

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

NAME: - SHAIK AKBAR BASHA

COLLEGE: - KALLAM HARANAHAREDDY INSTITUTE OF TECHNOLOGY

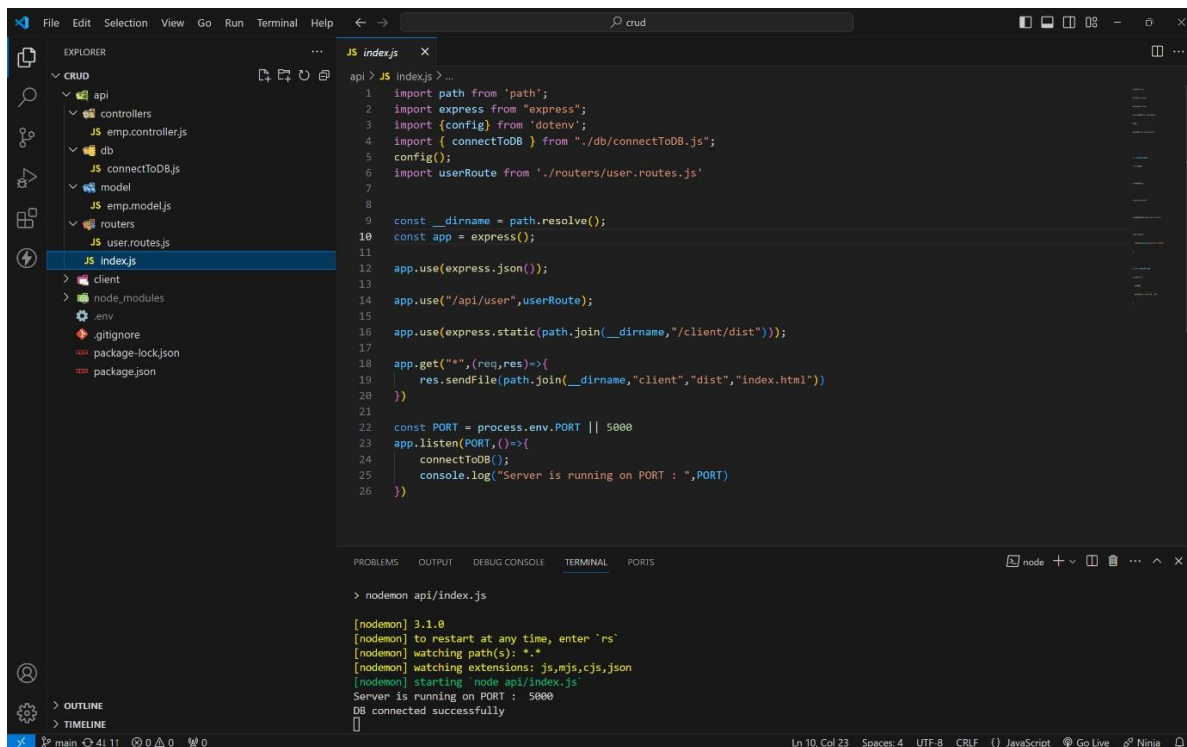
ROLL NO: - 208X1A05G0

MOBILE: - 8143489589

GMAIL: - 208x1a05g0@khitguntur.ac.in

source code :

index.js file :



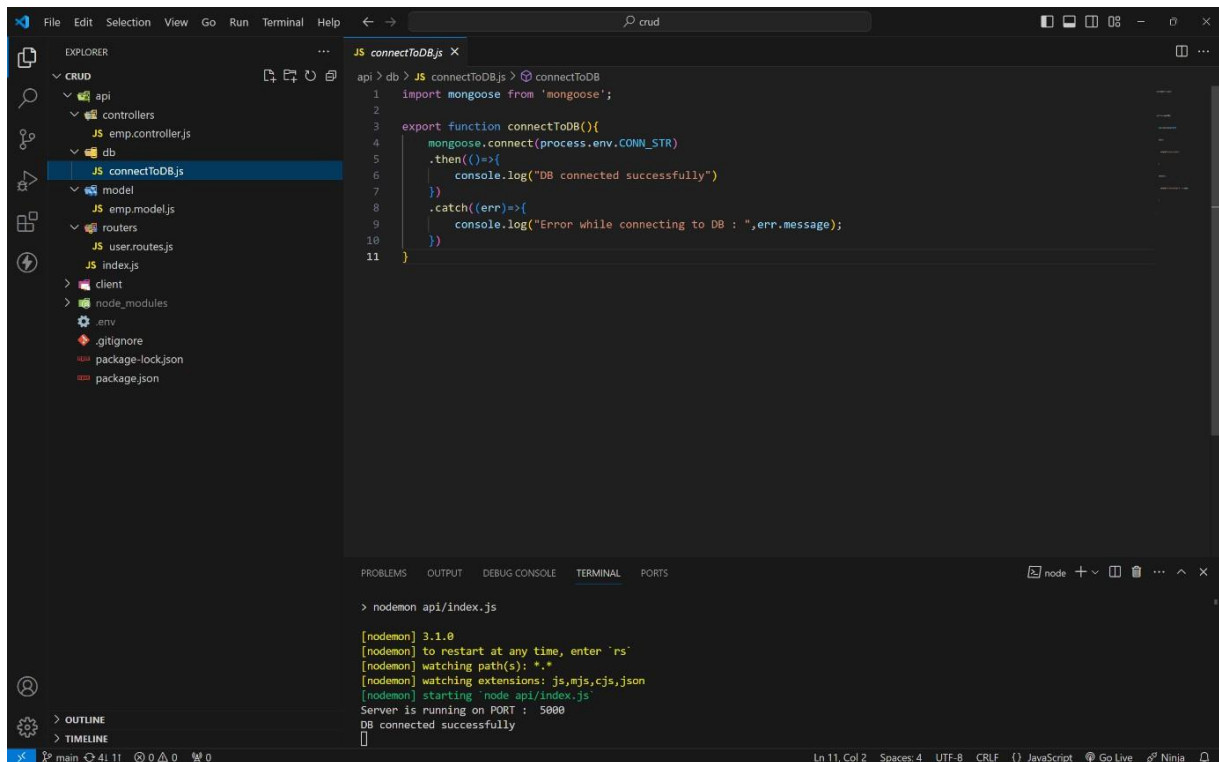
The screenshot shows a Visual Studio Code editor with a project structure on the left and a code editor in the center. The project structure includes a folder named 'api' containing subfolders 'controllers', 'db', 'model', 'routers', and 'index.js'. The 'index.js' file is selected and its code is displayed in the editor. The code is a Node.js script that uses Express.js to create a RESTful API. It imports necessary modules, configures the app, and sets up routes. The terminal at the bottom shows the command 'nodemon api/index.js' being executed, and the output indicates that the server is running successfully on port 5000.

```
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 })
```

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 :



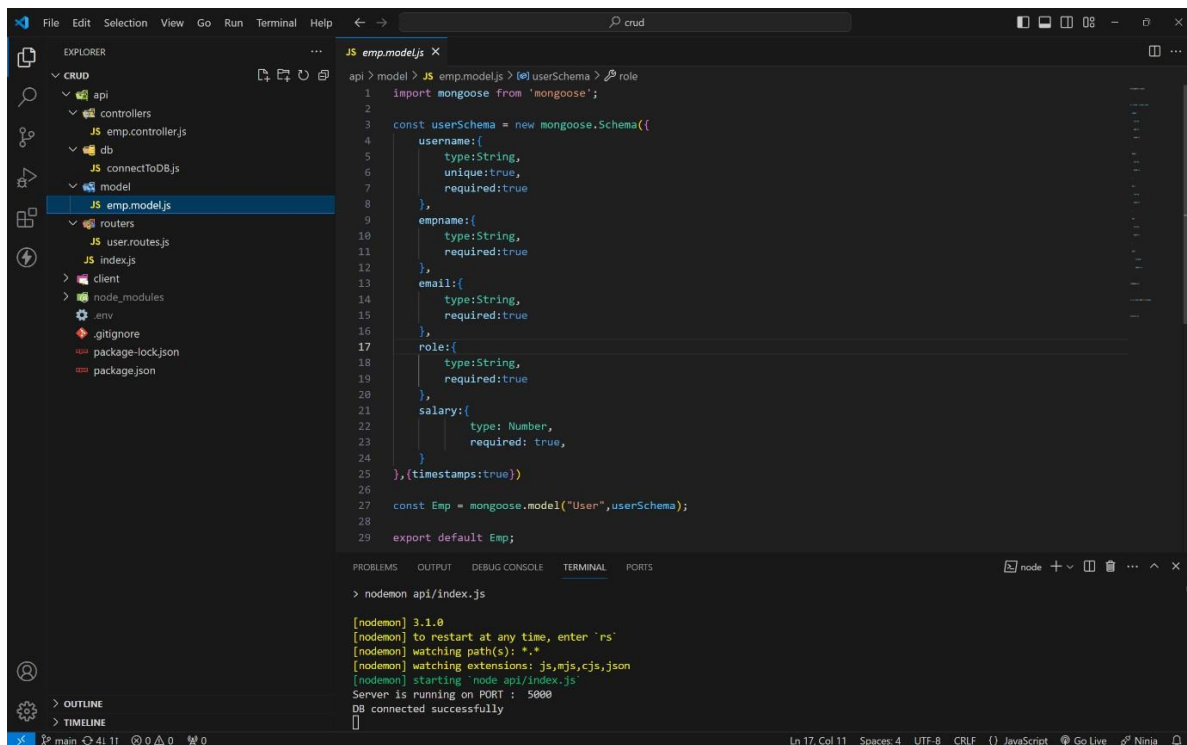
The screenshot shows the VS Code editor with the Explorer sidebar on the left. The file 'connectToDB.js' is selected under the 'db' folder. The main editor displays the code for 'connectToDB.js'. The terminal at the bottom shows the command 'nodemon api/index.js' and its output, indicating a successful MongoDB connection.

```
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 :



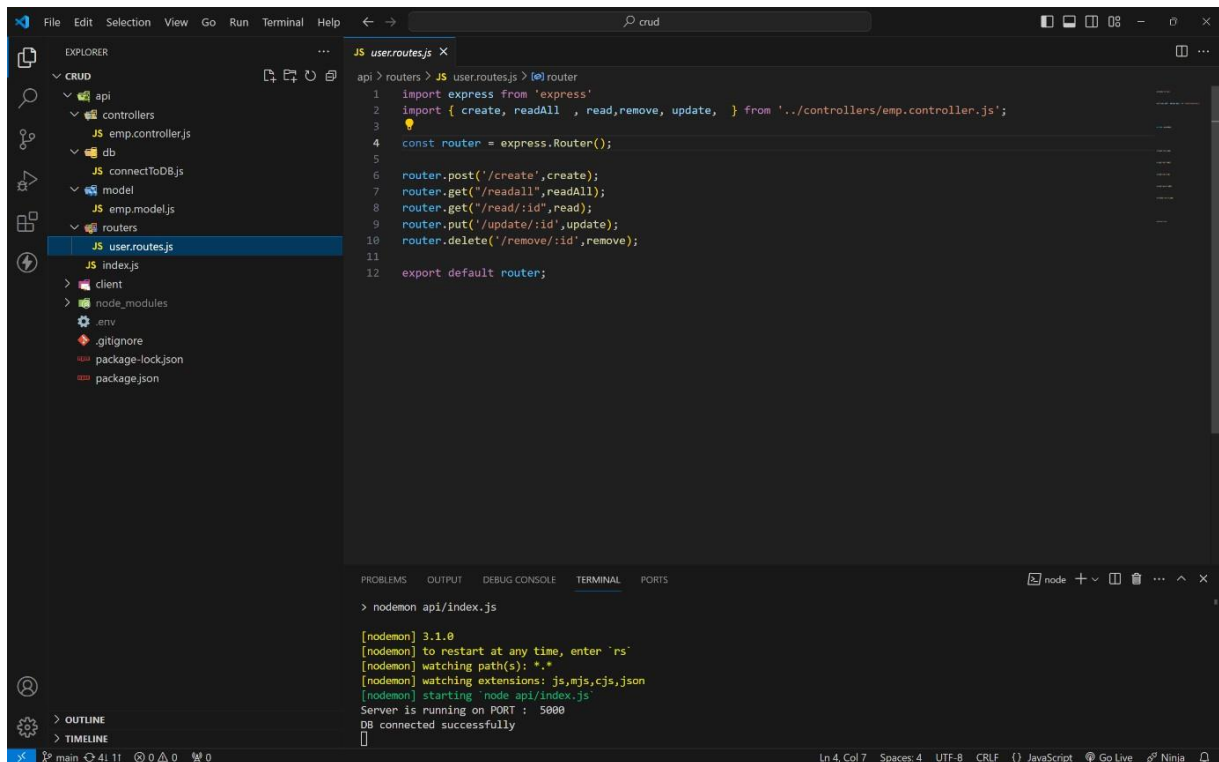
The screenshot shows the VS Code editor with the Explorer sidebar on the left. The file 'emp.model.js' is selected under the 'model' folder. The main editor displays the code for 'emp.model.js'. The terminal at the bottom shows the command 'nodemon api/index.js' and its output, indicating a successful MongoDB connection.

```
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 a 'crud' folder containing 'api', 'controllers', 'db', 'model', 'routes', and 'index.js'. The 'routes' folder is expanded, showing 'user.routes.js'. The main editor window displays the content of 'user.routes.js'.

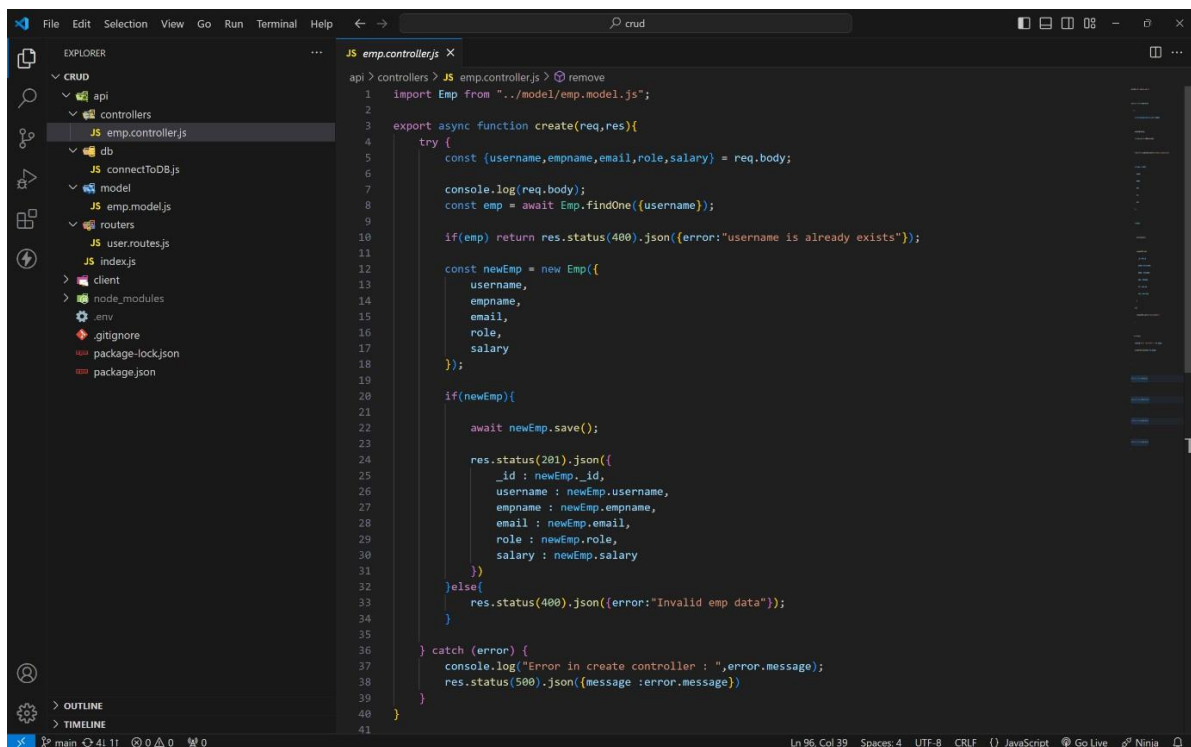
```
api > routes > JS user.routes.js > router
1 import express from 'express'
2 import { create, readAll, read, remove, update, } from '../controllers/emp.controller.js';
3
4 const router = express.Router();
5
6 router.post('/create', create);
7 router.get('/readall', readAll);
8 router.get('/read/:id',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

The terminal window 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
```

CONTROLLERS:

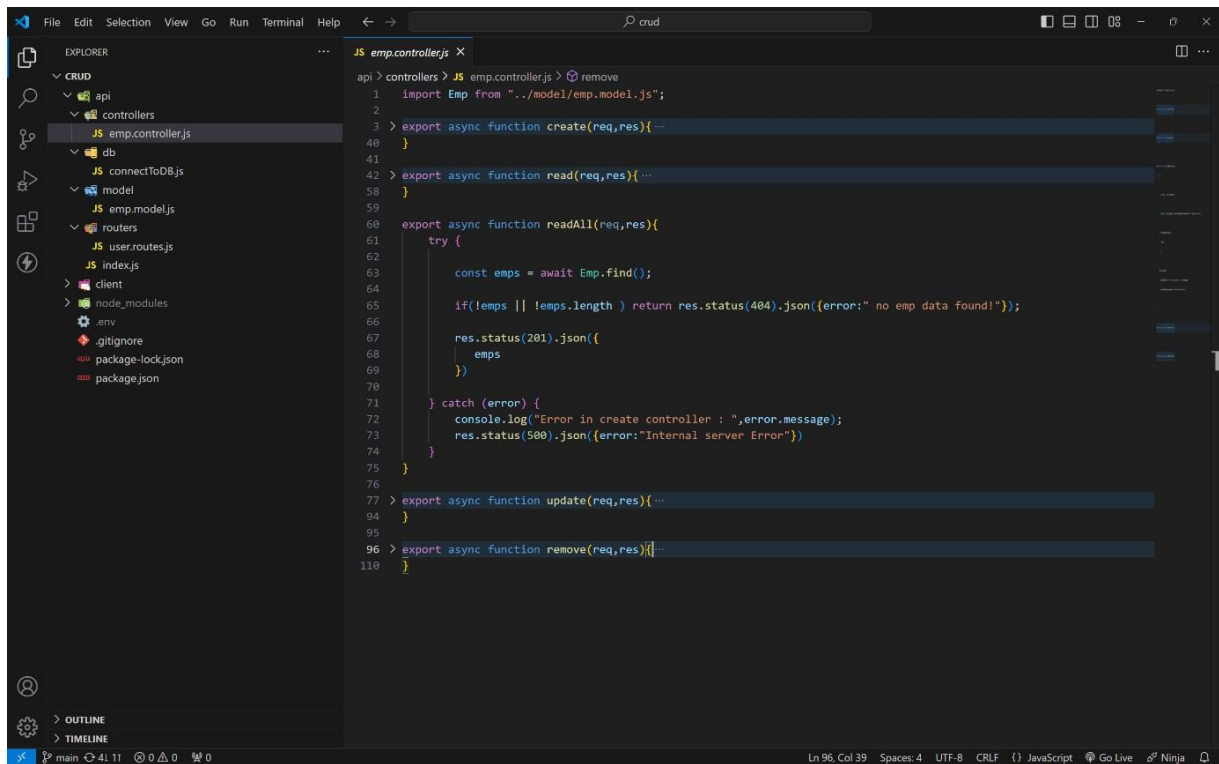
CREATE :



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', 'routes', and 'index.js'. The 'controllers' folder is expanded, showing 'emp.controller.js'. The main editor window 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    }
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
41 }
```

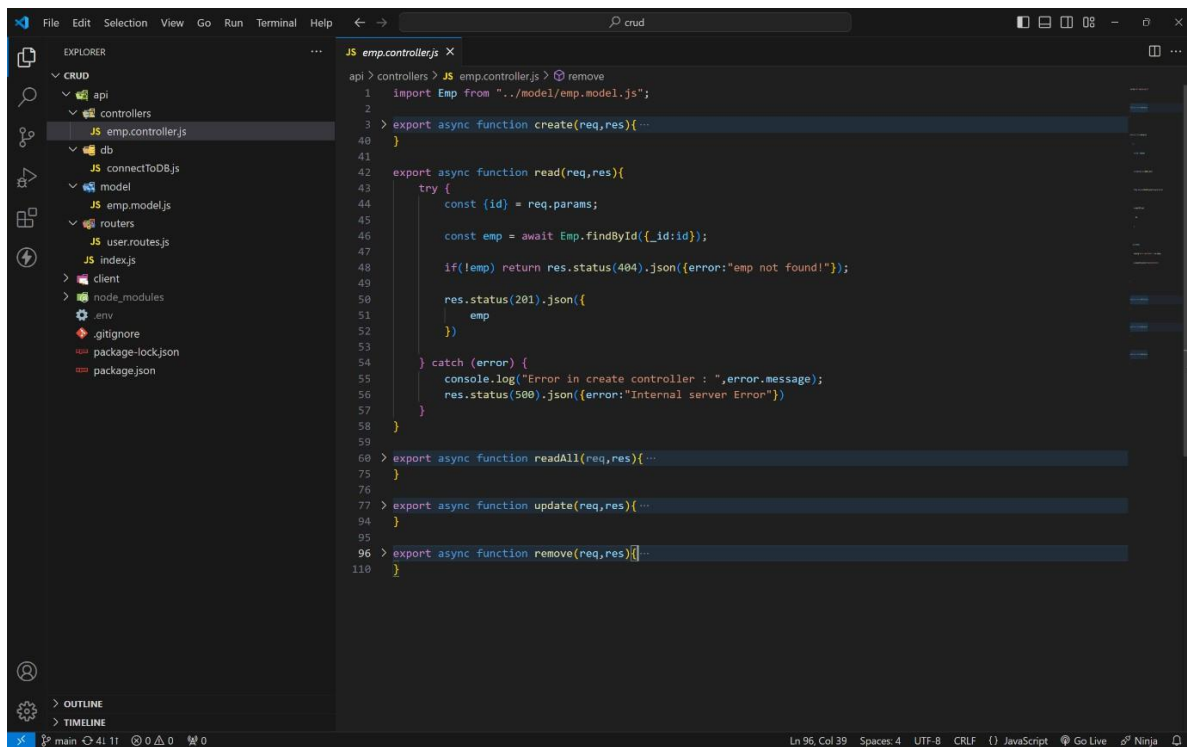
READALL:



The screenshot shows the VS Code editor with the file explorer on the left and the code editor in the center. The file explorer shows the project structure with folders like api, controllers, db, model, routers, and user.routes.js. The code editor displays the contents of `emp.controller.js`. The code includes imports for `Emp` from `../model/emp.model.js` and defines several asynchronous functions: `create`, `read`, `readAll`, `update`, and `remove`. The `readAll` function is the focus, showing a try-catch block that calls `Emp.find()` and returns the results in JSON format with a 201 status code. Error handling is implemented with `console.log` and `res.status(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){
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   }
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 > export async function update(req,res){ ...
94 }
95
96 > export async function remove(req,res){ ...
110 }
```

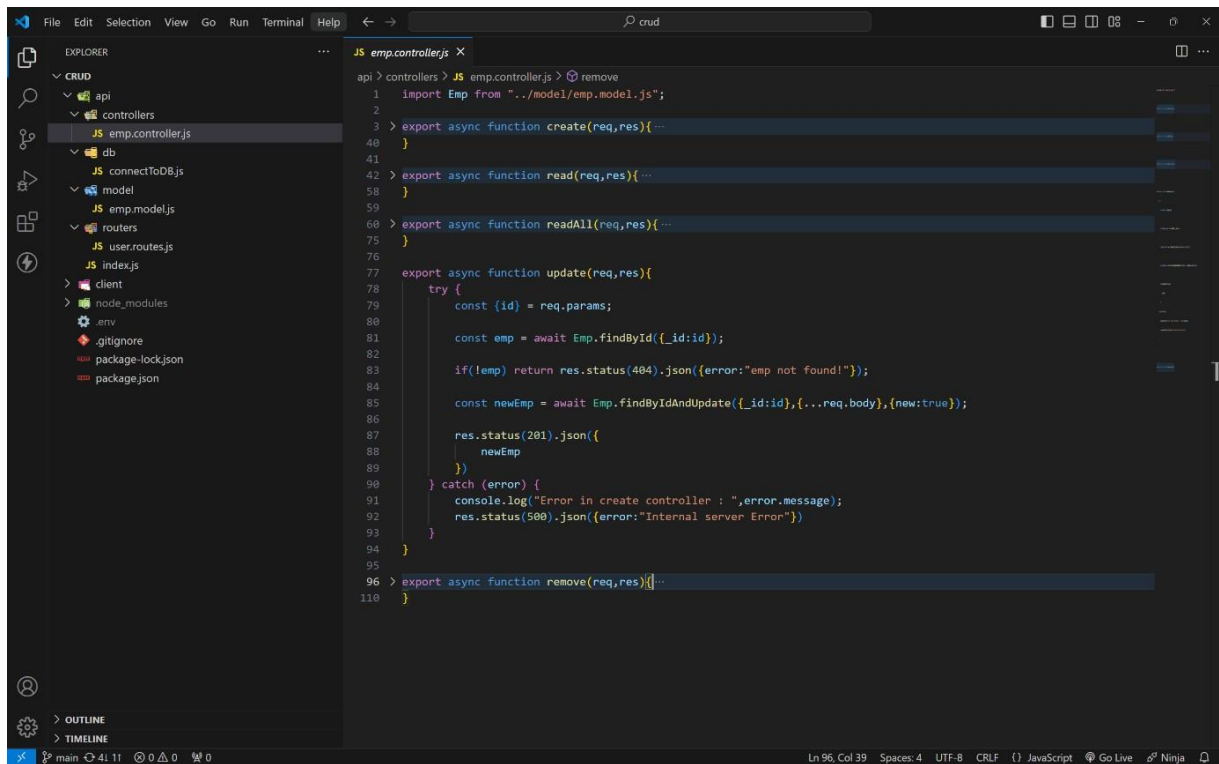
READONE :



The screenshot shows the VS Code editor with the file explorer on the left and the code editor in the center. The file explorer shows the project structure with folders like api, controllers, db, model, routers, and user.routes.js. The code editor displays the contents of `emp.controller.js`. The code includes imports for `Emp` from `../model/emp.model.js` and defines several asynchronous functions: `create`, `read`, `readAll`, `update`, and `remove`. The `read` function is the focus, showing a try-catch block that calls `Emp.findById()` with `req.params.id` and returns the result in JSON format with a 201 status code. Error handling is implemented with `console.log` and `res.status(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){
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   }
54   catch (error) {
55     console.log("Error in create controller : ",error.message);
56     res.status(500).json({error:"Internal server Error"});
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 }
```

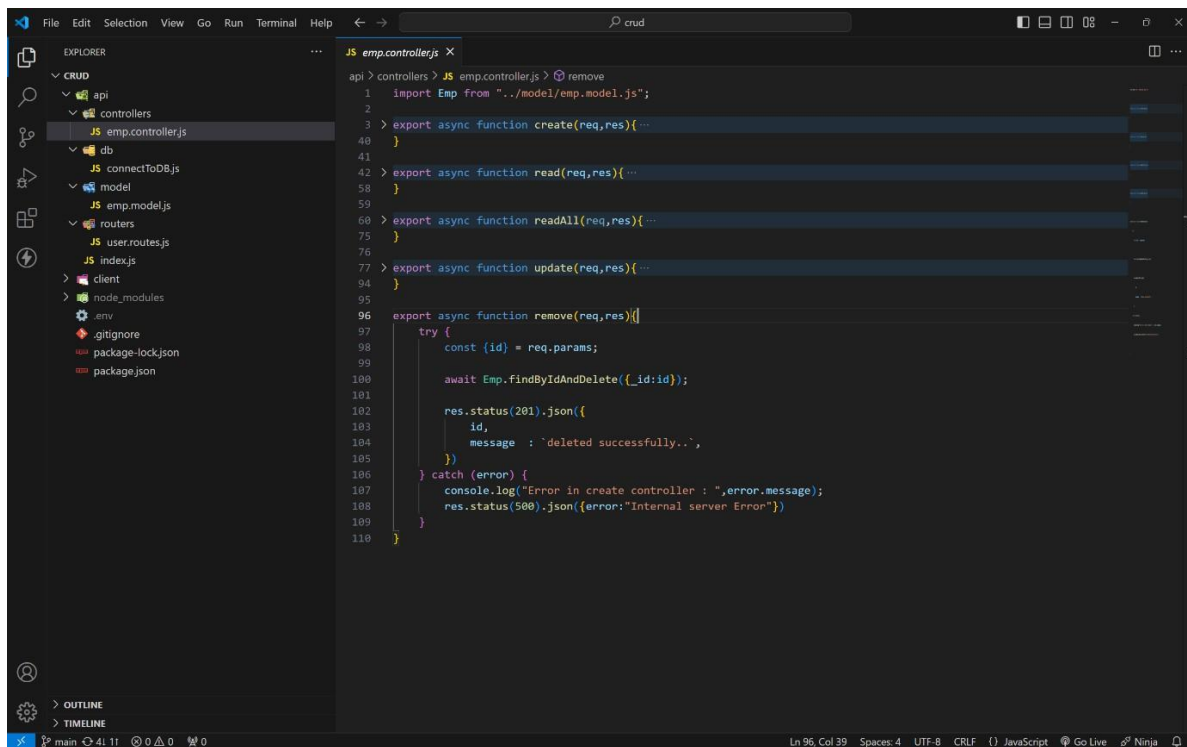
UPDATE :



The screenshot shows the VS 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 user.routes.js. The code editor displays the contents of `emp.controller.js`. The code includes imports for `Emp` from `../model/emp.model.js` and several async functions: `create`, `read`, `readAll`, `update`, and `remove`. The `update` function is the focus, showing a try-catch block that updates an employee record by ID. The status code 201 is returned on success, and 500 on error.

```
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 VS 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 user.routes.js. The code editor displays the contents of `emp.controller.js`. The code includes imports for `Emp` from `../model/emp.model.js` and several async functions: `create`, `read`, `readAll`, `update`, and `remove`. The `remove` function is the focus, showing a try-catch block that deletes an employee record by ID. The status code 201 is returned on success, and 500 on error.

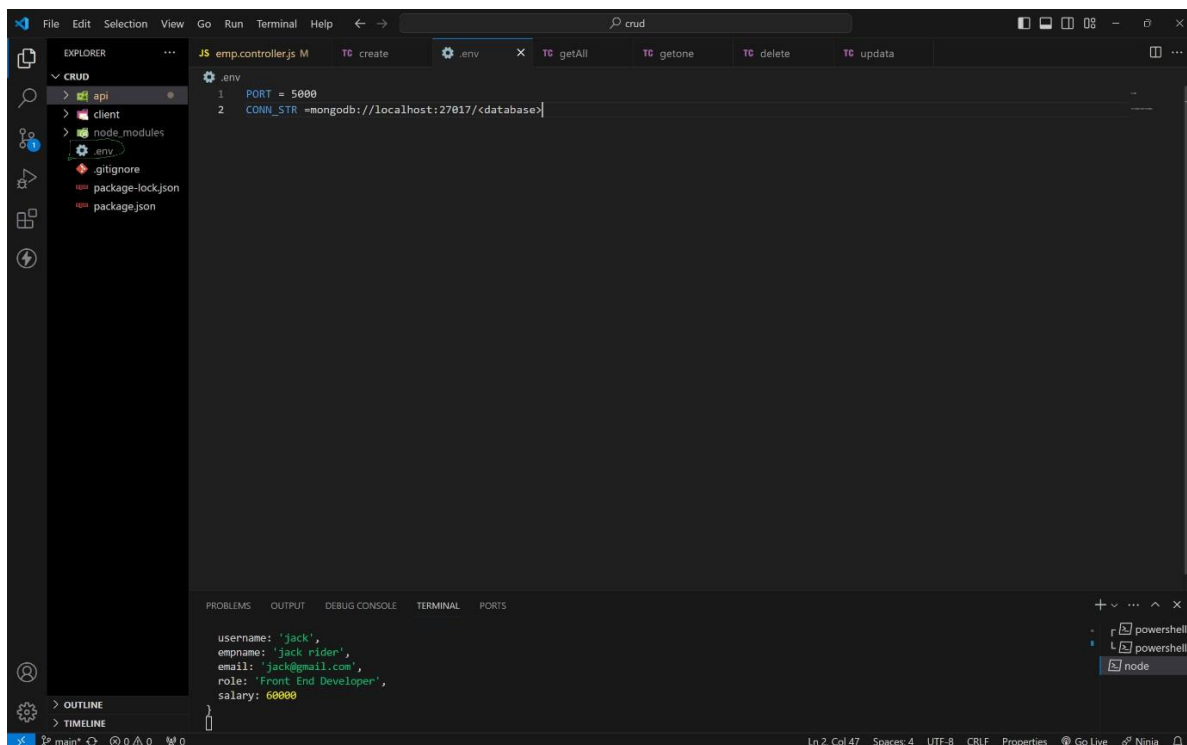
```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  > export async function create(req,res){ ...
40 }
41
42 > export async function read(req,res){ ...
58 }
59
60 > export async function readAll(req,res){ ...
75 }
76
77 > export async function update(req,res){ ...
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. Now move to crud folder (cd crud in terminal)
5. Ignore client folder.
6. Here crud is root folder.
7. 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<databasename>

8. 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)

9. 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

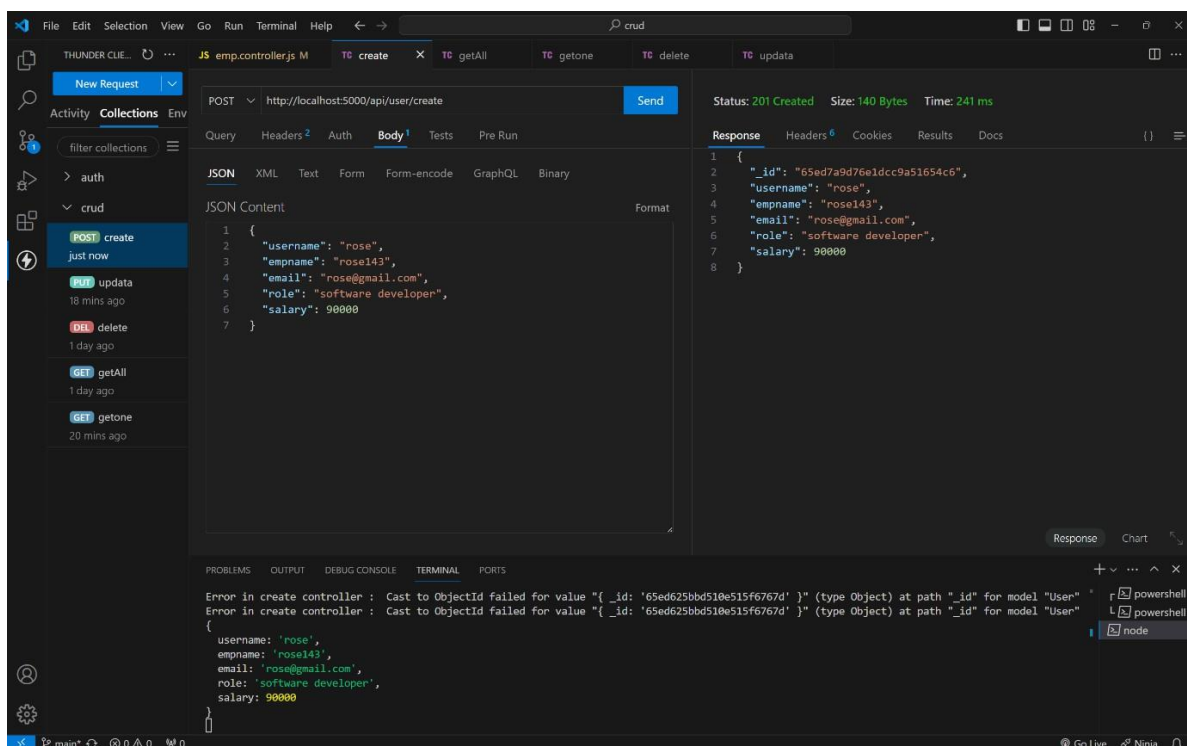
url as

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

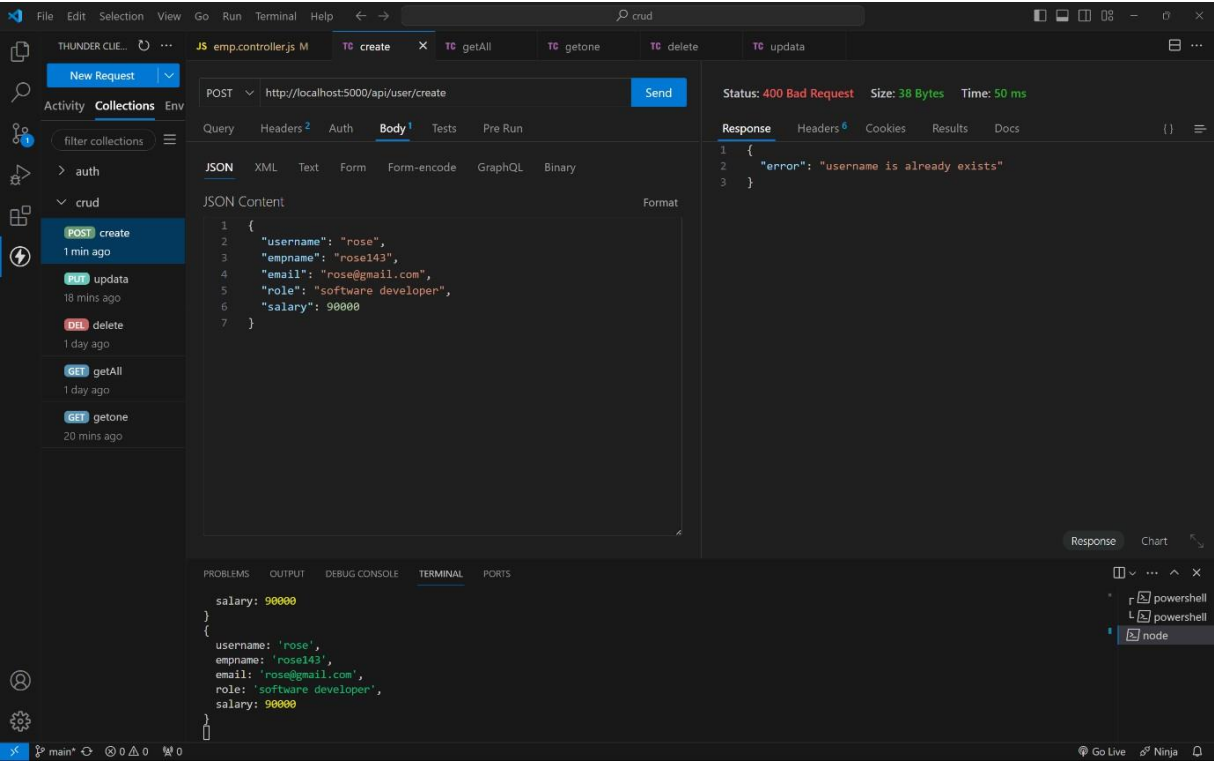
2. After sending you will deleted successfully as response.

OUTPUT :

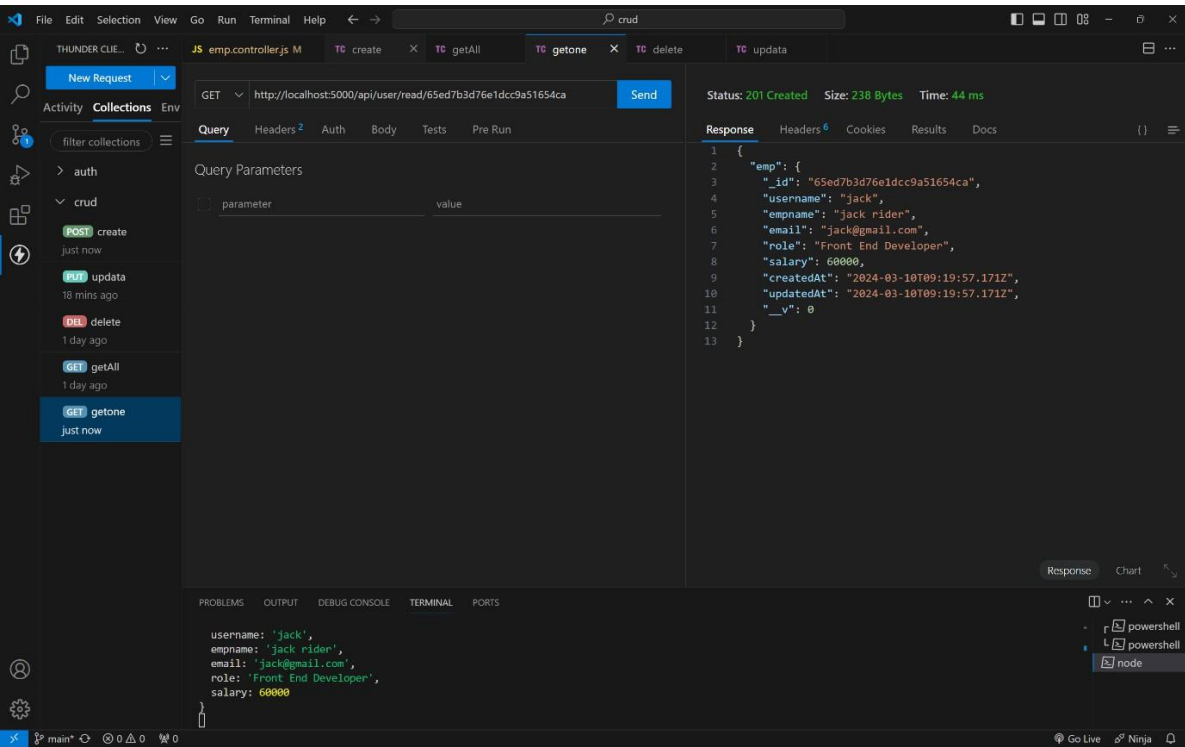
CREATE A NEW USER :



CREATING USER WITH EXISTING USERNAEM :



READONE :



READ ALL :

The screenshot shows the Thunder Client interface with a GET request to `http://localhost:5000/api/user/readall` successfully executed. The status is 201 Created, size is 468 Bytes, and time is 130 ms. The response is a JSON array of two employee objects. The terminal shows the output of the `getAll` function.

```
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 }
```

Terminal

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

UPDATE :

The screenshot shows a PUT request to `http://localhost:5000/api/user/update/65ed7b3d76e1dcc9a51654ca` with a JSON body. The status is 201 Created, size is 246 Bytes, and time is 213 ms. The response is a JSON object with a `newEmp` field. The terminal shows an error message: "Error in create controller : Cast to ObjectId failed for value '({ _id: '65ed625bbd510e515f6767d' })' (type Object) at path '_id' for model 'User'".

```
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 }
```

Terminal

```
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 `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`. The response is a 201 Created status with a JSON body containing the user ID and a success message.

Request Details:

- Method: DELETE
- URL: `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`
- Send button: Send

Response Details:

- 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
```