**Backend Developer Assignment: Simple Note-Taking API**

Steps to run on local machine :

1. Clone the repo using the link https://github.com/rrhan32/note-taking-app.git on git bash .

2. cd Note-Taking-app

3. npm i

4. node app.js

I would denote the working of the Api through **POSTMAN ,** as it is easier to highlight the working of each route and subsequent request .

The app would be running on localhost:3000

**TESTING :**

1. POST /create\_account

Purpose: Create a new user account.

Endpoint: /create\_account

Method: POST

Functionality:

Hashes the user's password using bcrypt.

Creates a new user with hashed password and saves it to the database.

Generates a JWT token for the user.

Sets the JWT token as a cookie in the response.

Response:

Status 201: User account created successfully.

Status 500: Internal server error if any issue occurs.

2. POST /login

Purpose: Authenticate user login.

Endpoint: /login

Method: POST

Functionality:

Finds the user by email in the database.

Compares the provided password with the hashed password stored in the database.

If the password matches, generates a new JWT token for the user.

Sets the JWT token as a cookie in the response.

Response:

Status 200: Login successful.

Status 404: User not found.

Status 500: Internal server error if any issue occurs.

3. GET /logout

Purpose: Logout user.

Endpoint: /logout

Method: GET

Functionality:

Clears the JWT cookie by setting an empty cookie with a short expiration time.

Response:

Status 200: Logout successful.

4. POST /

Purpose: Create a new note.

Endpoint: /

Method: POST

Functionality:

Creates a new note and saves it to the database.

Response:

Status 200: Note created successfully.

Status 400: Bad request if any issue occurs.

Purpose: Retrieve all notes.

Endpoint: /

Method: GET

Functionality:

Retrieves all notes from the database.

Response:

Status 200: Returns an array of notes.

Status 500: Internal server error if any issue occurs.

6. GET /:id

Purpose: Retrieve a single note by ID.

Endpoint: /:id

Method: GET

Functionality:

Retrieves a note by its ID from the database.

Response:

Status 200: Returns the requested note.

Status 404: Note not found.

Status 500: Internal server error if any issue occurs.

7. PUT /:id

Purpose: Update a note by ID.

Endpoint: /:id

Method: PUT

Functionality:

Updates a note by its ID in the database.

Response:

Status 200: Returns the updated note.

Status 404: Note not found.

Status 500: Internal server error if any issue occurs.

8. DELETE /:id

Purpose: Delete a note by ID.

Endpoint: /:id

Method: DELETE

Functionality:

Deletes a note by its ID from the database.

Response:

Status 200: Note deleted successfully.

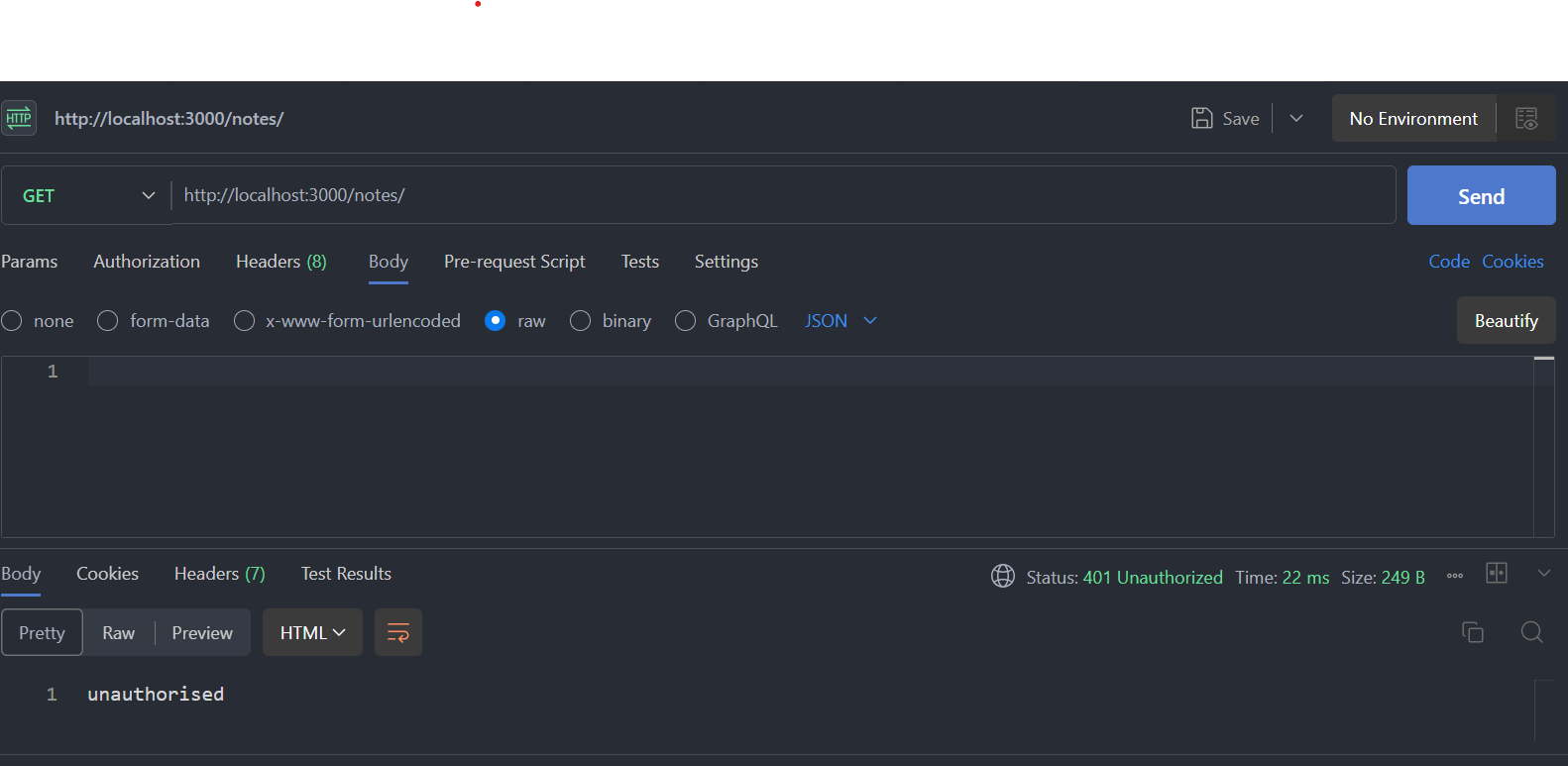
Status 404: Note not found.

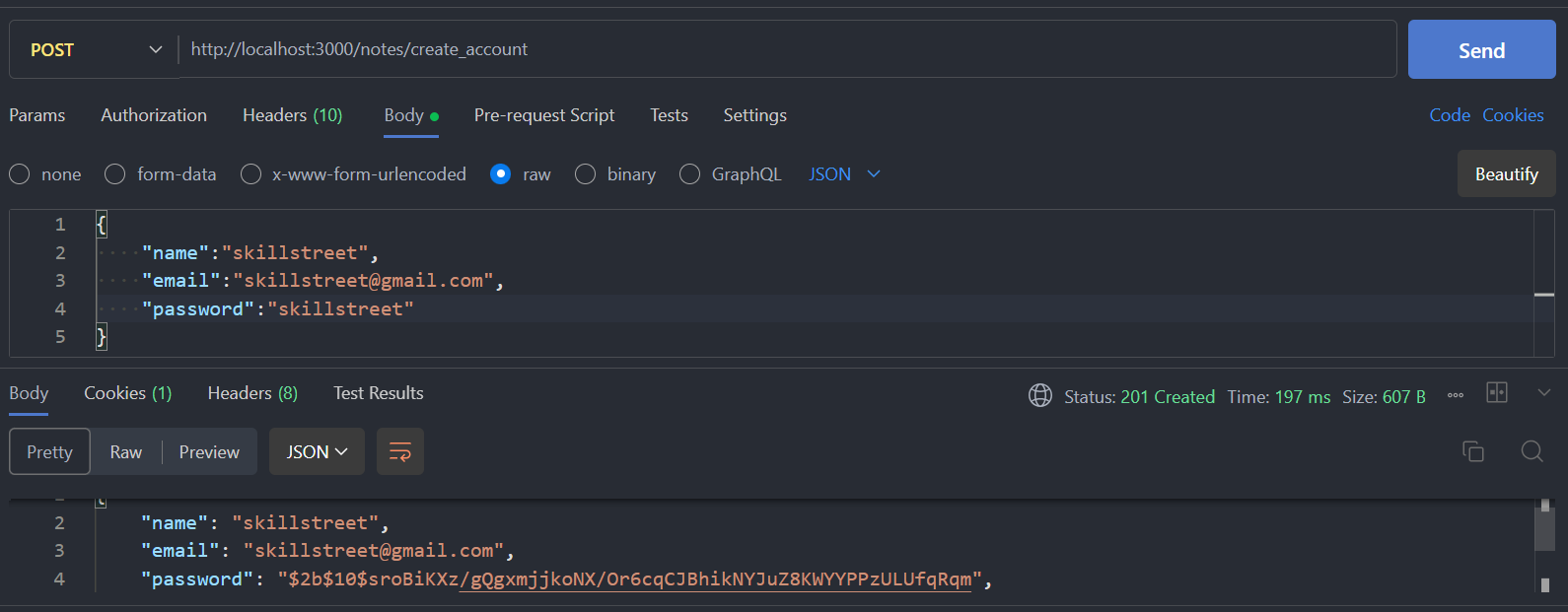
Status 500: Internal server error if any issue occurs.

AUTHENTICATION : All requests (except login and create\_account) are protected by JWT authentication using the verifytoken middleware, ensuring that only authenticated users can access these endpoints. If you try to access these routes before login or create\_account , tou would get "unauthorised" as the response .

For the purpose of Validating , I have attached the screenshots below :

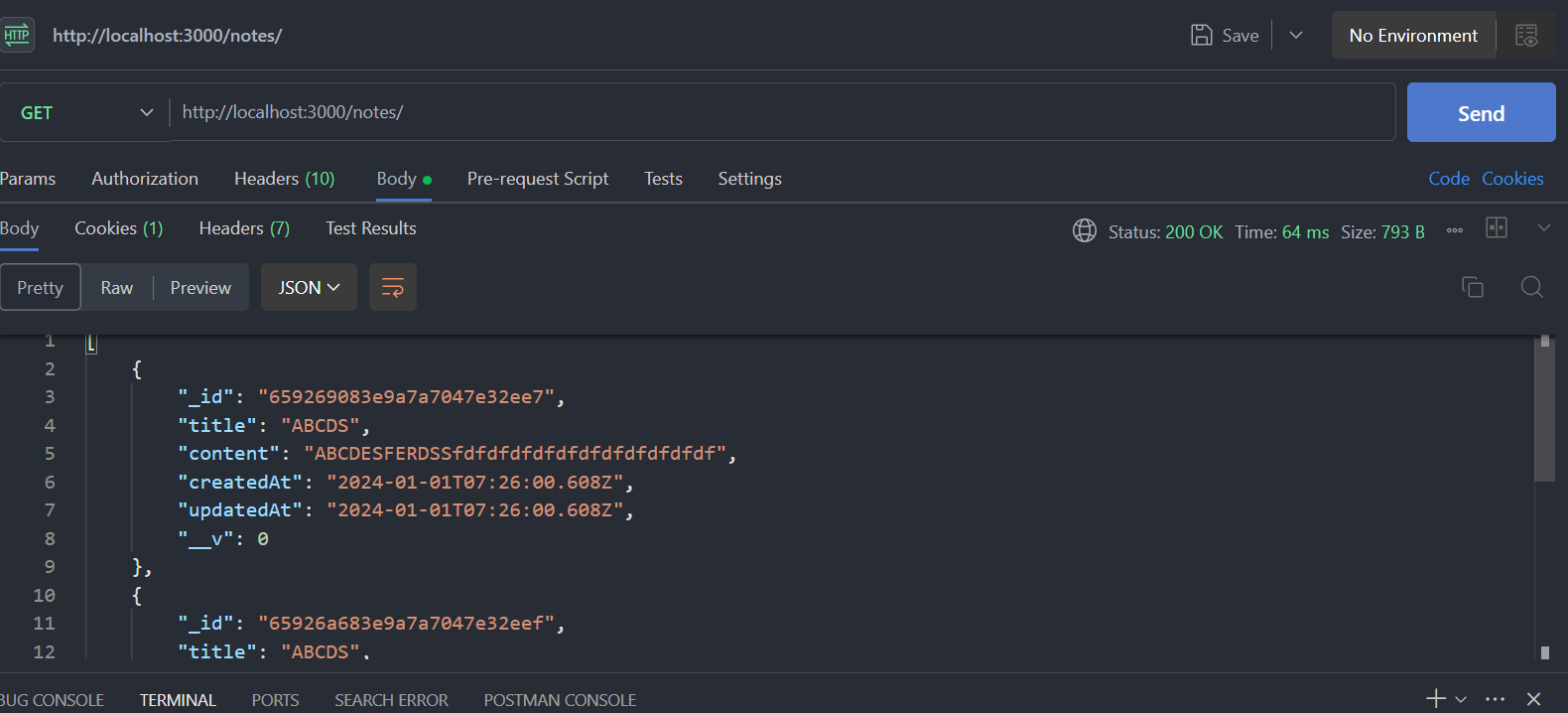
1.)

  
We tried to access the ‘/’ route before login , we got unauthorised as the response .

2.) 

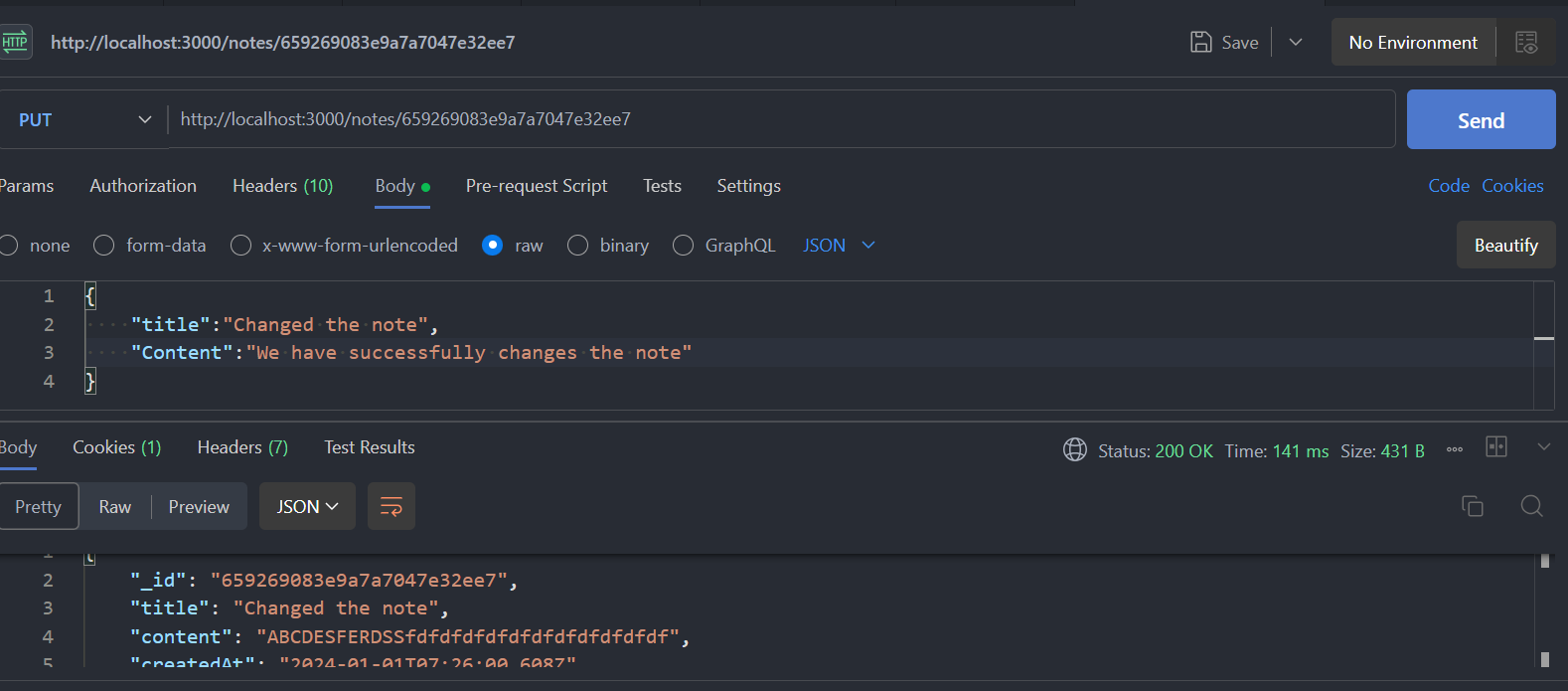
We have created the account , below I have printed the entry which is stored in our MONGODB atlas , and you can see that the password is **well-hashed** , which provides a two step security .

You can see **the cookies section** highlighting (1) , which is the jwt token which I have stored .

3.) 

Now , I have again tried to access the ‘/’ **get** route , and see we can see the different notes I have added here .

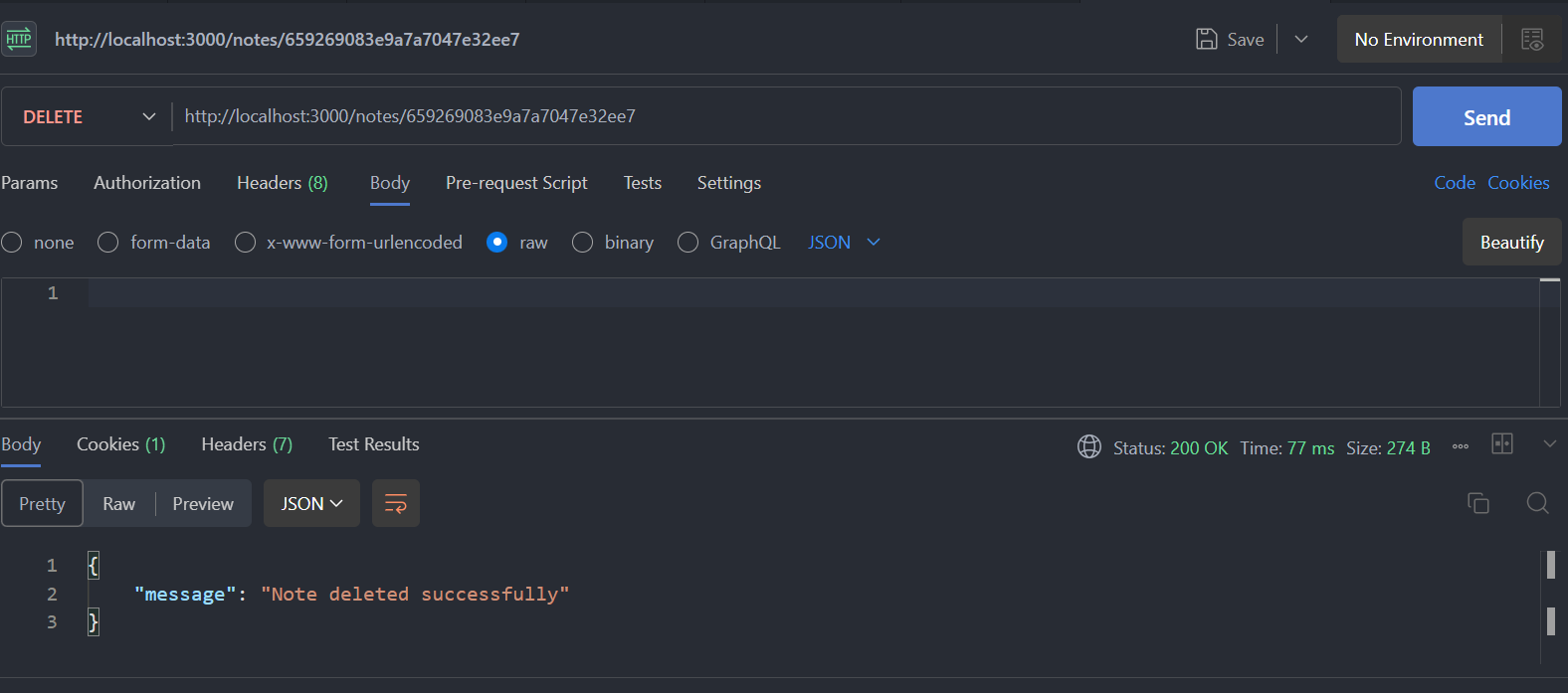
4.) Now , lets update the note which you can see on the above photo , using the put request .



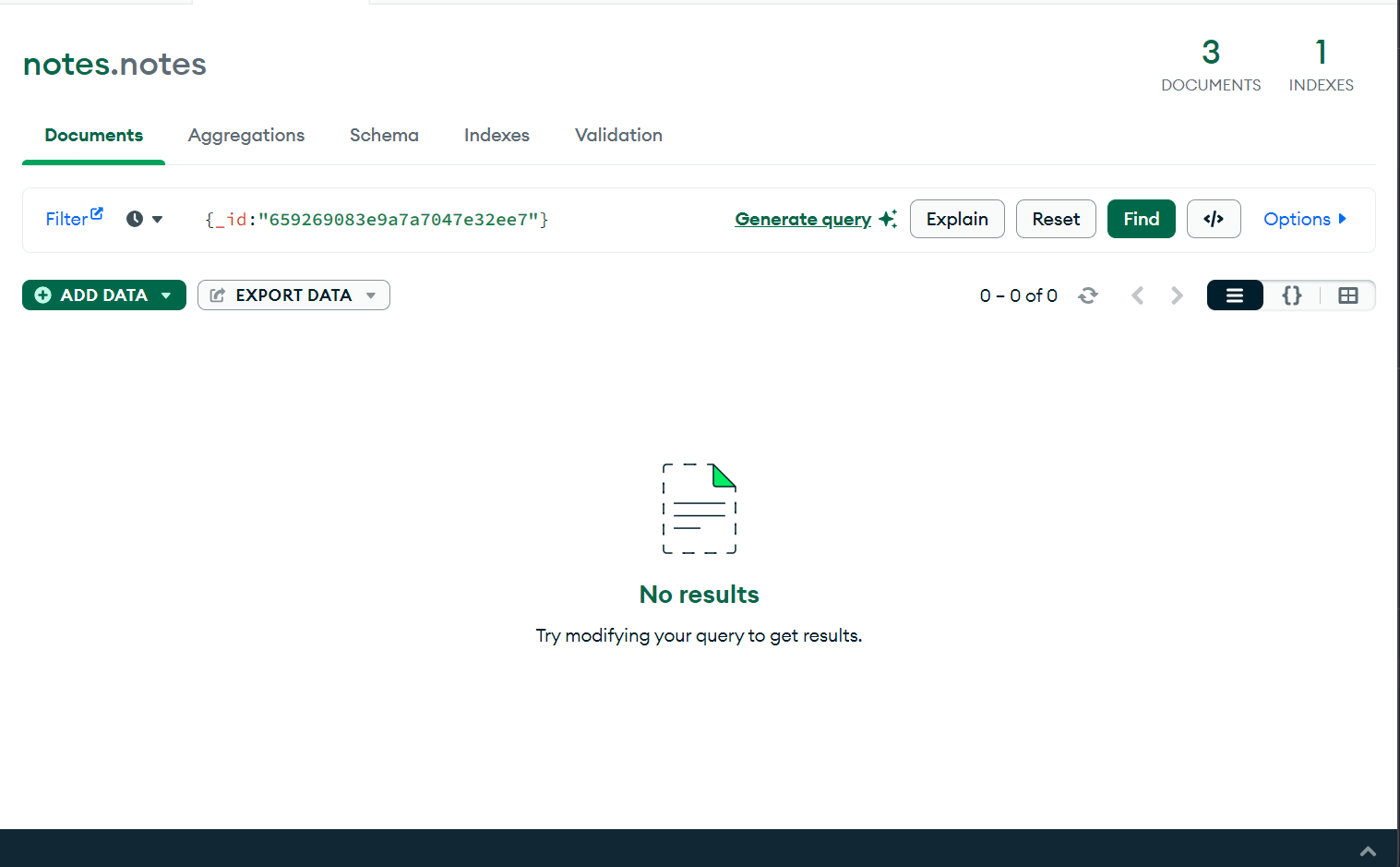
In the body , have added the content , and below you can see that the content is changes , which can be verified by seeing the \_id , which is same as :id in req.params .

5.) Testing Delete Route :

Now we try to delete the note with same :id



The note is deleted successfully , which I am verifying by looking in the MongoDB atlas database



So , the entry was deleted successfully .

So , out routes are working correctly as we want .

7.) When I use the logout route , the jwt is automatically set to empty , due to which , the user is unauthorised to access the routes , then protecting them .

In case of any confusion , would be glad to help and justify the step

Thanks