

ASSIGNMENT - 3

NAME : GANDHAM MANOJ KUMAR

E-MAIL: 2020cse.r60@svec.edu.in

PHN NO: 9121929522

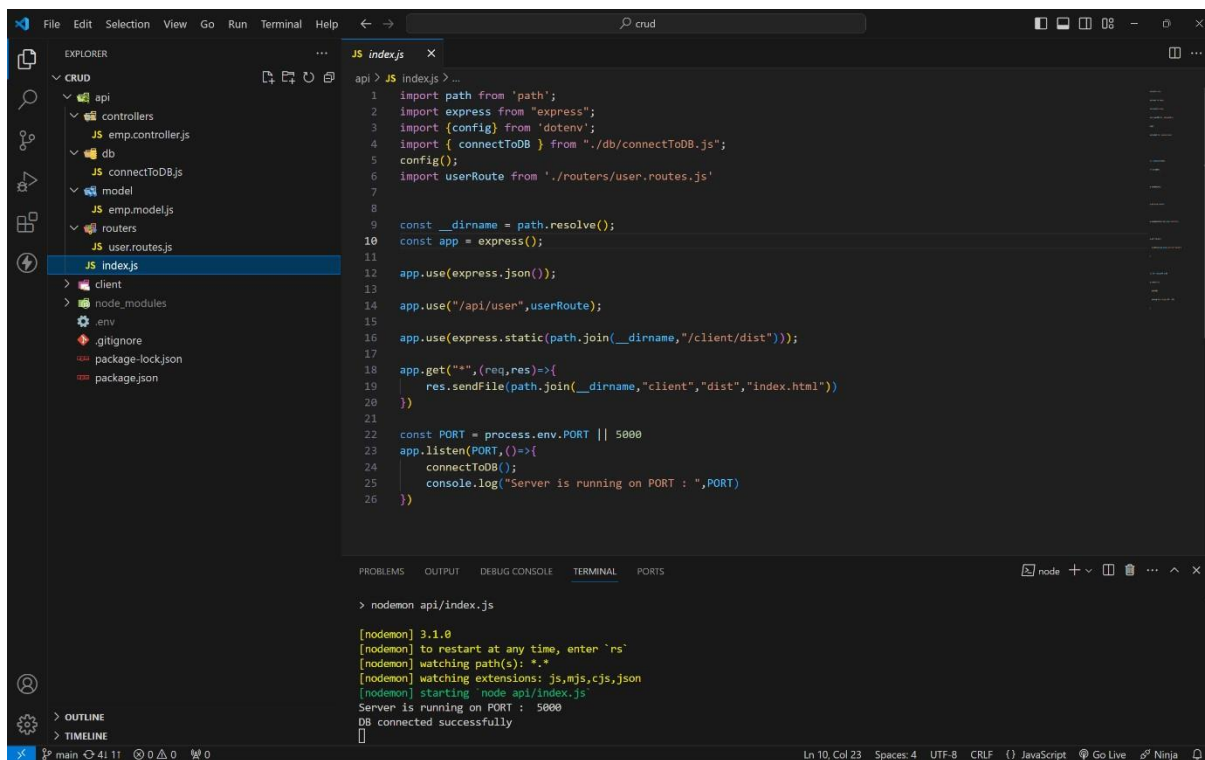
ROLL NO: 209E1A0560

COLLEGE: SRI VENKATESWARA ENGINEERING COLLEGE,TIRUPATI

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

source code :

index.js file :



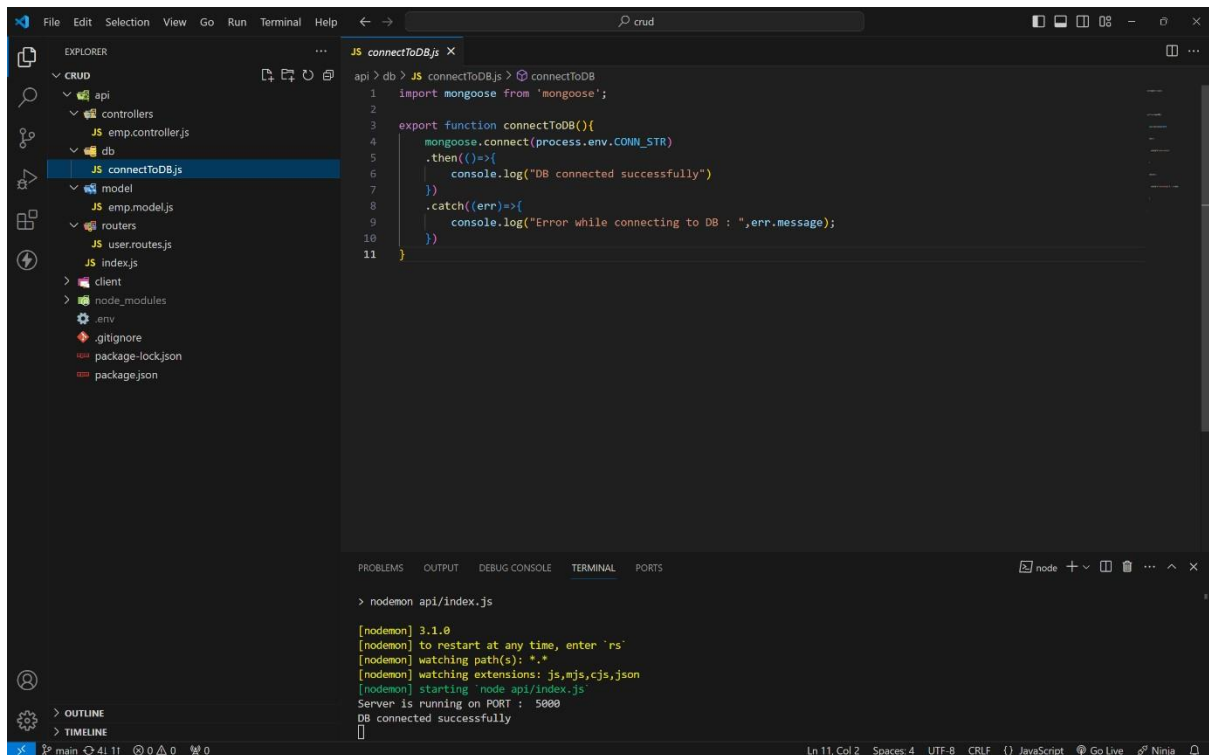
The screenshot shows a VS Code editor with a project structure on the left and a code editor on the right. The project structure includes a 'crud' folder with subfolders 'api', 'controllers', 'db', 'model', 'routers', and 'user.routes.js'. The 'api' folder contains 'index.js'. The code editor shows the content of '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 './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 })
```

The terminal output shows the command to run the application and the successful execution of the server.

```
> 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 `db` folder. 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` and logs the connection status. The terminal at the bottom shows the command `node api/index.js` being executed, and the output indicates that the server is running on port 5000 and the database connection was successful.

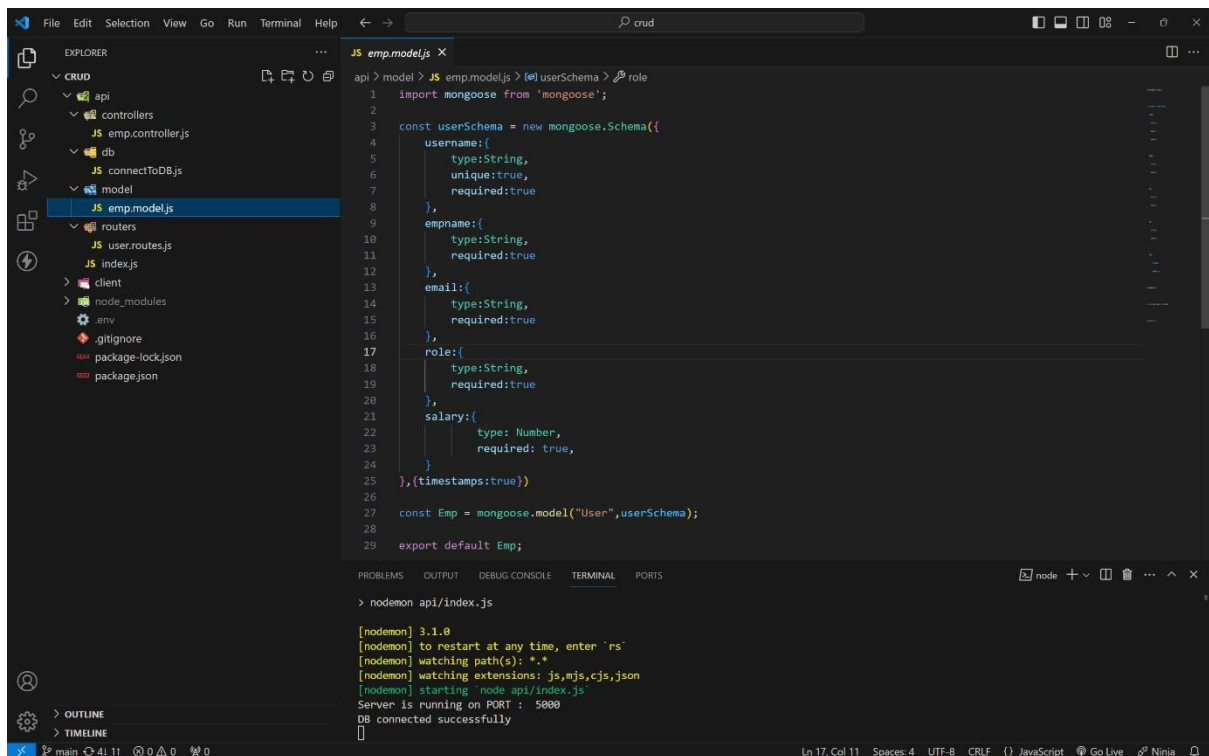
```
api > db > JS connectToDB.js > connectToDB
1 import mongoose from 'mongoose';
2
3 export function connectToDB(){
4   mongoose.connect(process.env.CONN_STR)
5     .then(()=>{
6       console.log("DB connected successfully")
7     })
8     .catch(err=>{
9       console.log("Error while connecting to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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
> 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 interface with the Explorer panel on the left displaying the project structure. The file `emp.model.js` is selected under the `model` folder. 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 `Emp` model is created using `mongoose.model` and exported as the default. The terminal at the bottom shows the command `node api/index.js` being executed, and the output indicates that the server is running on port 5000 and the database connection was successful.

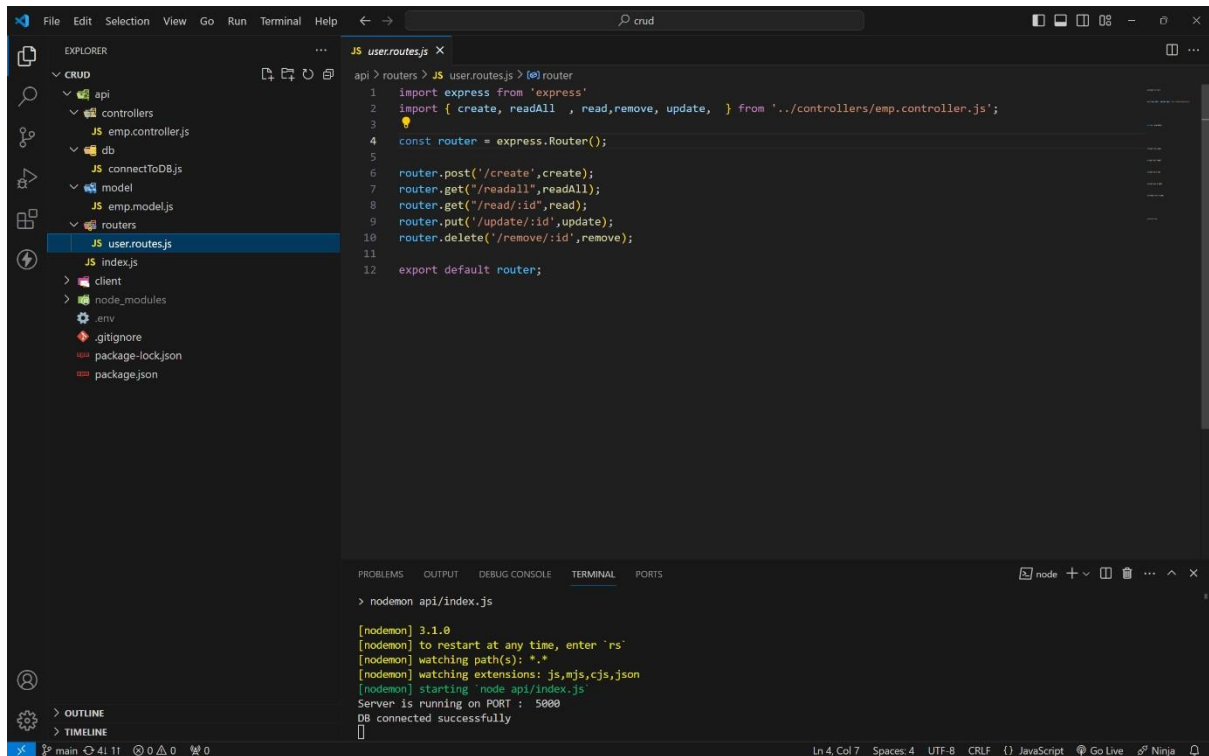
```
api > model > JS emp.model.js > userSchema > role
1 import mongoose from 'mongoose';
2
3 const userSchema = new mongoose.Schema({
4   username:{
5     type:String,
6     unique:true,
7     required:true
8   },
9   empname:{
10    type:String,
11    required:true
12  },
13  email:{
14    type:String,
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;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
> 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
```

ROUTES:



The screenshot shows a VS Code editor with a project structure on the left. The file explorer shows a directory named 'api' containing 'controllers', 'db', 'model', 'routes', and 'user.routes.js'. The 'user.routes.js' file is open in the editor, showing the following code:

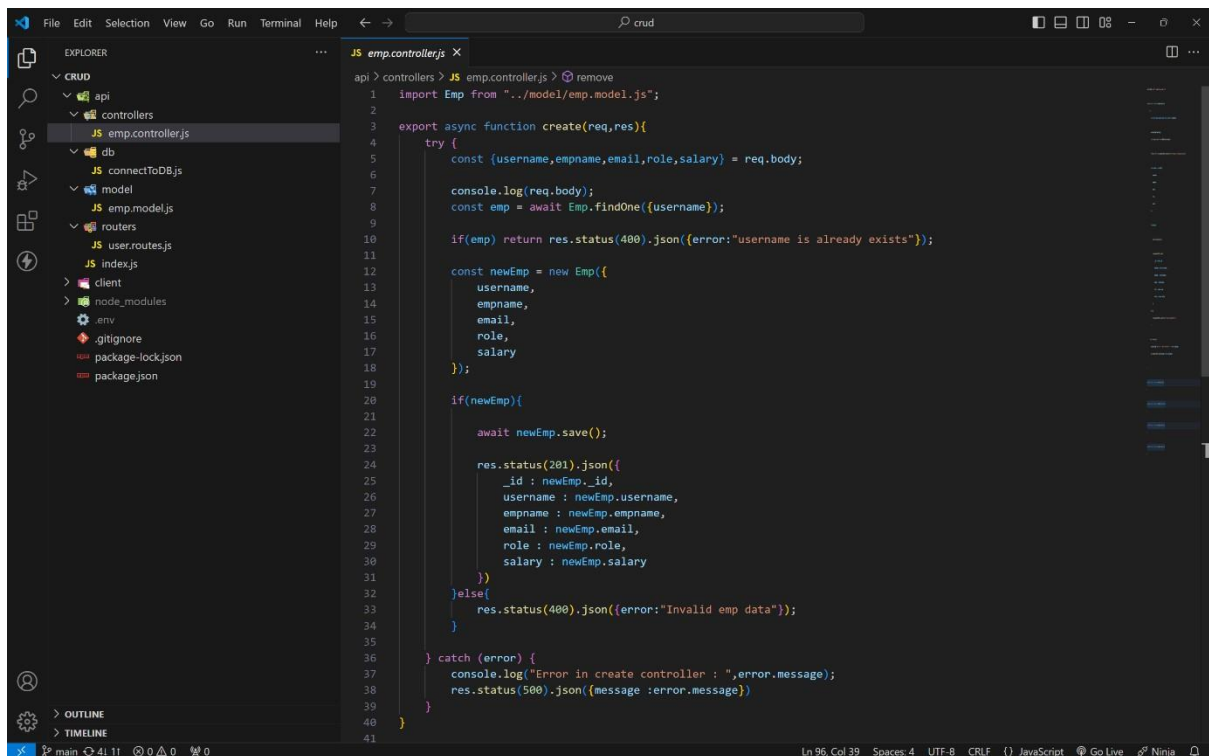
```
1 import express from 'express'
2 import { create, readAll, read, remove, update, } from '../controllers/emp.controller.js';
3
4 const router = express.Router();
5
6 router.post('/create', create);
7 router.get('/readall', readAll);
8 router.get('/read/:id', read);
9 router.put('/update/:id', update);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

Below the editor, the terminal window shows the command 'nodemon api/index.js' and the output:

```
[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 :

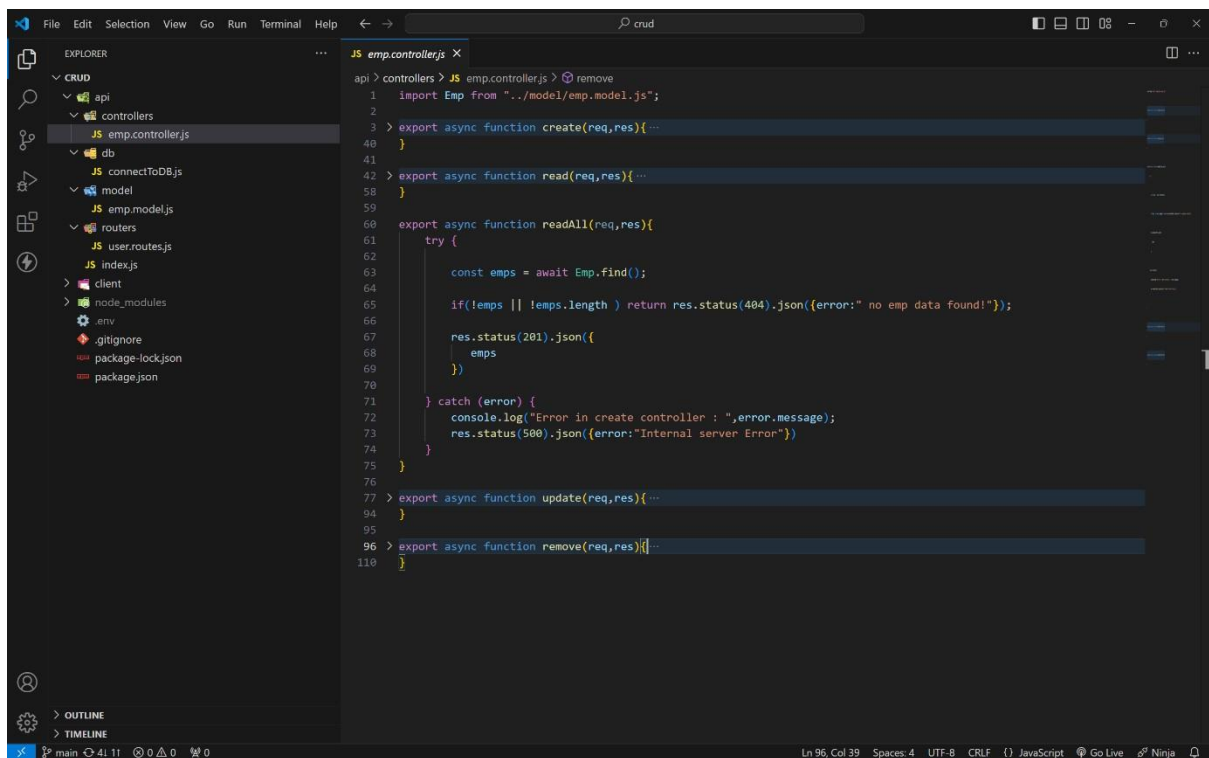
CREATE :



The screenshot shows a VS Code editor with a project structure on the left. The file explorer shows a directory named 'api' containing 'controllers', 'db', 'model', 'routes', and 'user.routes.js'. The 'emp.controller.js' file is open in the editor, showing 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    }
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 }
```

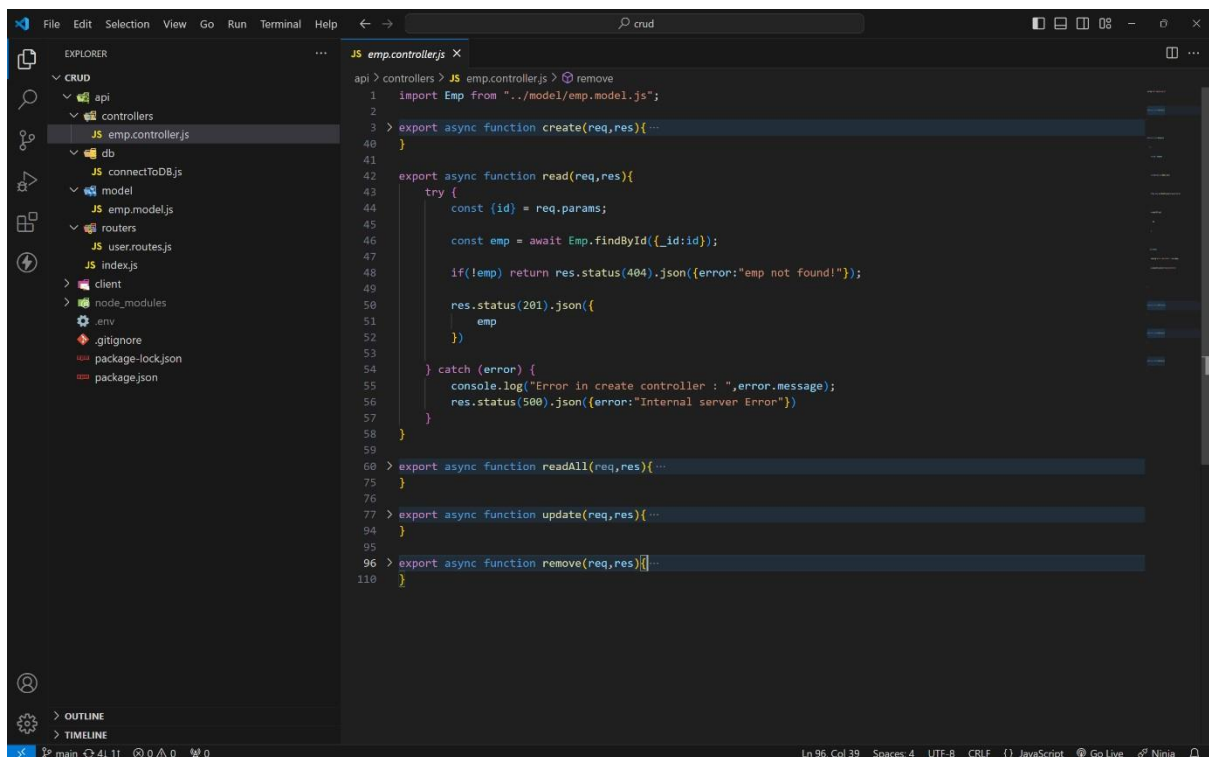
READALL:



The image shows a VS Code editor with the file explorer on the left displaying a project structure for a CRUD application. The main editor window shows the `emp.controller.js` file. The code 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 a JSON response with the list of employees or a 404 status if none are found. Error handling is implemented with `console.log` and a 500 status for internal 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){
61   try {
62     const emps = await Emp.find();
63     if(!emps || !emps.length) return res.status(404).json({error:" no emp data found!});
64     res.status(201).json({
65       emps
66     })
67   } catch (error) {
68     console.log("Error in create controller : ",error.message);
69     res.status(500).json({error:"Internal server Error"});
70   }
71 }
72
73 > export async function update(req,res){ ...
94 }
95
96 > export async function remove(req,res){ ...
110 }
```

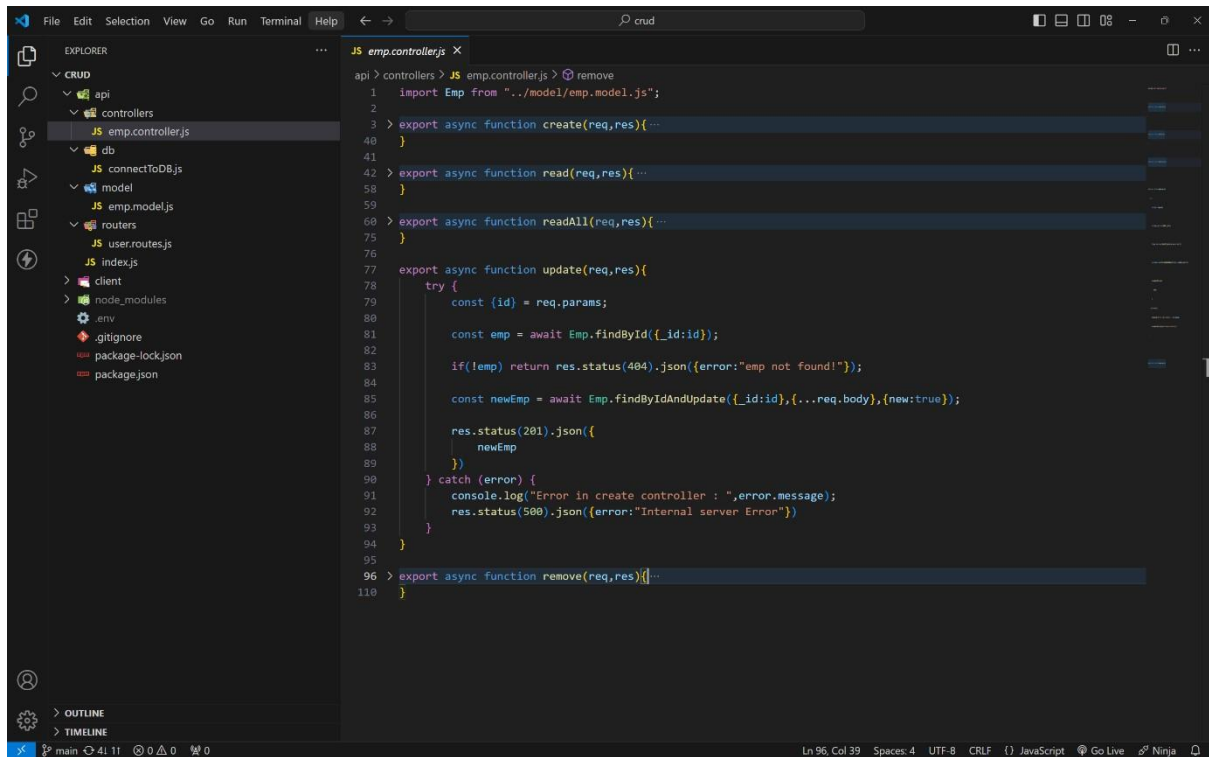
READONE :



The image shows the same VS Code editor environment, but the `read` function in `emp.controller.js` is the focus. This function uses `req.params` to get the employee ID, calls `Emp.findById()`, and returns the single employee object or a 404 status if not found. It also includes the same error handling as the `readAll` function.

```
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     if(!emp) return res.status(404).json({error:"emp not found!});
47     res.status(201).json({
48       emp
49     })
50   } catch (error) {
51     console.log("Error in create controller : ",error.message);
52     res.status(500).json({error:"Internal server Error"});
53   }
54 }
55
56 > 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 }
```

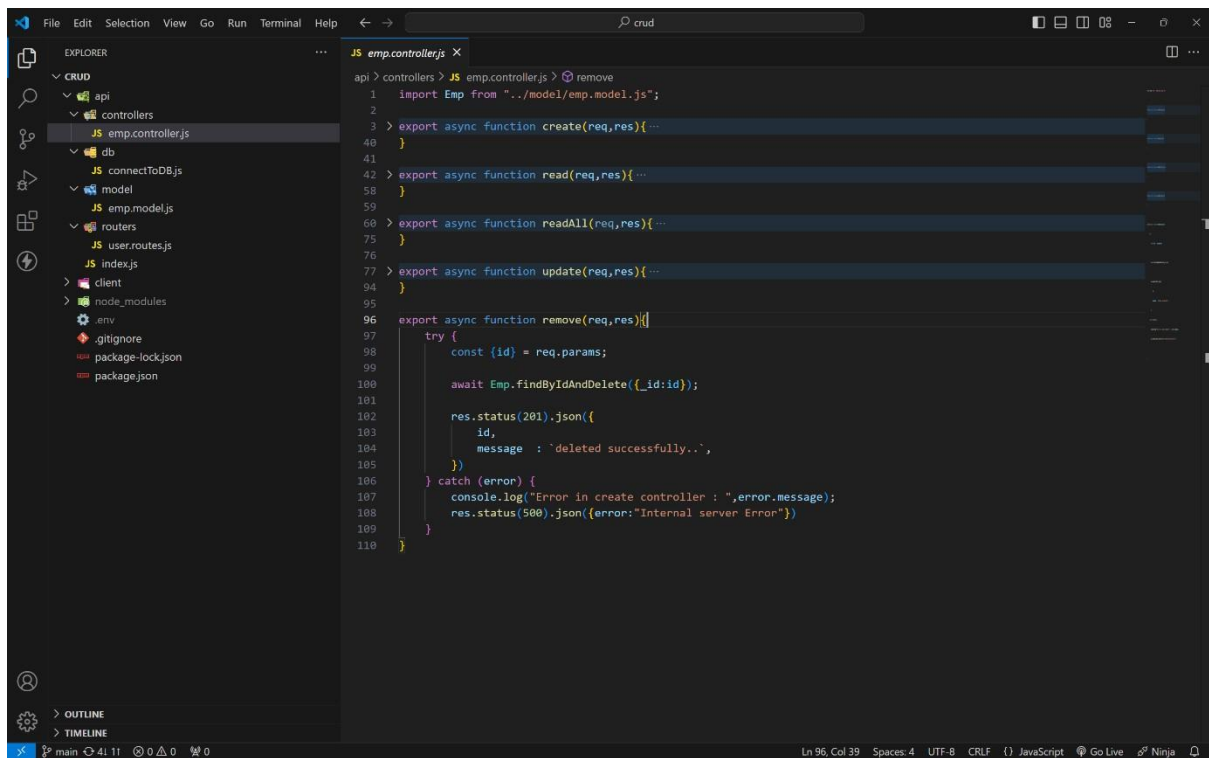
UPDATE :



The screenshot shows the Visual Studio Code editor with the file explorer on the left. 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 'JS emp.controller.js' is selected in the 'controllers' folder. The main editor window shows the code for the 'update' function in 'emp.controller.js'. The code is as follows:

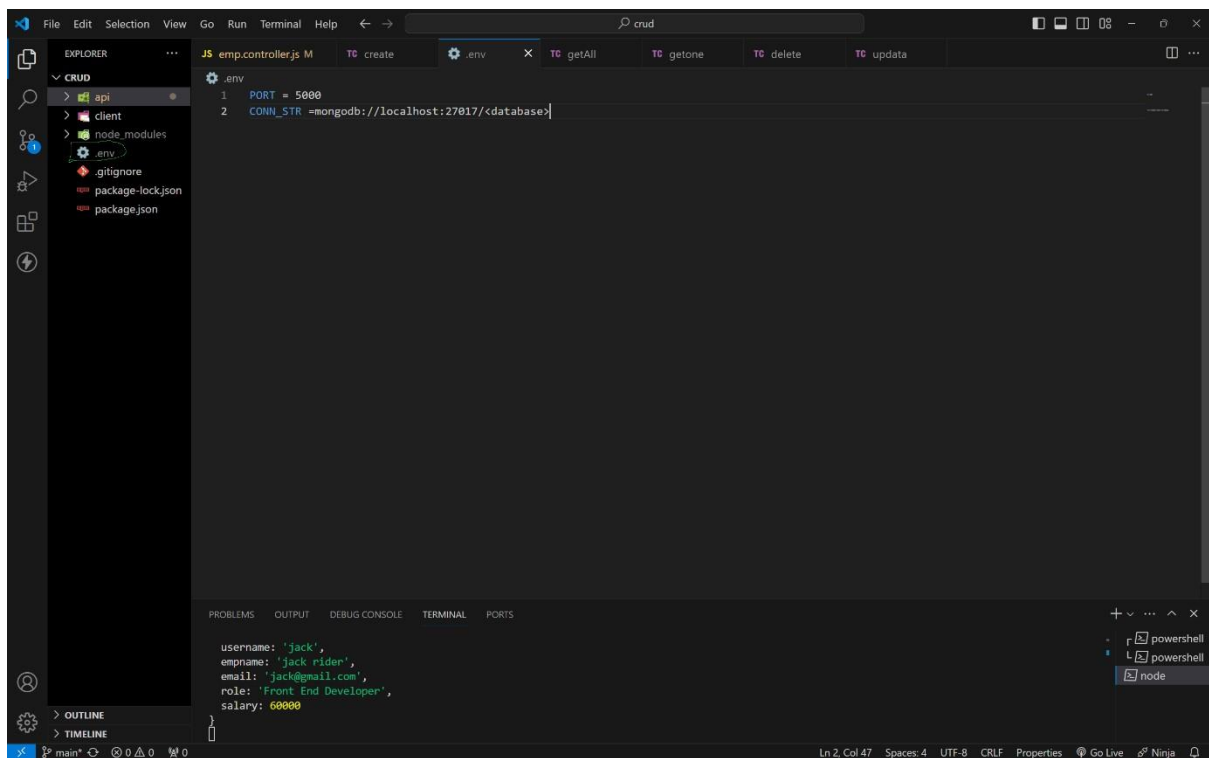
```
api > controllers > JS emp.controller.js > remove
1  import Emp from "../model/emp.model.js";
2
3  > export async function create(req,res){ ...
40 }
41
42 > export async function read(req,res){ ...
58 }
59
60 > export async function readAll(req,res){ ...
75 }
76
77 export async function update(req,res){
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. 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 'JS emp.controller.js' is selected in the 'controllers' folder. The main editor window shows the code for the 'remove' function in 'emp.controller.js'. The code is as follows:

```
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 }
```



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

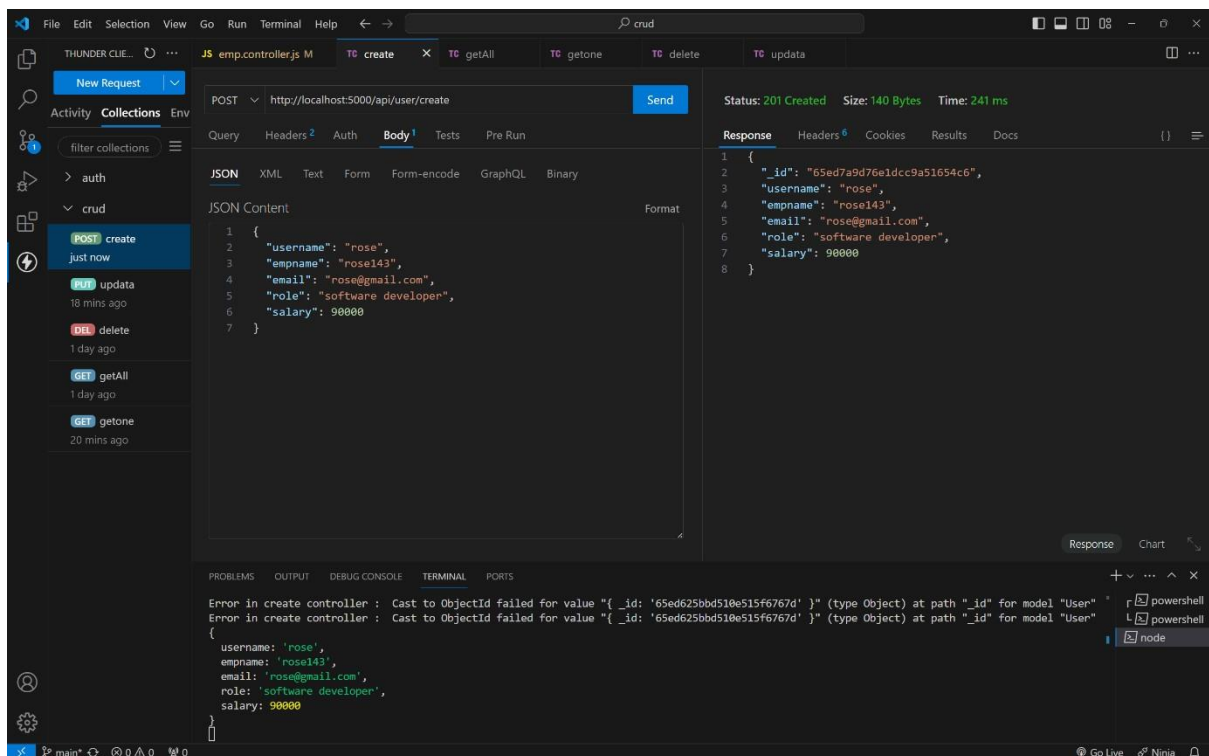
url as

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

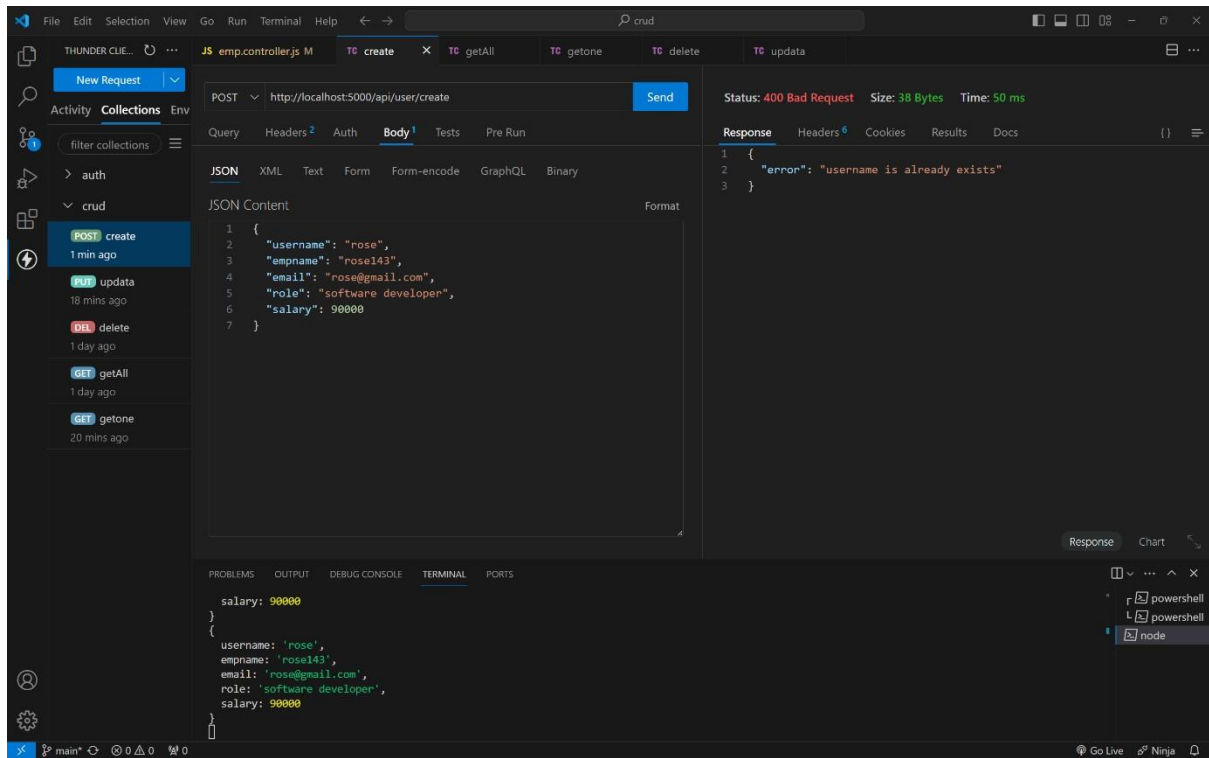
2 . After sending you will deleted successfully as response.

OUTPUT :

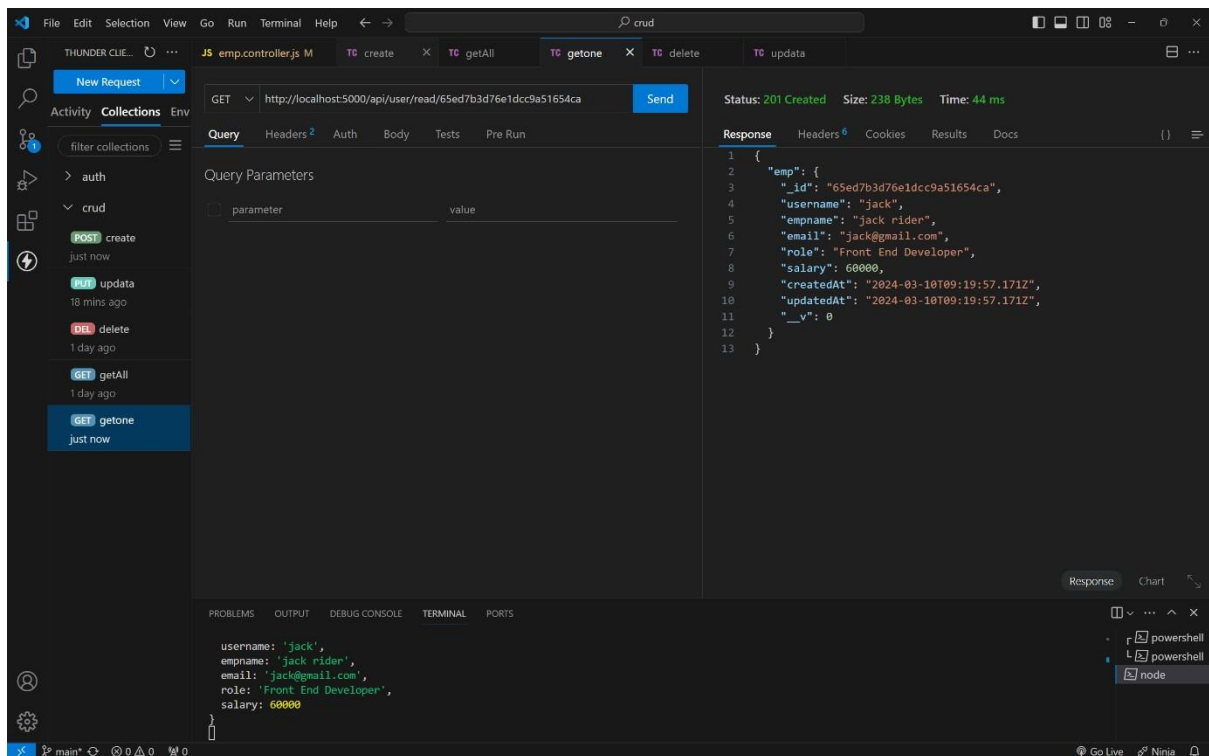
CREATE A NEW USER :



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. The terminal shows the output of the response.

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

Status: 201 Created Size: 468 Bytes Time: 130 ms

Response:

```
1 {
2   "emps": [
3     {
4       "_id": "65ed7a9d76e1dcc9a51654c6",
5       "username": "rose",
6       "empname": "rosel43",
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 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 request body is a JSON object with updated employee details. The response is a JSON object with the updated employee details. The terminal shows an error message.

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

Status: 201 Created Size: 246 Bytes Time: 213 ms

Request Body (JSON):

```
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 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 68-byte body and a response time of 111 ms. The response body is a JSON object: `{ "id": "65ed7b3d76e1dcc9a51654ca", "message": "deleted successfully.." }`.

Request Details:

- Method: DELETE
- URL: `http://localhost:5000/api/user/remove/65ed7b3d76e1dcc9a51654ca`
- Query Parameters: None

Response Details:

- Status: 201 Created
- Size: 68 Bytes
- Time: 111 ms
- Body:

```
{
  "id": "65ed7b3d76e1dcc9a51654ca",
  "message": "deleted successfully.."
}
```

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
```