



Assignment-II

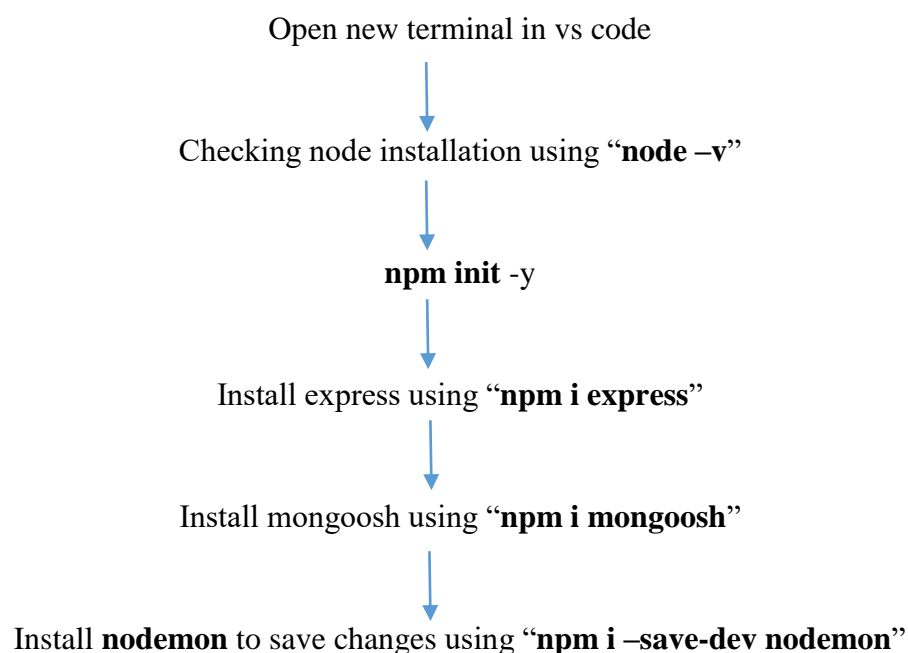
Build a RESTful API with MongoDB using Node.js and Express

Objective:

Expand on the previous assignment by integrating MongoDB as a database, testing concepts such as database interaction, CRUD operations, and advanced middleware handling.

∞ Connect to MongoDB ∞

Initiate MongoDB project:



Set-up package.json file:

- Add a start script –**"nodemon index.js"**
- Check dependencies and dev-dependencies
- Change type is commonjs to **module**.

Mongo-Compass:

- Install Mongo-Compass as per the official instructions
- Make connection
- Open command prompt
- Give commands to manage database

Commands for command prompt

1. node -v
2. mongosh
3. db.createCollection("users")
4. show dbs

```
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=m
ongosh+2.5.1
Using MongoDB:      8.0.9
Using Mongosh:      2.5.1
mongosh 2.5.2 is available for download: https://www.mongodb.com/try/download/shell

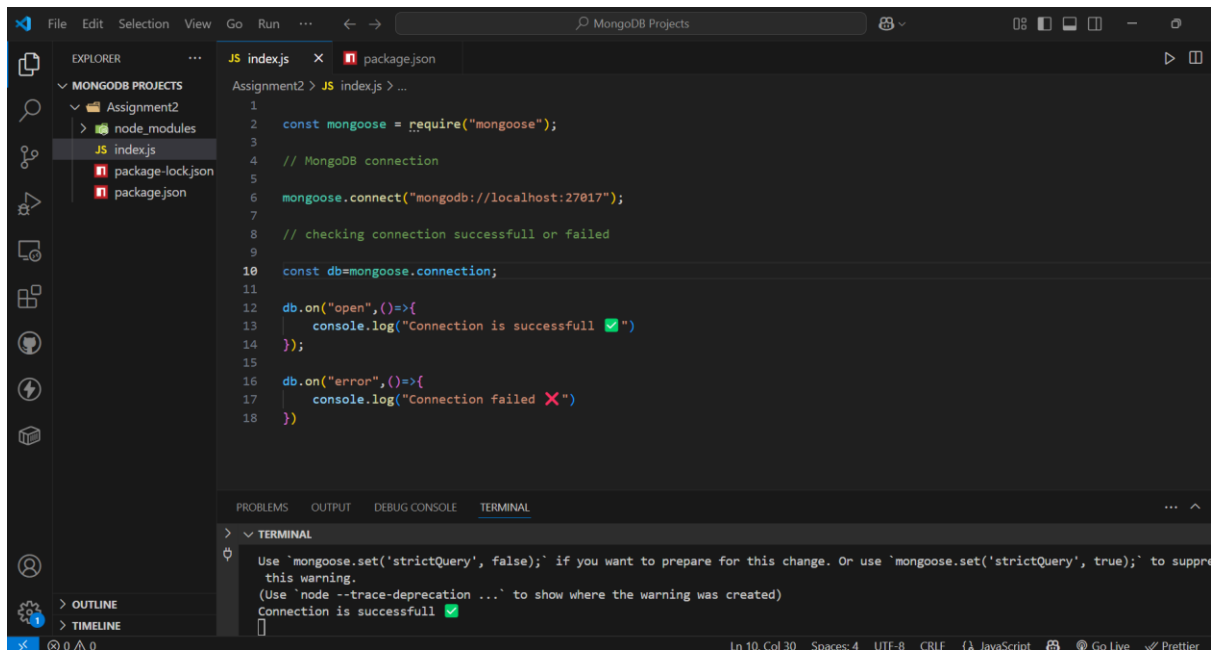
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

-----
The server generated these startup warnings when booting
2025-05-29T11:15:06.578+05:30: Access control is not enabled for the database. Read and write access to data
and configuration is unrestricted
-----

test> db.createCollection("users")
{ ok: 1 }
test> db.users.insertOne({ name:"Hemalatha",age:23,email:"themalatha2810@gmail.com",isActive:true})
{
  acknowledged: true,
  insertedId: ObjectId('684d33ea4823201a5b6c4bd0')
}
test> show collection
MongoshInvalidInputError: [COMMON-10001] 'collection' is not a valid argument for "show".
test> show dbs
admin      40.00 KiB
config     48.00 KiB
local      40.00 KiB
products   72.00 KiB
test       40.00 KiB
test>
```

Screenshot

Set-up and initiate MongoDB project

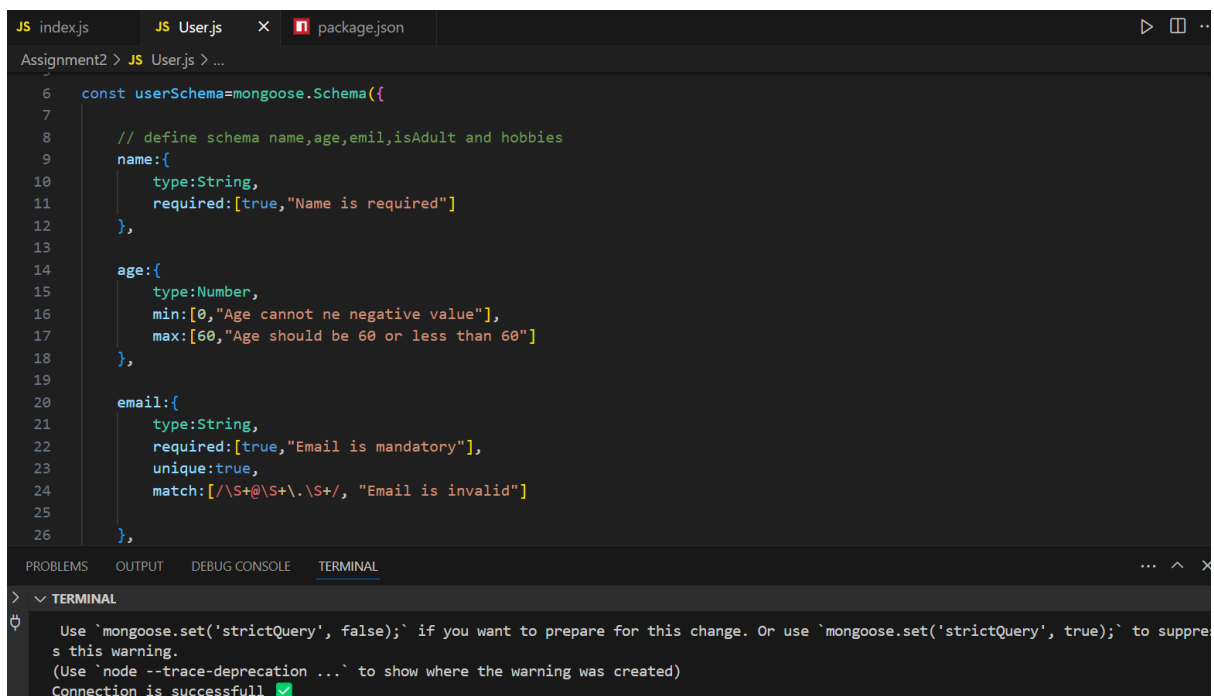


```
File Edit Selection View Go Run ... MongoDB Projects
EXPLORER
  MONGODB PROJECTS
    Assignment2
      node_modules
      JS index.js
      package-lock.json
      package.json
  OUTLINE
  TIMELINE

JS index.js
1  const mongoose = require("mongoose");
2
3  // MongoDB connection
4
5  mongoose.connect("mongodb://localhost:27017");
6
7  // checking connection successfull or failed
8
9
10 const db=mongoose.connection;
11
12 db.on("open",()=>{
13   console.log("Connection is successfull ✅")
14 });
15
16 db.on("error",()=>{
17   console.log("Connection failed ❌")
18 })

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> TERMINAL
Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
Connection is successfull ✅
Ln 10, Col 30 Spaces: 4 UTF-8 CRLF JavaScript Go Live Prettier
```

Created a schema for storing user data

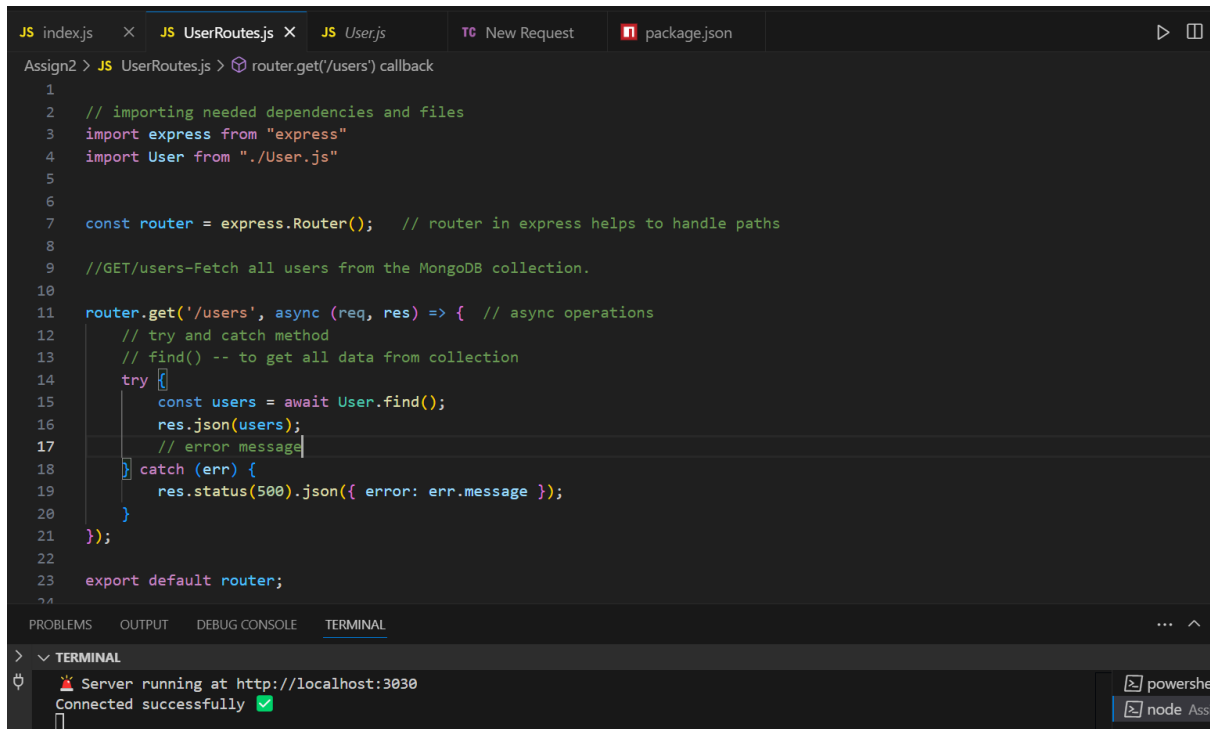


```
JS index.js JS User.js package.json
Assignment2 > JS User.js > ...
6  const userSchema=mongoose.Schema({
7
8    // define schema name,age,emil,isAdult and hobbies
9    name:{
10     type:String,
11     required:[true,"Name is required"]
12   },
13
14   age:{
15     type:Number,
16     min:[0,"Age cannot ne negative value"],
17     max:[60,"Age should be 60 or less than 60"]
18   },
19
20   email:{
21     type:String,
22     required:[true,"Email is mandatory"],
23     unique:true,
24     match:[/^\S+@\S+\.\S+/, "Email is invalid"]
25   },
26 },
27 );

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> TERMINAL
Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
Connection is successfull ✅
```

Define REST API Routes with MongoDB

- 1) **GET/users**– Fetch all users from the MongoDB collection.

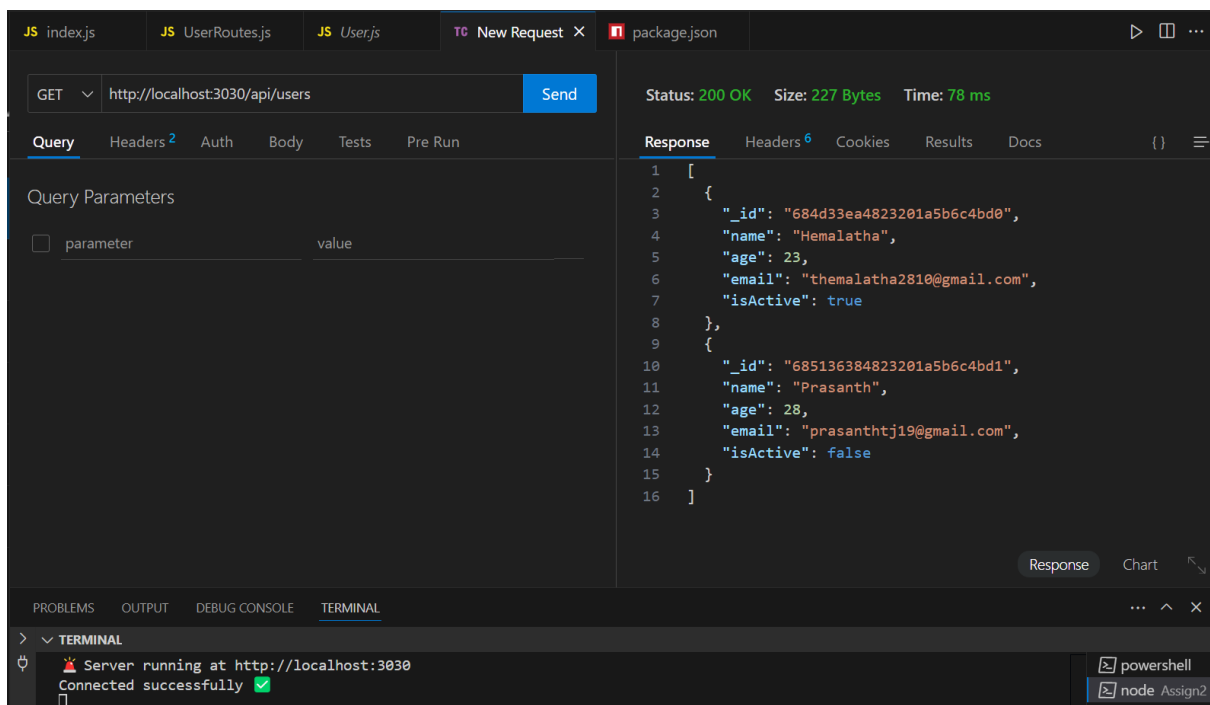


```
JS index.js x JS UserRoutes.js x JS User.js TC New Request package.json
Assign2 > JS UserRoutes.js > router.get('/users') callback
1
2 // importing needed dependencies and files
3 import express from "express"
4 import User from "./User.js"
5
6
7 const router = express.Router(); // router in express helps to handle paths
8
9 //GET/users-Fetch all users from the MongoDB collection.
10
11 router.get('/users', async (req, res) => { // async operations
12   // try and catch method
13   // find() -- to get all data from collection
14   try {
15     const users = await User.find();
16     res.json(users);
17     // error message
18   } catch (err) {
19     res.status(500).json({ error: err.message });
20   }
21 });
22
23 export default router;
24
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Server running at http://localhost:3030
Connected successfully

Response in Thunder-Client



```
JS index.js JS UserRoutes.js JS User.js TC New Request x package.json
GET http://localhost:3030/api/users Send
Status: 200 OK Size: 227 Bytes Time: 78 ms
Query Headers 2 Auth Body Tests Pre Run
Query Parameters
parameter value
Response Headers 6 Cookies Results Docs {}
1 [
2   {
3     "_id": "684d33ea4823201a5b6c4bd0",
4     "name": "Hemalatha",
5     "age": 23,
6     "email": "themalatha2810@gmail.com",
7     "isActive": true
8   },
9   {
10    "_id": "685136384823201a5b6c4bd1",
11    "name": "Prasanth",
12    "age": 28,
13    "email": "prasanthtj19@gmail.com",
14    "isActive": false
15  }
16 ]
Response Chart
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Server running at http://localhost:3030
Connected successfully
```

Commands in terminal

```
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

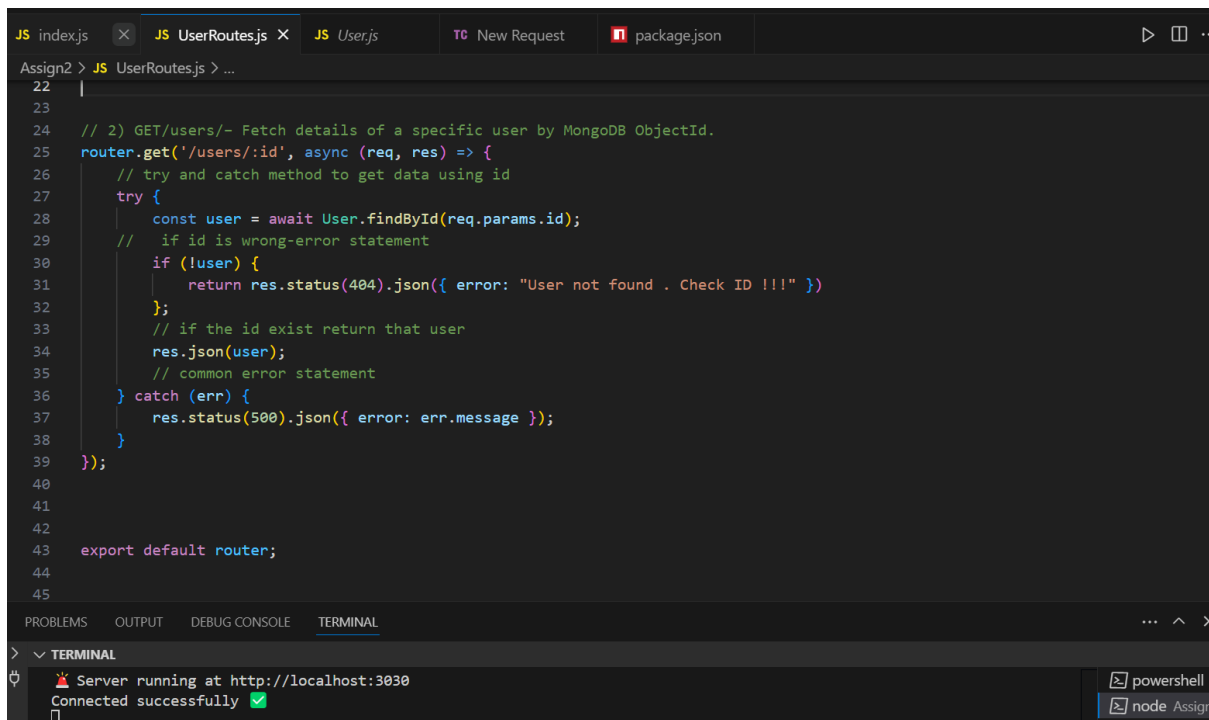
-----
The server generated these startup warnings when booting
2025-05-29T11:15:06.578+05:30: Access control is not enabled for the database. Read and write access to data
and configuration is unrestricted
-----

test> db.createCollection("users")
{ ok: 1 }
test> db.users.insertOne({ name:"Hemalatha",age:23,email:"themalatha2810@gmail.com",isActive:true})
{
  acknowledged: true,
  insertedId: ObjectId('684d33ea4823201a5b6c4bd0')
}
test> show collection
MongoshInvalidInputError: [COMMON-10001] 'collection' is not a valid argument for "show".
test> show dbs
admin      40.00 KiB
config    48.00 KiB
local     40.00 KiB
products  72.00 KiB
test      40.00 KiB
test> db.users.insertOne({ name:"Prasanth",age:28,email:"prasanthtj19@gmail.com",isActive:false})
{
  acknowledged: true,
  insertedId: ObjectId('685136384823201a5b6c4bd1')
}
test>
```

Mongo-Compass response

The screenshot shows the MongoDB Compass web interface. On the left, the 'CONNECTIONS' sidebar lists the 'test' database under 'localhost:27017'. The main panel displays the 'users' collection with two documents. The first document is for 'Hemalatha' with age 23 and email 'themalatha2810@gmail.com', where 'isActive' is true. The second document is for 'Prasanth' with age 28 and email 'prasanthtj19@gmail.com', where 'isActive' is false. The interface includes a search bar, a query editor, and various action buttons like 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'.

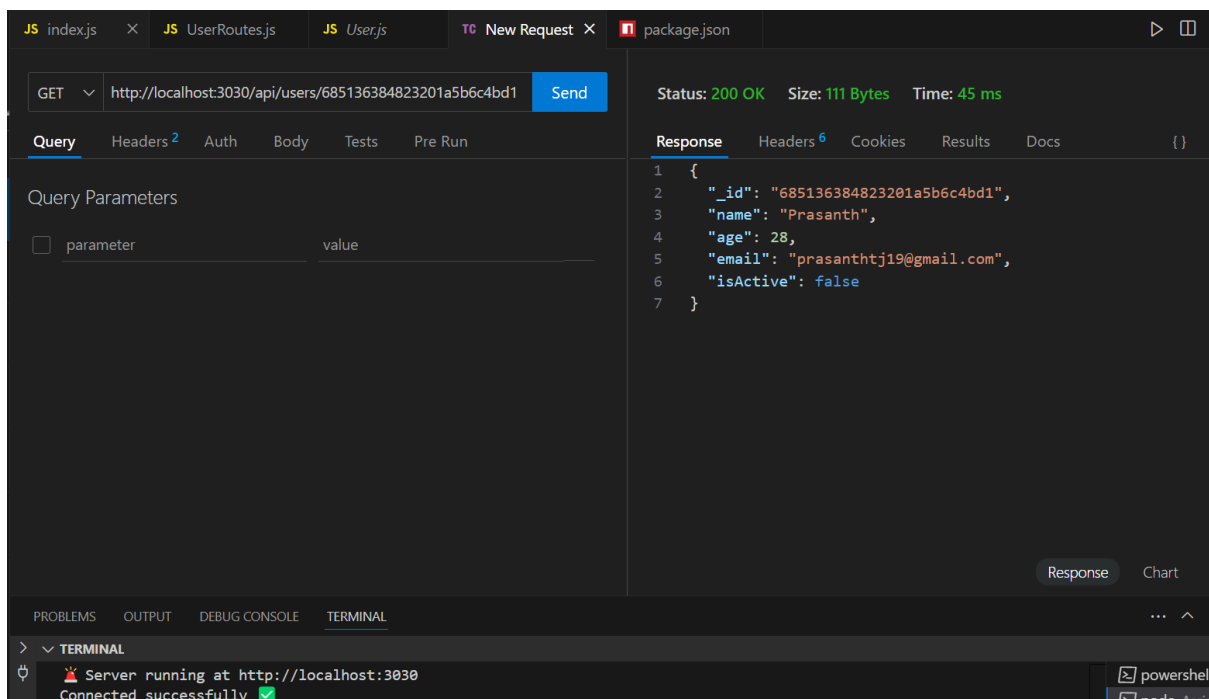
2) GET/users/- Fetch details of a specific user by MongoDB ObjectId



The screenshot shows a VS Code editor with the file `UserRoutes.js` open. The code implements a GET endpoint `/users/:id` using Express.js. It uses `async/await` to fetch a user by their ID from a MongoDB collection. Error handling is implemented with `try/catch` blocks, returning 404 for 'User not found' and 500 for server errors. The terminal at the bottom shows the server is running on `http://localhost:3030` and connected successfully.

```
22 |
23 |
24 | // 2) GET/users/- Fetch details of a specific user by MongoDB ObjectId.
25 | router.get('/users/:id', async (req, res) => {
26 |   // try and catch method to get data using id
27 |   try {
28 |     const user = await User.findById(req.params.id);
29 |     // if id is wrong-error statement
30 |     if (!user) {
31 |       return res.status(404).json({ error: "User not found . Check ID !!!" });
32 |     };
33 |     // if the id exist return that user
34 |     res.json(user);
35 |     // common error statement
36 |   } catch (err) {
37 |     res.status(500).json({ error: err.message });
38 |   }
39 | });
40 |
41 |
42 |
43 | export default router;
44 |
45 |
```

Response in Thunder-Client



The screenshot shows the Thunder Client interface. A GET request has been sent to `http://localhost:3030/api/users/685136384823201a5b6c4bd1`. The response is a 200 OK status with a size of 111 Bytes and a time of 45 ms. The response body is a JSON object containing user details. The terminal at the bottom shows the server is running on `http://localhost:3030` and connected successfully.

GET `http://localhost:3030/api/users/685136384823201a5b6c4bd1` Send

Status: 200 OK Size: 111 Bytes Time: 45 ms

Query Parameters

parameter	value
-----------	-------

Response

```
1 {
2   "_id": "685136384823201a5b6c4bd1",
3   "name": "Prasanth",
4   "age": 28,
5   "email": "prasanthtj19@gmail.com",
6   "isActive": false
7 }
```

Error message if that user id is not exist in DB

The screenshot displays the VS Code interface with a REST client request and response. The request is a GET to `http://localhost:3030/api/users/685136384823201a5b6c4bd8`. The response is a 404 Not Found with a JSON error message: `{\"error\": \"User not found . Check ID !!!\"}`. The terminal shows the server is running at `http://localhost:3030`.

Request:

```
GET http://localhost:3030/api/users/685136384823201a5b6c4bd8
```

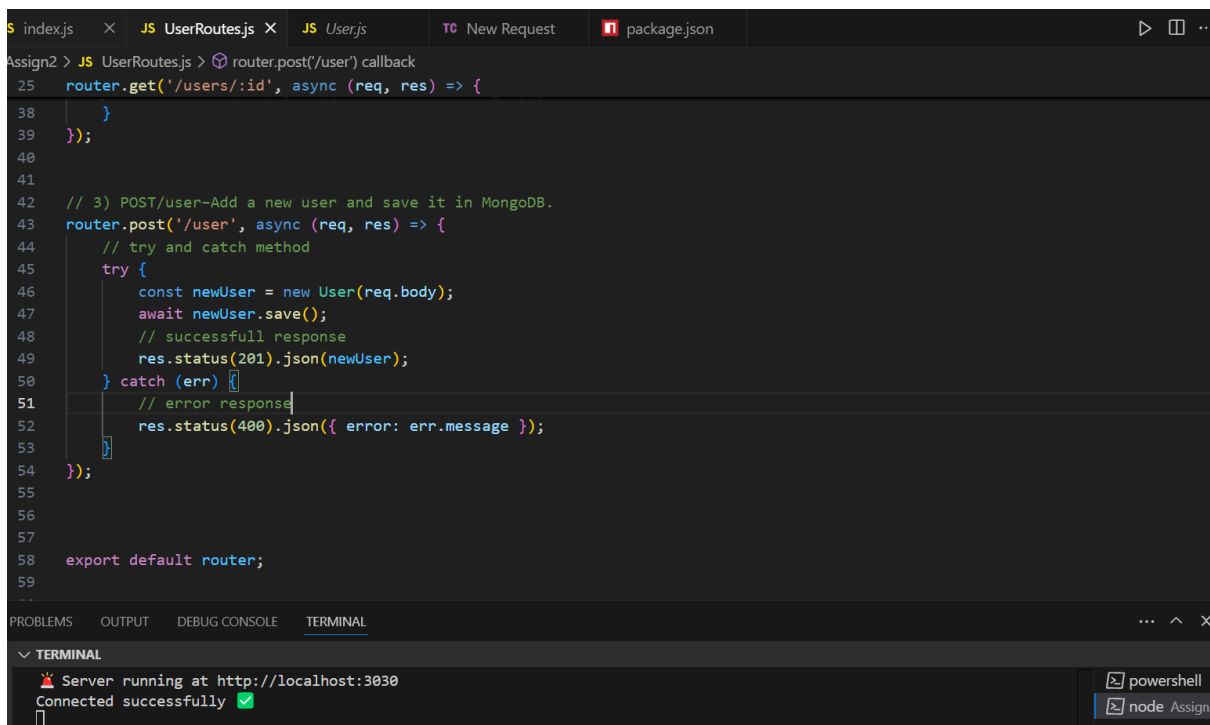
Response:

```
Status: 404 Not Found Size: 41 Bytes Time: 35 ms
{
  "error": "User not found . Check ID !!!"
}
```

Terminal:

```
Server running at http://localhost:3030
Connected successfully
```

3) POST/user—Add a new user and save it in MongoDB.



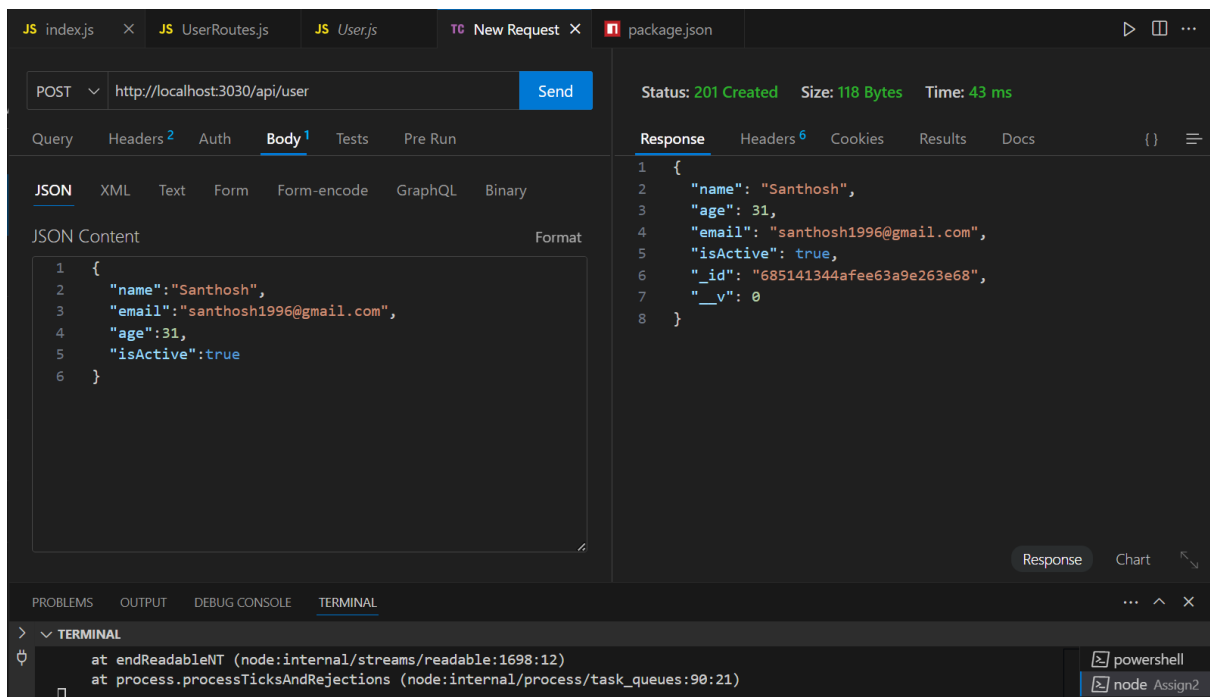
The screenshot shows a VS Code editor with a file named `UserRoutes.js`. The code implements a REST API endpoint for adding a new user. It uses `router.post` to handle the request. Inside the callback, it uses `try-catch` to handle the database operation. If successful, it returns a 201 status code with the user object. If there's an error, it returns a 400 status code with the error message. The terminal at the bottom shows the server is running on `http://localhost:3030` and is connected successfully.

```
25 router.get('/users/:id', async (req, res) => {
38   }
39 });
40
41
42 // 3) POST/user—Add a new user and save it in MongoDB.
43 router.post('/user', async (req, res) => {
44   // try and catch method
45   try {
46     const newUser = new User(req.body);
47     await newUser.save();
48     // successfull response
49     res.status(201).json(newUser);
50   } catch (err) {
51     // error response
52     res.status(400).json({ error: err.message });
53   }
54 });
55
56
57 export default router;
58
59
...
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Server running at http://localhost:3030
Connected successfully

Response in Thunder-Client



The screenshot shows the Thunder-Client interface. A POST request to `http://localhost:3030/api/user` has been sent. The response is a JSON object with the following structure:

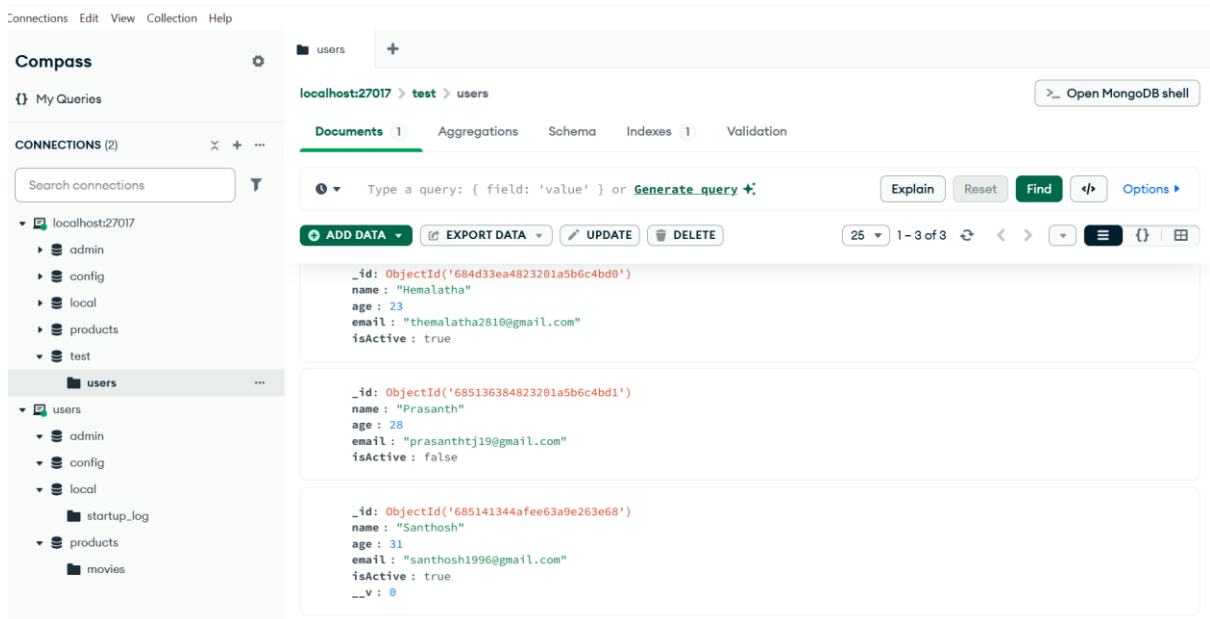
```
{
  "name": "Santhosh",
  "age": 31,
  "email": "santhosh1996@gmail.com",
  "isActive": true,
  "_id": "685141344afee63a9e263e68",
  "__v": 0
}
```

The status is 201 Created, the size is 118 Bytes, and the time is 43 ms. The terminal at the bottom shows the server is running on `http://localhost:3030` and is connected successfully.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

at endReadableNT (node:internal/streams/readable:1698:12)
at process.processTicksAndRejections (node:internal/process/task_queues:90:21)

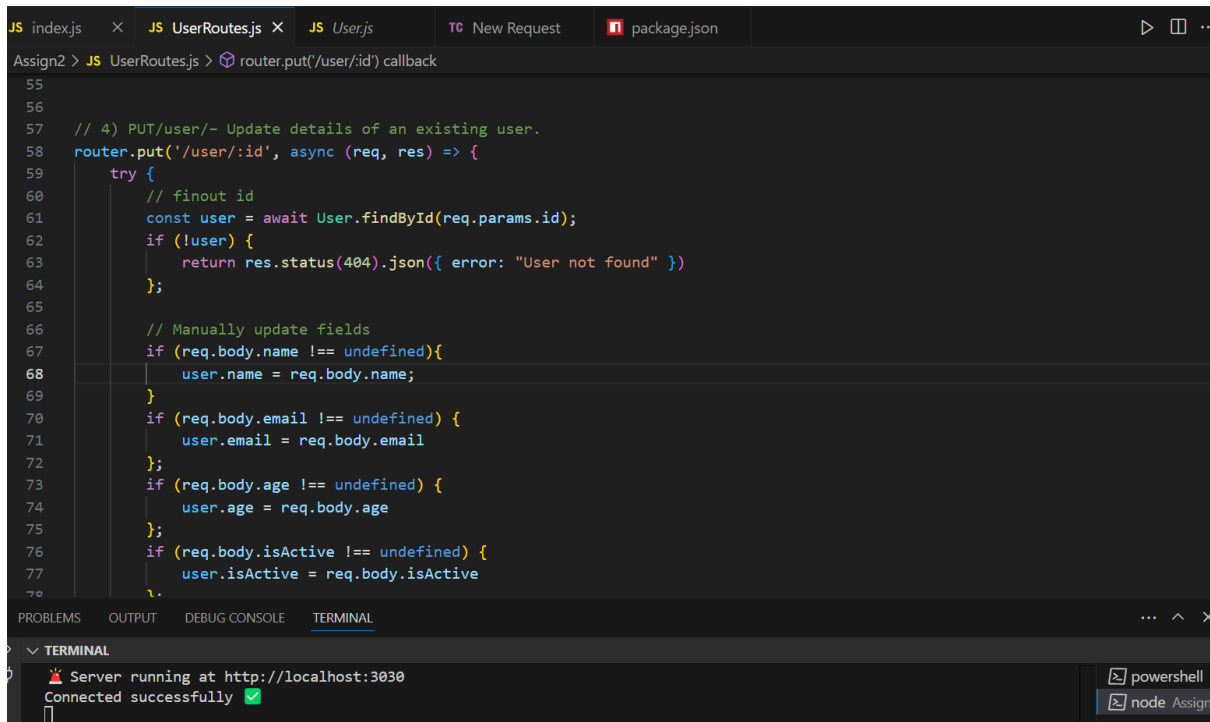
Added user data into MongoDB users data



The screenshot shows the MongoDB Compass interface. On the left, the 'CONNECTIONS (2)' panel lists 'localhost:27017' with a tree view of databases including 'users'. The main panel displays the 'users' collection from the 'test' database on 'localhost:27017'. The 'Documents' tab is active, showing three documents. Each document contains fields for '_id', 'name', 'age', 'email', and 'isActive'.

Document 1	Document 2	Document 3
<pre>{ "_id": ObjectId("684d33ea4823201a5b6c4bd0"), "name": "Hemalatha", "age": 23, "email": "themalatha2810@gmail.com", "isActive": true }</pre>	<pre>{ "_id": ObjectId("685136384823201a5b6c4bd1"), "name": "Prasanth", "age": 28, "email": "prasanthtj19@gmail.com", "isActive": false }</pre>	<pre>{ "_id": ObjectId("685141344afee63a9e263e68"), "name": "Santhosh", "age": 31, "email": "santhosh1996@gmail.com", "isActive": true }</pre>

4) PUT/user/- Update details of an existing user.

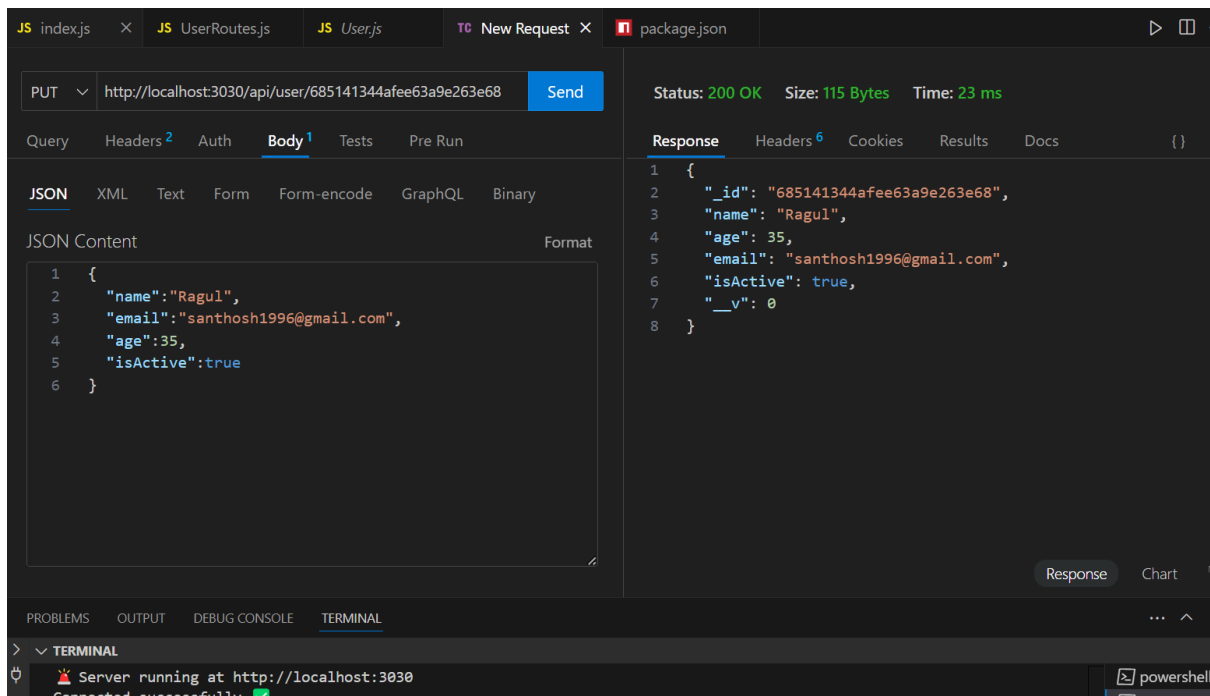


```
55
56
57 // 4) PUT/user/- Update details of an existing user.
58 router.put('/user/:id', async (req, res) => {
59   try {
60     // findout id
61     const user = await User.findById(req.params.id);
62     if (!user) {
63       return res.status(404).json({ error: "User not found" });
64     };
65
66     // Manually update fields
67     if (req.body.name !== undefined){
68       user.name = req.body.name;
69     }
70     if (req.body.email !== undefined) {
71       user.email = req.body.email
72     };
73     if (req.body.age !== undefined) {
74       user.age = req.body.age
75     };
76     if (req.body.isActive !== undefined) {
77       user.isActive = req.body.isActive
78     };
79   }
80 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Server running at http://localhost:3030
Connected successfully

Response in Thunder-Client



PUT http://localhost:3030/api/user/685141344afee63a9e263e68 Send

Status: 200 OK Size: 115 Bytes Time: 23 ms

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "name": "Ragul",
3   "email": "santhosh1996@gmail.com",
4   "age": 35,
5   "isActive": true
6 }
```

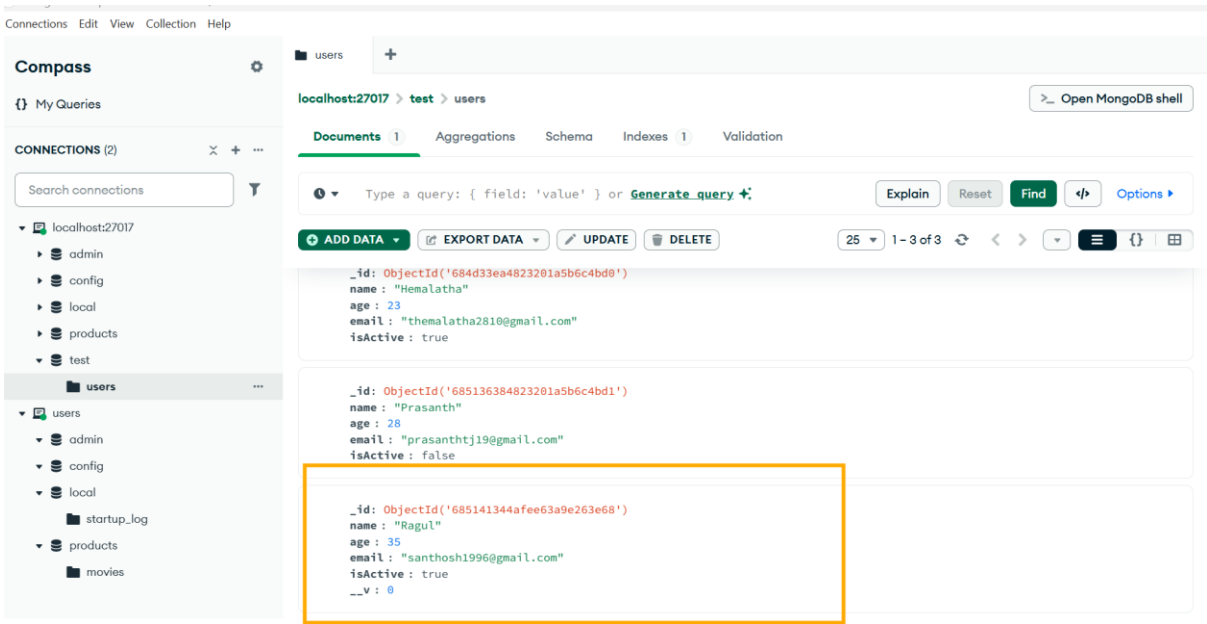
Response Headers 6 Cookies Results Docs {}

```
1 {
2   "_id": "685141344afee63a9e263e68",
3   "name": "Ragul",
4   "age": 35,
5   "email": "santhosh1996@gmail.com",
6   "isActive": true,
7   "__v": 0
8 }
```

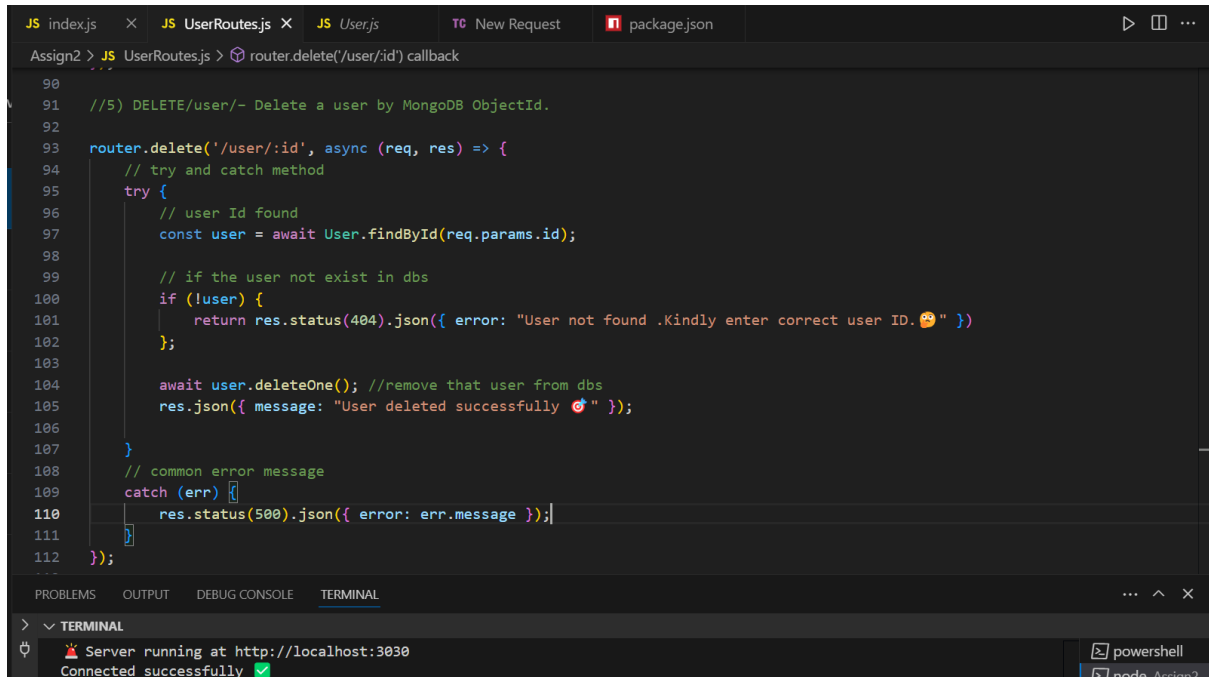
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Server running at http://localhost:3030
Connected successfully

Updated that user data in MongoDB



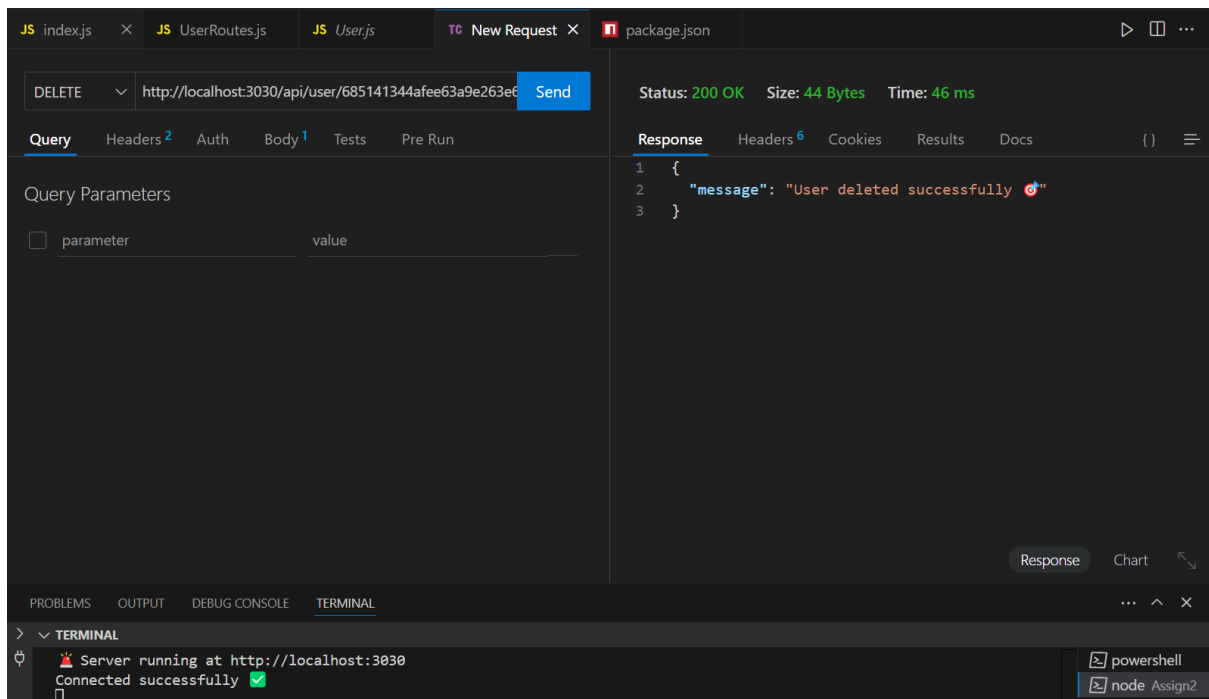
5) DELETE/user/- Delete a user by MongoDB ObjectId



The screenshot shows a VS Code editor with the file `UserRoutes.js` open. The code implements a `router.delete('/user/:id')` endpoint. It uses `async (req, res) => {}` syntax. Inside a `try` block, it finds the user by ID using `User.findById(req.params.id)`. If the user is not found, it returns a 404 status with a message. If found, it calls `user.deleteOne()` to remove the user from the database and returns a 200 status with a success message. A `catch` block handles any errors, returning a 500 status with the error message. The terminal at the bottom shows the server is running at `http://localhost:3030` and is connected successfully.

```
90
91 //5) DELETE/user/- Delete a user by MongoDB ObjectId.
92
93 router.delete('/user/:id', async (req, res) => {
94   // try and catch method
95   try {
96     // user Id found
97     const user = await User.findById(req.params.id);
98
99     // if the user not exist in dbs
100    if (!user) {
101      return res.status(404).json({ error: "User not found .Kindly enter correct user ID. 😞" });
102    };
103
104    await user.deleteOne(); //remove that user from dbs
105    res.json({ message: "User deleted successfully 🍕" });
106
107   }
108   // common error message
109   catch (err) {
110     res.status(500).json({ error: err.message });
111   }
112 });
```

Response in Thunder-Client



The screenshot shows the Thunder Client interface. A DELETE request has been sent to `http://localhost:3030/api/user/685141344afee63a9e263e6`. The response is shown in the right pane, indicating a status of 200 OK, size of 44 Bytes, and time of 46 ms. The response body is a JSON object: `{ "message": "User deleted successfully 🍕" }`. The terminal at the bottom shows the server is running at `http://localhost:3030` and is connected successfully.

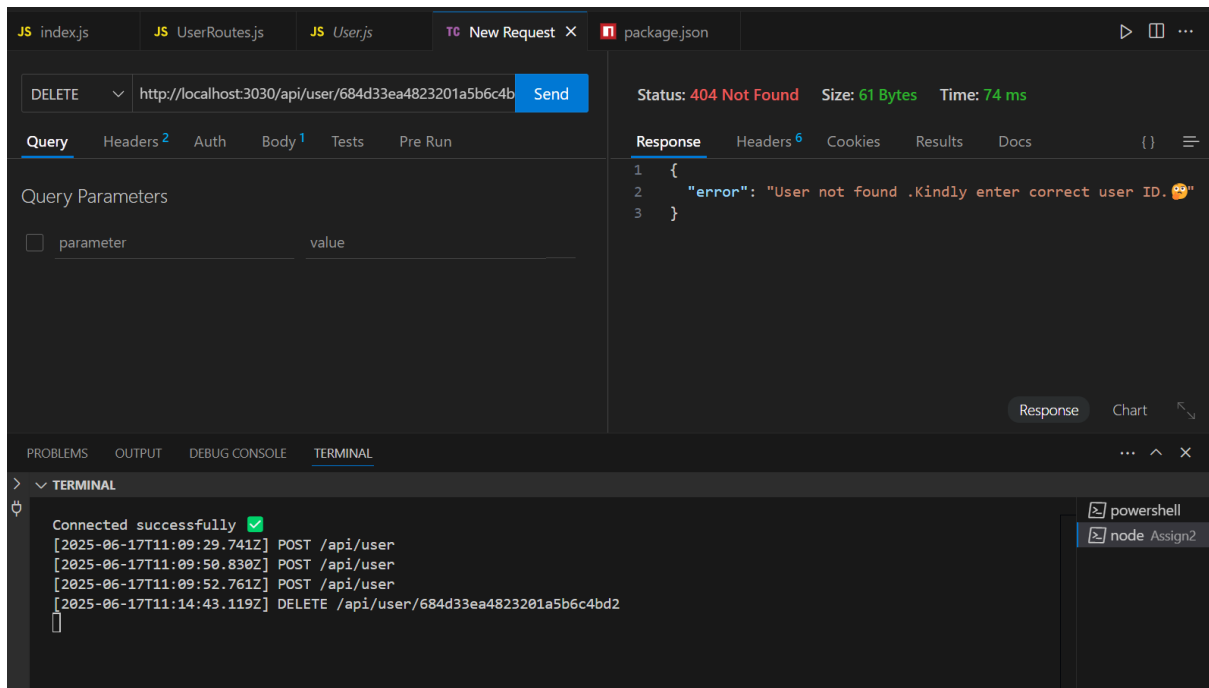
DELETE `http://localhost:3030/api/user/685141344afee63a9e263e6` Send

Status: 200 OK Size: 44 Bytes Time: 46 ms

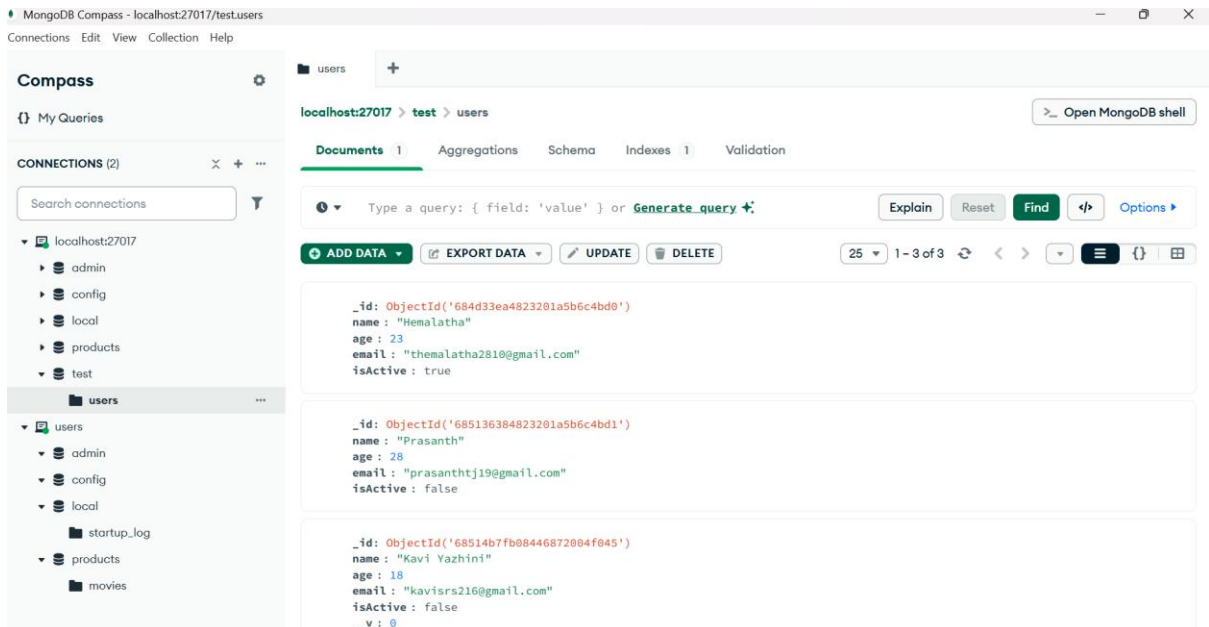
Response

```
1 {
2   "message": "User deleted successfully 🍕"
3 }
```

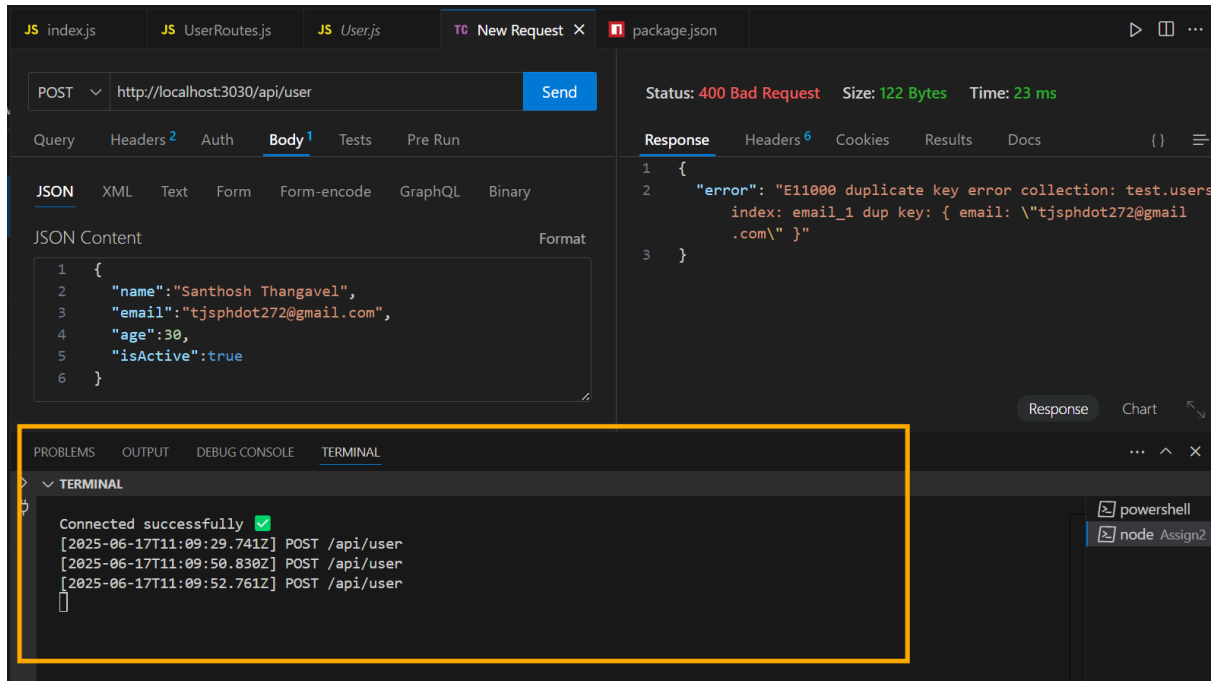
If the user not exist-wrong id is given



View the response in MongoDB compass



Logging details of task performance in terminal



Conclusion

- I have connected to NodeJS application using Mongoose to a MongoDB application
- I have created schema with proper validation
- I have performed CRUD operations
- I have defined REST API routes with MongoDB
- Provided clear comments

GitHub link

<https://github.com/Hema2802/RESTful-API-MongoDB-NodeJS-Express>