RESTful API with Node.js and Express

Introduction

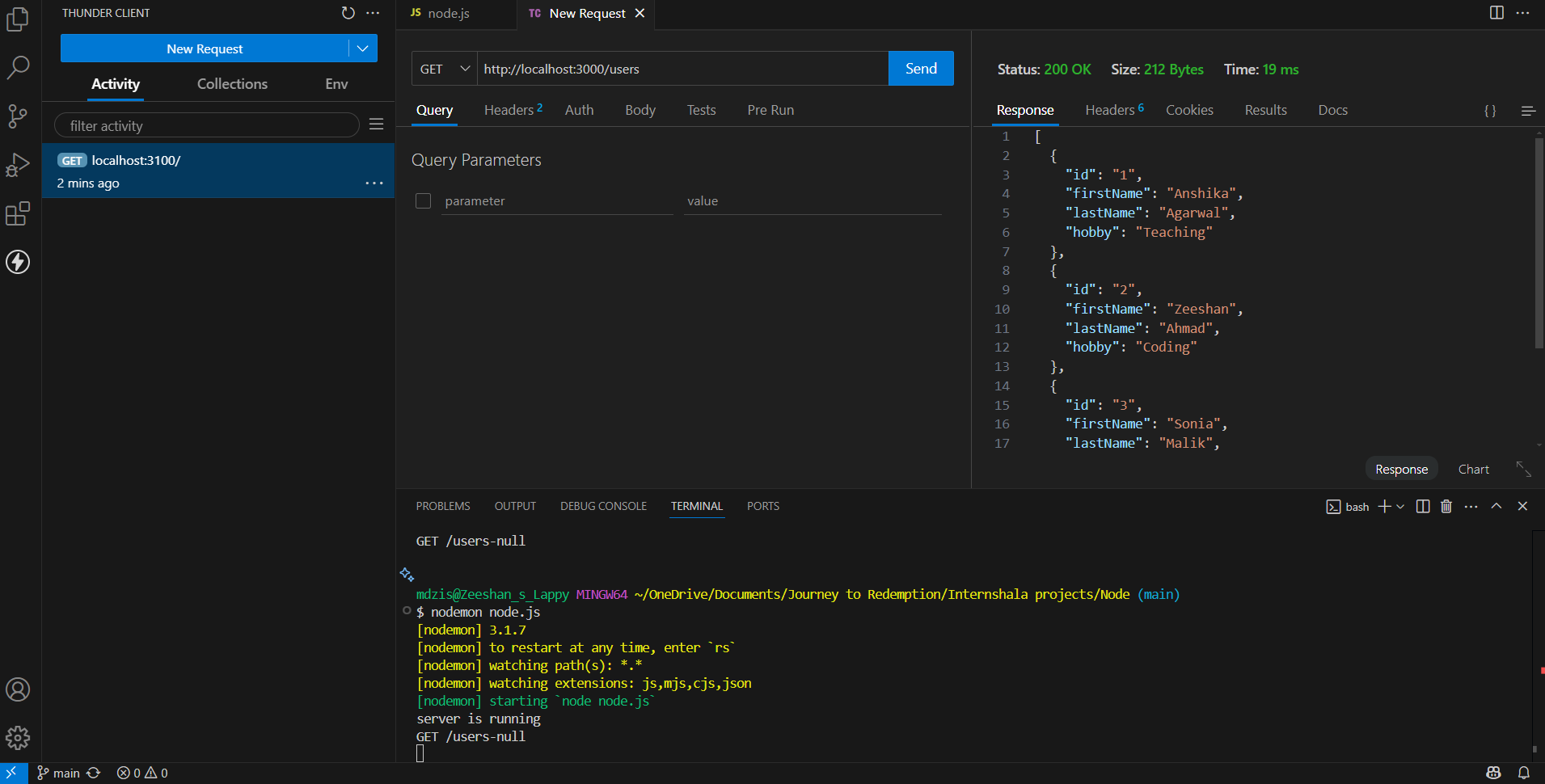
This project demonstrates the implementation of a simple RESTful API using Node.js and Express.js. The API performs CRUD operations on user data stored in memory. It incorporates routing, middleware, error handling, and proper use of HTTP methods and status codes. The API was tested using ThunderClient.

Code Summary and Comments

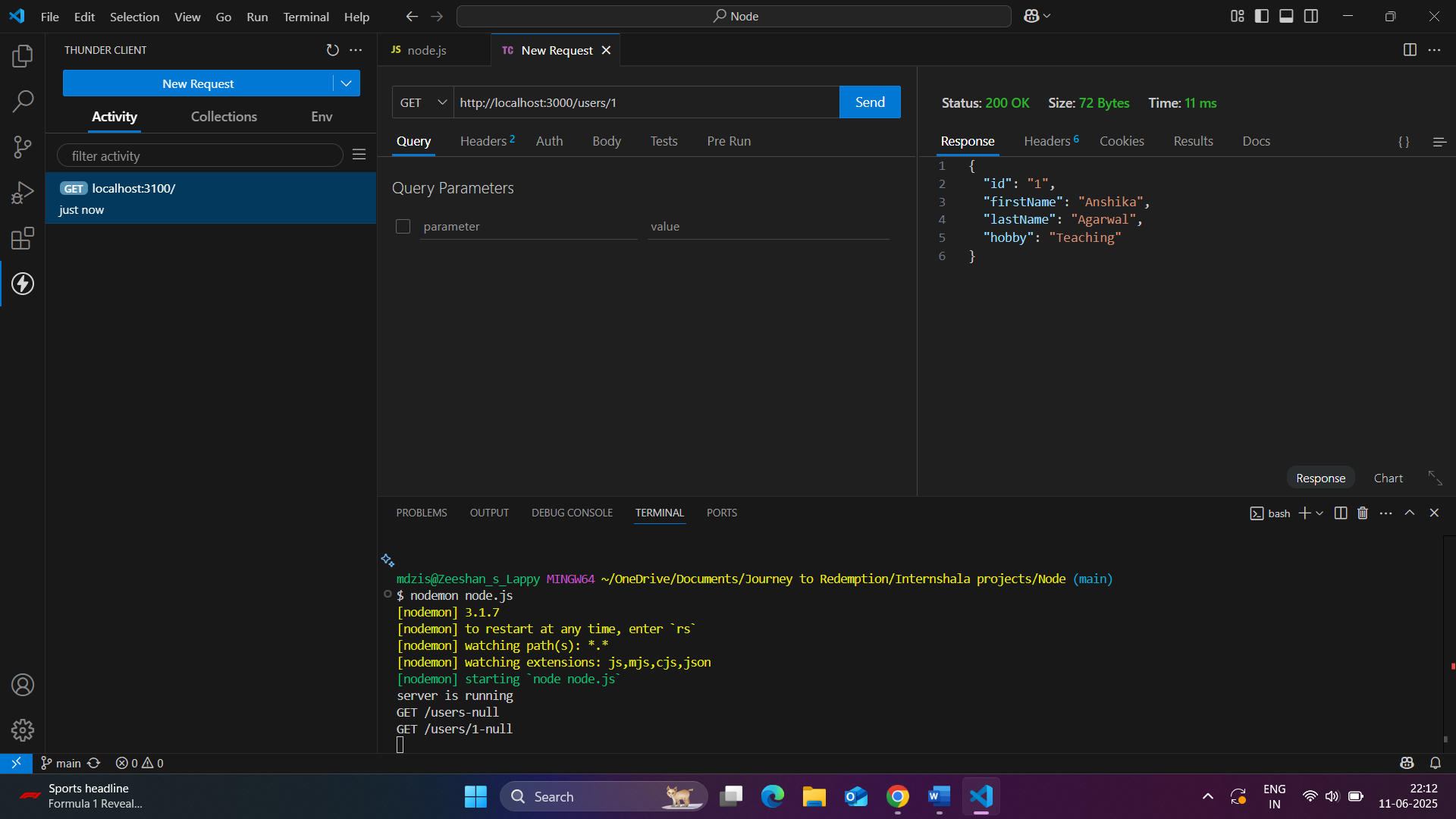
// Importing express and initializing the app  
import express from "express";  
const app = express();  
  
// Middleware to parse JSON requests  
app.use(express.json());  
  
// Middleware to log HTTP method, URL, and response status  
app.use((req, res, next) => {  
 res.on("finish", () => {  
 console.log(`${req.method} ${req.url}-${req.statusCode}`);  
 });  
 next();  
});  
  
// In-memory user data array  
let userData = [  
 { id: "1", firstName: "Anshika", lastName: "Agarwal", hobby: "Teaching" },  
 { id: "2", firstName: "Zeeshan", lastName: "Ahmad", hobby: "Coding" },  
 { id: "3", firstName: "Sonia", lastName: "Malik", hobby: "teaching" }  
];  
  
// Validation middleware for POST/PUT  
const validate = (req, res, next) => {  
 const { firstName, lastName, hobby } = req.body;  
 if (  
 typeof firstName !== "string" ||  
 typeof lastName !== "string" ||  
 typeof hobby !== "string"  
 ) {  
 return res.status(400).send("Please enter values in strings");  
 }  
 next();  
};  
  
// GET /users - Fetch all users  
app.get("/users", (req, res) => res.send(userData));  
  
// GET /users/:id - Fetch a user by ID  
app.get("/users/:id", (req, res) => {  
 const userID = req.params.id;  
 const user = userData.find((user) => user.id == userID);  
 if (!user) return res.status(404).json({ message: "User not found" });  
 res.status(200).send(user);  
});  
  
// POST /user - Add new user  
app.post("/user", validate, (req, res) => {  
 const { firstName, lastName, hobby } = req.body;  
 const newUser = {  
 id: Math.floor(Math.random() \* 1000).toString(),  
 firstName,  
 lastName,  
 hobby,  
 };  
 userData.push(newUser);  
 res.status(201).send(userData);  
});  
  
// PUT /user/:id - Update user details  
app.put("/user/:id", validate, (req, res) => {  
 const userID = req.params.id;  
 if (Object.keys(req.body).length === 0)  
 return res.status(404).json({ message: "No fields to update" });  
 const editUser = userData.find((user) => user.id == userID);  
 if (!editUser)  
 return res.status(404).json({ message: "user not found" });  
 const keys = Object.keys(req.body);  
 keys.forEach((key) => {  
 editUser[key] = req.body[key];  
 });  
 res.send(userData);  
});  
  
// DELETE /user/:id - Delete user by ID  
app.delete("/user/:id", (req, res) => {  
 const userID = req.params.id;  
 const initialLength = userData.length;  
 userData = userData.filter((user) => user.id != userID);  
 if (userData.length === initialLength)  
 return res.status(404).json({ message: "User not found" });  
 res.send({ message: "User deleted ", users: userData });  
});  
  
// Start server  
app.listen(3000, () => console.log("server is running"));

API Testing Screenshots

## Screenshot 1: GET /users – Response with list of users



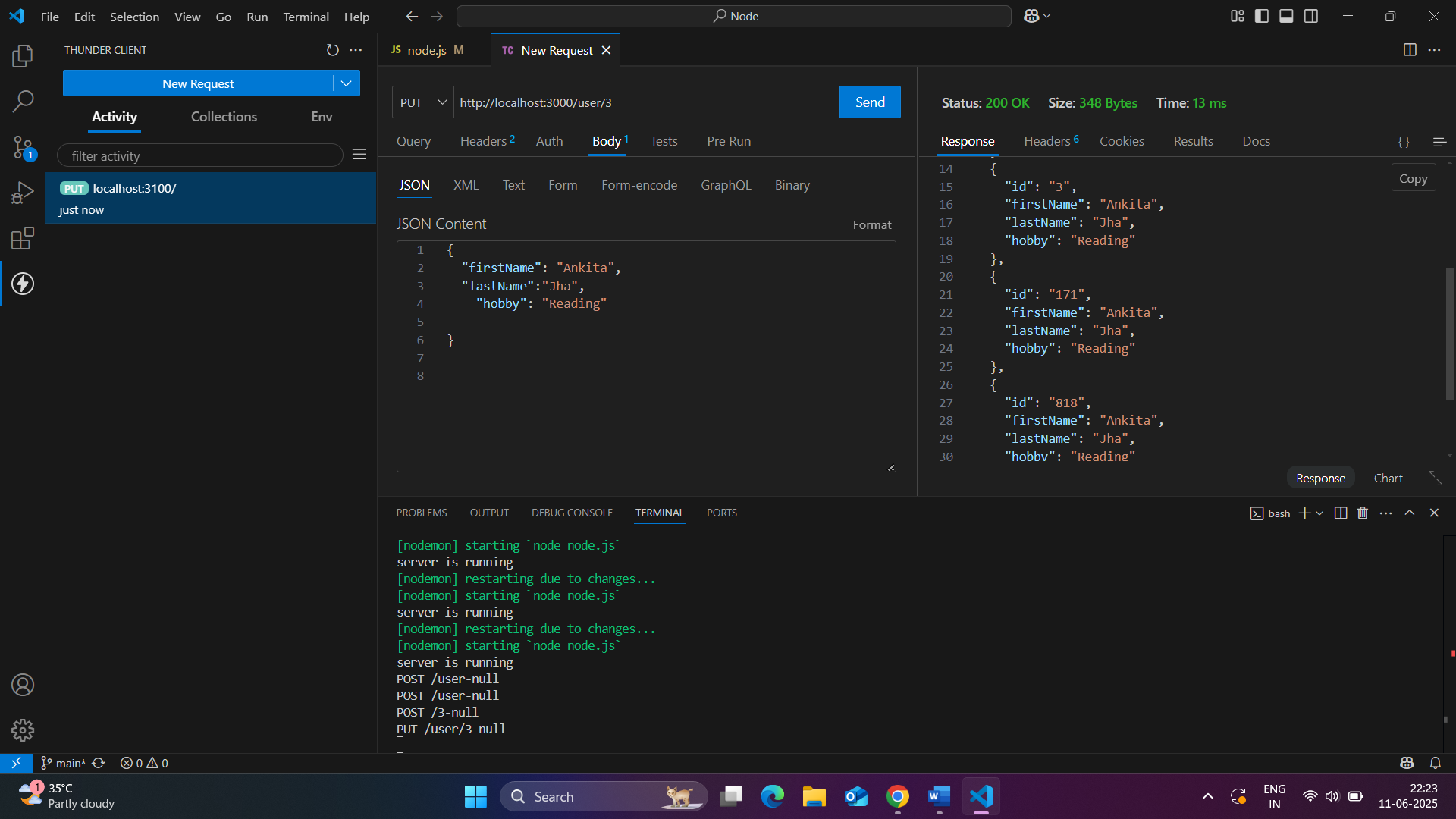
## Screenshot 2: GET /users/:id – Response with a specific user



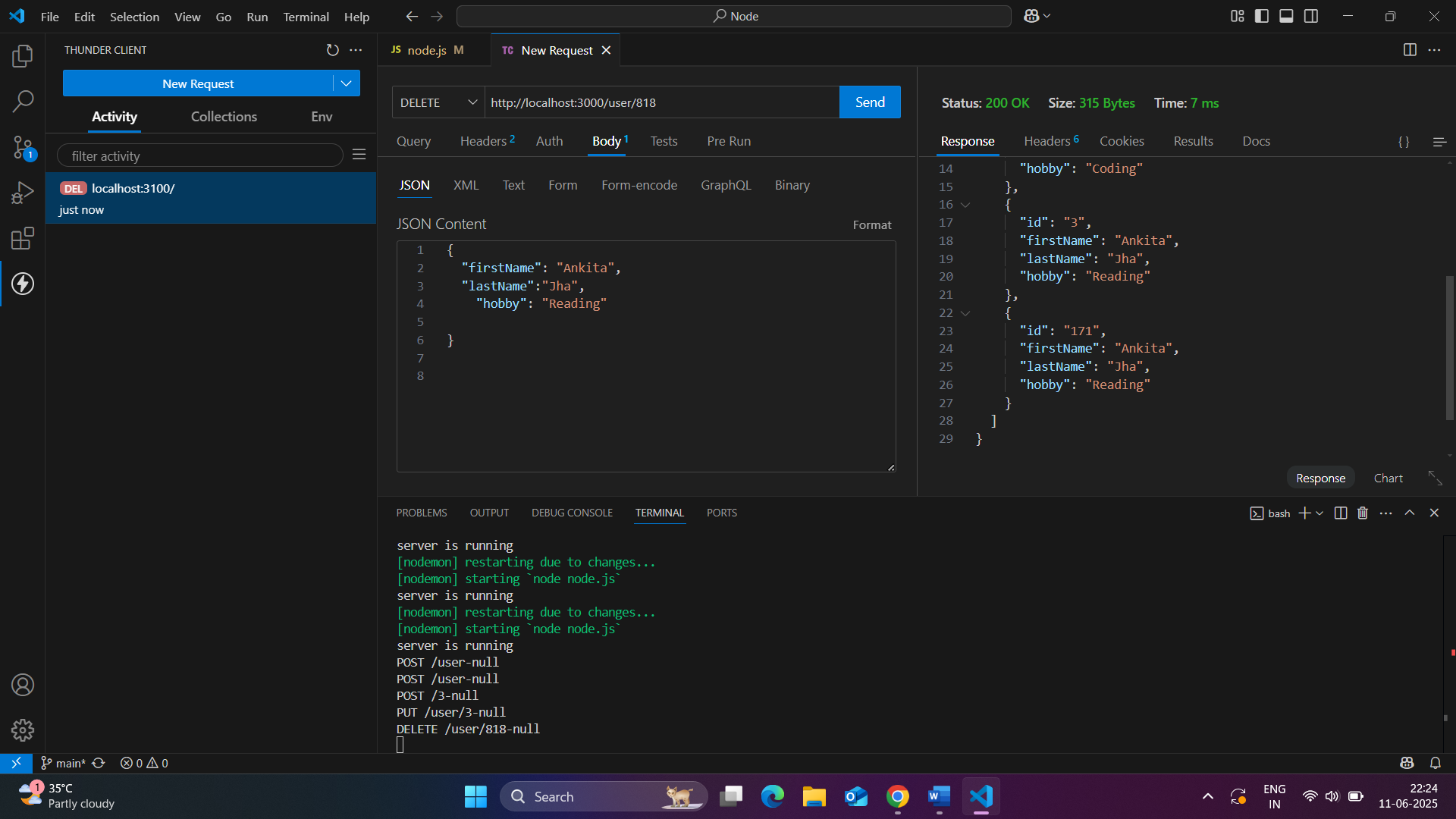
## Screenshot 3: POST /user – Successfully adding a user

## 

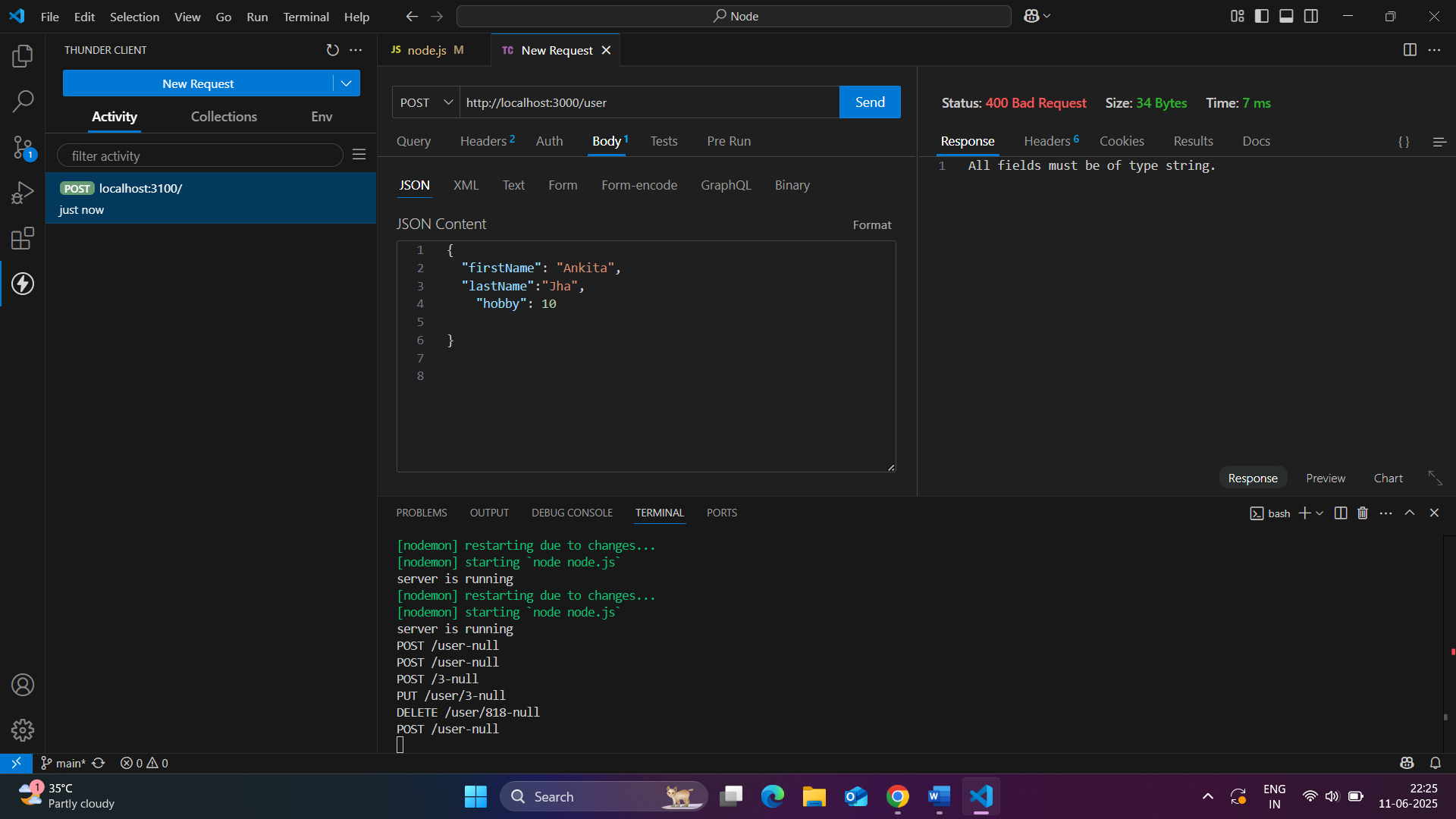
## Screenshot 4: PUT /user/:id – Successfully updating a user



## Screenshot 5: DELETE /user/:id – Successfully deleting a user

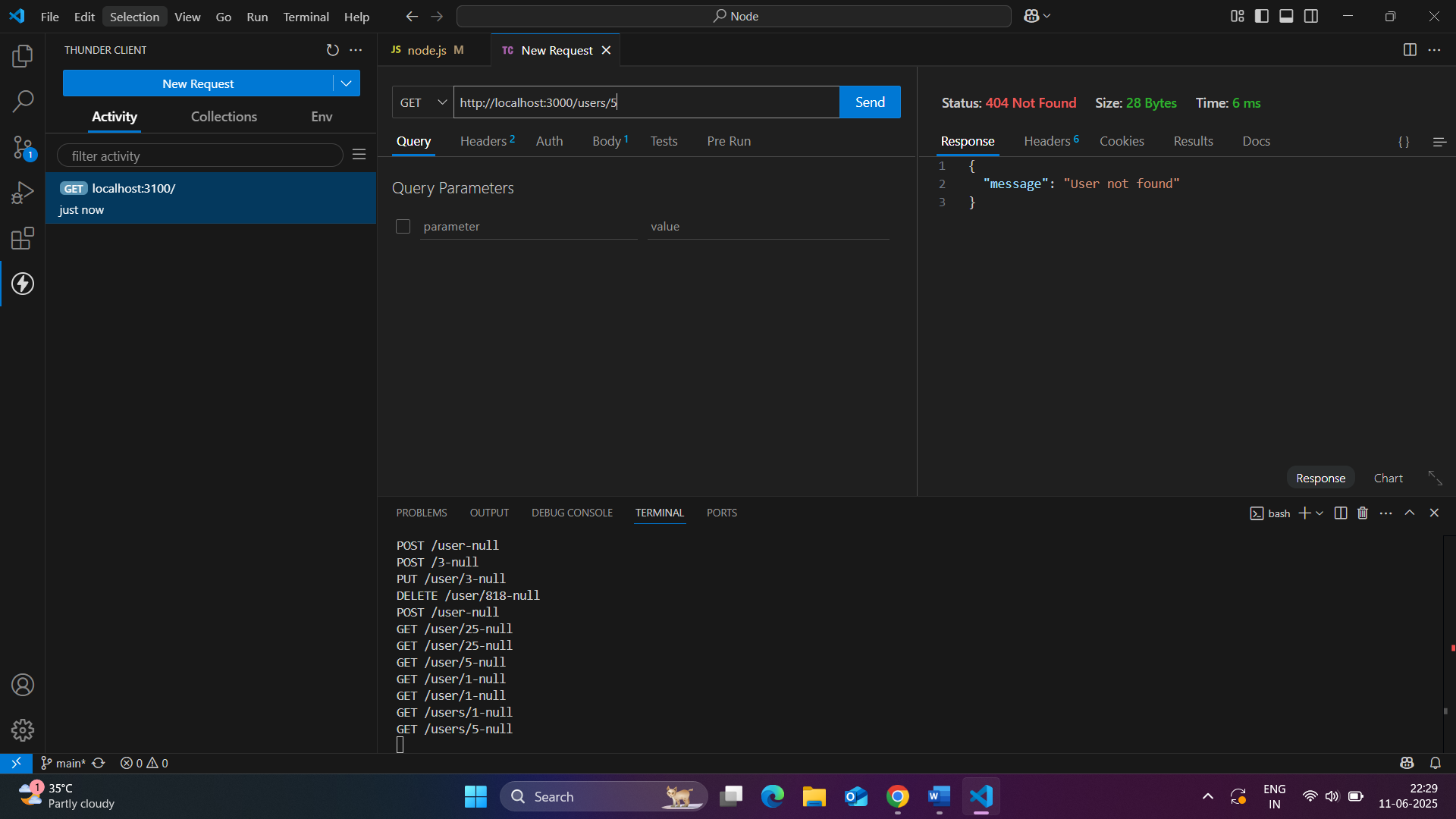


## Screenshot 6: Invalid POST – Missing/incorrect fields (should return 400)

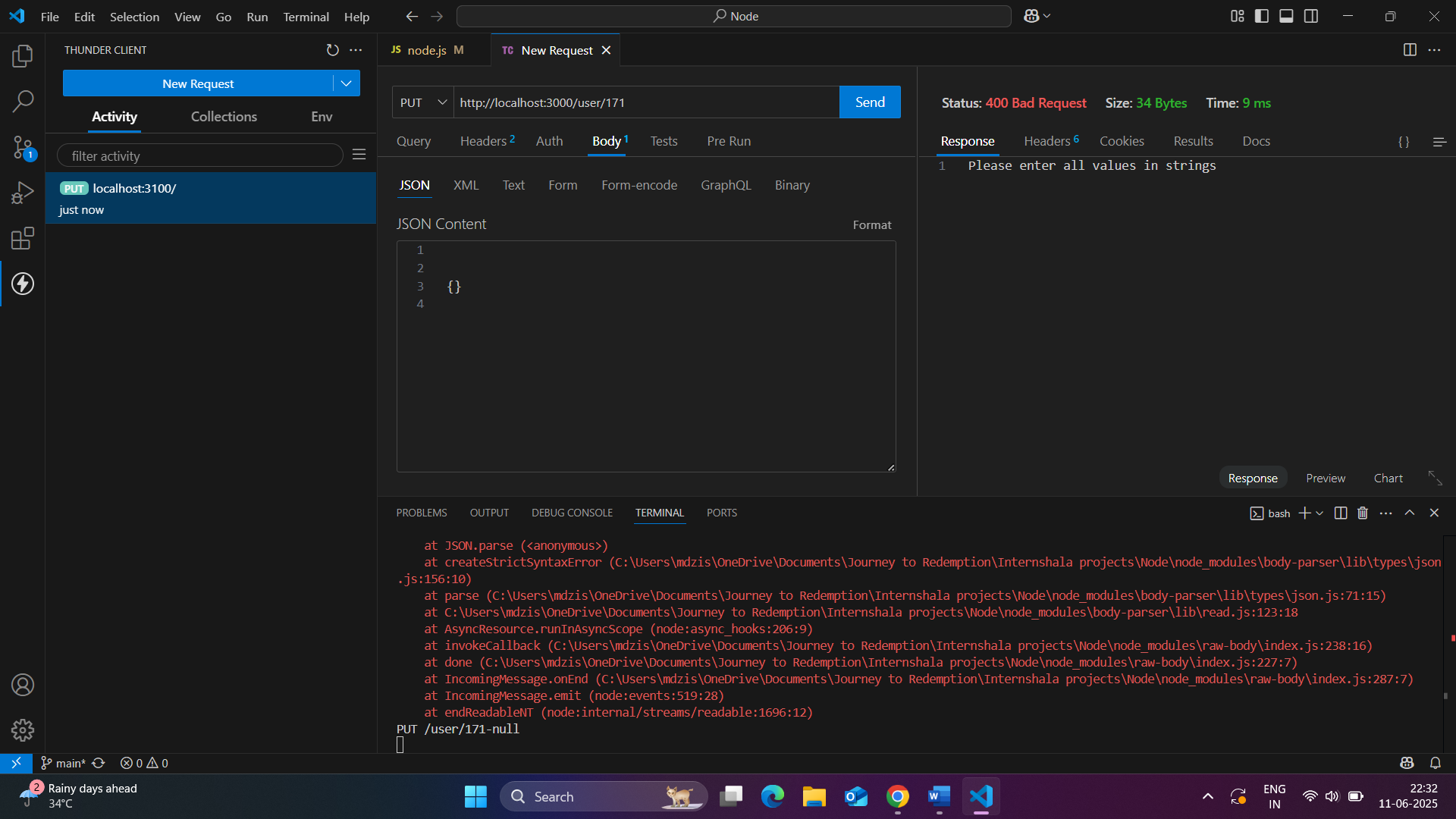


## Screenshot 7: GET/PUT/DELETE non-existent ID – Should return 404

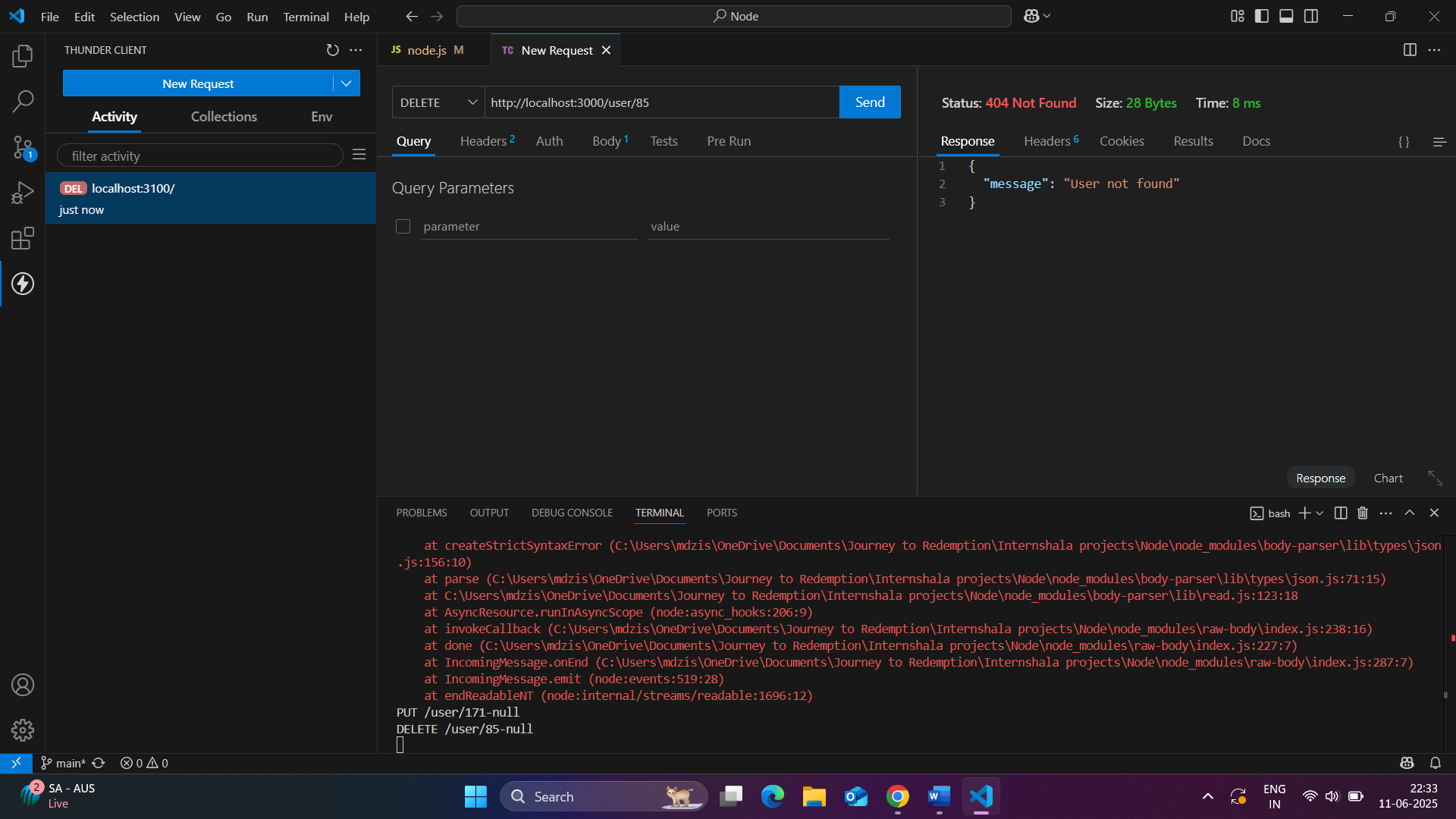
## Get



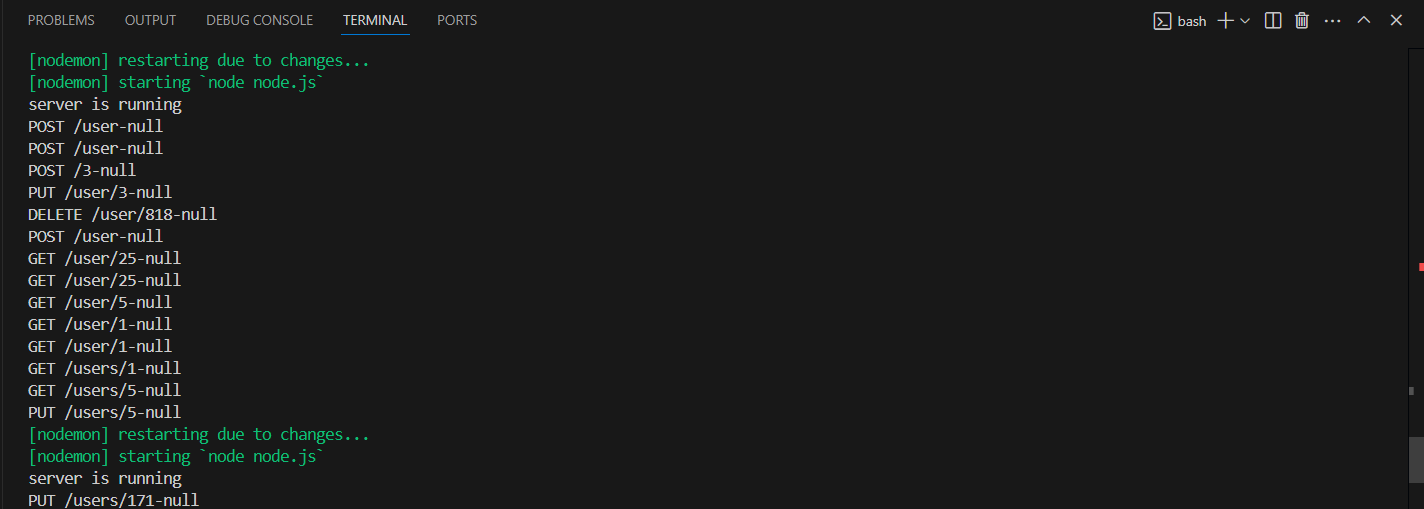
**B) Put**

****

**C) Delete**

****

## Screenshot 8: Terminal Output – Show logs printed from middleware



# Conclusion

This project helped me understand how to set up a RESTful API using Express, implement middleware, validate data, and handle errors properly. The API is fully tested using ThunderClient, and all routes function as expected. It simulates CRUD operations using an in-memory array, ideal for learning REST concepts before connecting to a real database.