Cloud Computing Coursework

**GitHub Repo Link:** https://github.com/ViplaviWade/Piazza.git

# 1.0 Introduction:

Piazza is a dynamic social media platform designed to provide users with a personalized space for sharing thoughts, ideas, and engaging with a community. Leveraging modern web technologies, Piazza offers features such as user registration, authentication, and a robust set of APIs for creating, updating, and interacting with posts.

**2.0 Architecture Overview:** The architecture of Piazza follows a server-client model:

Server: Implemented using Node.js and Express, the server handles HTTP requests and interacts with the MongoDB database.

Database: MongoDB is employed as the primary database for its flexibility and scalability. It stores user information in the 'users' collection and post-related data in the 'posts' collection.

Authentication: OAuth v2 is integrated to secure user authentication, ensuring a seamless and secure login process.

# 3.0 Technology Stack:

Server-Side: Node.js with Express for building a fast and efficient server.

Database**:** MongoDB for its NoSQL capabilities, facilitating flexibility in data storage.

Authentication**:** OAuth v2 ensures secure user authentication with support for third-party login providers.

# Database Schema

* 1. **Users Collection**

The 'users' collection contains:

* + 1. **username**: User's unique identifier.
    2. **email**: User's email for communication and identification.
    3. **password**: Encrypted password for secure authentication.
    4. **date**: Registration date for tracking user activity.

# Posts Collection

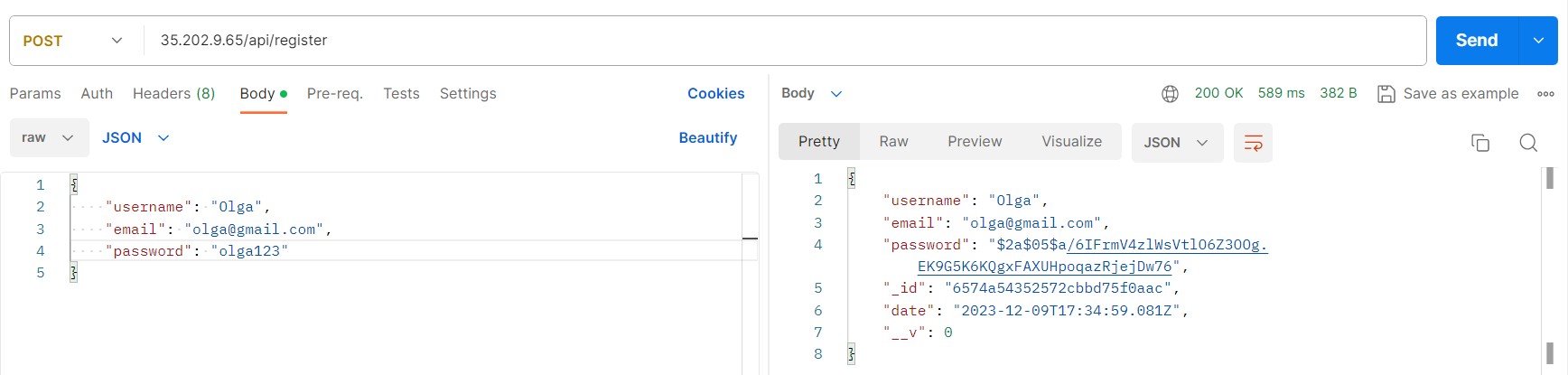
The 'posts' collection includes:

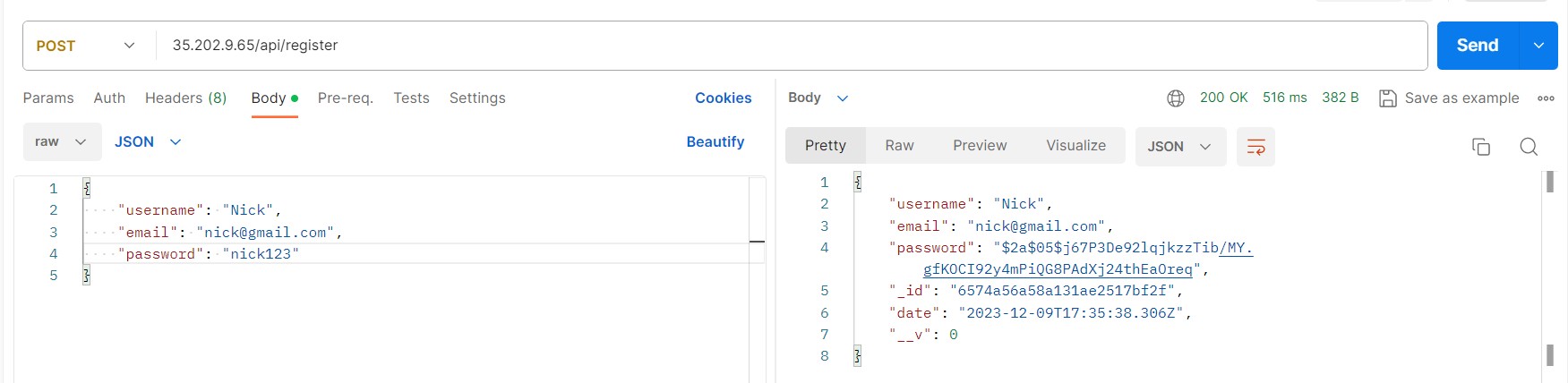
* + 1. **post\_title**: Title of the post.
    2. **post\_topic**: Topic or category to which the post belongs.
    3. **message**: Content of the post.
    4. **expiration\_time**: Time after which the post is marked as 'EXPIRED'.
    5. **status**: Current status of the post ('LIVE' or 'EXPIRED').
    6. **like\_users**, **dislike\_users**: Arrays storing users who liked or disliked the post.
    7. **likes\_count**, **dislikes\_count**: Counts of likes and dislikes.
    8. **comments**: Array storing comments on the post.
    9. **comments\_count**: Count of comments on the post.
    10. **post\_id**: Unique identifier for the post.
    11. **timestamp**: Timestamp indicating when the post was created.
    12. **postOwner**: Username of the post creator.

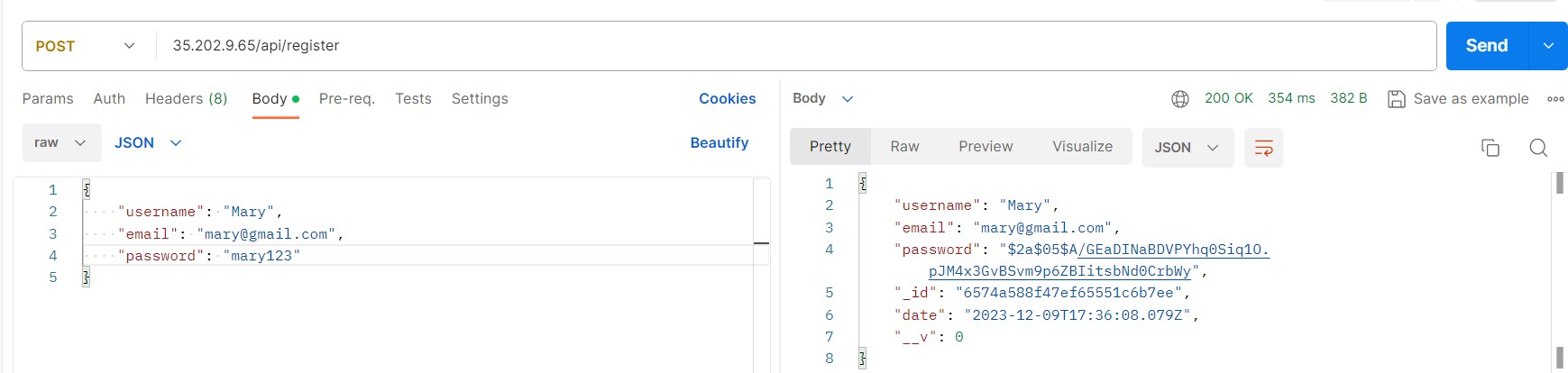
# API Endpoints using test cases: (used Kubernetes deployment Endpoint IP address)

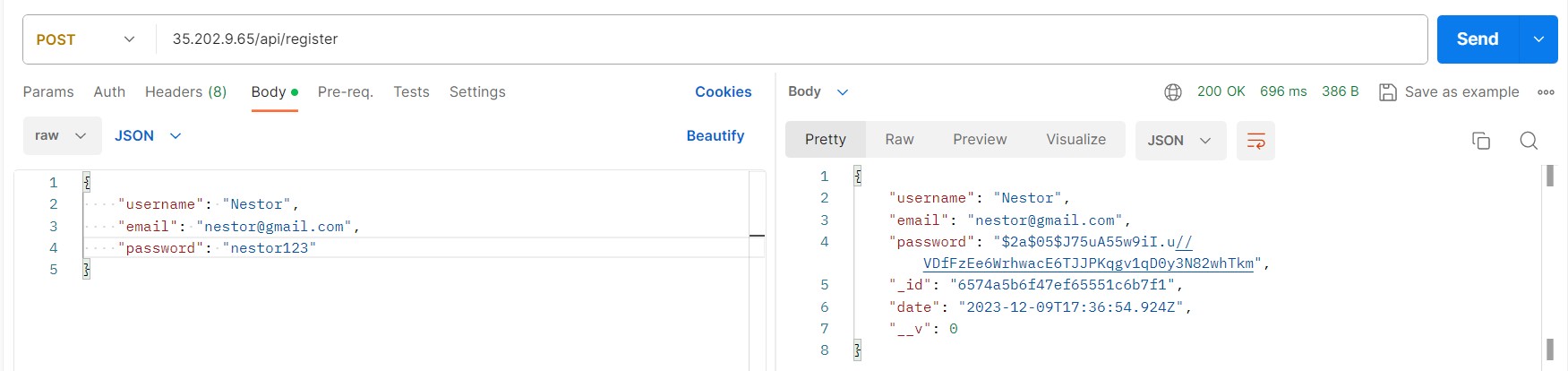
* 1. **Test Case-1:** Olga, Nick, Mary, and Nestor register and are ready to access the Piazza API.

Olga registers to Piazza API

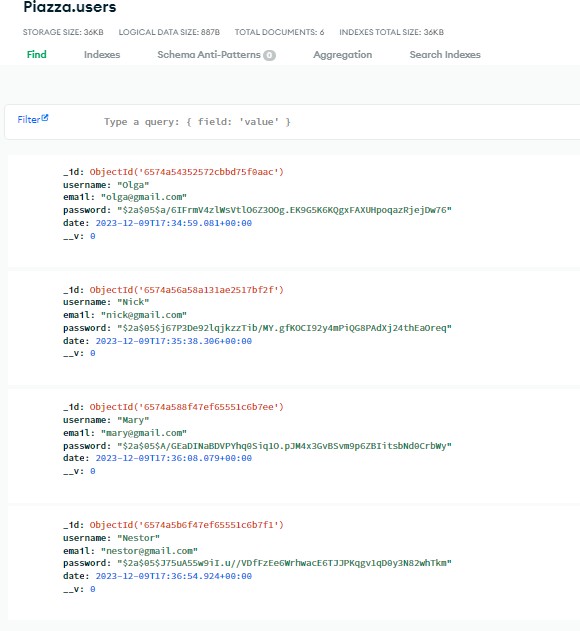


Nick registers to Piazza API

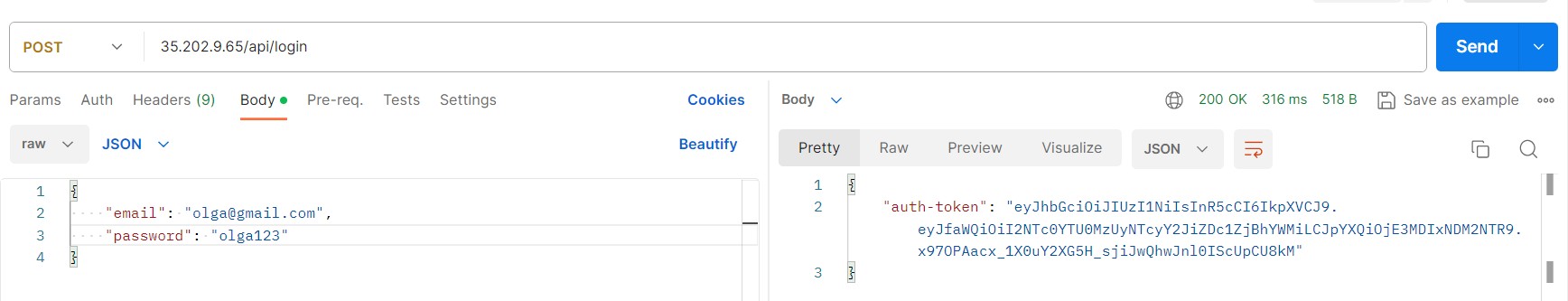
Mary registers to Piazza API

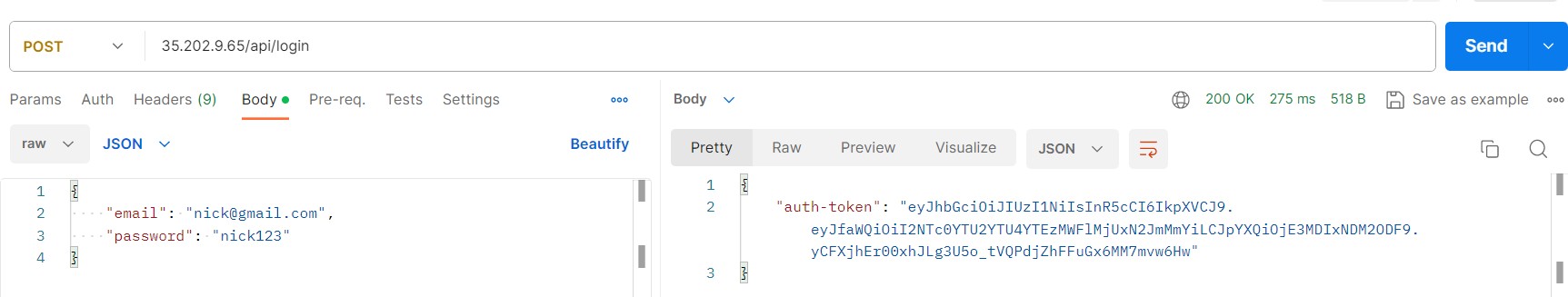
Nestor registers to Piazza API

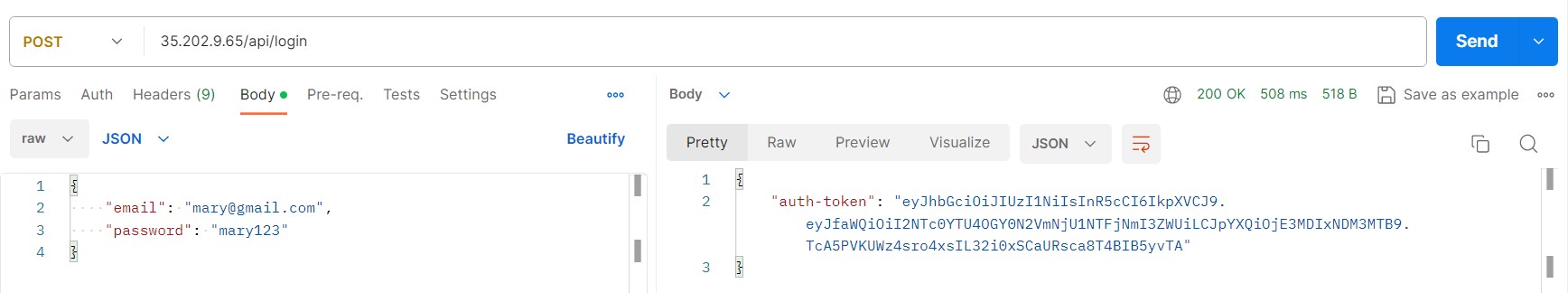
All the users: Olga, Nick, Mary and Nestor are registered in the **Piazza API.**



* 1. **Test Case 2:** Olga, Nick, Mary, and Nestor use the oAuth v2 authorisation service to register and get their tokens.

Olga receives a token for her login

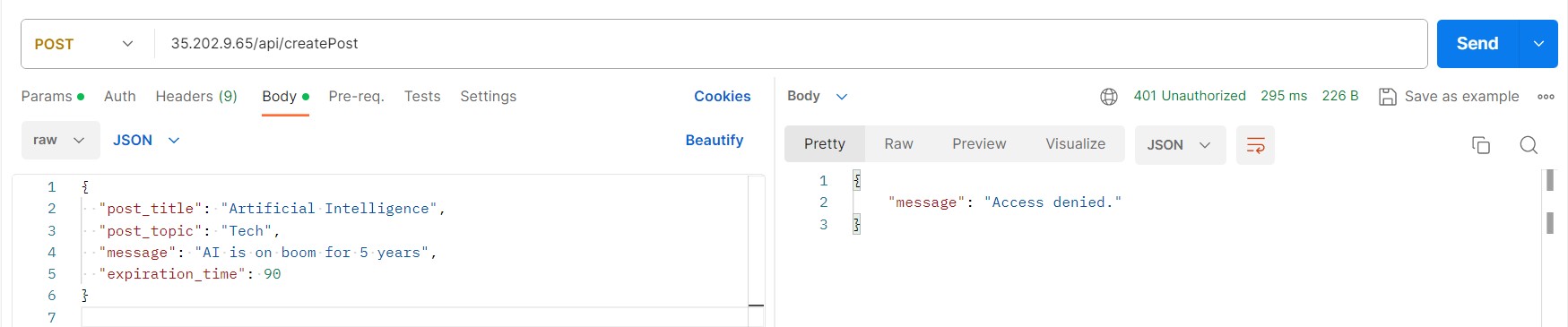
Nick receives a token for his login

Mary receives a token for her login

Nestor receives a token for her login

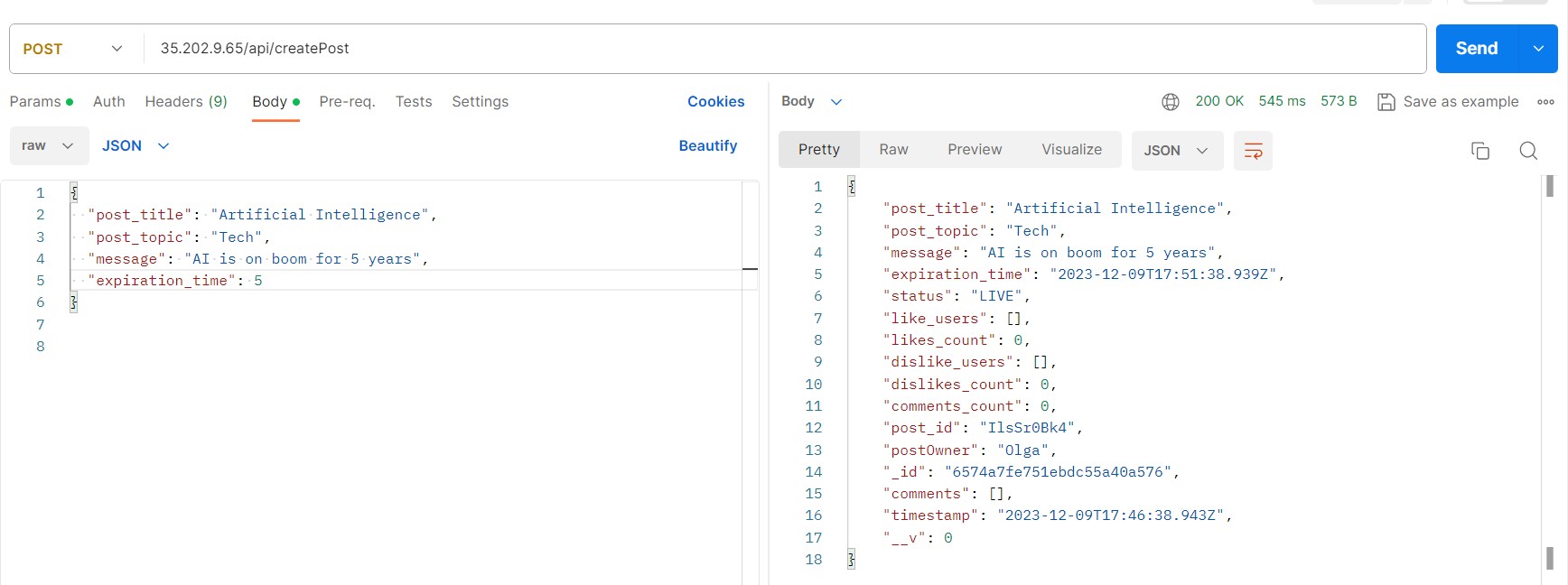
All the users used the **oAuth v2 authorization s**ervice to register and receive their tokens.

* 1. **Test Case 3:** Olga makes a call to the API without using her token. This call should be unsuccessful as the user is unauthorized.

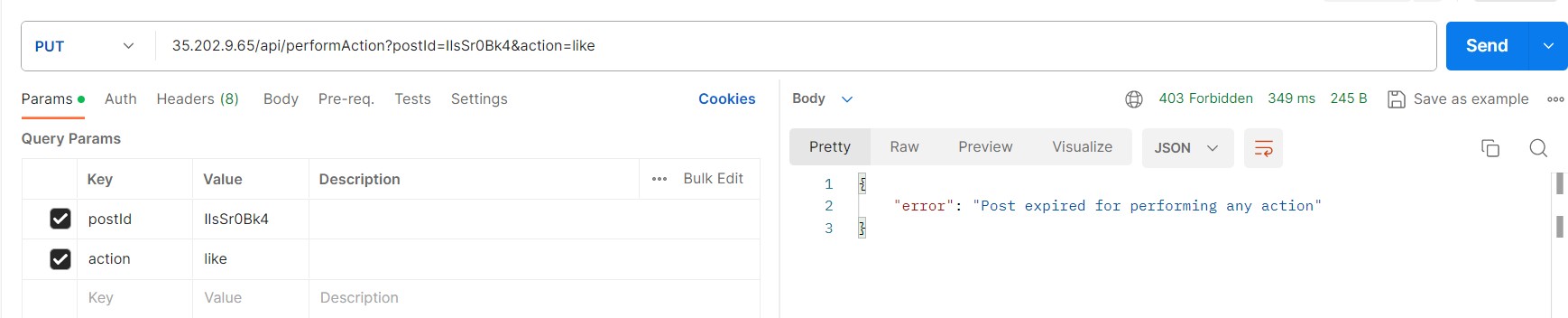


When Olga tried to make a call without her token, the API call failed as the user was not authorized. And the API returned the error message as “**Access Denied**”.

* 1. **Test Case 4:** Olga posts a message in the Tech topic with an expiration time (e.g. 5 minutes) using her token. After the end of the expiration time, the message will not accept any further user interactions (likes, dislikes, or comments).

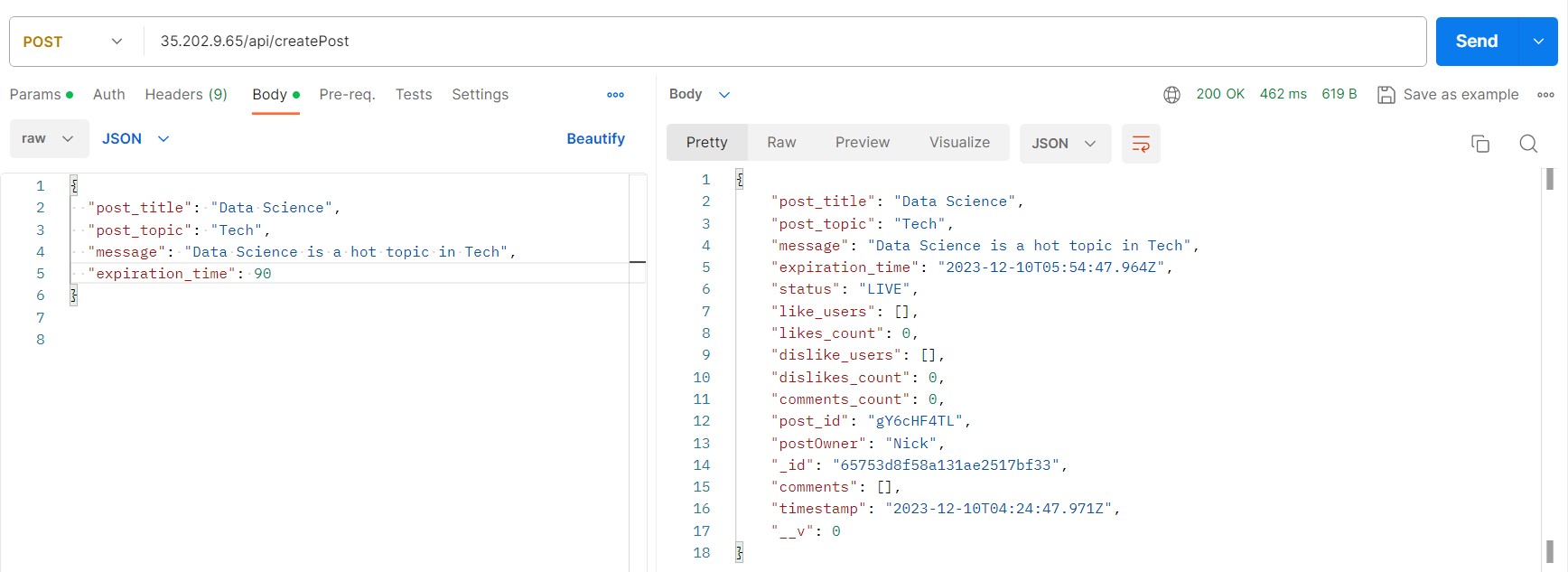


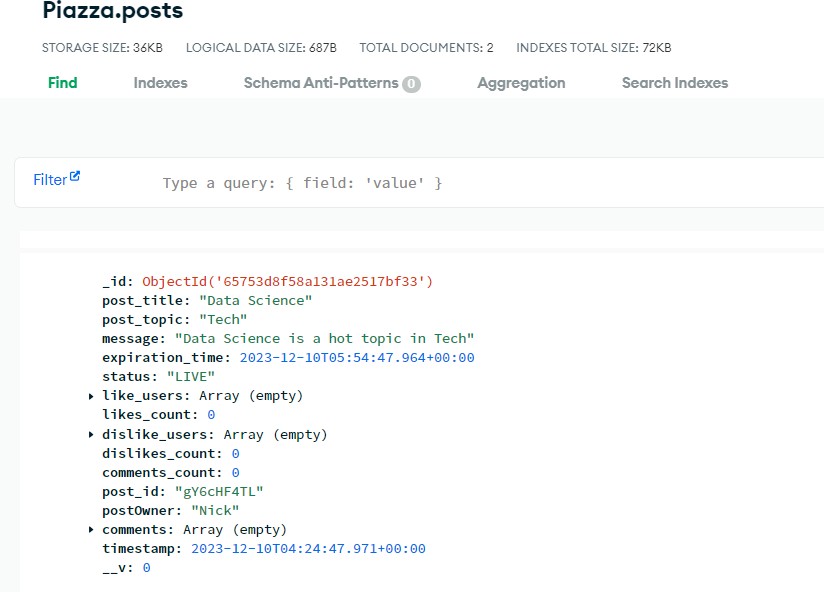
After 5 minutes (expiration time) the post will not accept any further interactions (like/dislike/comments)

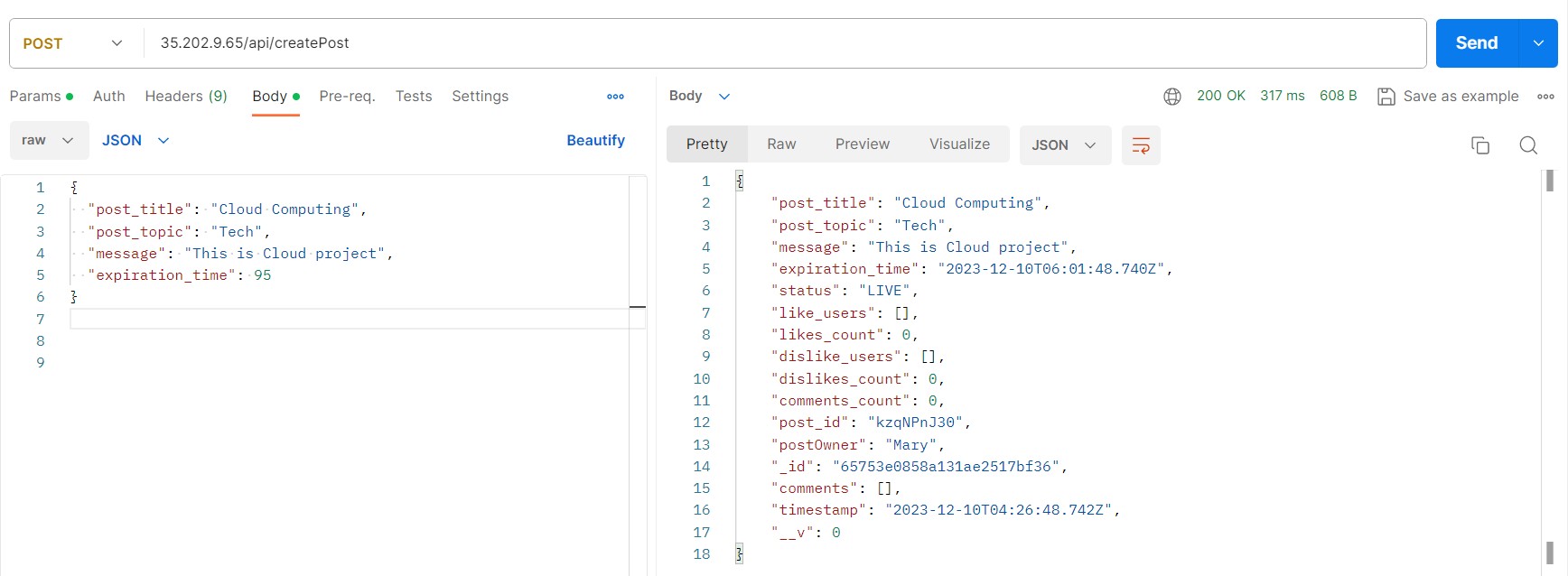


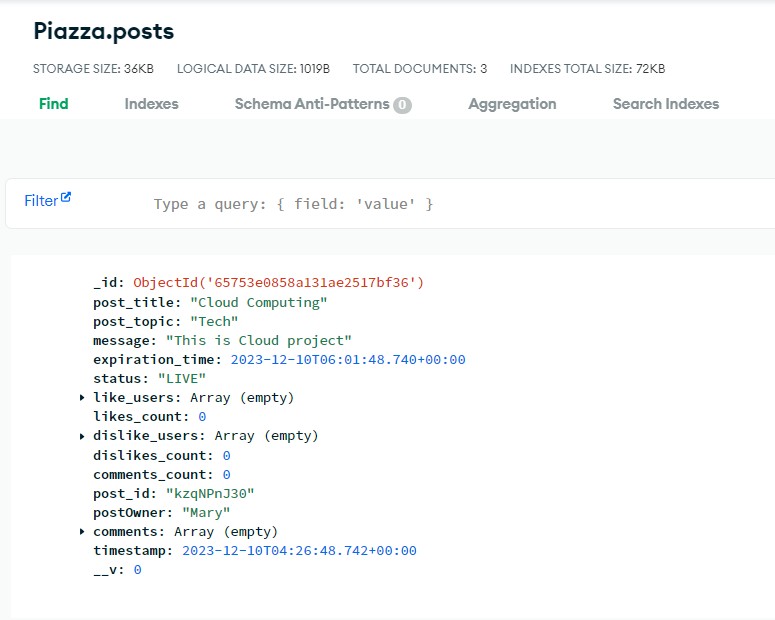
* 1. **Test Case 5:** Nick posts a message in the Tech topic with an expiration time using his token.

Nick posts a message in Tech topic with a n expiration time of 90 minutes



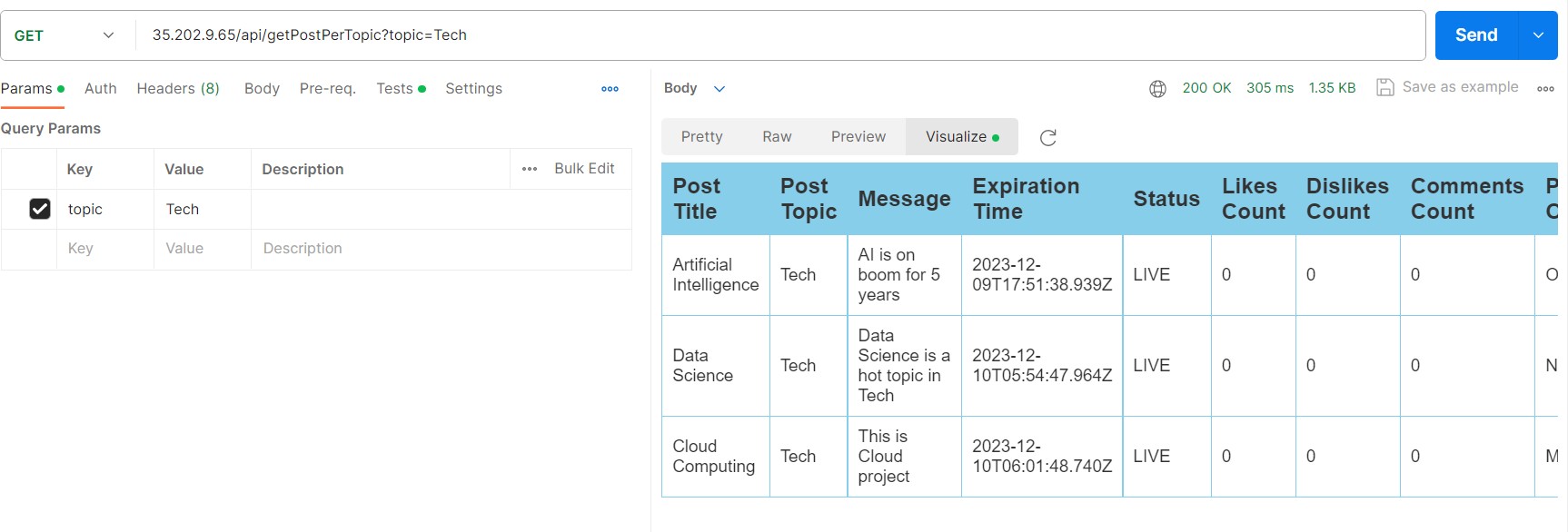


* 1. **Test Case 6:** Mary posts a message in the Tech topic with an expiration time using her token.

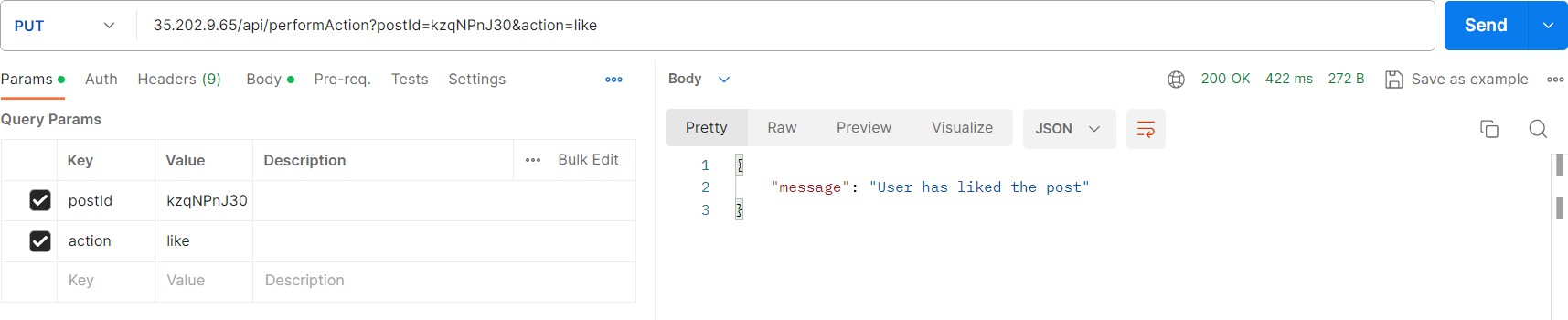


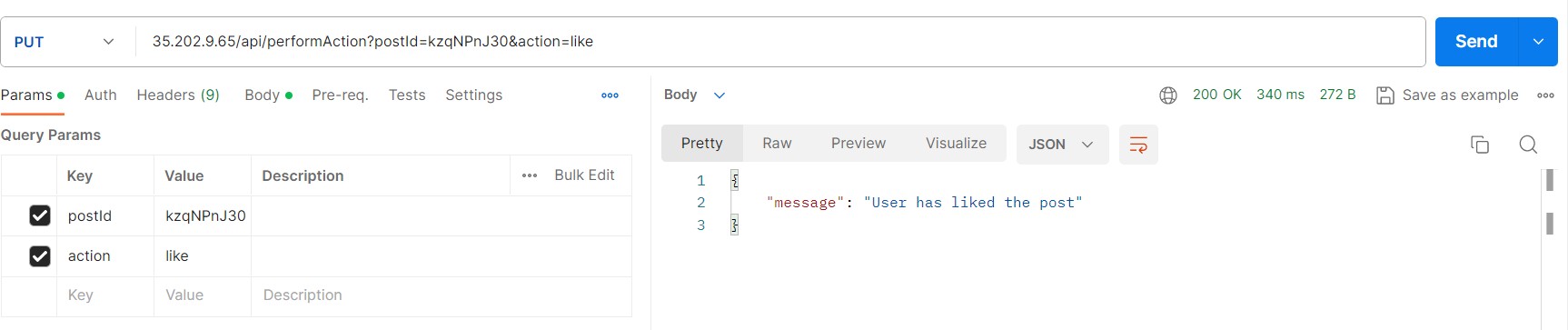
These two messages are added in the database as shown above

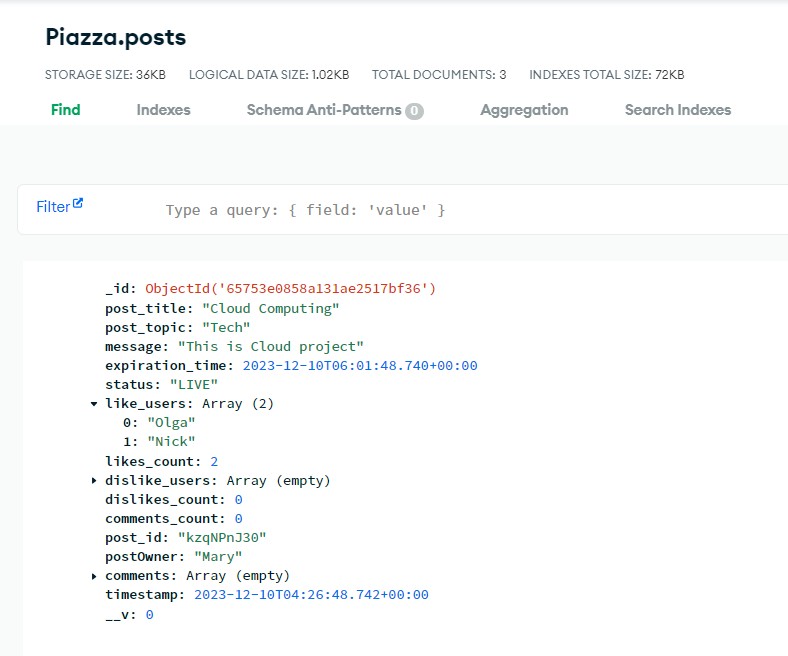
* 1. **Test Case 7:** Nick and Olga browse all the available posts in the Tech topic; three posts should be available with zero likes, zero dislikes and no comments.



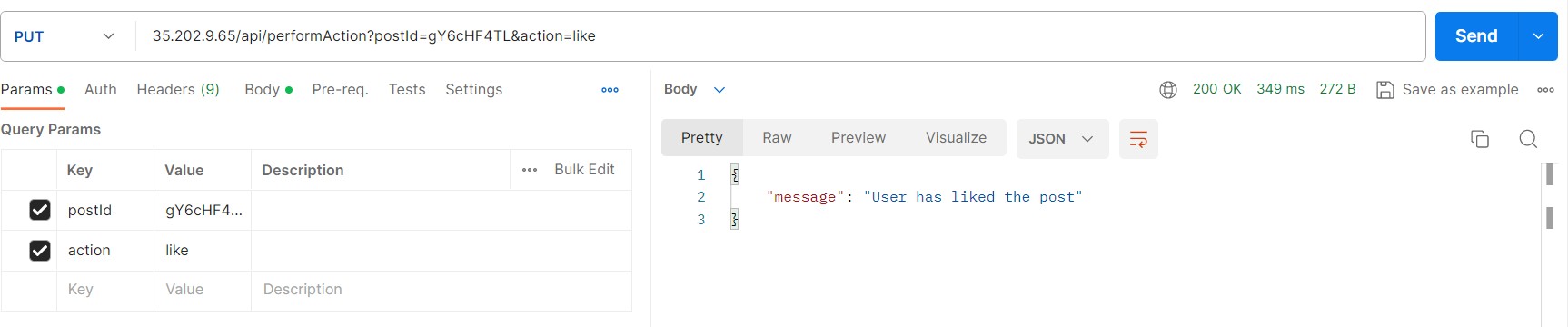
* 1. **Test Case 8:** Nick and Olga “like” Mary’s post on the Tech topic.

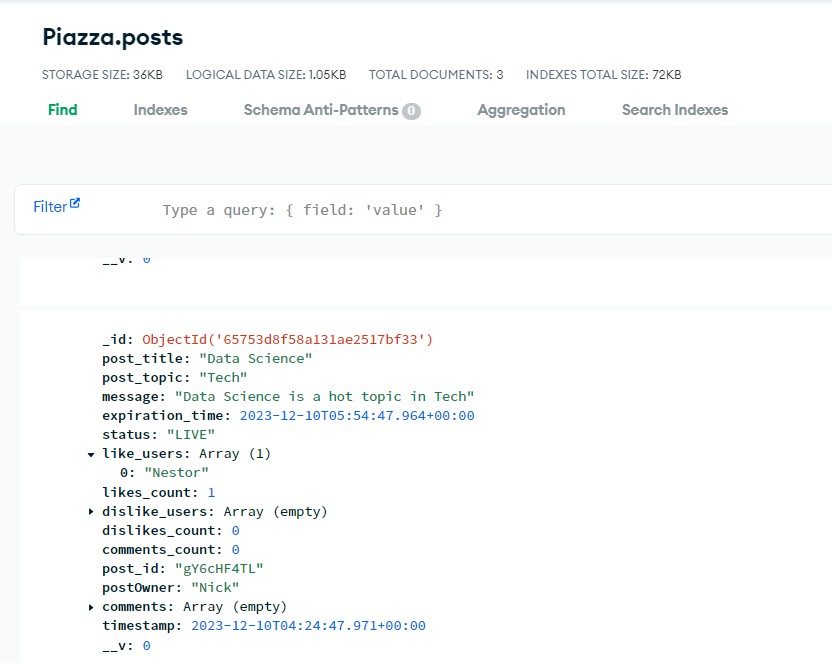
Olga likes Mary’s post on Tech topic

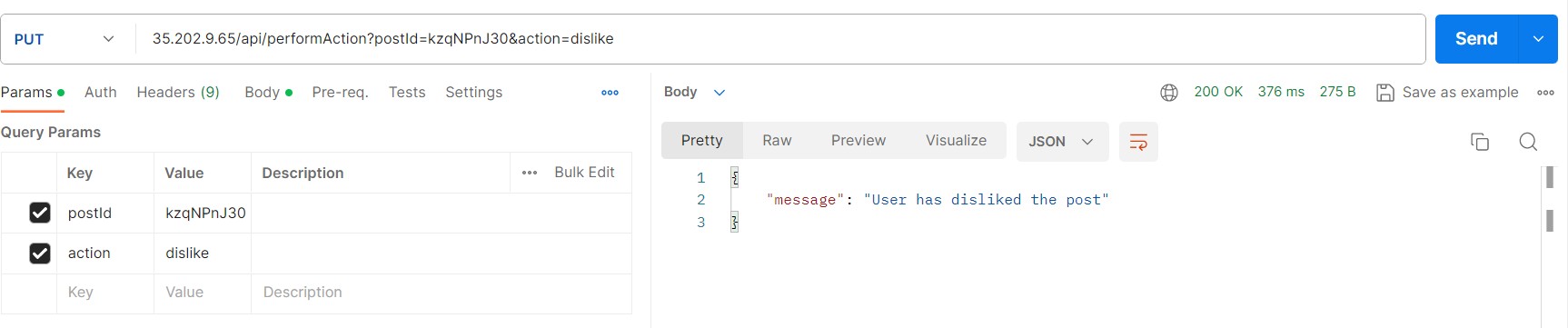
Nick likes Mary’s post on Tech topic

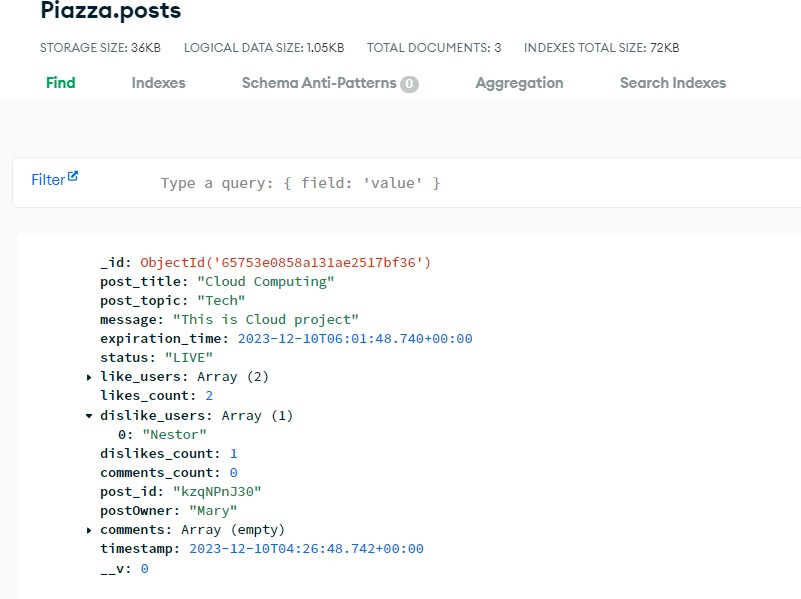
In the DB we can see likes\_count as 2 and like\_users are { Nick and Olga }

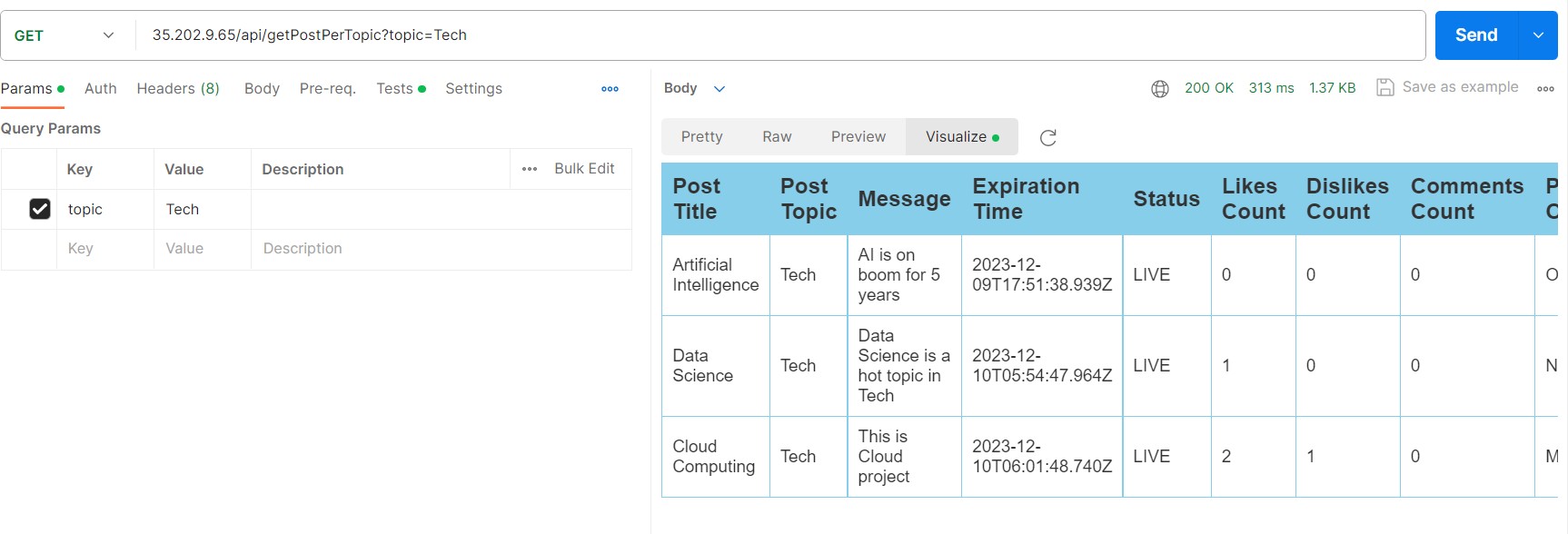
* 1. **Test Case 9:** Nestor “likes” Nick’s post and “dislikes” Mary’s on the Tech topic

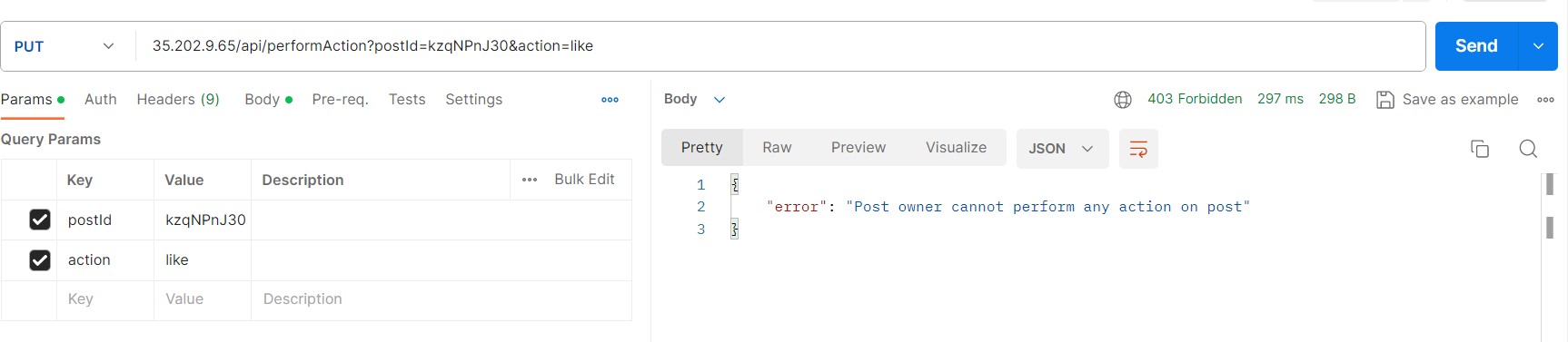
Nestor likes Nicks post on Tech



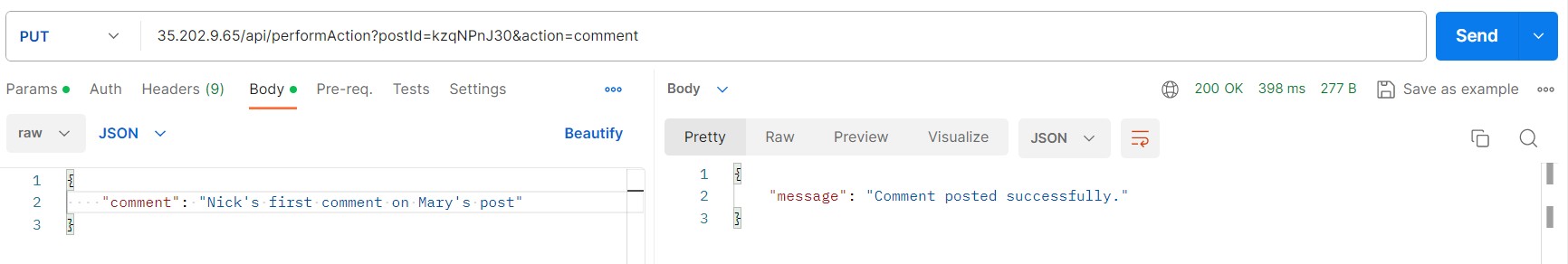
Nestor dislikes Mary’s topic on tech

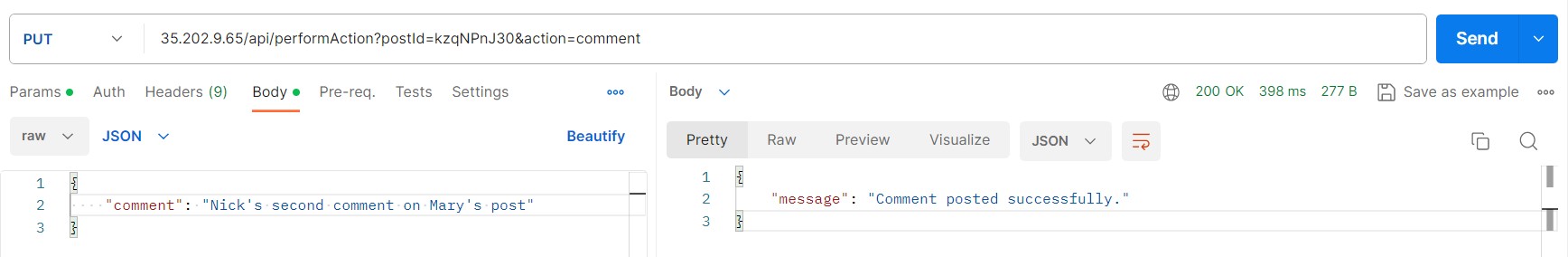


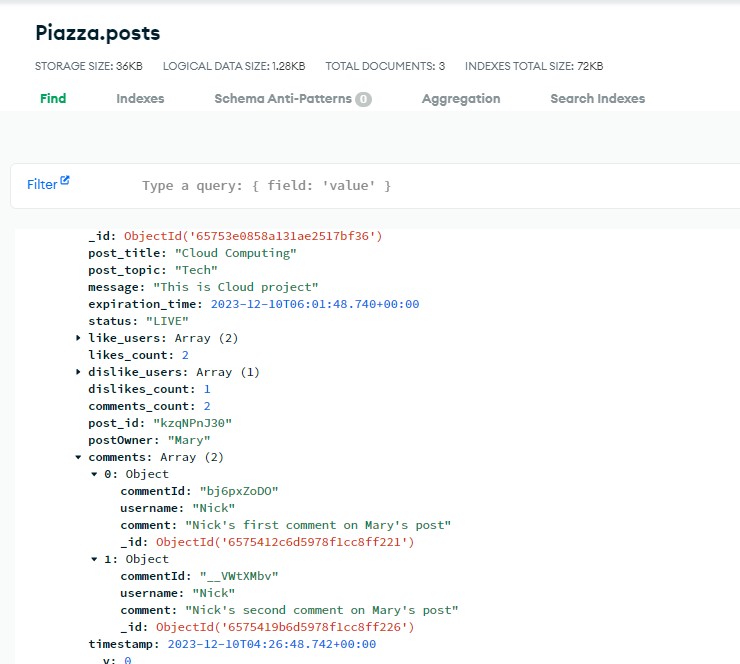
* 1. **Test Case 10:** Nick browses all the available posts on the Tech topic; at this stage, he can see the number of likes and dislikes for each post (Mary has two likes and one dislike, and Nick has one like). There are no comments made yet.
  2. **Test Case 11:** Mary likes her post on the Tech topic. This call should be unsuccessful; in Piazza, a post owner cannot like their messages.



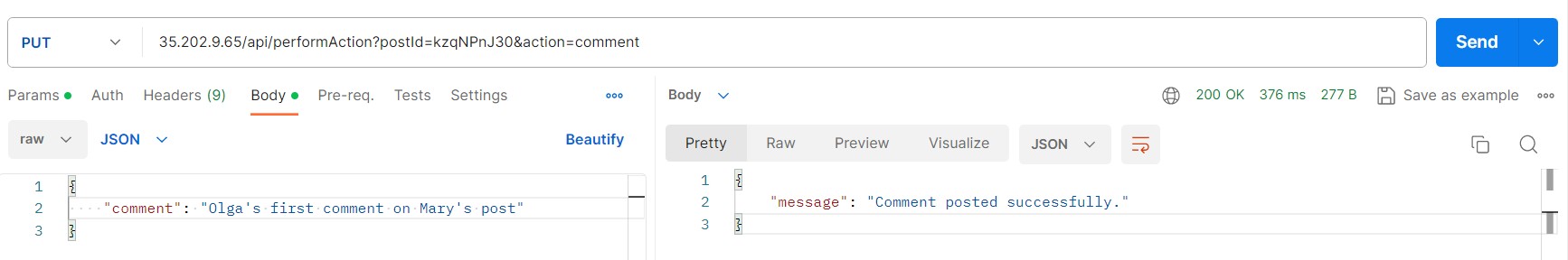
* 1. **Test Case 12:** Nick and Olga comment on Mary’s post on the Tech topic in a round-robin fashion (one after the other, adding at least two comments each).

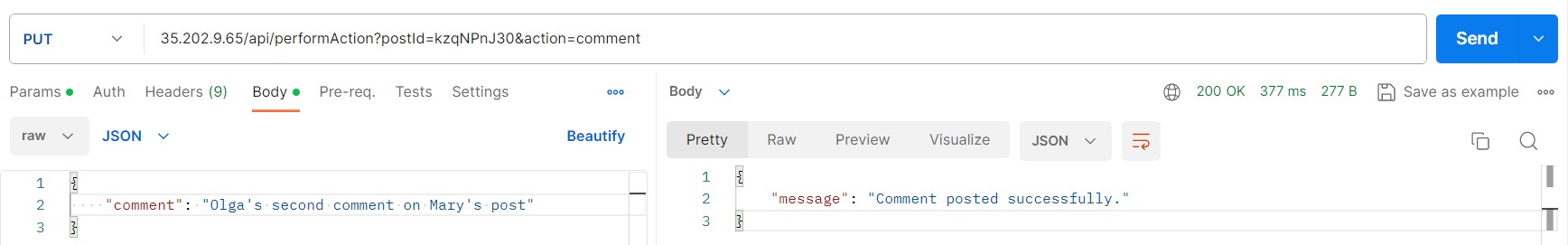
Nick comments on Mary’s post

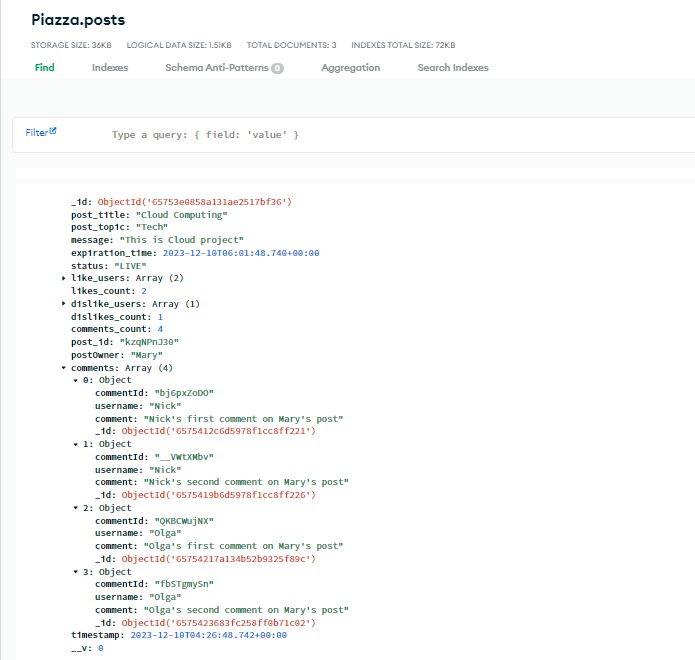




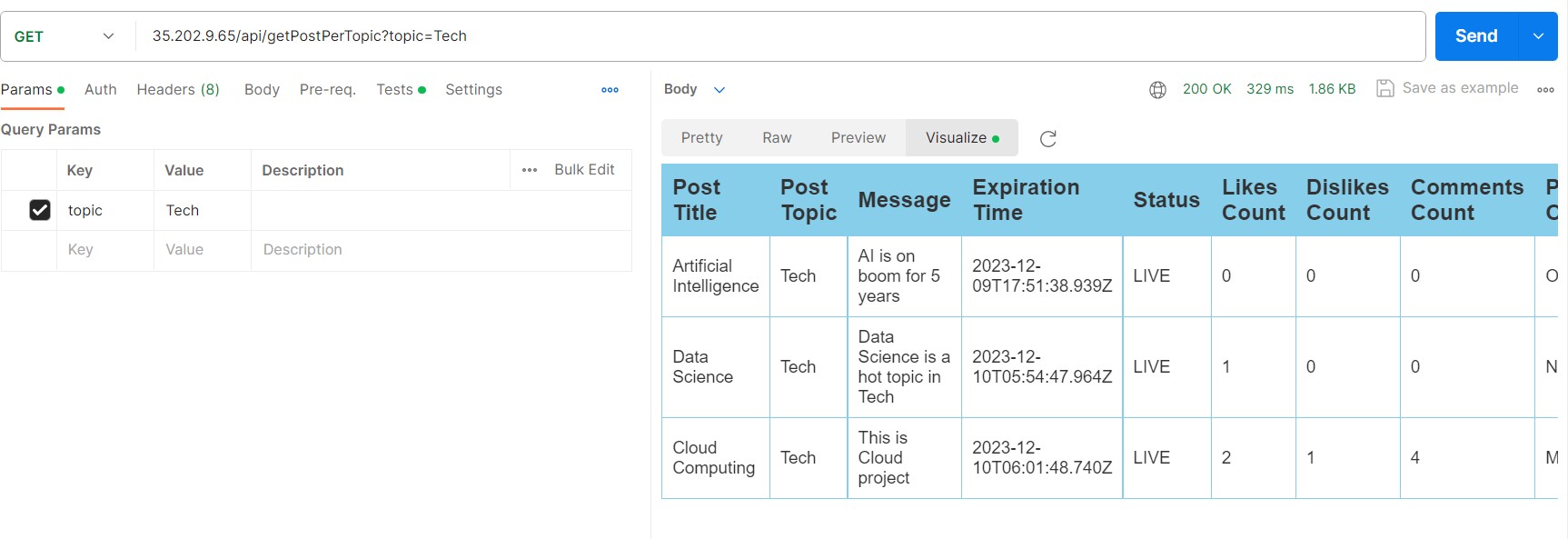
Olga comments on Mary’s post





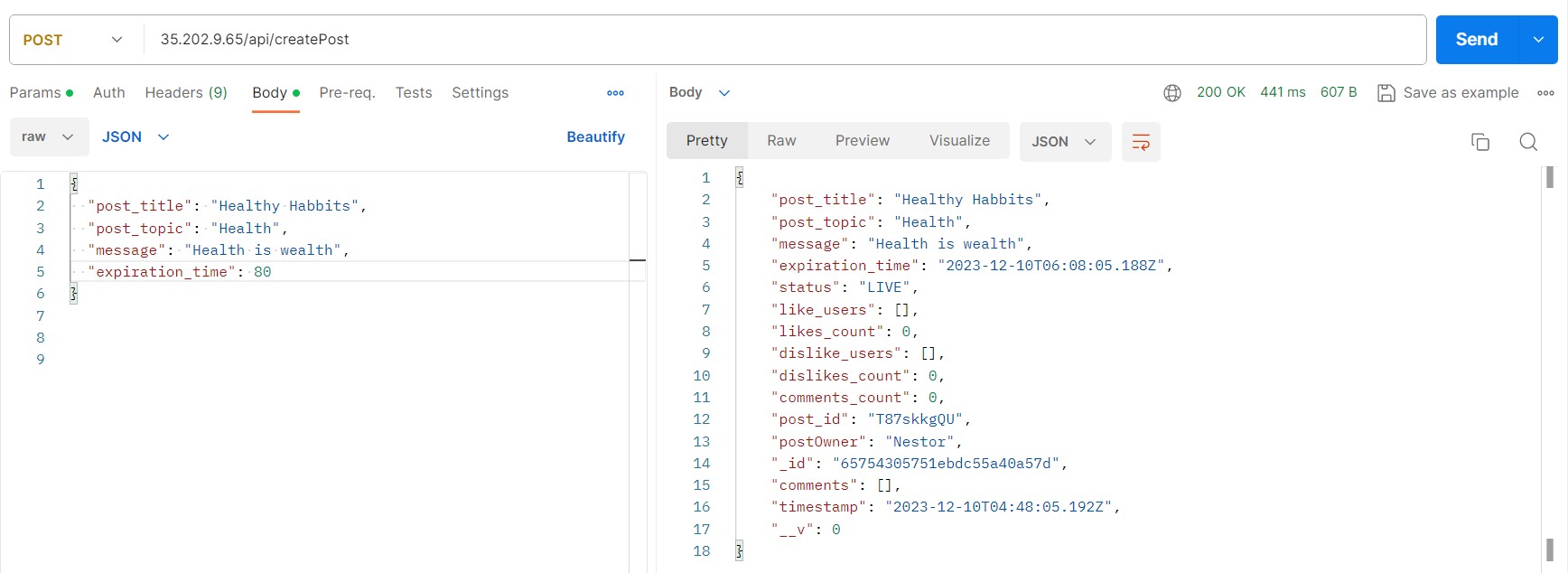


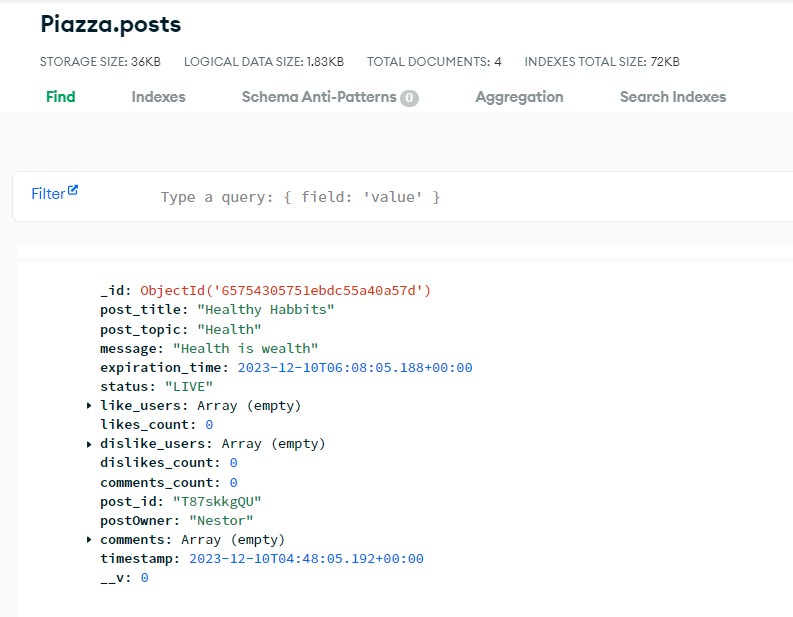
* 1. **Test Case 13:** Nick browses all the available posts in the Tech topic; at this stage, he can see the number of likes and dislikes of each post and the comments made.



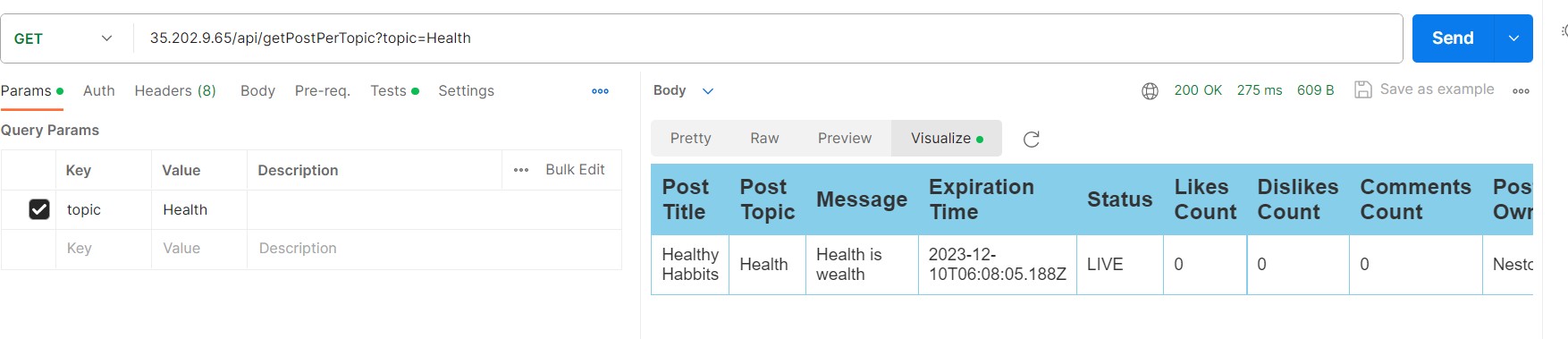
* 1. **Test Case 14:** Nestor posts a message in the Health topic with an expiration time using her

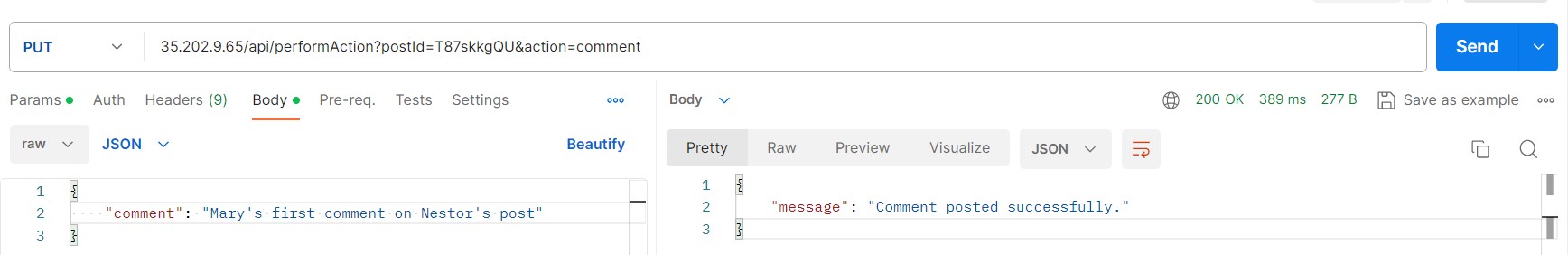
token.

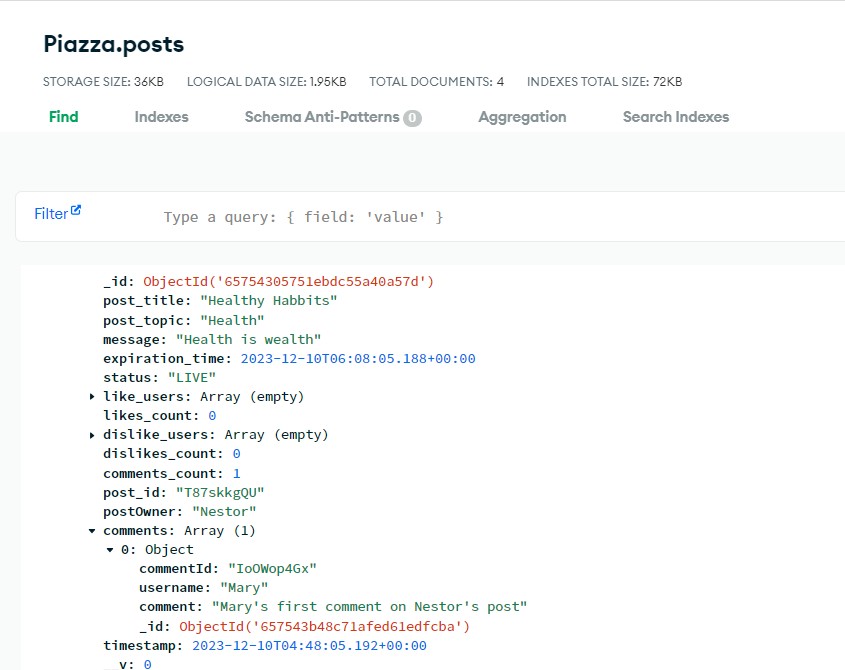




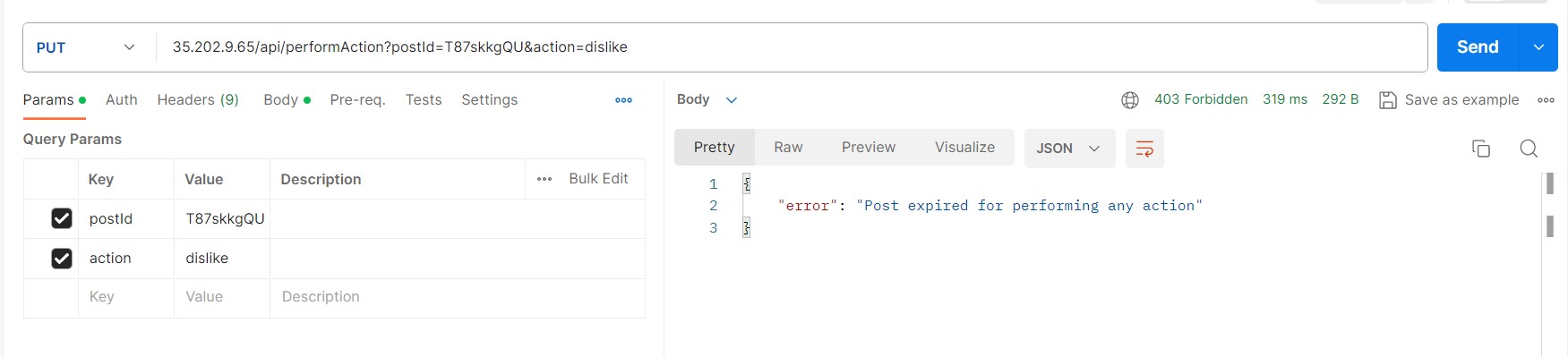
* 1. **Test Case 15:** Mary browses all the available posts on the Health topic; at this stage, she can see only Nestor’s post.



* 1. **Test Case 16:** Mary posts a comment in Nestor’s message on the Health topic.

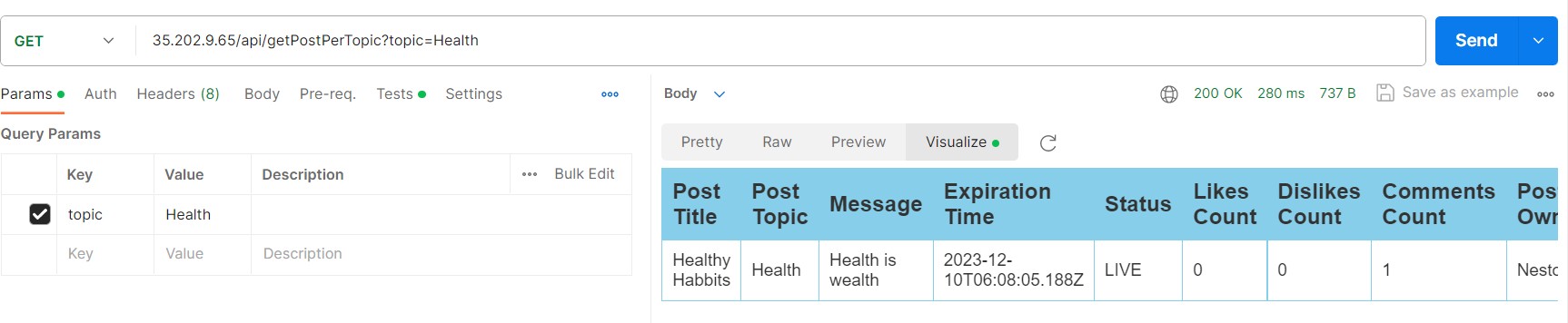


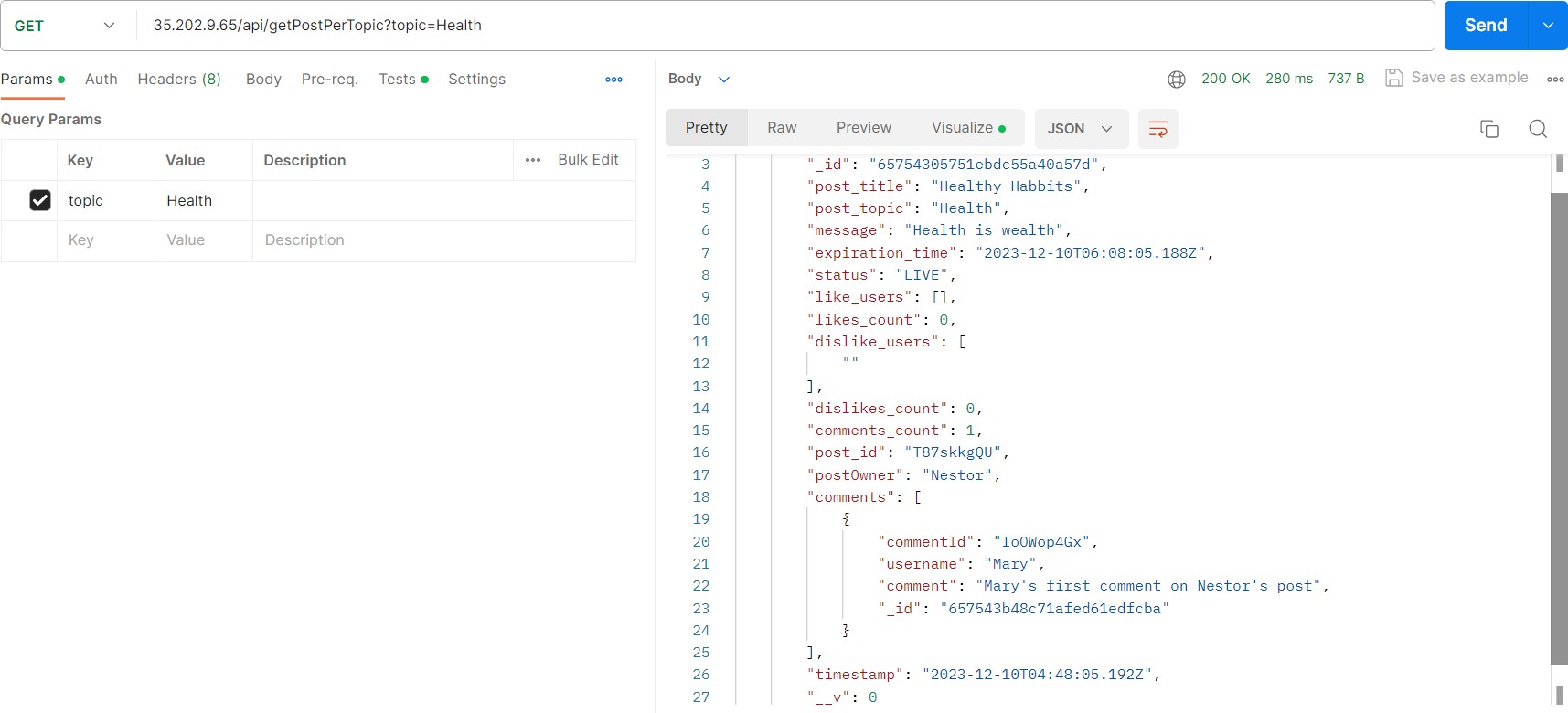
* 1. **Test Case 17:** Mary dislikes Nestor’s message on the Health topic after the end of post-expiration time. This should fail



Mary dislikes Nestor’s message on Health topic after the end of post-expiration, and this API call fails as the post is expired for performing any actions such as (likes/ dislikes/ comments)

* 1. **Test Case 18:** Nestor browses all the messages on the Health topic. There should be only one post (his own) with one comment (Mary’s).

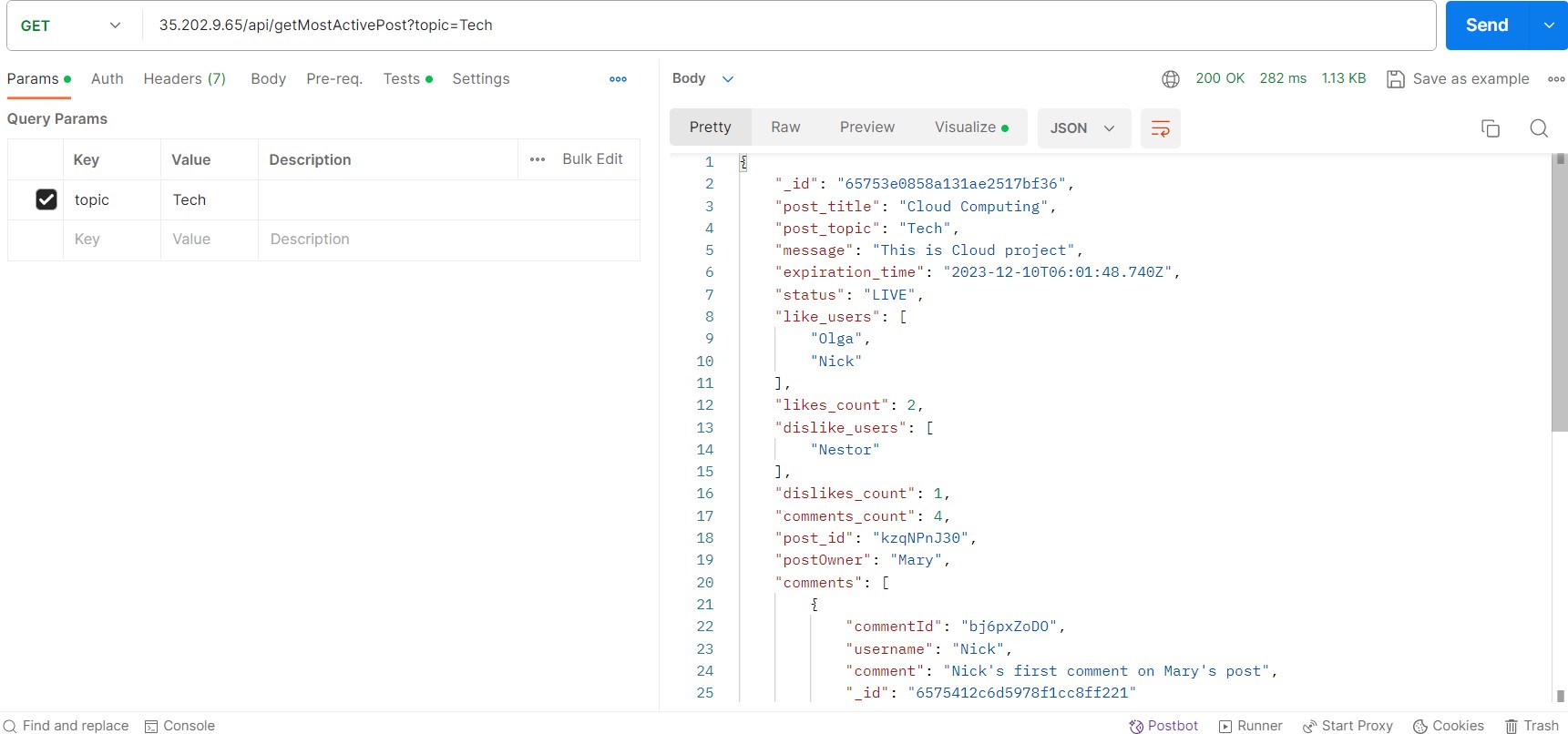


Json view of the response:

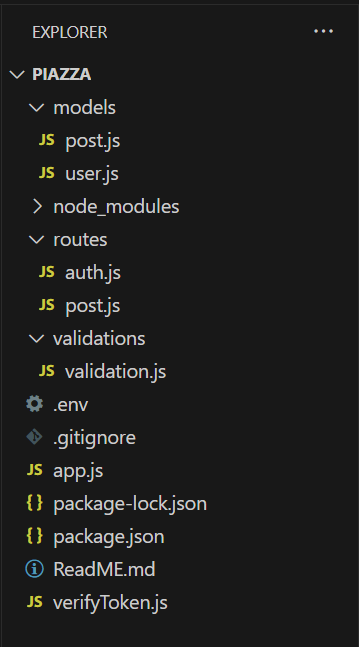
* 1. **Test Case 19:** Nick browses all the expired messages on the Sports topic. These should be

empty.

* 1. **Test Case 20:** Nestor queries for an active post with the highest interest (maximum number of likes and dislikes) in the Tech topic. This should be Mary’s post.



# 6.0 File Structure:



**The Models folder:** has the schema models for posts and users schema.

**The Node Modules:** has the imported node modules required for node project

**The Routes folder:** has two routers Auth and Post where Auth Router handles the registration of a new user to the Piazza API and login for the registered users.

**The Validation folder:** has a validation.js file which incorporates the validation of the user inputs using the JOI module, which is popular for data validation.

**The .env file:** has the Database connector URI (DB\_CONNECTOR) for connection for the database to the Mongoose and the (TOKEN\_SECRET) for the user.

**The .gitignore file:** To ignore the node\_modules, npm-debug.logs and .DS\_Store while committing the codebase to GitHub.

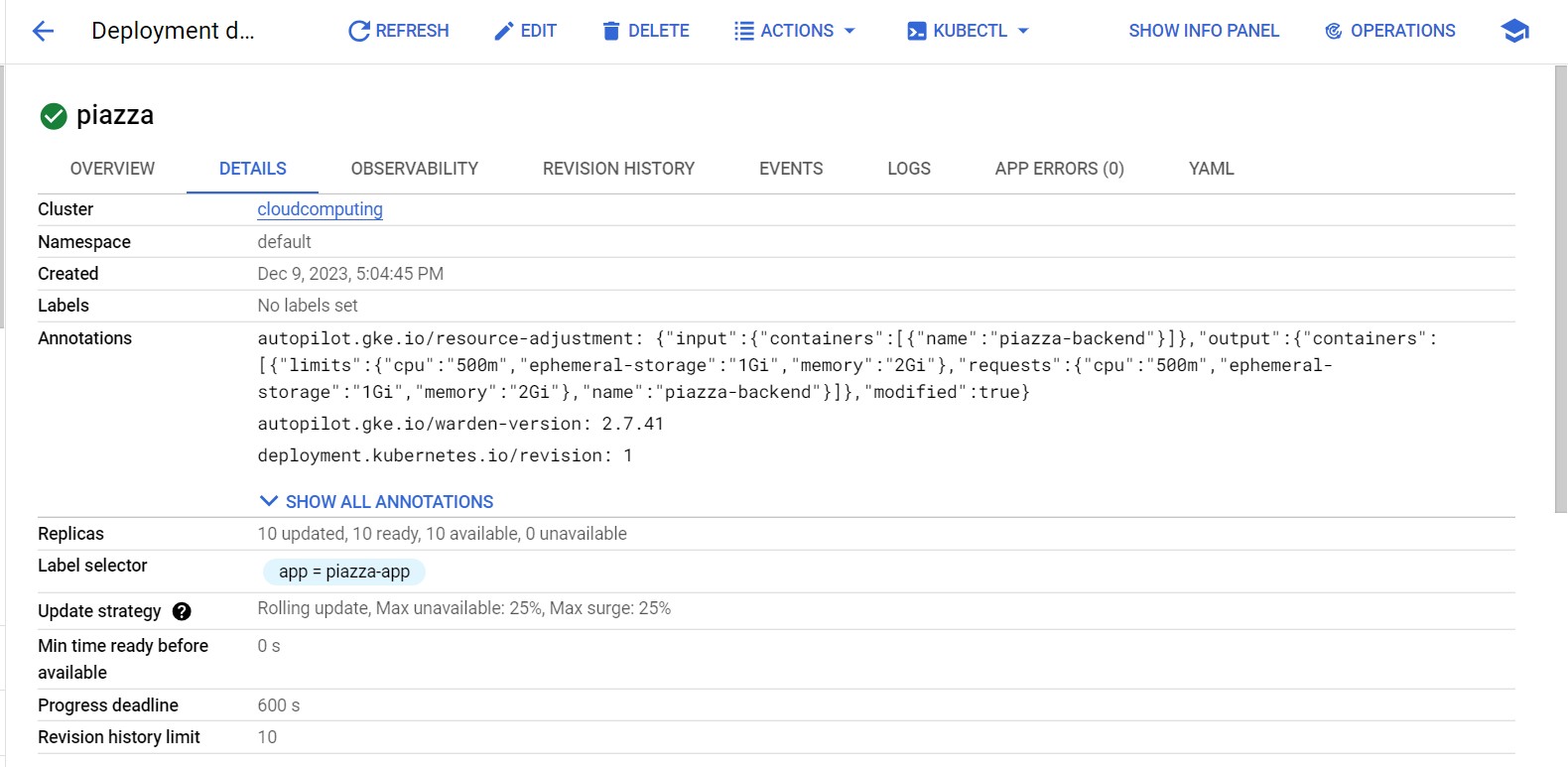
**The app.js file:** has the router path for both Auth and Post users, the database connection and the port localhost:3000 server configurations.

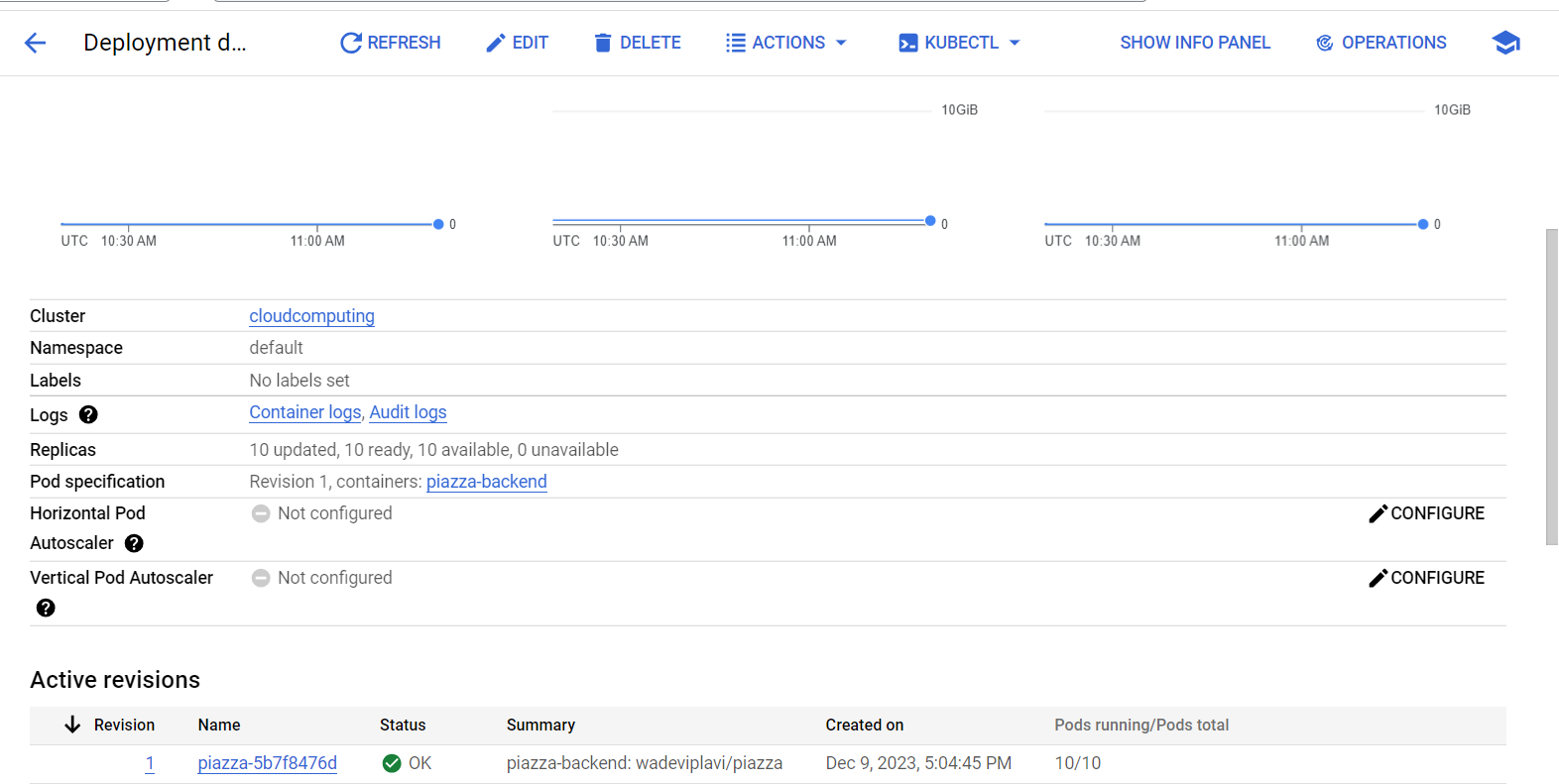
**The package-lock.json and package.json:** has the necessary packages required for implementation.

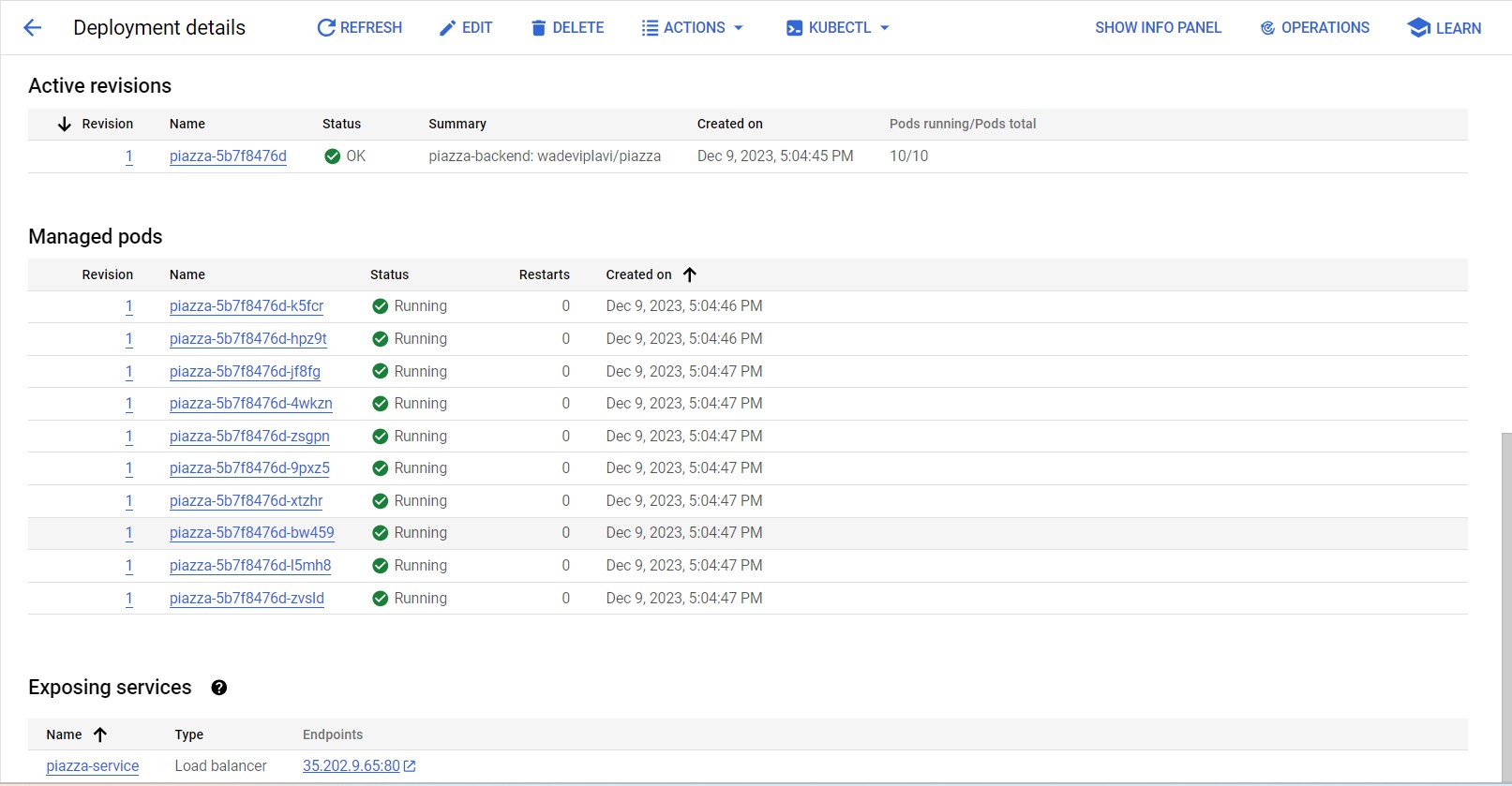
**The ReadME.md file:** has all the commands used for implementing the project

**The verifyToken.js:** handles the verification and authentication of users in Piazza API

# 7.0 Deployment to Kubernetes





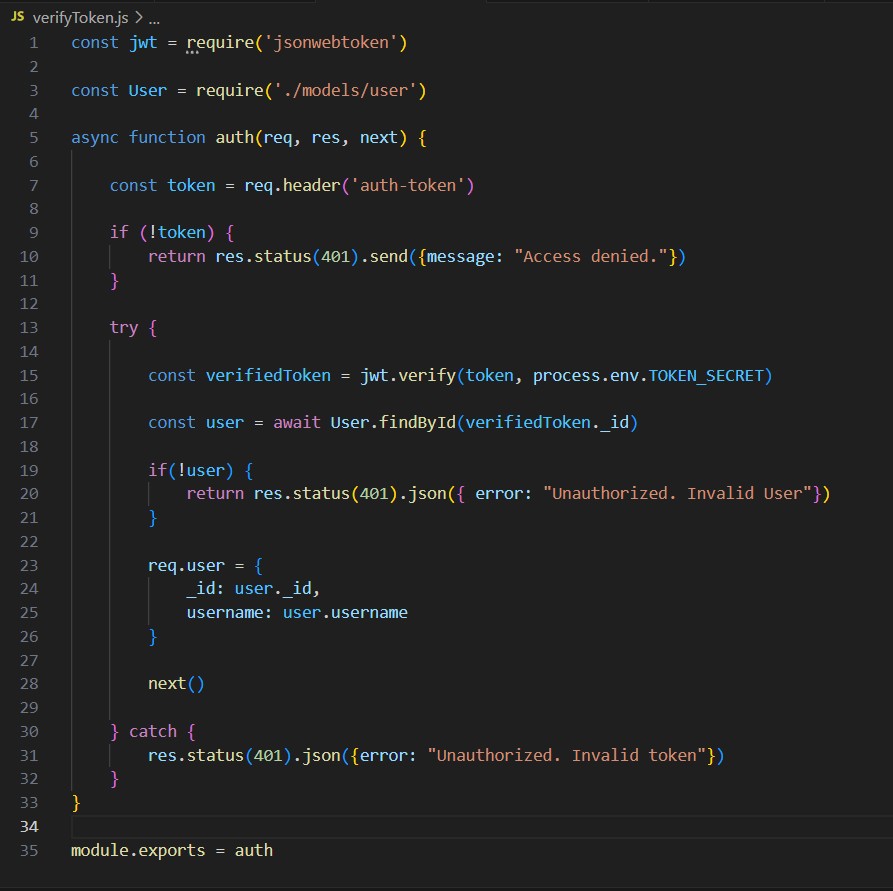


**7.0 Validation and Authentication**

Piazza implements rigorous data validation to ensure data integrity and user security. OAuth v2 is employed for secure user authentication, safeguarding user accounts from unauthorized access.

JWTs are a good way of securely transmitting information between parties because they can be signed, which means the users are identified with the unique tokens.

The async auth function covers the authentication part which authenticates the users with a unique token and maintains the user.id and user.username for the signed in user.

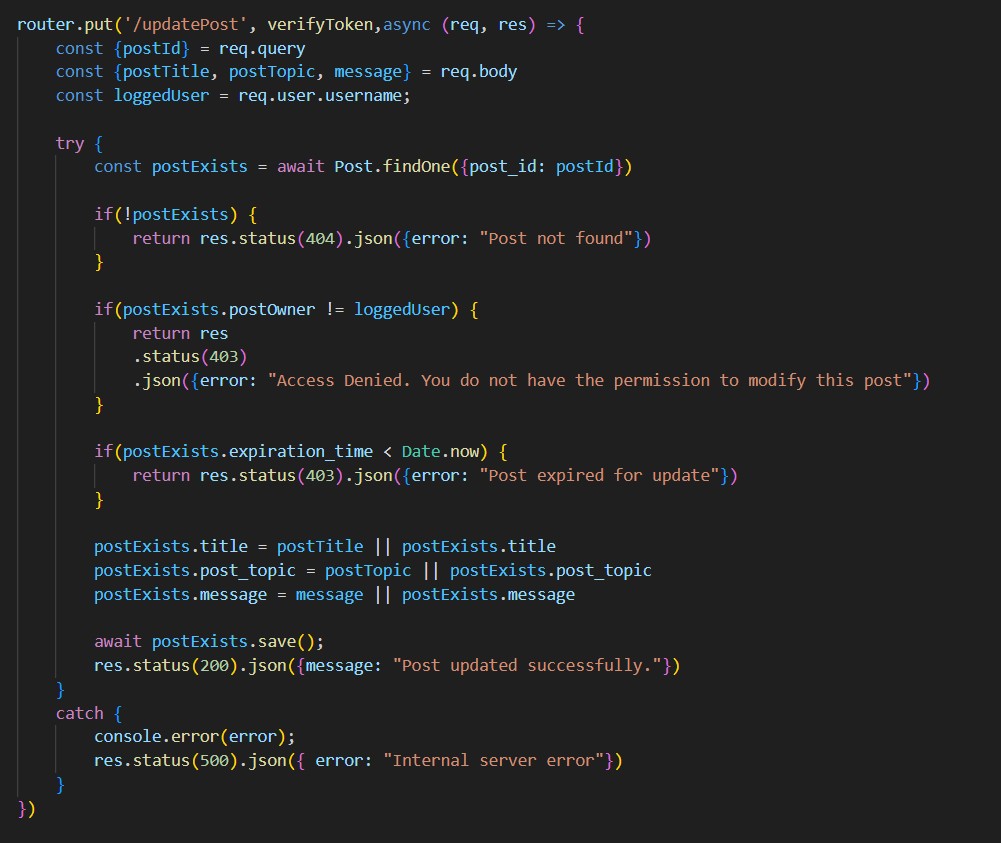


# API Endpoints:

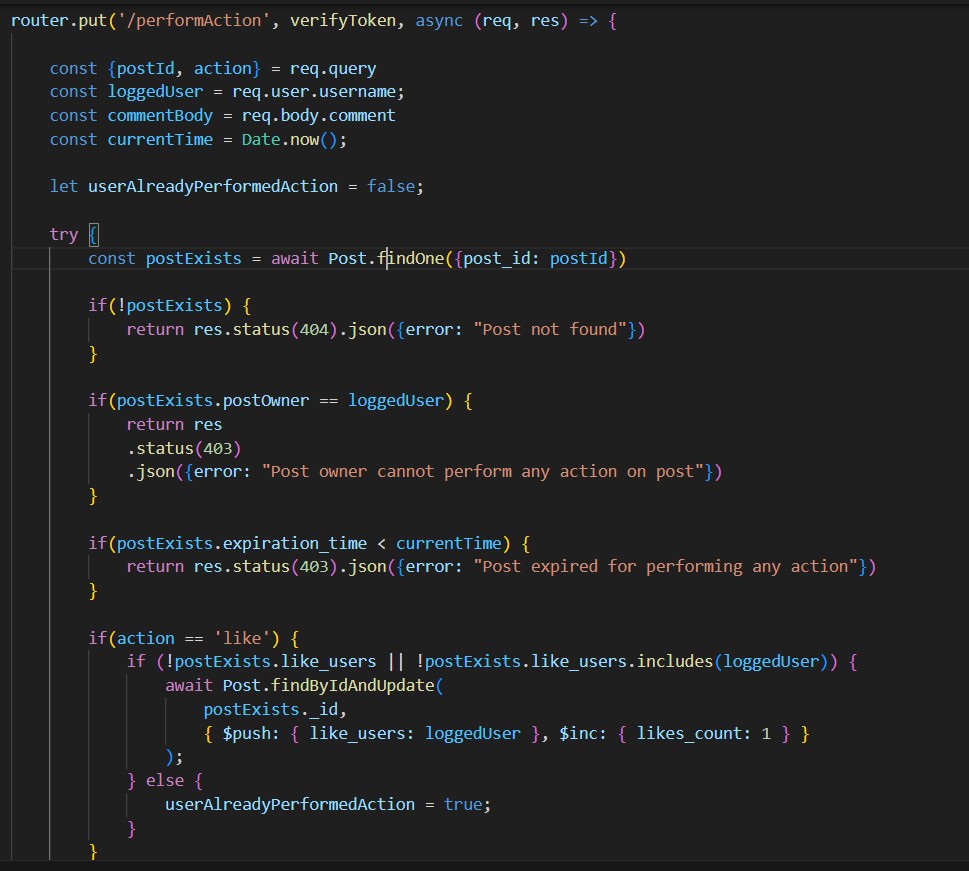
* + 1. **CreatePost:**

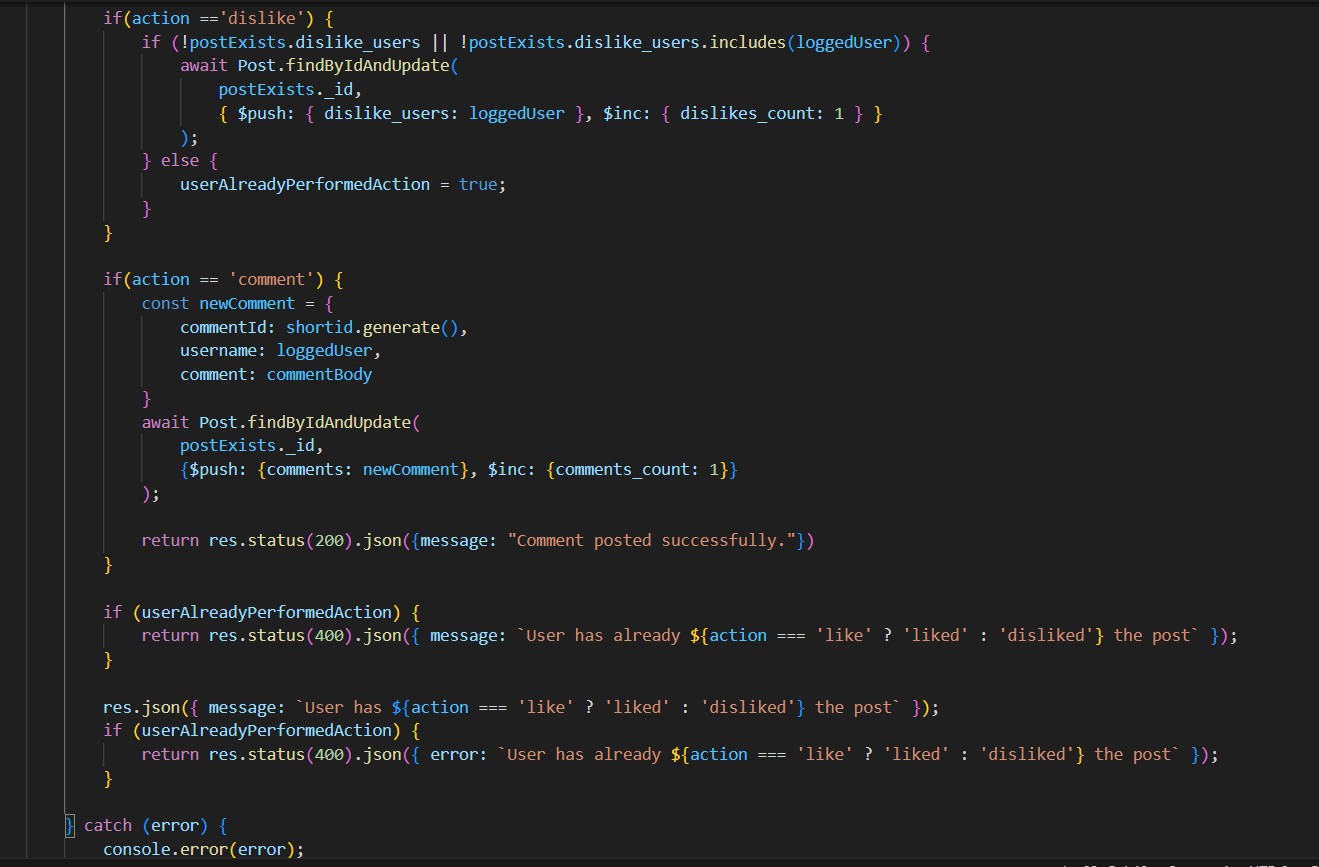


# Update Post:



* + 1. **Perform Actions (Like/ Dislike/ Comment):**





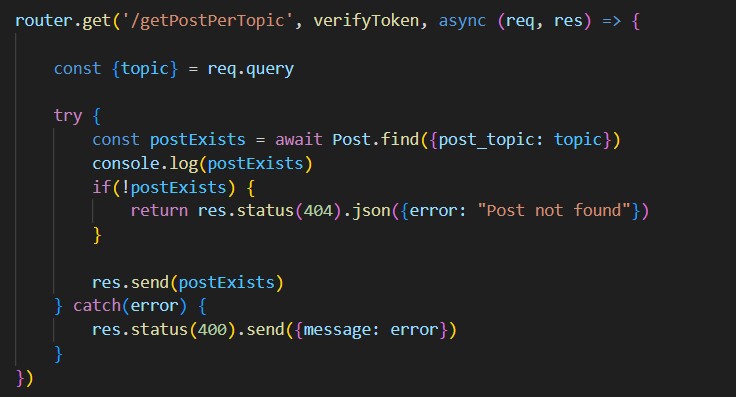
# Get a Post by ID:



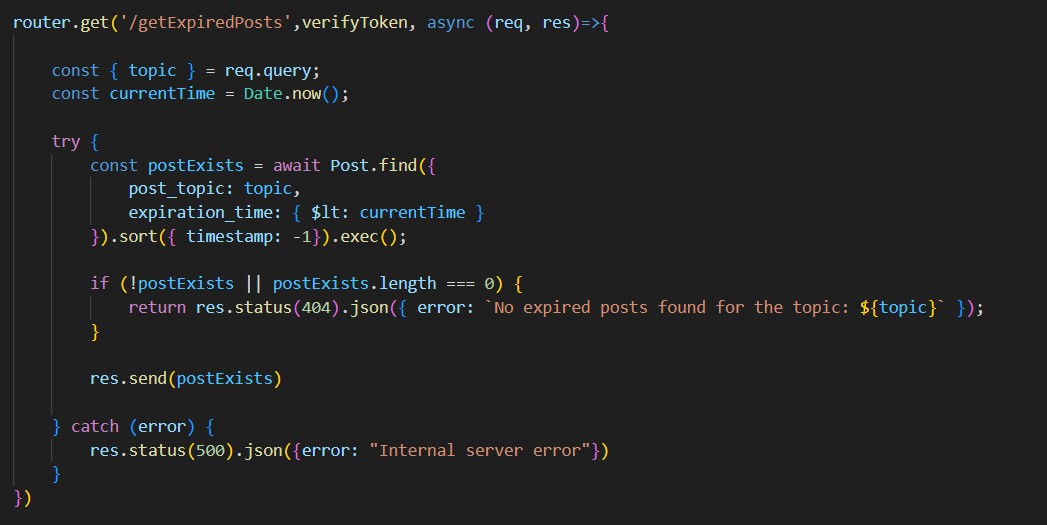
* + 1. **Get Most Active Post:**



# Get Post Per Topic:



* + 1. **Get Expired Posts:**



# Get All Posts:



**9.0 Conclusion**

In conclusion, Piazza offers a feature-rich social media experience with secure user authentication, interactive post functionalities, and efficient data storage using MongoDB. The implementation of OAuth v2 ensures a secure environment for users to engage and share content.