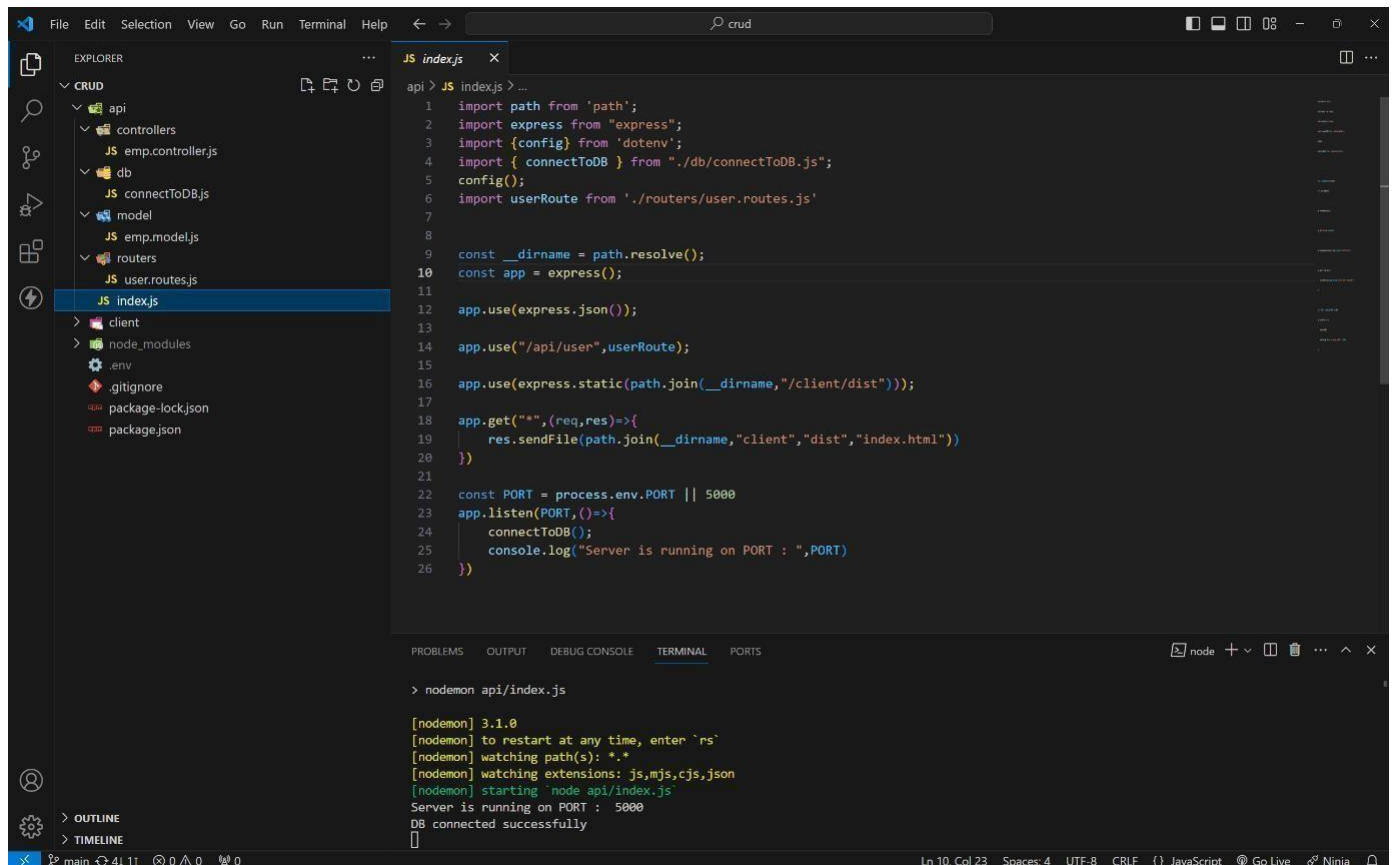
Phone no : 6301142009

Roll NO : 20HU1A0402

College Name : Chebrolu Engineering College

source code :

index.js file :



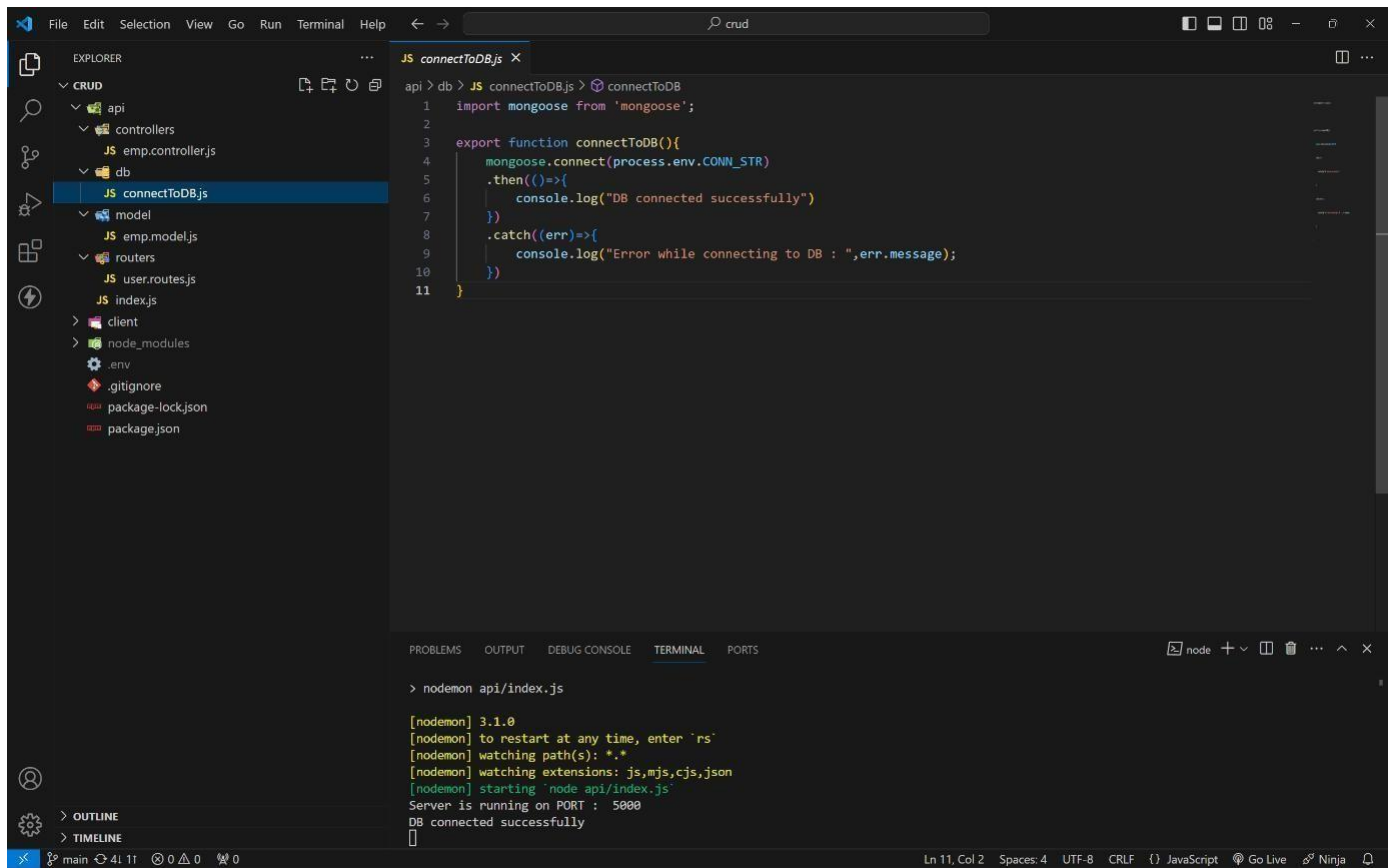
The screenshot shows a VS Code editor with a project named 'crud'. The Explorer sidebar on the left shows the file structure, with 'index.js' selected under the 'api' directory. The main editor displays the content of 'index.js', which is a Node.js application using Express.js. The code imports necessary modules, connects to a MongoDB database, and sets up routes for API endpoints and static client files. 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 connection is successful.

```
api > JS 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 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 :



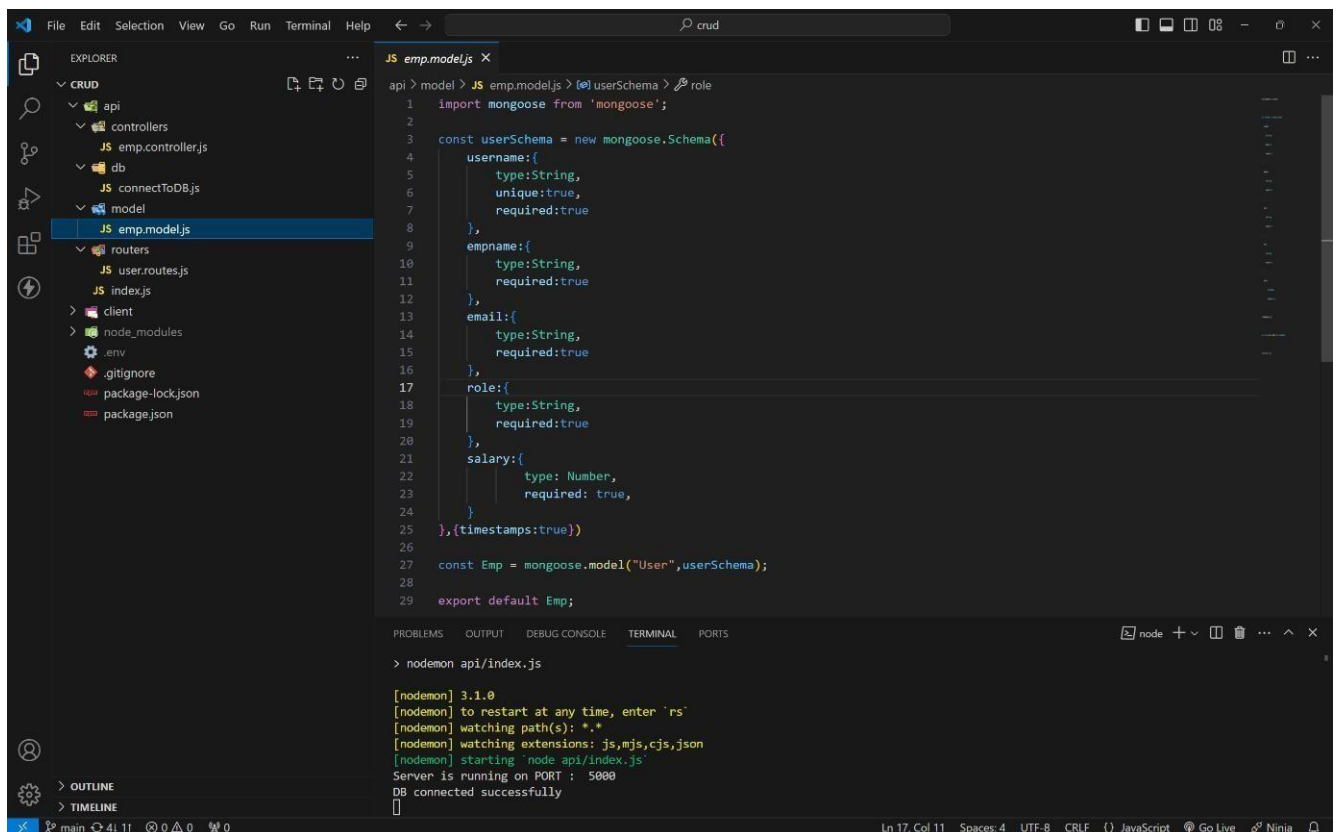
The screenshot shows a VS Code editor with a project named 'crud'. The Explorer panel on the left shows the file structure: api > db > JS connectToDB.js. The main editor displays the code for connectToDB.js, which imports mongoose and exports a connectToDB function. The function uses mongoose.connect with process.env.CONN_STR, logs a success message, and catches errors. The terminal at the bottom shows the command 'nodemon api/index.js' and the output of the application, including 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 :



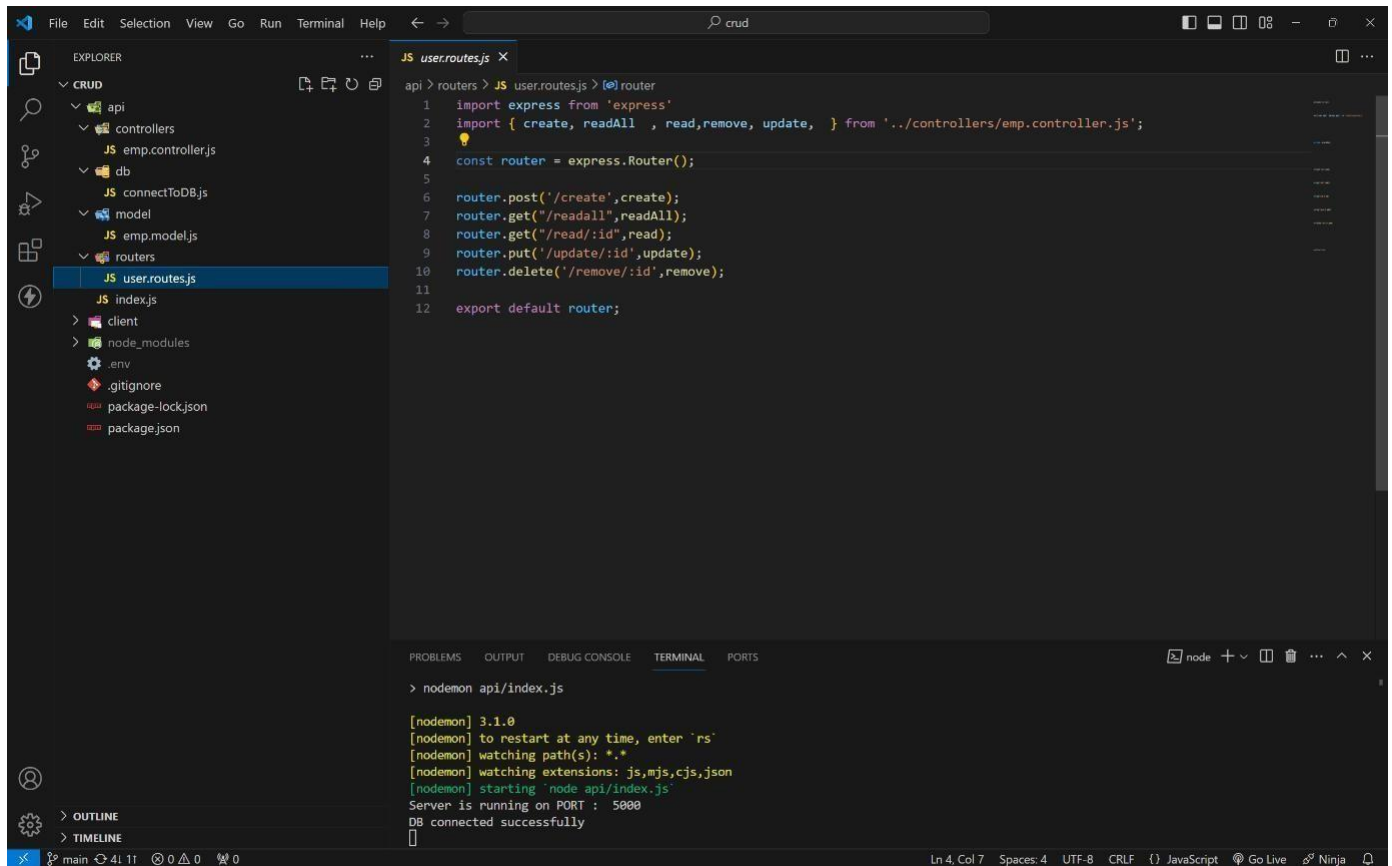
The screenshot shows a VS Code editor with the same project. The Explorer panel shows the file structure: api > model > JS emp.model.js. The main editor displays the code for emp.model.js, which imports mongoose and defines a userSchema with fields: username (String, unique, required), empname (String, required), email (String, required), role (String, required), and salary (Number, required). It also defines a timestamps field. The schema is then used to create a mongoose model 'User' and exported as 'Emp'.

```
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 a VS Code editor with the Explorer sidebar on the left. The 'api' folder is expanded, showing subfolders 'controllers', 'db', 'model', and 'routes'. The 'routes' folder is selected, and 'user.routes.js' is highlighted. The main editor displays the content of 'user.routes.js'.

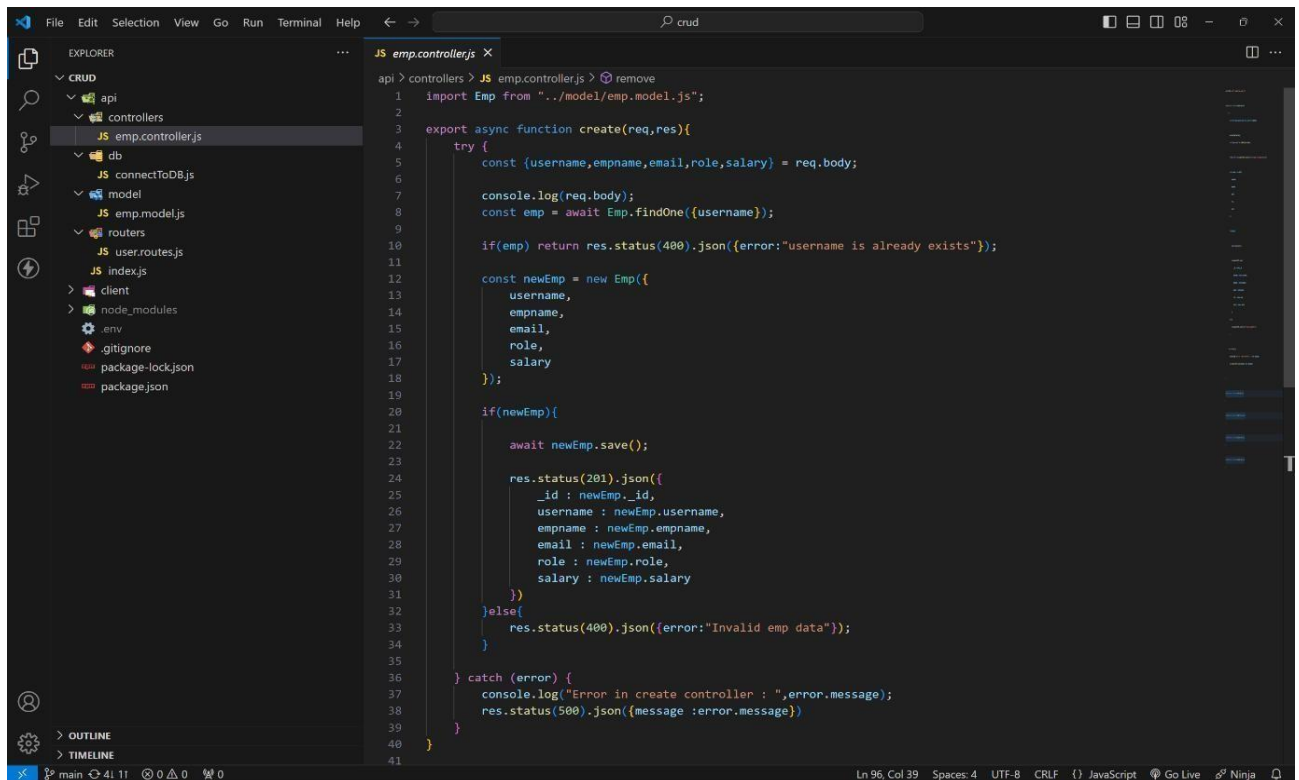
```
api > routes > JS user.routes.js > @ router
1  import express from 'express'
2  import { create, readAll, read, remove, update, } from '../controllers/emp.controller.js';
3
4  const router = express.Router();
5
6  router.post('/create', create);
7  router.get("/readall", readAll);
8  router.get("/read/:id", 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 command 'nodemon api/index.js' being executed. The output indicates that the server is running on port 5000 and is connected successfully.

```
> 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
```

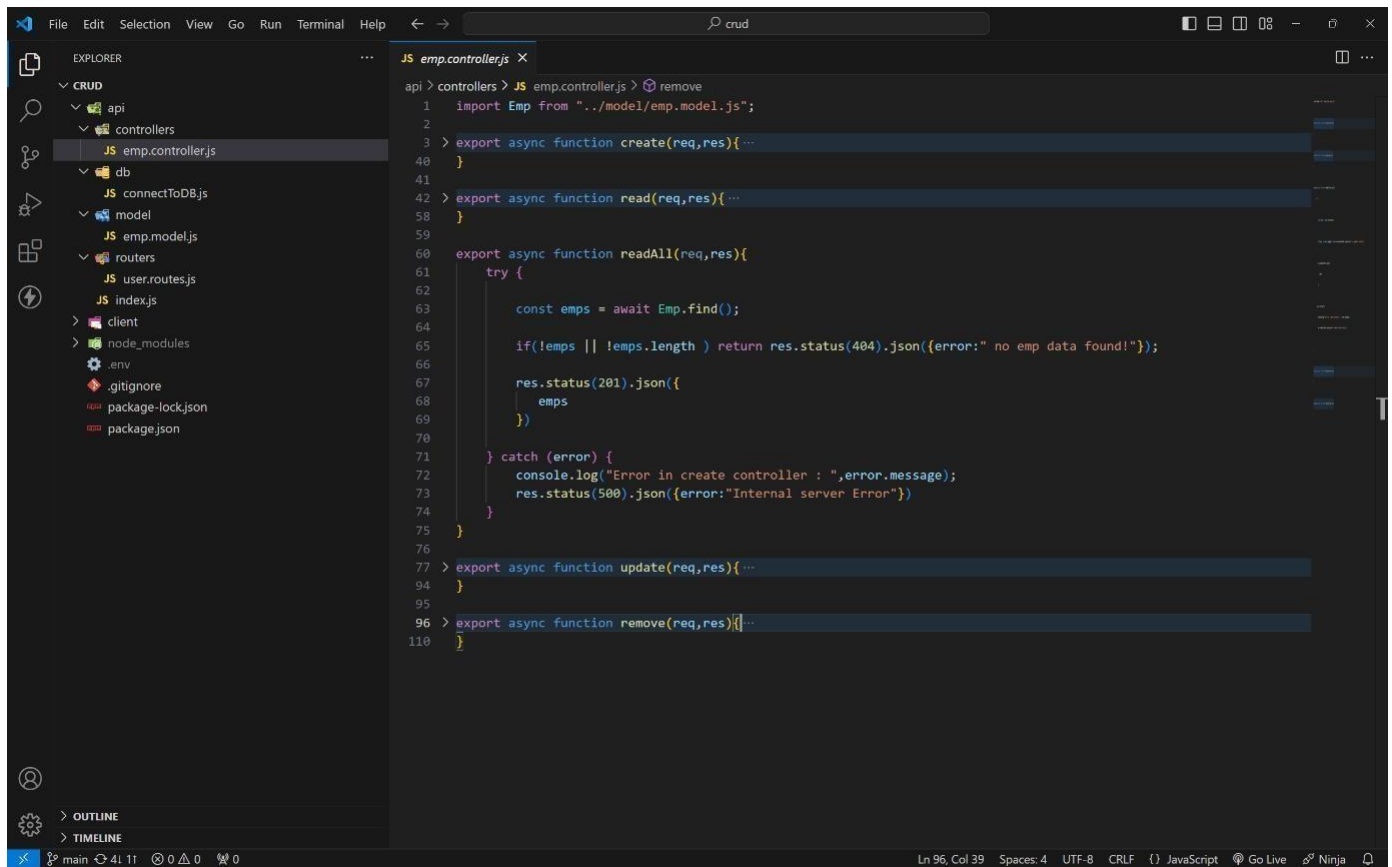
CONTROLLERS: CREATE :



The screenshot shows a VS Code editor with the Explorer sidebar on the left. The 'api' folder is expanded, showing subfolders 'controllers', 'db', 'model', and 'routes'. The 'controllers' folder is selected, and 'emp.controller.js' is highlighted. The main editor displays the content of 'emp.controller.js'.

```
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
```

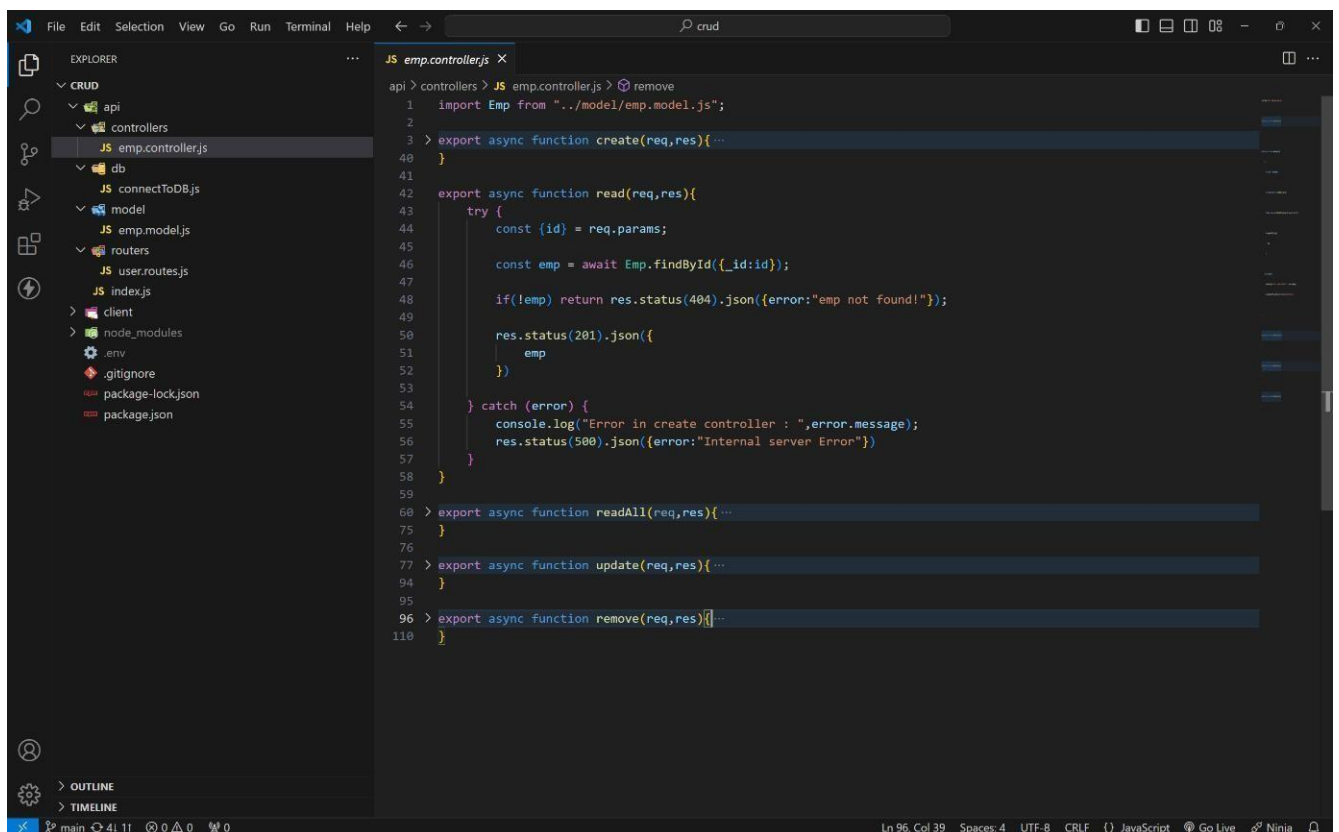
READALL:



The screenshot shows the 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', 'model', 'routers', and 'index.js'. The 'api' folder contains 'emp.controller.js'. The code editor shows the implementation of the 'readAll' function in 'emp.controller.js'.

```
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
63     const emps = await Emp.find();
64
65     if(!emps || !emps.length ) return res.status(404).json({error:" no emp data found!"});
66
67     res.status(201).json({
68       emps
69     })
70   } catch (error) {
71     console.log("Error in create controller : ",error.message);
72     res.status(500).json({error:"Internal server Error"})
73   }
74 }
75
76
77 > export async function update(req,res){ ...
94 }
95
96 > export async function remove(req,res){ ...
110 }
```

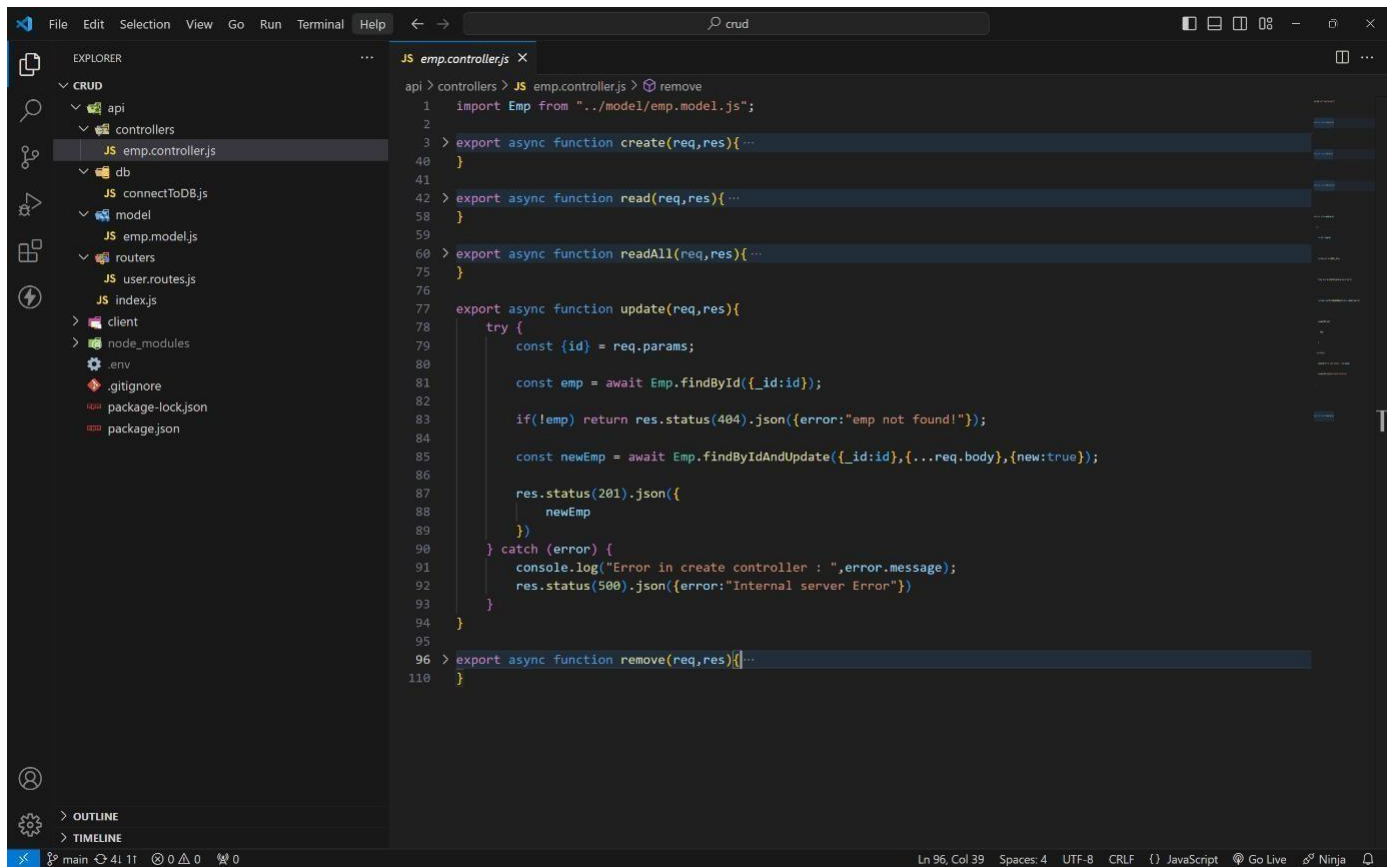
READONE :



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 a project structure with a 'crud' folder containing 'api', 'controllers', 'db', 'model', 'routers', and 'index.js'. The 'api' folder contains 'emp.controller.js'. The code editor shows the implementation of the 'read' function in 'emp.controller.js'.

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

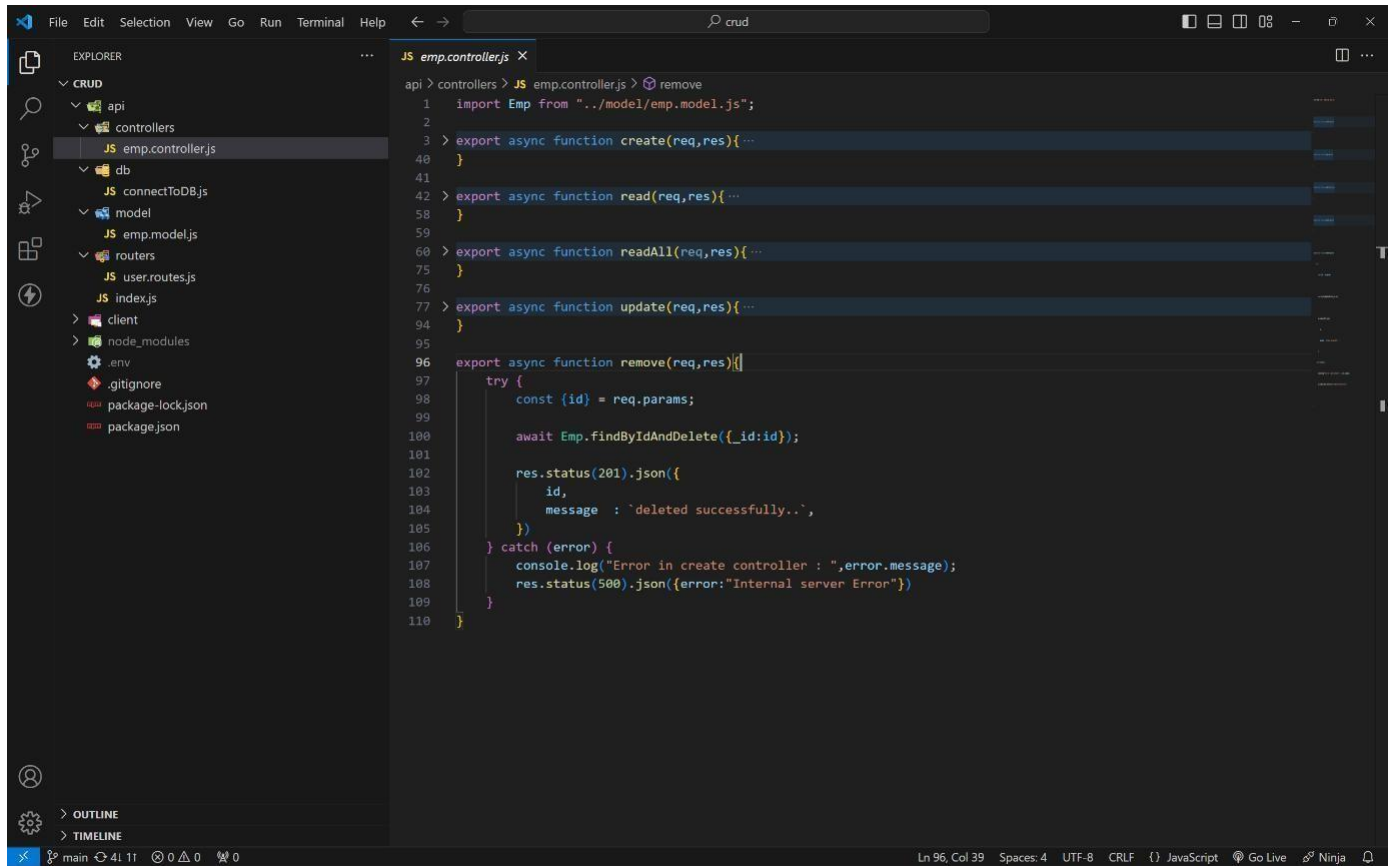
UPDATE :



The screenshot shows the Visual Studio Code editor with the file `emp.controller.js` open. The file is part of a project structure where `crud` is the current directory. The code defines several asynchronous functions for a REST API: `create`, `read`, `readAll`, `update`, and `remove`. The `update` function is the focus, showing a try-catch block that updates an employee record by ID. It uses `Emp.findByIdAndUpdate` to update the record and returns the updated object. Error handling is implemented with `console.log` and `res.status` for both client and server errors.

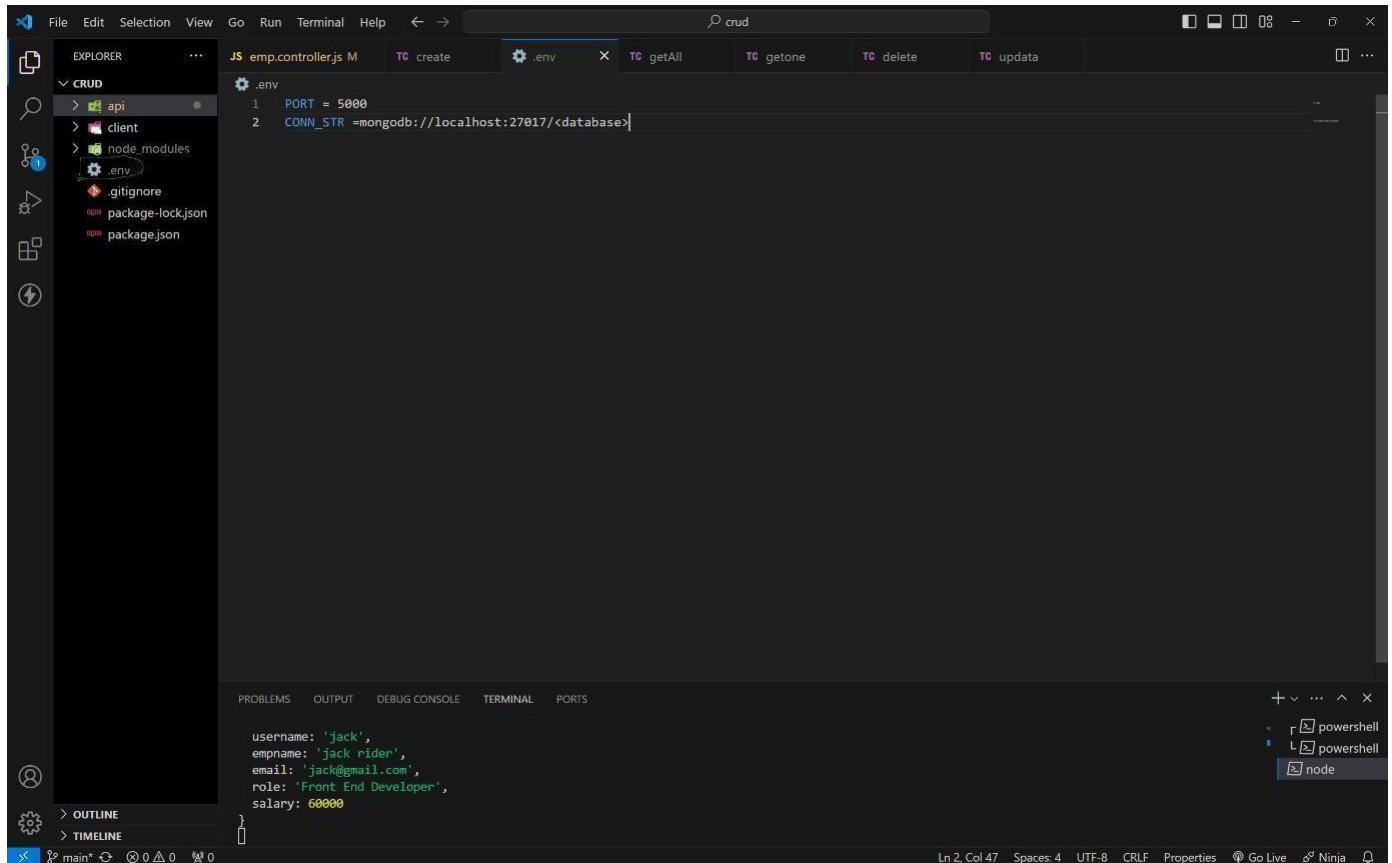
```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  > export async function create(req,res){ ...
40 }
41
42 > export async function read(req,res){ ...
58 }
59
60 > export async function readAll(req,res){ ...
75 }
76
77 export async function update(req,res){
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 :



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 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 :

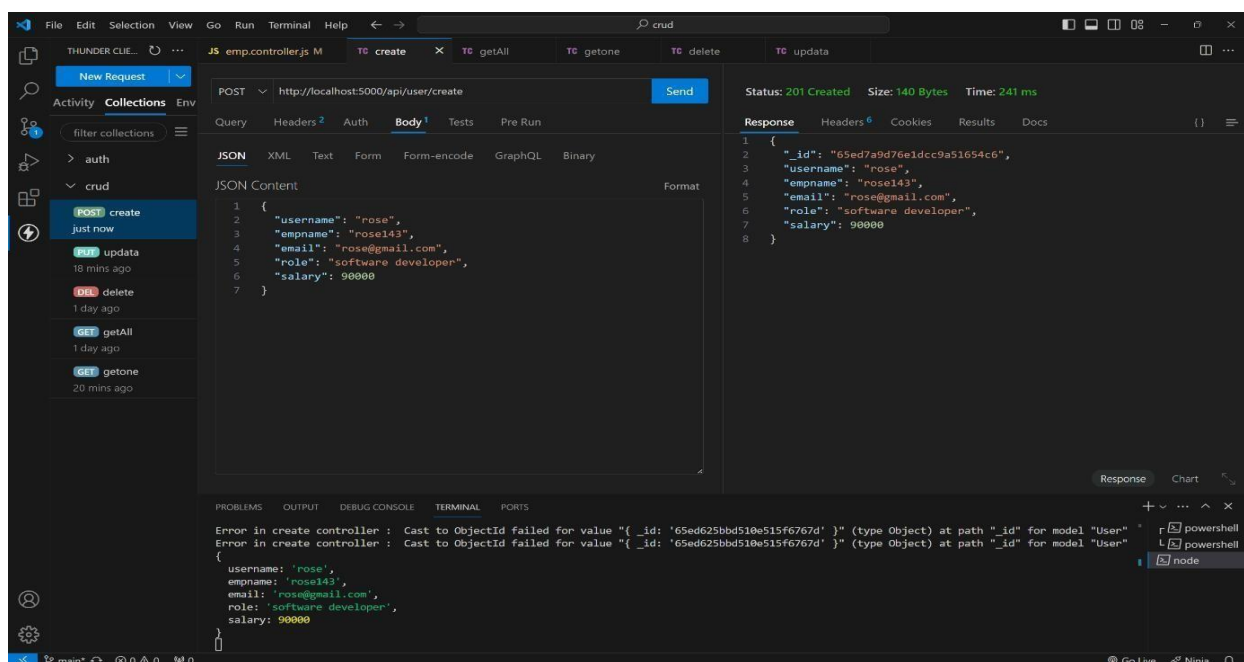
1 . This route is used to delete specific user by passing that user id as a param. method as delete url as

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

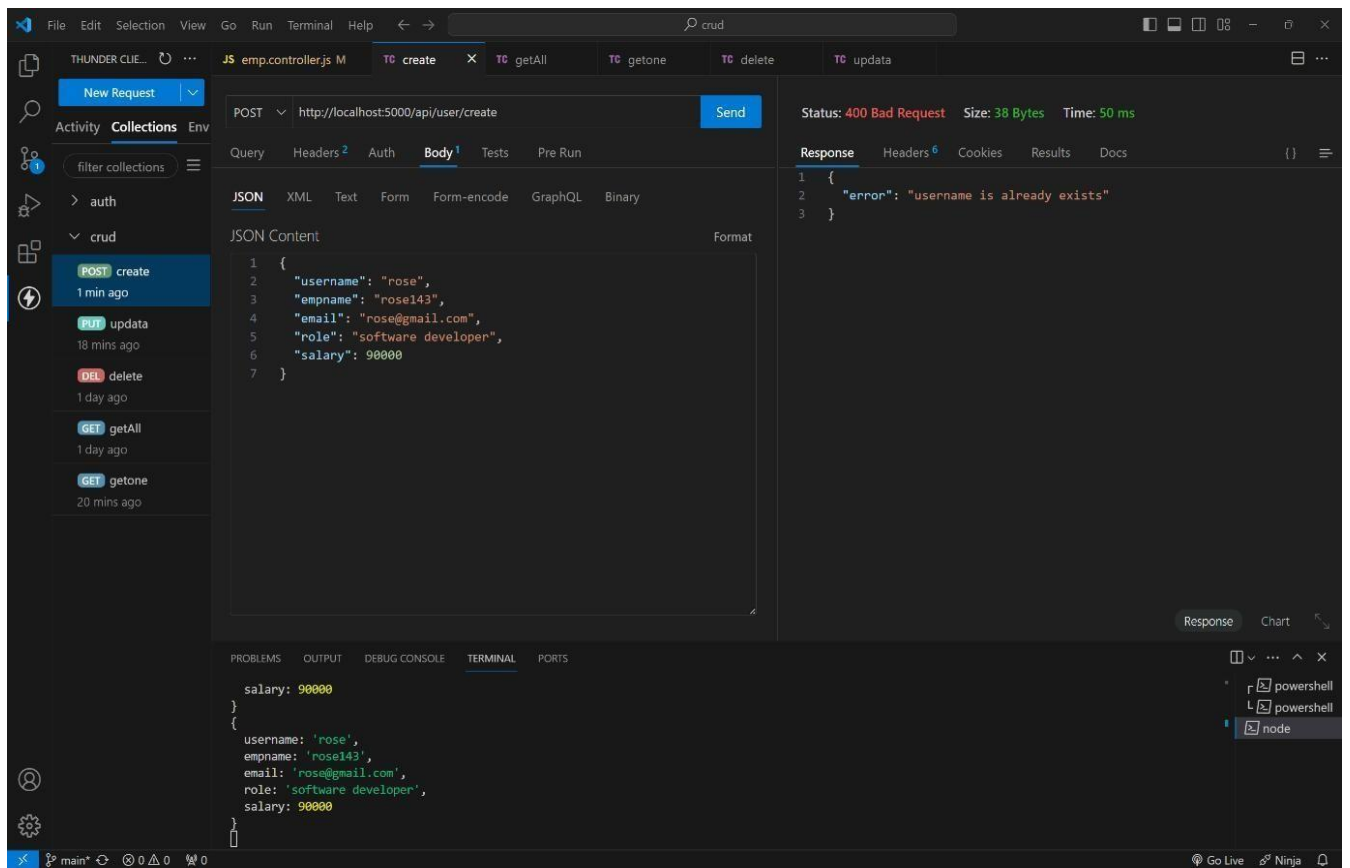
2 . After sending you will deleted successfully as response.

OUTPUT :

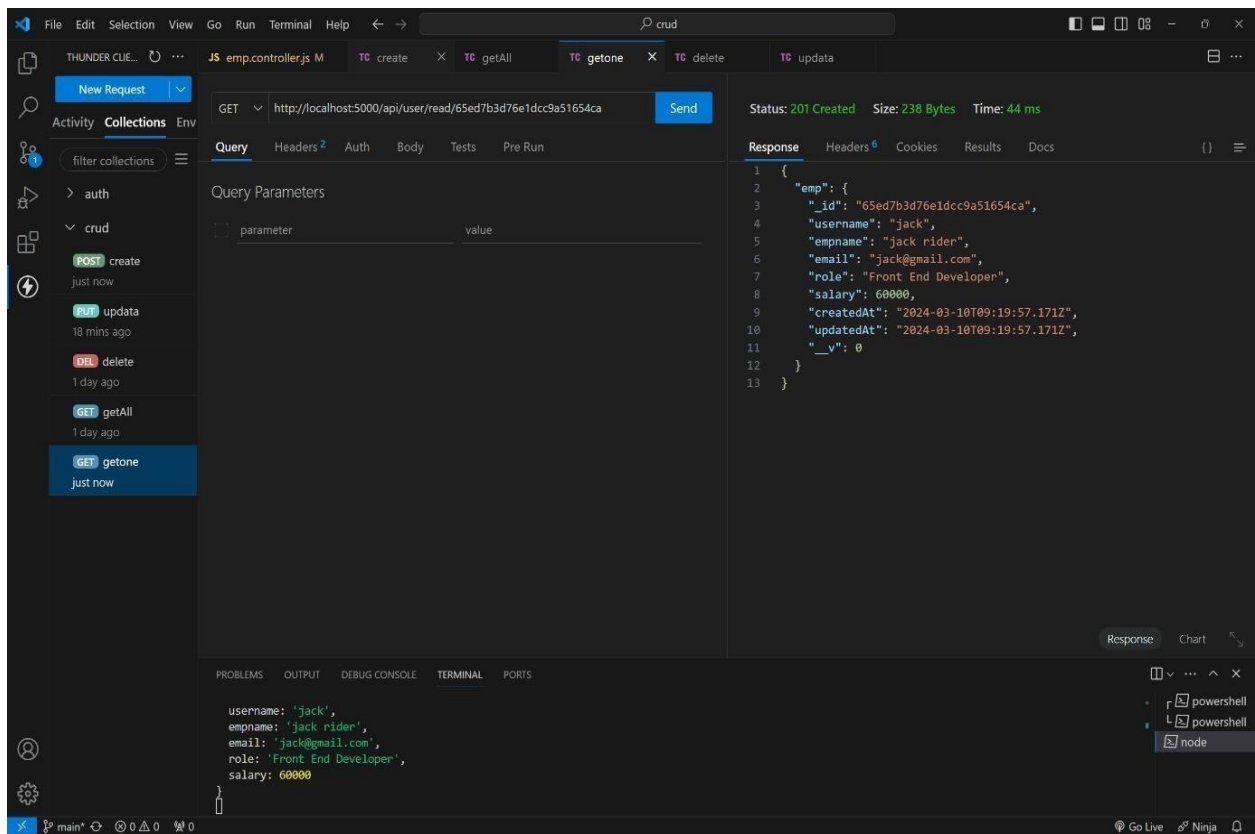
CREATE A NEW USER :



CREATING USER WITH EXISTING USERNAME :



READONE :



READ ALL :

The screenshot shows the Thunder Client interface with a collection named 'crud'. The 'getAll' request is selected, showing a GET request to `http://localhost:5000/api/user/readall`. The response is a JSON array of two user objects, with a status of 201 Created, size of 468 Bytes, and time of 130 ms.

```
GET http://localhost:5000/api/user/readall
```

Query Parameters

| parameter | value |
|-----------|-------|
|-----------|-------|

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": "2024-03-10T09:17:17.904Z",
11      "updatedAt": "2024-03-10T09:17:17.904Z",
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 }
```

Problems

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

UPDATE :

The screenshot shows the Thunder Client interface with a collection named 'crud'. The 'update' request is selected, showing a PUT request to `http://localhost:5000/api/user/update/65ed7b3d76e1dcc9a51654ca`. The request body is a JSON object with user details. The response is a JSON object with the updated user details, with a status of 201 Created, size of 246 Bytes, and time of 213 ms.

```
PUT http://localhost:5000/api/user/update/65ed7b3d76e1dcc9a51654ca
```

JSON Content

```
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 }
```

Problems

```
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 DELETE request configured and executed. The request is sent to the URL `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`. The response is a 201 Created status with a JSON body containing an ID and a success message.

Request Details:

- Method: DELETE
- URL: `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`
- Status: 201 Created
- Size: 68 Bytes
- Time: 111 ms

Response Body:

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

Terminal Output:

```
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
```