## **EXPERIMENT 17: Testing Web Application REST APIs Using Postman**

### **Aim**

To test and validate the RESTful API endpoints of the **Interview Simulator Website** using **Postman**.  
 The objective is to ensure that all backend routes (CRUD operations) built with **Node.js, Express, and MongoDB** are functioning correctly, returning proper status codes, and handling requests and responses efficiently.

### **Theory**

**1. Introduction to Postman** **Postman** is an API development and testing tool used to send HTTP requests, inspect responses, automate API tests, and debug APIs.  
 It allows developers to test REST APIs directly without needing a frontend, which helps verify server-side logic before integration.

**2. Why Use Postman?**

* **Testing CRUD operations:** Easily send GET, POST, PUT, DELETE requests.
* **Visualize Responses:** View JSON responses, headers, and status codes.
* **Environment Management:** Store variables like base URLs and tokens.
* **Automation:** Create test collections for regression testing.
* **Debugging:** Identify backend issues such as incorrect routes or payloads.

**3. API Request Components** Every Postman request includes:

* **URL:** The endpoint (e.g., http://localhost:5000/api/questions)
* **Method:** HTTP verb (GET, POST, PUT, DELETE)
* **Headers:** Metadata (e.g., Content-Type: application/json)
* **Body:** Data sent to the server for POST/PUT requests
* **Response:** The server’s reply (JSON data + status code)

### **Procedure**

#### **Step 1 – Start the Backend Server**

Run your Express + MongoDB application:

node server.js

Confirm that MongoDB is connected and the server is running:

Server running on port 5000

MongoDB connected

#### **Step 2 – Open Postman and Create a New Collection**

1. Open **Postman Desktop App** or go to https://postman.com.
2. Click **New → Collection** and name it Interview Simulator API.

Add your base URL as an environment variable, e.g.:  
  
 {{base\_url}} = http://localhost:5000

#### **Step 3 – Test API Endpoints**

##### **(a) GET – Retrieve All Questions**

* **Method:** GET
* **URL:** {{base\_url}}/api/questions
* **Expected Response:** List of all questions from MongoDB.  
   Example Output:

[

{

"\_id": "671c2f1e9a1a73c2b3f2f2d7",

"title": "Explain closures in JavaScript",

"difficulty": "Medium",

"category": "JavaScript"

}

]

* **Status Code:** 200 OK

##### **(b) POST – Add a New Question**

* **Method:** POST
* **URL:** {{base\_url}}/api/questions

##### **(c) PUT – Update a Question**

* **Method:** PUT
* **URL:** {{base\_url}}/api/questions/<question\_id>

**Expected Response:** {

"\_id": "671c3d6b8a1e5b64d4a72b90",

"title": "What is React useEffect hook?",

"difficulty": "Hard",

"category": "React"

}

* **Status Code:** 200 OK

##### **(d) DELETE – Remove a Question**

* **Method:** DELETE
* **URL:** {{base\_url}}/api/questions/<question\_id>

**Expected Response:** { "message": "Question deleted" }

* **Status Code:** 200 OK

#### **Step 4 – Verify Error Handling**

Test invalid requests (e.g., missing required fields or wrong ID) to confirm proper error responses:

{ "message": "Question validation failed" }

**Status Code:** 400 Bad Request or 404 Not Found

#### **Step 5 – Automate Tests (Optional)**

In Postman’s **Tests** tab, add simple assertions:

pm.test("Status code is 200", () => {

pm.response.to.have.status(200);

});

pm.test("Response has title field", () => {

const jsonData = pm.response.json();

pm.expect(jsonData[0]).to.have.property("title");

});

Save and run your test collection with **Postman Runner**.

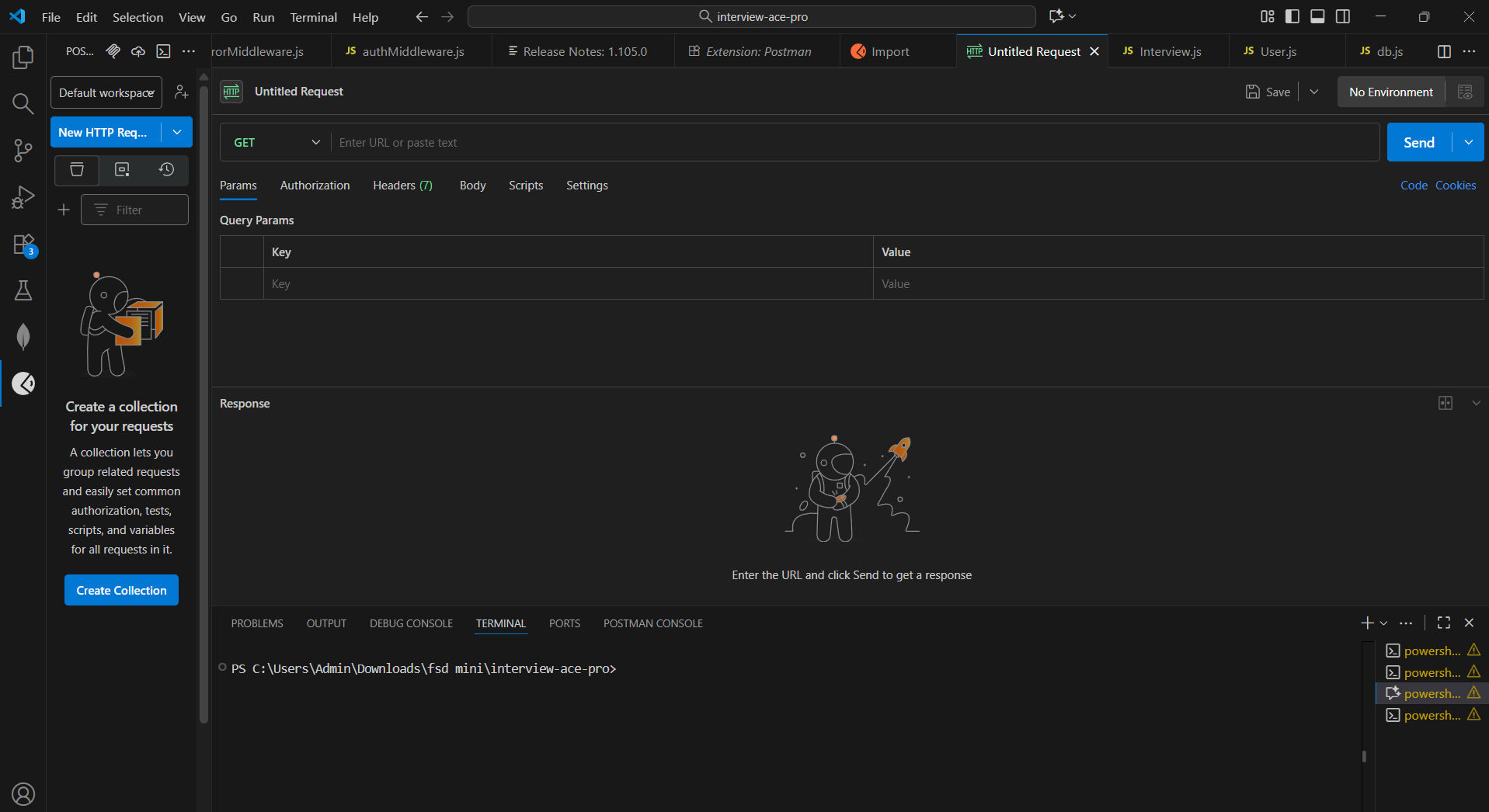
### **Step 6 – Integrate Postman with CI/CD (Optional)**

You can export the collection and use it in GitHub Actions or Newman (Postman’s CLI) for automated API testing:

npm install -g newman

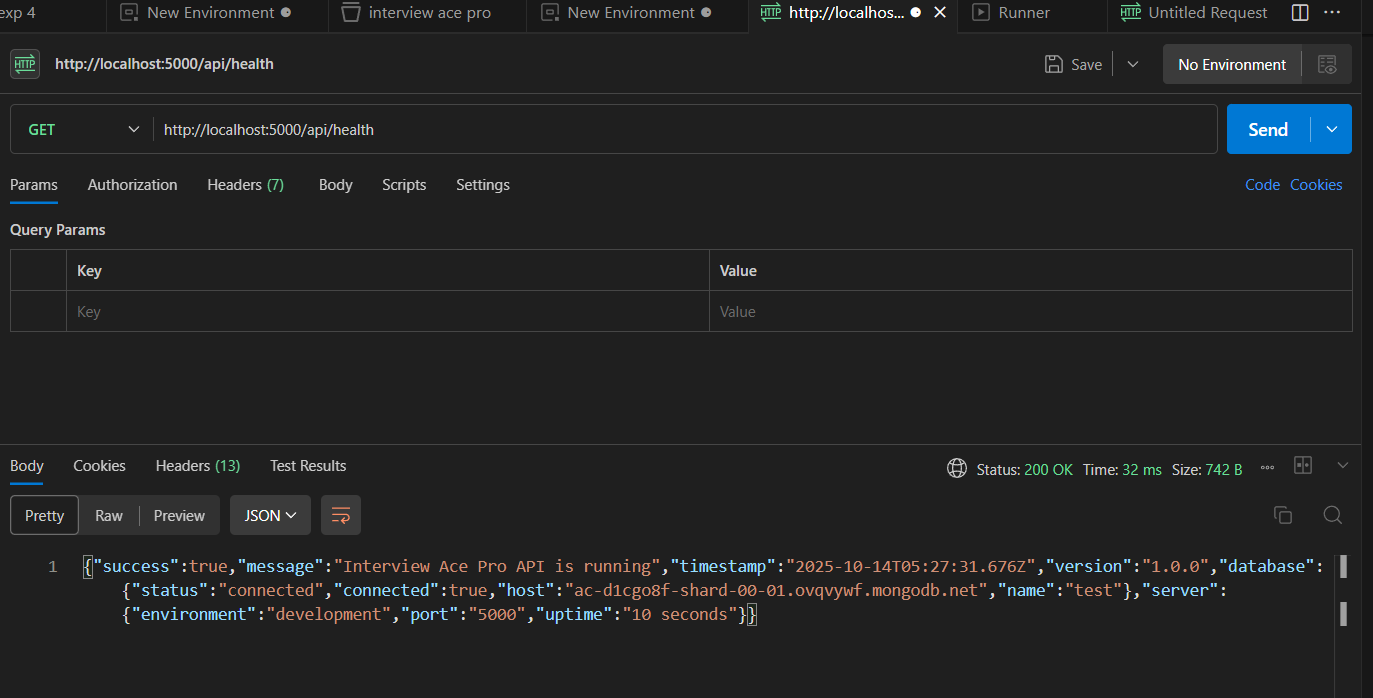
newman run "Interview Simulator API.postman\_collection.json"

Dashboard of postman

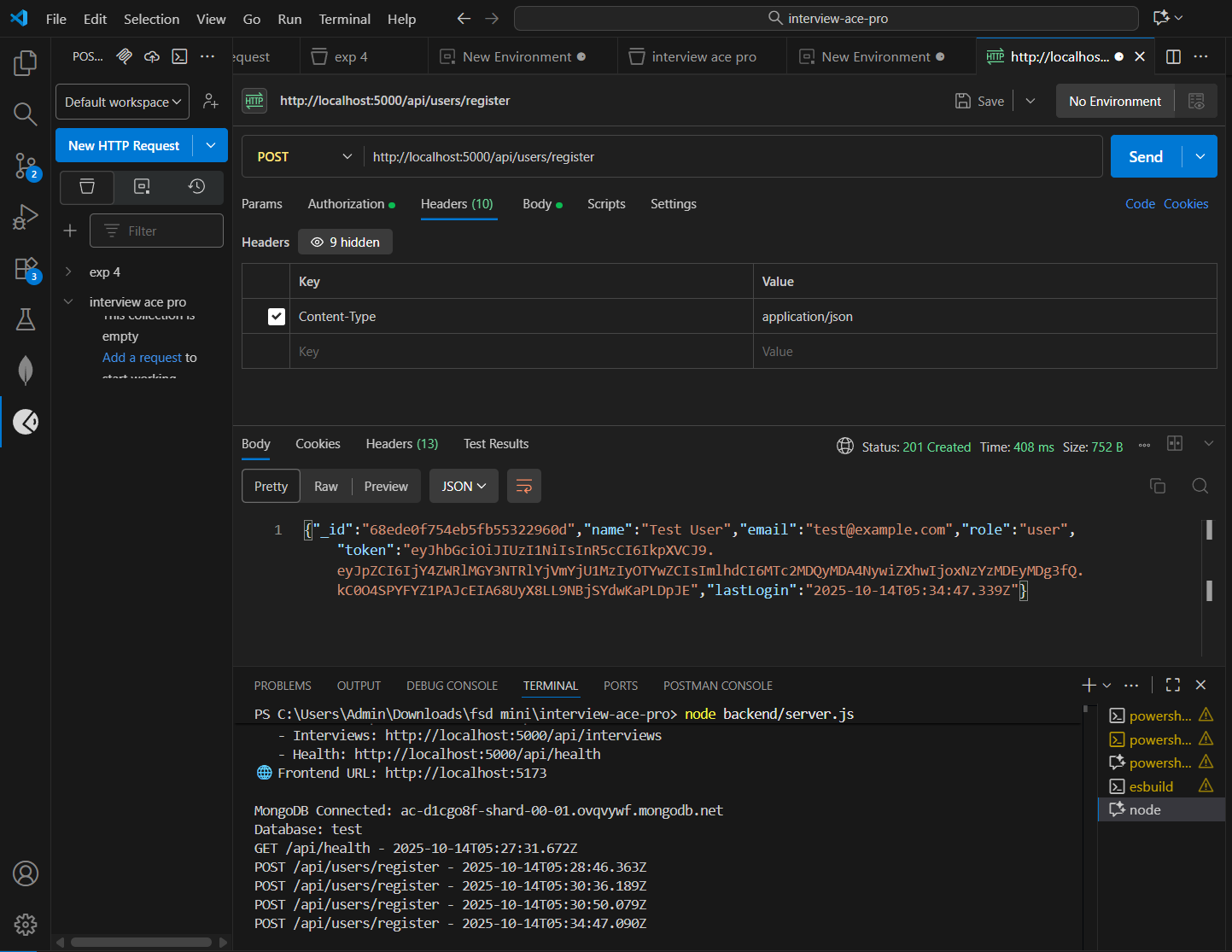


Get

(Testing whether Api is working properly)

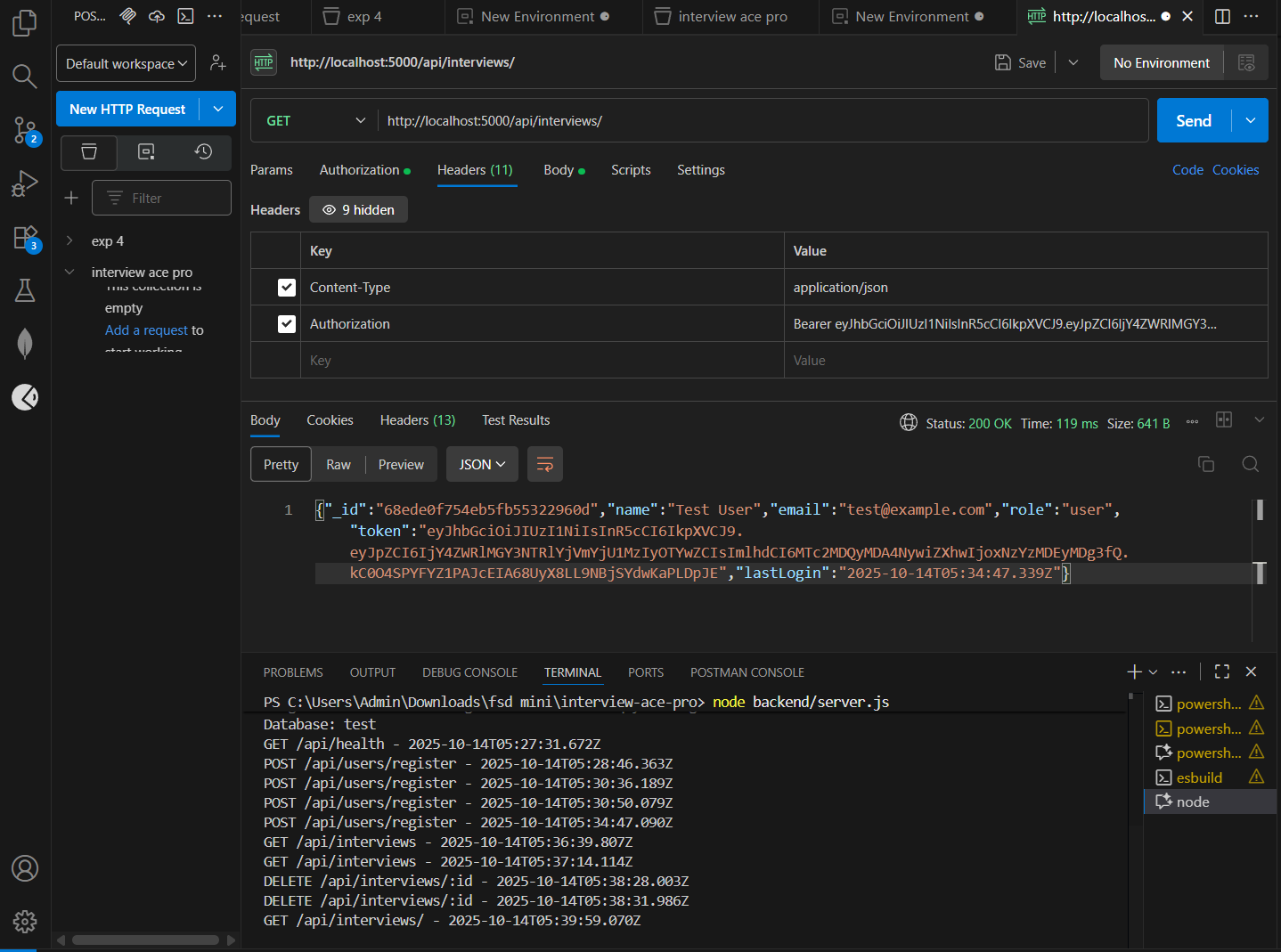


Post ( a dummy user created )

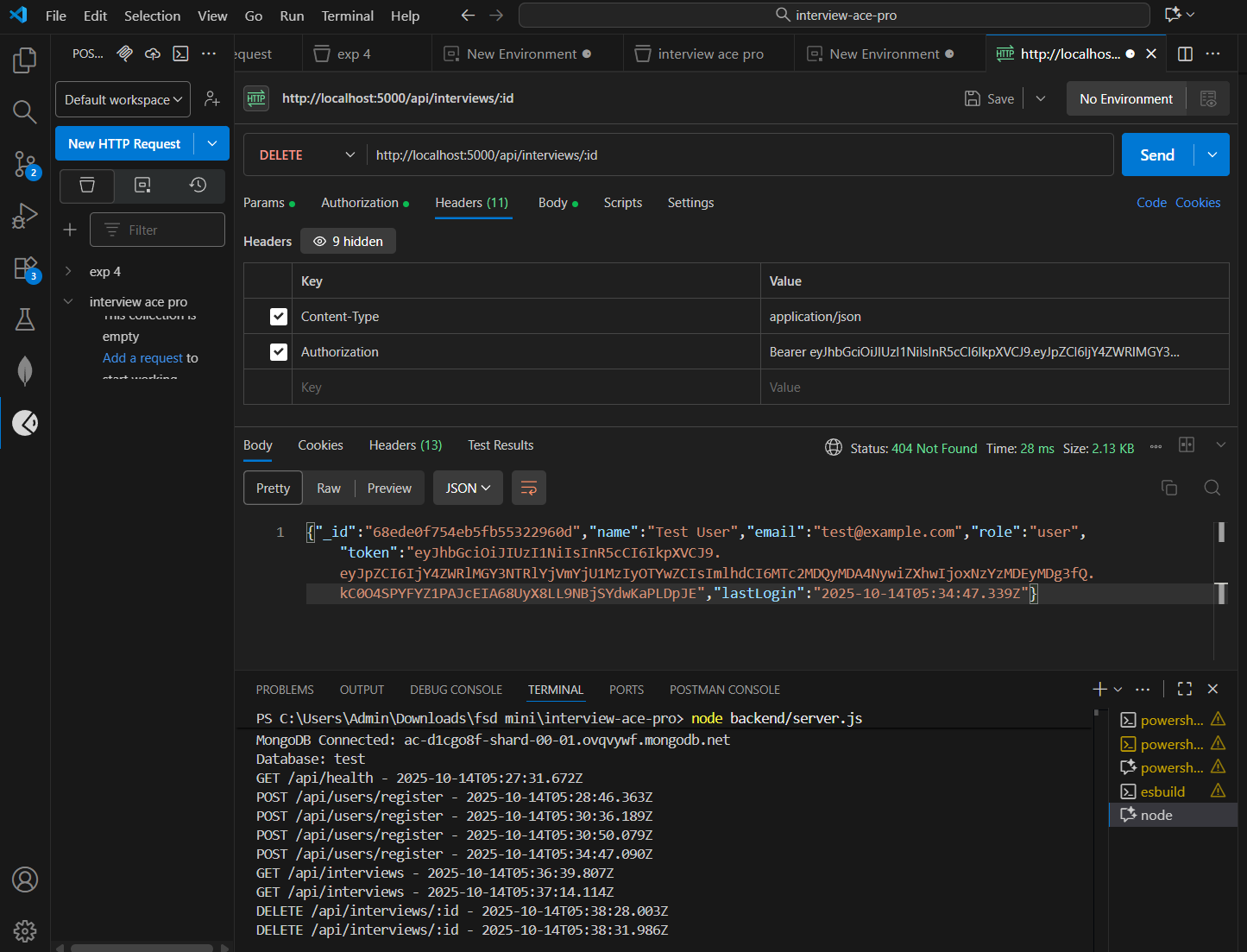


After post of user , we get its user id , JWT token and role which can used for fetching

we get all the user from ‘GET’ command



Delete



### **Conclusion**

This experiment successfully validated the RESTful API endpoints of the Interview Simulator Website using **Postman**.  
 Each CRUD operation (Create, Read, Update, Delete) was tested, and responses were verified for accuracy and proper status codes.

By performing API testing with Postman:

* The backend was confirmed to function as expected before frontend integration.
* Error handling and response structures were validated.
* API reliability and data integrity were ensured.

Postman thus served as a crucial step in the development workflow, helping verify that the backend was production-ready and seamlessly integrable with the frontend React interface.