

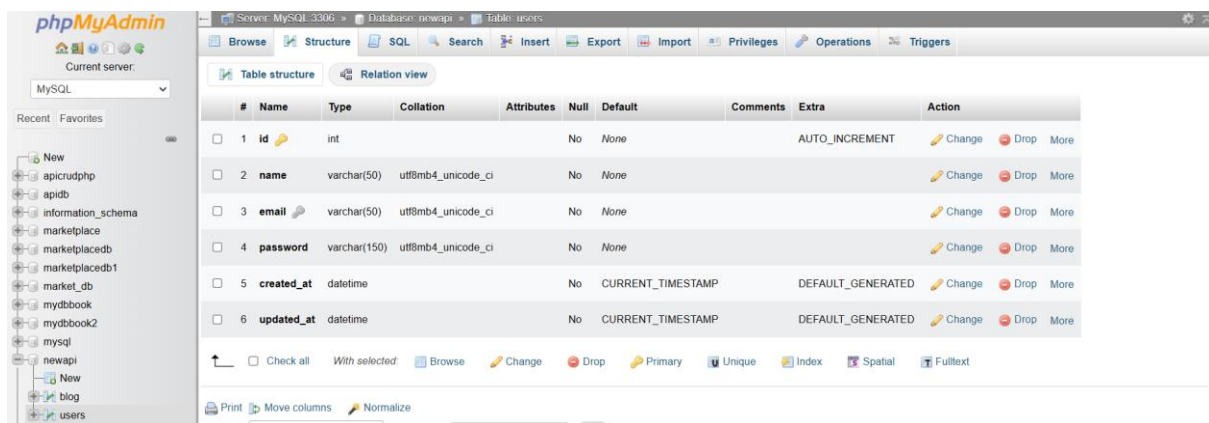
DATABASE CREATION

I have created a relational database to store data related to users, and blog by using phpMyAdmin. For each table there will be primary keys which are unique identifiers for records and a data type for each column in the tables.

Structures:

The structure for table **users**:

For the "users" table, the structure should include columns like id (as the primary key), name, email, and password to store user-related information, facilitating user registration and authentication processes.



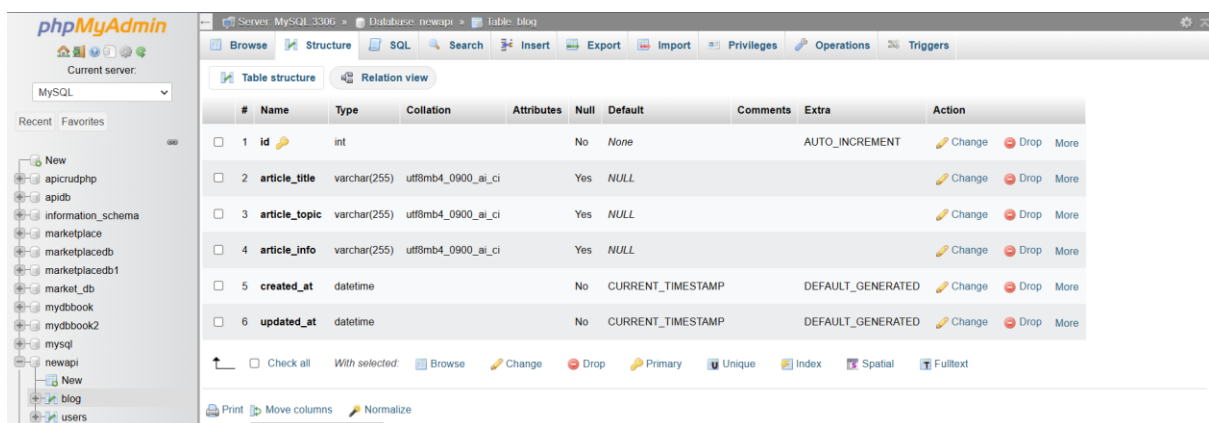
The screenshot shows the phpMyAdmin interface with the 'users' table structure displayed. The table has six columns: id (int, primary key, AUTO_INCREMENT), name (varchar(50)), email (varchar(50)), password (varchar(150)), created_at (datetime, DEFAULT_GENERATED), and updated_at (datetime, DEFAULT_GENERATED). The interface includes a sidebar with a database tree and a top navigation bar with options like Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers.

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|---|------------|--------------|--------------------|------------|------|-------------------|----------|-------------------|------------------|
| 1 | id | int | | | No | None | | AUTO_INCREMENT | Change Drop More |
| 2 | name | varchar(50) | utf8mb4_unicode_ci | | No | None | | | Change Drop More |
| 3 | email | varchar(50) | utf8mb4_unicode_ci | | No | None | | | Change Drop More |
| 4 | password | varchar(150) | utf8mb4_unicode_ci | | No | None | | | Change Drop More |
| 5 | created_at | datetime | | | No | CURRENT_TIMESTAMP | | DEFAULT_GENERATED | Change Drop More |
| 6 | updated_at | datetime | | | No | CURRENT_TIMESTAMP | | DEFAULT_GENERATED | Change Drop More |

Figure 1

The structure for table **blog**:

For the "blog" table, the structure includes columns for post information, such as id (as the primary key), article_title, article_topic and article_info enabling the storage and management of various posts.



The screenshot shows the phpMyAdmin interface with the 'blog' table structure displayed. The table has six columns: id (int, primary key, AUTO_INCREMENT), article_title (varchar(255)), article_topic (varchar(255)), article_info (varchar(255)), created_at (datetime, DEFAULT_GENERATED), and updated_at (datetime, DEFAULT_GENERATED). The interface includes a sidebar with a database tree and a top navigation bar with options like Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers.

| # | Name | Type | Collation | Attributes | Null | Default | Comments | Extra | Action |
|---|---------------|--------------|--------------------|------------|------|-------------------|----------|-------------------|------------------|
| 1 | id | int | | | No | None | | AUTO_INCREMENT | Change Drop More |
| 2 | article_title | varchar(255) | utf8mb4_0900_ai_ci | | Yes | NULL | | | Change Drop More |
| 3 | article_topic | varchar(255) | utf8mb4_0900_ai_ci | | Yes | NULL | | | Change Drop More |
| 4 | article_info | varchar(255) | utf8mb4_0900_ai_ci | | Yes | NULL | | | Change Drop More |
| 5 | created_at | datetime | | | No | CURRENT_TIMESTAMP | | DEFAULT_GENERATED | Change Drop More |
| 6 | updated_at | datetime | | | No | CURRENT_TIMESTAMP | | DEFAULT_GENERATED | Change Drop More |

Figure 2

The tables are then populated with sample data:

Table users:

```
INSERT INTO `users` (`id`, `name`, `email`, `password`, `created_at`, `updated_at`) VALUES  
(1, 'shameem', 'nubee@gmail.com', '$2y$10$PMGp9zRtJZCVPXxakudZZuuSspgtDC1UfKKFfE0mkUcrsn.PWXQDW', '2023-09-08 11:23:58', '2023-09-08 11:23:58');
```

Figure 3

Table blog:

```
INSERT INTO `blog` (`id`, `article_title`, `article_topic`, `article_info`, `created_at`, `updated_at`) VALUES  
(1, 'Venom 3', 'movie', 'I have watch venom the last dance and it was interesting.', '2012-06-01 02:12:30', '2023-09-07 15:07:00');
```

Figure 4

API DEVELOPMENT

API Development is crucial as it serves as the bridge between the frontend and the database, enabling data retrieval, manipulation, and interaction with the blog application. It ensures that the application functions smoothly and securely while providing users with the expected functionalities.

In the folder newapi/api we have the classes, controllers, vendor and the models like login, blogs and then the index which has the routes and the endpoint.

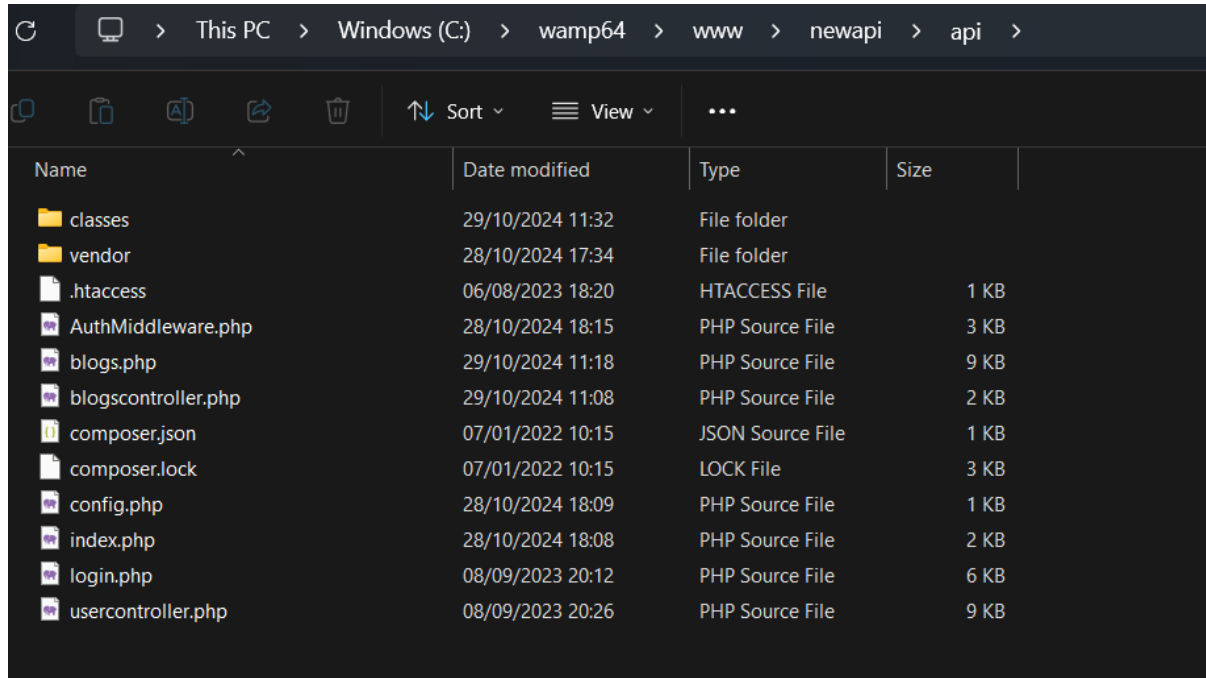


Figure 5

And in the folder classes we have e.g., users: Handles user authentication operations such as registration.

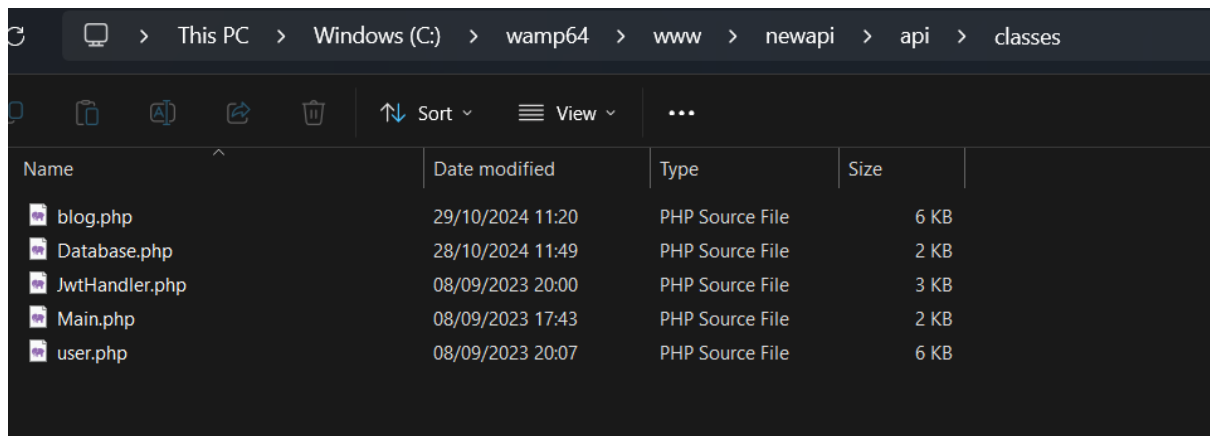


Figure 6

The folder vendor contains the composer and the firebase which perform the authentication part.

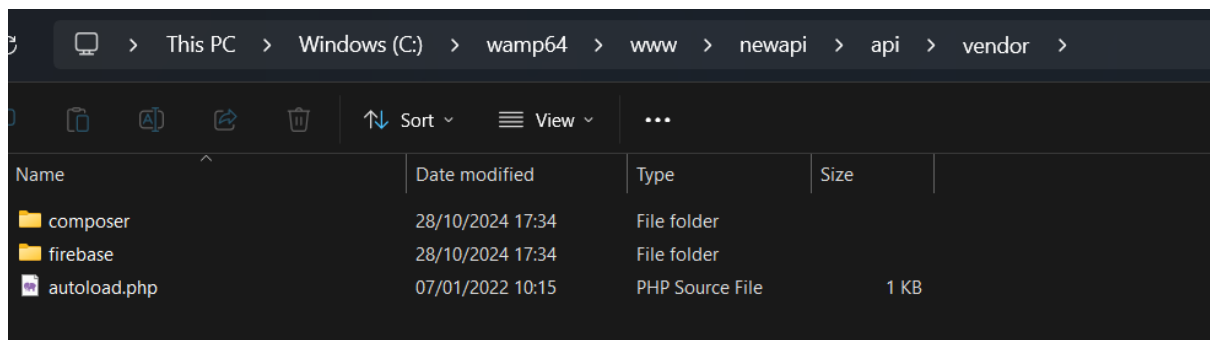
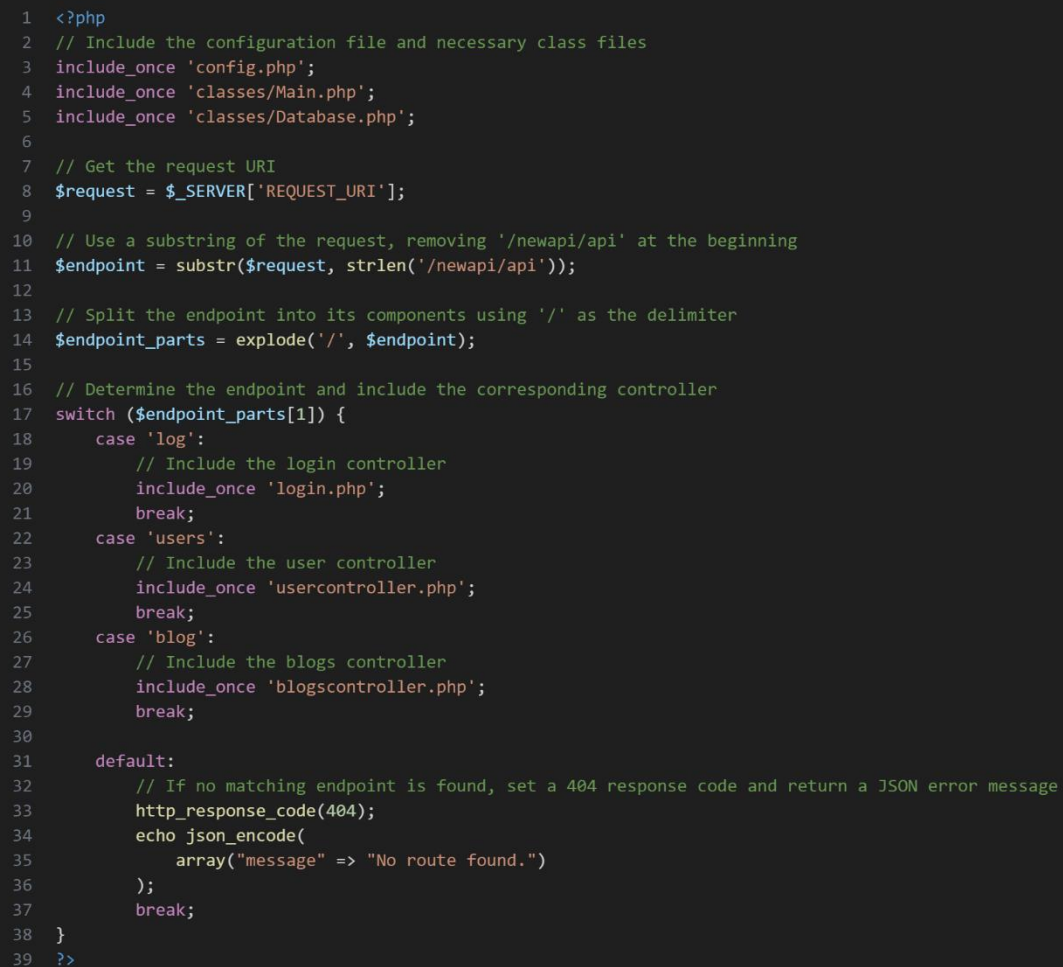


Figure 7

ROUTE HANDLING

Route Handling serves as the traffic controller for incoming HTTP requests, ensuring they are directed to the relevant parts of the application for processing. It plays a pivotal role in organizing and managing the flow of data and operations in the backend. In "index.php" file there's a router that directs incoming requests to their respective controllers based on the requested endpoint. Endpoints, such as /log or /users, are associated with specific controllers (e.g., login.php or usercontroller.php).



```
1  <?php
2  // Include the configuration file and necessary class files
3  include_once 'config.php';
4  include_once 'classes/Main.php';
5  include_once 'classes/Database.php';
6
7  // Get the request URI
8  $request = $_SERVER['REQUEST_URI'];
9
10 // Use a substring of the request, removing '/newapi/api' at the beginning
11 $endpoint = substr($request, strlen('/newapi/api'));
12
13 // Split the endpoint into its components using '/' as the delimiter
14 $endpoint_parts = explode('/', $endpoint);
15
16 // Determine the endpoint and include the corresponding controller
17 switch ($endpoint_parts[1]) {
18     case 'log':
19         // Include the login controller
20         include_once 'login.php';
21         break;
22     case 'users':
23         // Include the user controller
24         include_once 'usercontroller.php';
25         break;
26     case 'blog':
27         // Include the blogs controller
28         include_once 'blogscontroller.php';
29         break;
30
31     default:
32         // If no matching endpoint is found, set a 404 response code and return a JSON error message
33         http_response_code(404);
34         echo json_encode(
35             array("message" => "No route found.")
36         );
37         break;
38 }
39 ?>
```

Figure 8

CONTROLLERS

Each controller is responsible for parsing incoming HTTP requests, executing necessary actions (e.g., database operations), and generating appropriate responses.

We have the functions like the (GET, POST, PUT, DELETE) and for each controller (blog & User) it performed the same functions. Here is an example for the blog controller:

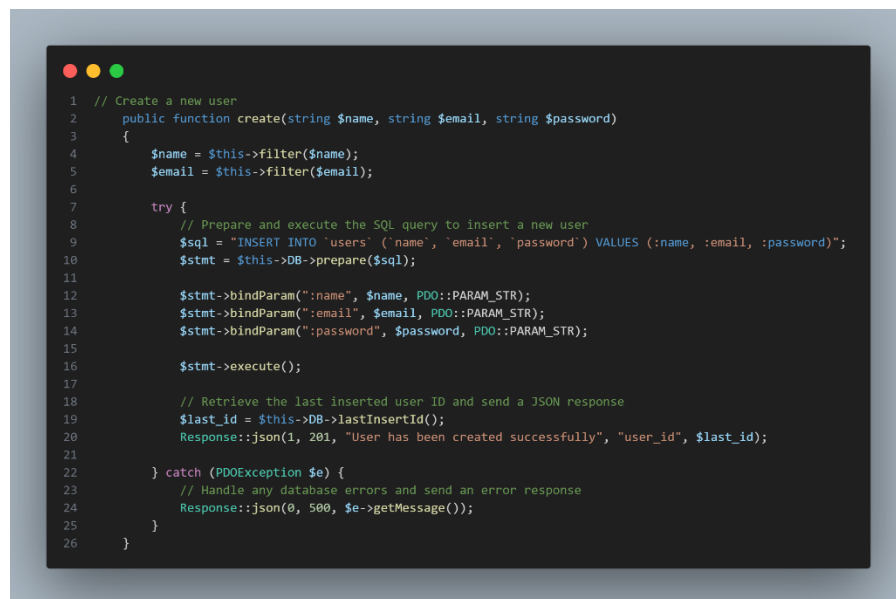


```
1 // Handle different request methods
2 switch($requestMethod) {
3     case 'GET':
4         // Create an instance of the BlogAuth class for GET requests
5         $auth = new BlogAuth($conn, $allHeaders);
6
7         // Echo the JSON response of the getBlog method
8         echo json_encode($auth->getBlog());
9         break;
10
11     case 'POST':
12         // Create an instance of the BlogAuth class for POST requests
13         $auth1 = new BlogAuth($conn, $allHeaders);
14
15         // Echo the JSON response of the InsertBlog method
16         echo json_encode($auth1->InsertBlog());
17         break;
18
19     case 'PUT':
20         // Create an instance of the BlogAuth class for PUT requests
21         $auth3 = new BlogAuth($conn, $allHeaders);
22
23         // Echo the JSON response of the UpdateBlog method
24         echo json_encode($auth3->UpdateBlog());
25         break;
26
27     case 'DELETE':
28         // Create an instance of the BlogAuth class for DELETE requests
29         $auth2 = new BlogAuth($conn, $allHeaders);
30
31         // Echo the JSON response of the DeleteBlog method
32         echo json_encode($auth2->DeleteBlog());
33         break;
34 }
35 ?>
```

Figure 9

USER AUTHENTICATION

User Authentication is a fundamental security measure that safeguards user data and ensures that interactions with the e-commerce application are secure and personalized. It relies on a token-based system to authenticate users and protect against unauthorized access. The authentication system includes two essential methods: `register()` and `login()` within the user class where the `register()` method allows new users to create an account by providing essential details such as name, email, and password whereas the `login()` method validates user credentials (email and password), and upon successful validation, it generates a unique authentication token. This token serves as proof of authentication and is required for accessing protected endpoints and personalizing the user experience. All CRUD (Create, Read, Update, Delete) operations within the application verify the authenticity of the user by checking the validity of this authentication token. This is done to ensure that only authorized users can access sensitive data and functionalities.



```
1 // Create a new user
2 public function create(string $name, string $email, string $password)
3 {
4     $name = $this->filter($name);
5     $email = $this->filter($email);
6
7     try {
8         // Prepare and execute the SQL query to insert a new user
9         $sql = "INSERT INTO 'users' ( 'name', 'email', 'password' ) VALUES (:name, :email, :password)";
10        $stmt = $this->DB->prepare($sql);
11
12        $stmt->bindParam(":name", $name, PDO::PARAM_STR);
13        $stmt->bindParam(":email", $email, PDO::PARAM_STR);
14        $stmt->bindParam(":password", $password, PDO::PARAM_STR);
15
16        $stmt->execute();
17
18        // Retrieve the last inserted user ID and send a JSON response
19        $last_id = $this->DB->lastInsertId();
20        Response::json(1, 201, "User has been created successfully", "user_id", $last_id);
21
22    } catch (PDOException $e) {
23        // Handle any database errors and send an error response
24        Response::json(0, 500, $e->getMessage());
25    }
26 }
```

Figure 10



Figure 11

TESTING USING POSTMAN

Table **users** in database “newapi”:

| id | name | email | password | created_at | updated_at |
|----|------|-------|----------|------------|------------|
|----|------|-------|----------|------------|------------|

Figure 12

1. Creating user:

Creating a user using POST. It will require a name, an email and a password that consists of at least 8 characters long:

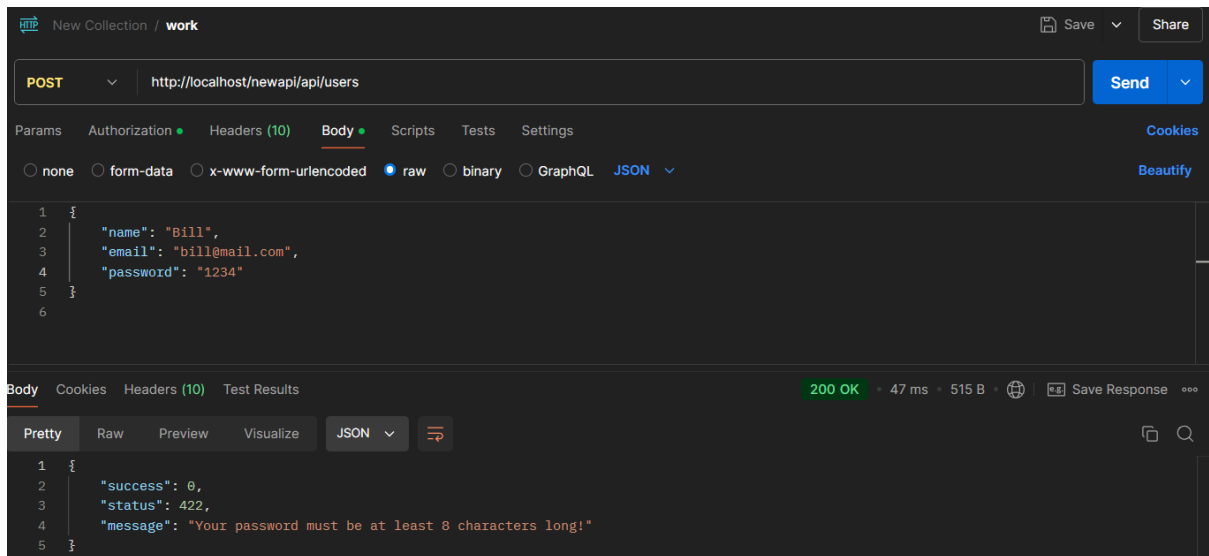


Figure 13

Once created, a respond message is prompted that states that it has been registered successfully:

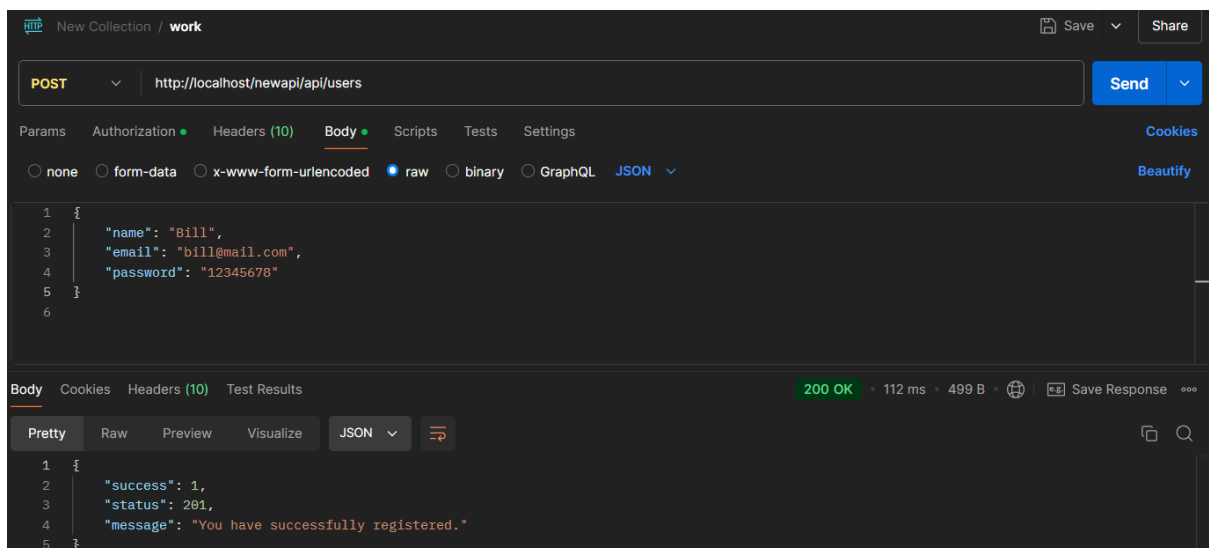


Figure 14

The record has also been registered in the database “newapi” in phpMyAdmin:

| | | | | id | name | email | password | created_at | updated_at |
|--------------------------|------|------|--------|----|---------|-----------------|--|---------------------|---------------------|
| <input type="checkbox"/> | Edit | Copy | Delete | 1 | shameem | nubee@gmail.com | \$2y\$10\$PMGp9zRiJZCVPXxakudZZuuSspgtDC1UfKKFfEOmkUc... | 2023-09-08 11:23:58 | 2023-09-08 11:23:58 |
| <input type="checkbox"/> | Edit | Copy | Delete | 3 | Wick | john@mail.com | \$2y\$10\$jCeQE/3aO2iZltw3UE22wAIsuE6/c/exKHnjtLR9t... | 2024-10-28 12:55:56 | 2024-10-28 12:57:19 |
| <input type="checkbox"/> | Edit | Copy | Delete | 4 | Ben | ben@mail.com | \$2y\$10\$HwdNT3mDqQPMUquN9jD8FuHMr0md3atk5WnowOCZy7k... | 2024-10-29 11:23:29 | 2024-10-29 11:23:29 |
| <input type="checkbox"/> | Edit | Copy | Delete | 5 | Bill | bill@mail.com | \$2y\$10\$WjBBjwqTGfB4QqiaI3VWt0Wyl6d1GWrJpQpGbxri... | 2024-10-29 16:10:21 | 2024-10-29 16:10:21 |

Figure 15

2. Update user:

The record before updating a user in database:

| | | id | name | email | password | created_at | updated_at |
|--------------------------|------------------|----|---------|-----------------|--|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | shameem | nubee@gmail.com | \$2y\$10\$PMGp9zRtJZCVPXxakudZZuuSspgtDC1UfKKFfEOmkUc... | 2023-09-08 11:23:58 | 2023-09-08 11:23:58 |
| <input type="checkbox"/> | Edit Copy Delete | 3 | Wick | john@mail.com | \$2y\$10\$JCeQE/3aO2IZltw3UE22w.AISuE6/c/exKHnjt.R9t... | 2024-10-28 12:55:56 | 2024-10-28 12:57:19 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Ben | ben@mail.com | \$2y\$10\$HwdNT3mDqQPMUquN9jD8FuHMr0md3atk5WnowOCZy7k... | 2024-10-29 11:23:29 | 2024-10-29 11:23:29 |
| <input type="checkbox"/> | Edit Copy Delete | 5 | Bill | bill@mail.com | \$2y\$10\$WjBBjvwqTGfB4QqiaI3VW.t0Wyl6d1GWrJJpQpGbxri... | 2024-10-29 16:10:21 | 2024-10-29 16:10:21 |

Figure 16

Updating a user using PUT. It will require a valid id in order to update any user details. Once updated, a respond message is prompted that states that it has been updated successfully:

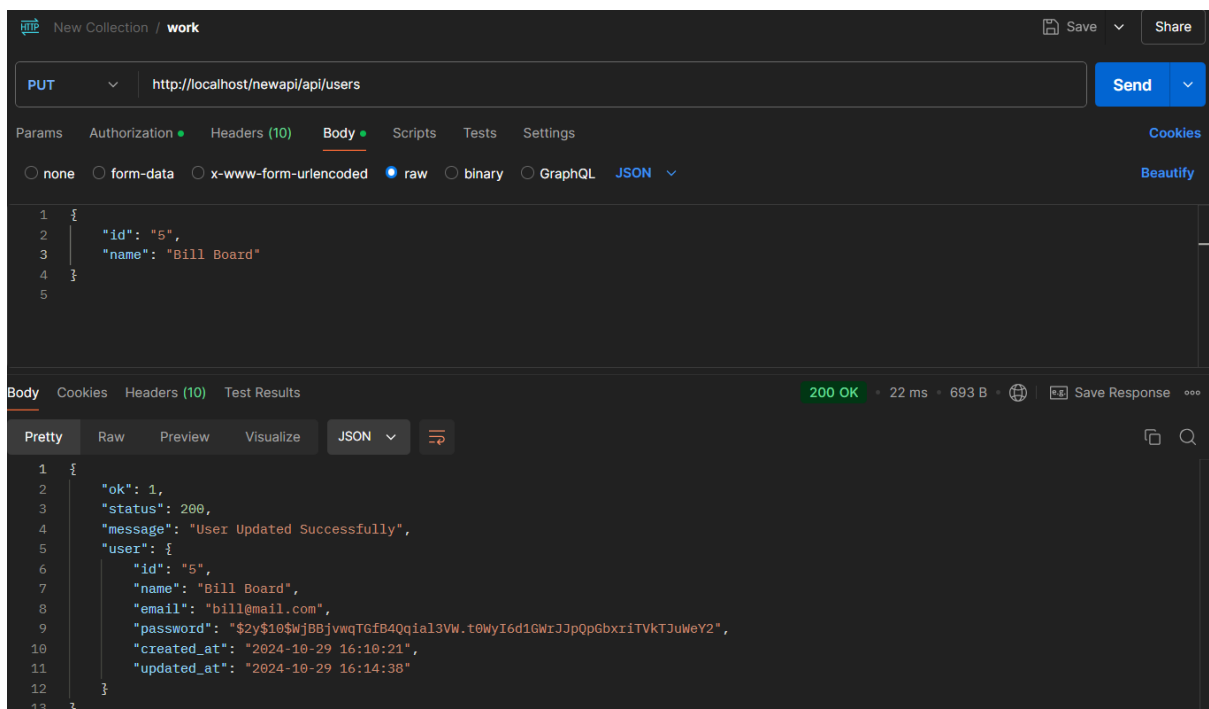


Figure 17

The record after updating user "id:5" in database:

| | | id | name | email | password | created_at | updated_at |
|--------------------------|------------------|----|------------|-----------------|--|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | shameem | nubee@gmail.com | \$2y\$10\$PMGp9zRtJZCVPXxakudZZuuSspgtDC1UfKKFfEOmkUc... | 2023-09-08 11:23:58 | 2023-09-08 11:23:58 |
| <input type="checkbox"/> | Edit Copy Delete | 3 | Wick | john@mail.com | \$2y\$10\$JCeQE/3aO2IZltw3UE22w.AISuE6/c/exKHnjt.R9t... | 2024-10-28 12:55:56 | 2024-10-28 12:57:19 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Ben | ben@mail.com | \$2y\$10\$HwdNT3mDqQPMUquN9jD8FuHMr0md3atk5WnowOCZy7k... | 2024-10-29 11:23:29 | 2024-10-29 11:23:29 |
| <input type="checkbox"/> | Edit Copy Delete | 5 | Bill Board | bill@mail.com | \$2y\$10\$WjBBjvwqTGfB4QqiaI3VW.t0Wyl6d1GWrJJpQpGbxri... | 2024-10-29 16:10:21 | 2024-10-29 16:14:38 |

Figure 18

3. Delete user:

The record before deleting a user in database:

| | | id | name | email | password | created_at | updated_at |
|--------------------------|------------------|----|------------|-----------------|---|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | shameem | nubee@gmail.com | \$2y\$10\$PMGp9zRtJZCVPXxakudZZuuSsgtDC1UfKKFfEOmkUc... | 2023-09-08 11:23:58 | 2023-09-08 11:23:58 |
| <input type="checkbox"/> | Edit Copy Delete | 3 | Wick | john@mail.com | \$2y\$10\$jCeqE/3aO2IZItw3UE22w.AISuE6/c/exKHnjt.R9t... | 2024-10-28 12:55:56 | 2024-10-28 12:57:19 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Ben | ben@mail.com | \$2y\$10\$HwdNT3mDqQPMUquN9jD8FuHMr0md3atk5WnowOCZy7k... | 2024-10-29 11:23:29 | 2024-10-29 11:23:29 |
| <input type="checkbox"/> | Edit Copy Delete | 5 | Bill Board | bill@mail.com | \$2y\$10\$WjBBjvwqTGfB4Qqial3VWw.t0Wyl6d1GWrJJpQpGbxri... | 2024-10-29 16:10:21 | 2024-10-29 16:14:38 |

Figure 19

Deleting a user using DELETE. It will require a valid id in order to delete a user. Once deleted, a respond message is prompted that states that it has been deleted successfully:

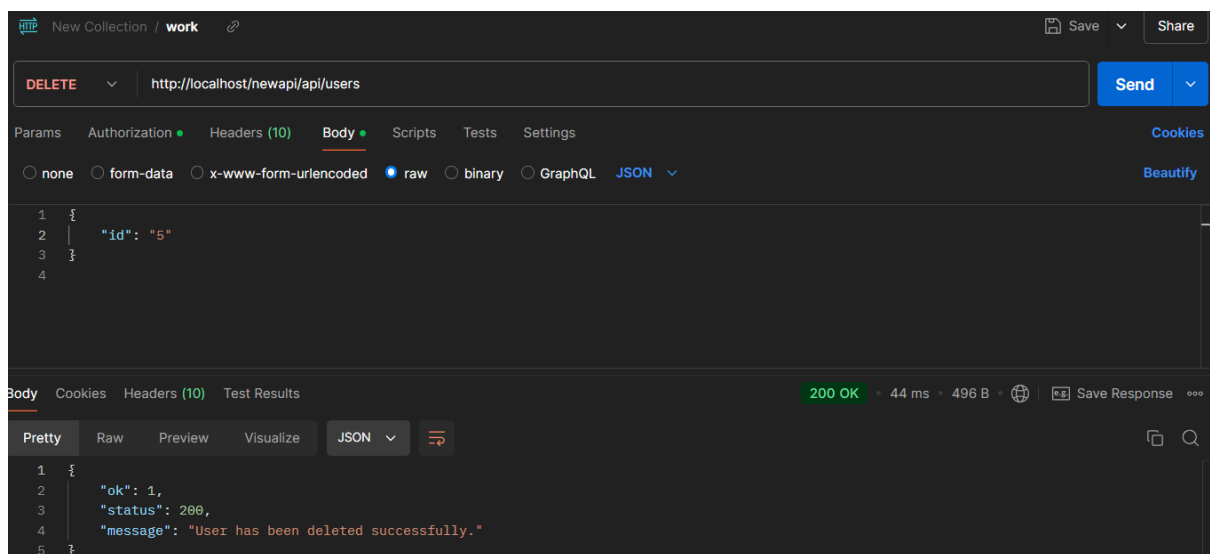


Figure 20

The record after deleting user “id:5” in database:

| | | id | name | email | password | created_at | updated_at |
|--------------------------|------------------|----|---------|-----------------|--|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | shameem | nubee@gmail.com | \$2y\$10\$PMGp9zRtJZCVPXxakudZZuuSsgtDC1UfKKFfEOmkUc... | 2023-09-08 11:23:58 | 2023-09-08 11:23:58 |
| <input type="checkbox"/> | Edit Copy Delete | 3 | Wick | john@mail.com | \$2y\$10\$jCeqE/3aO2IZItw3UE22w.AISuE6/c/exKHnjt.R9t... | 2024-10-28 12:55:56 | 2024-10-28 12:57:19 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Ben | ben@mail.com | \$2y\$10\$HwdNT3mDqQPMUquN9jD8FuHMr0md3atk5WnowOCZy7k... | 2024-10-29 11:23:29 | 2024-10-29 11:23:29 |

Figure 21

4. Get user:

Read all users using GET:

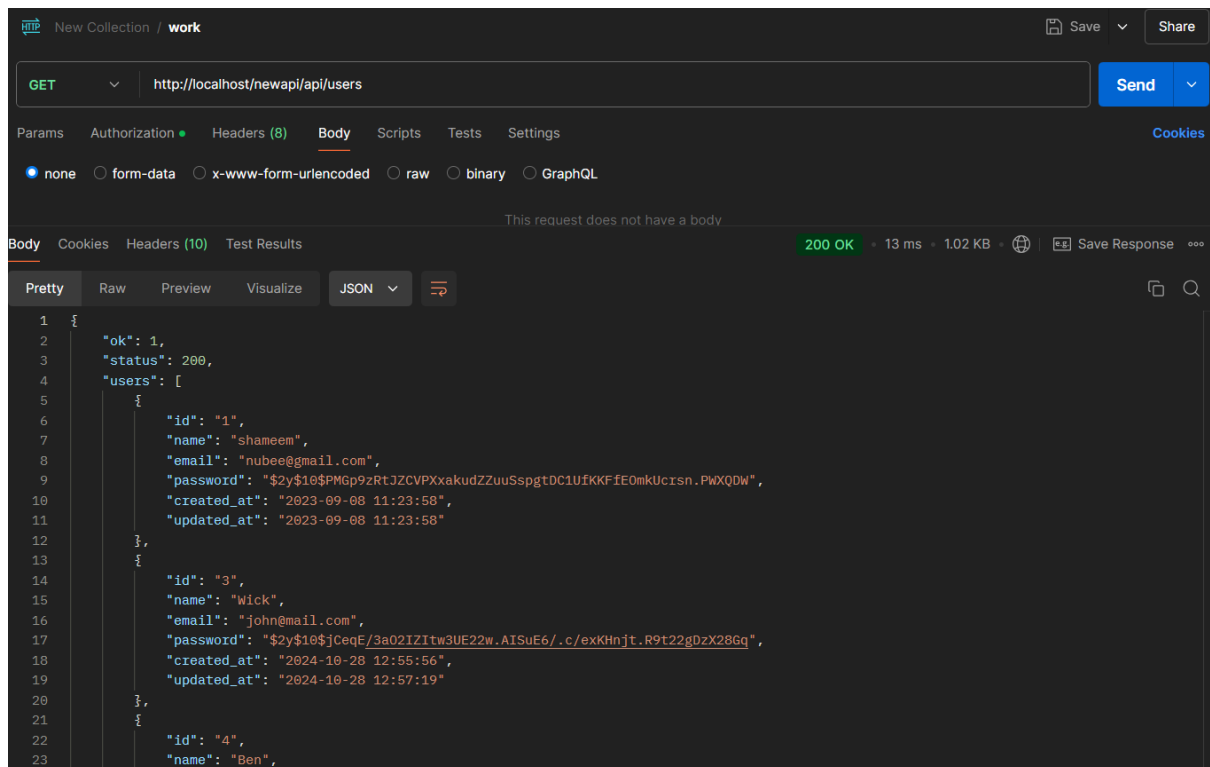


Figure 22

5. Login user:

A user can login using POST by providing an email and a password. Once validated, a respond message is prompted that states that it has been logged in successfully and a token is generated:

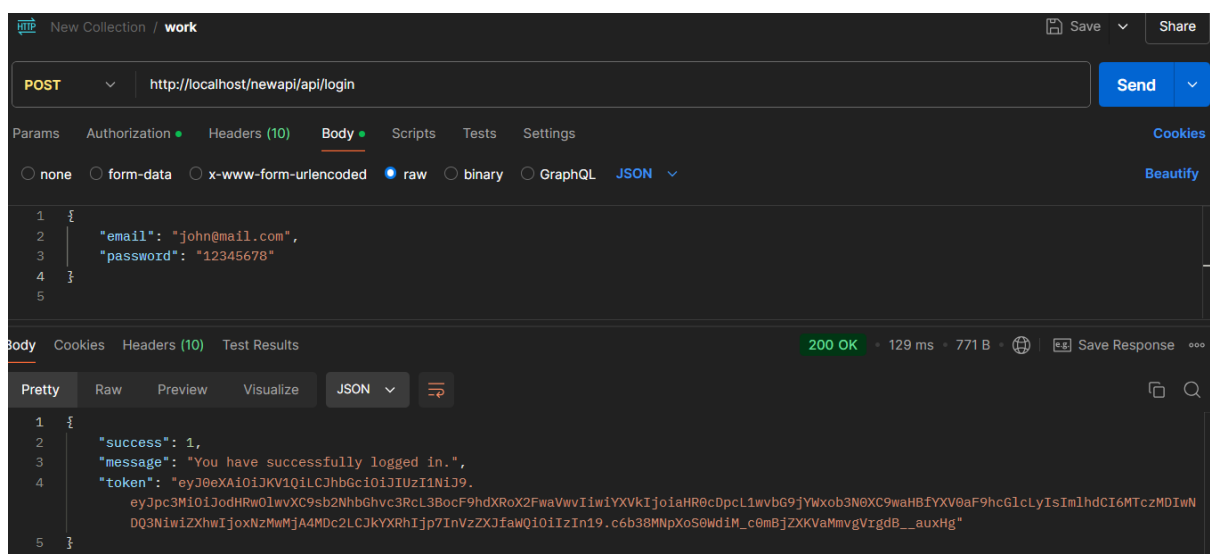


Figure 23

Authenticate user

When a user tries to use GET in login and provides a valid token, it will give status 200 ok and will provide the user's name and email:

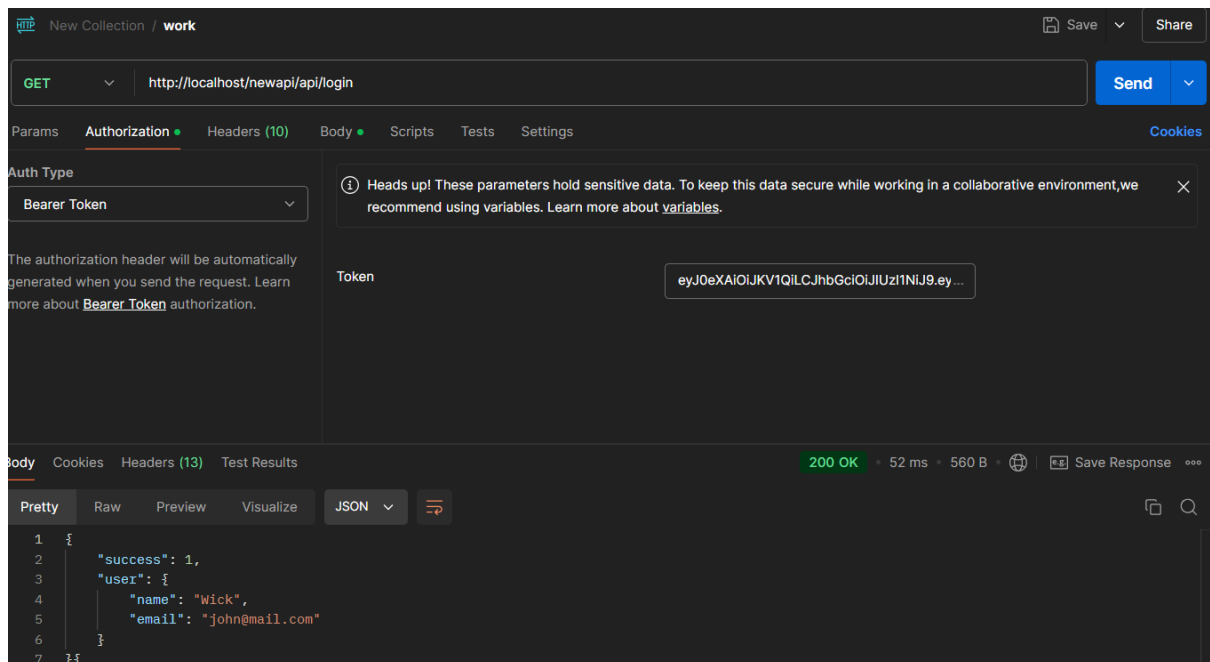


Figure 24

6. Blog:

Table blog in database “newapi”:



Figure 25

7. Create Blog:

A user can create a blog using POST by providing article_title, article_topic and article_info. Once validated, a respond message is prompted that states that the blog has been successfully created:

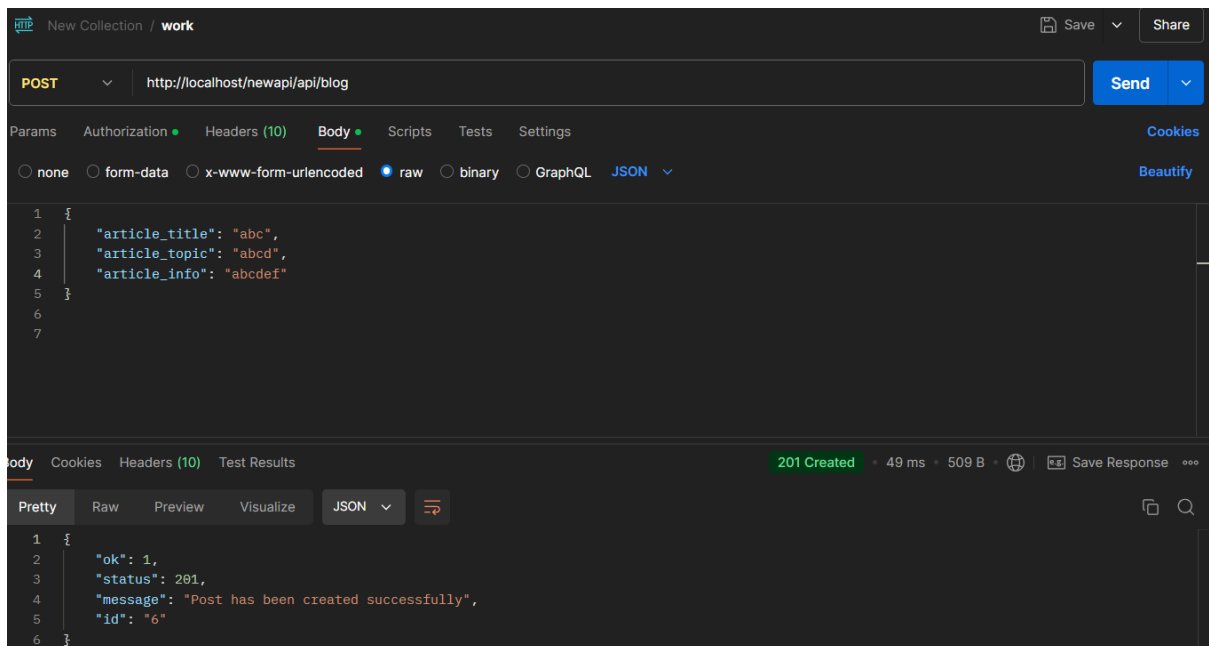


Figure 26

The record after creating a blog in database:

| | | id | article_title | article_topic | article_info | created_at | updated_at |
|--------------------------|------------------|----|---------------|---------------|---|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | Venom 3 | movie | I have watch venom the last dance and it was inter... | 2012-06-01 02:12:30 | 2023-09-07 15:07:00 |
| <input type="checkbox"/> | Edit Copy Delete | 2 | Damso Concert | Music | I've attended his concert and it was very cool. | 2024-10-28 17:35:52 | 2024-10-28 17:42:15 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Candide | Book | This literature book was amazing. | 2024-10-29 11:25:03 | 2024-10-29 11:26:43 |
| <input type="checkbox"/> | Edit Copy Delete | 6 | abc | abcd | abcdef | 2024-10-29 16:29:03 | 2024-10-29 16:29:03 |

Figure 27

8. Get Blog:

Read all blogs using GET with a valid token:

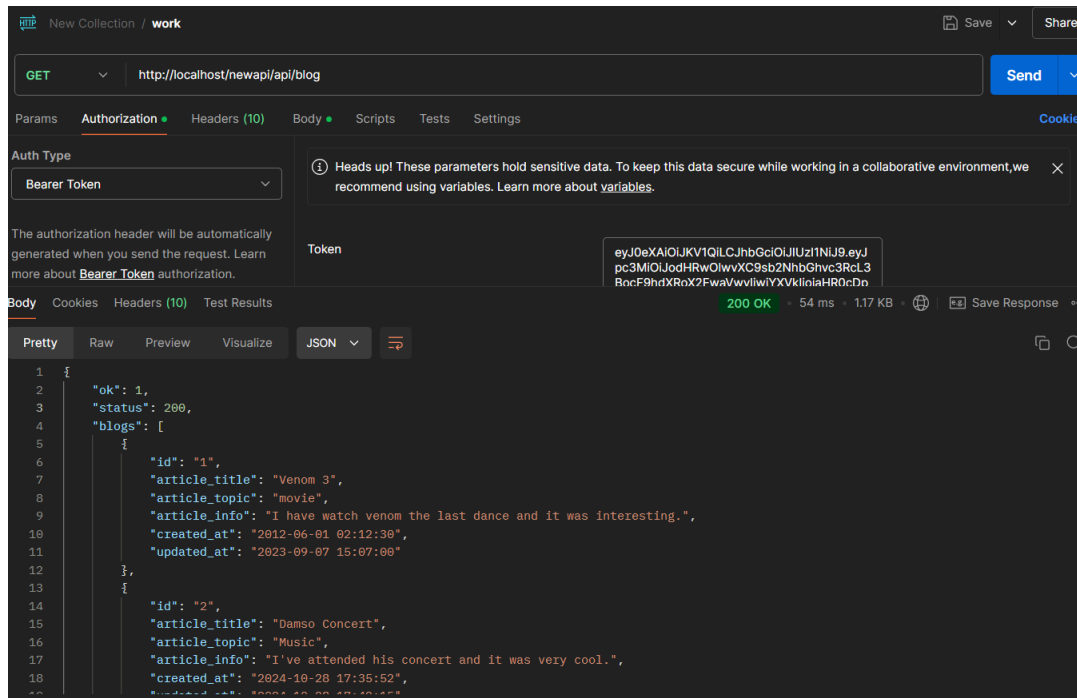


Figure 28

9. Update Blog:

The record before updating a blog in database:

| | | id | article_title | article_topic | article_info | created_at | updated_at |
|--------------------------|------------------|----|---------------|---------------|---|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | Venom 3 | movie | I have watch venom the last dance and it was inter... | 2012-06-01 02:12:30 | 2023-09-07 15:07:00 |
| <input type="checkbox"/> | Edit Copy Delete | 2 | Damso Concert | Music | I've attended his concert and it was very cool. | 2024-10-28 17:35:52 | 2024-10-28 17:42:15 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Candide | Book | This literature book was amazing. | 2024-10-29 11:25:03 | 2024-10-29 11:26:43 |
| <input type="checkbox"/> | Edit Copy Delete | 6 | abc | abcd | abcdef | 2024-10-29 16:29:03 | 2024-10-29 16:29:03 |

Figure 29

Updating a blog using PUT. It will require a valid id in order to update any blog details. Once updated, a respond message is prompted that states that it has been updated successfully:

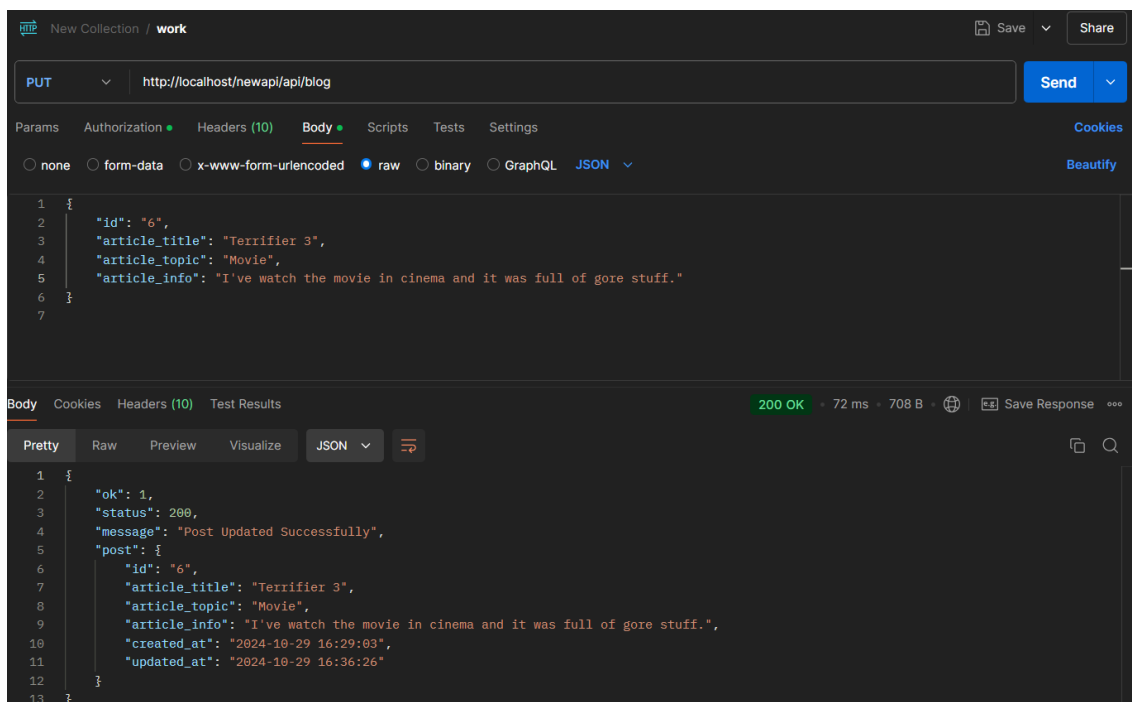


Figure 30

The record after updating a blog in database:

| | | id | article_title | article_topic | article_info | created_at | updated_at |
|--------------------------|------------------|----|---------------|---------------|---|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | Venom 3 | movie | I have watch venom the last dance and it was inter... | 2012-06-01 02:12:30 | 2023-09-07 15:07:00 |
| <input type="checkbox"/> | Edit Copy Delete | 2 | Damso Concert | Music | I've attended his concert and it was very cool. | 2024-10-28 17:35:52 | 2024-10-28 17:42:15 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Candide | Book | This literature book was amazing. | 2024-10-29 11:25:03 | 2024-10-29 11:26:43 |
| <input type="checkbox"/> | Edit Copy Delete | 6 | Terrifier 3 | Movie | I've watch the movie in cinema and it was full of ... | 2024-10-29 16:29:03 | 2024-10-29 16:36:26 |

Figure 31

10. Delete Blog:

The record before deleting a blog in database:

| | | id | article_title | article_topic | article_info | created_at | updated_at |
|--------------------------|------------------|----|---------------|---------------|---|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | Venom 3 | movie | I have watch venom the last dance and it was inter... | 2012-06-01 02:12:30 | 2023-09-07 15:07:00 |
| <input type="checkbox"/> | Edit Copy Delete | 2 | Damso Concert | Music | I've attended his concert and it was very cool. | 2024-10-28 17:35:52 | 2024-10-28 17:42:15 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Candide | Book | This literature book was amazing. | 2024-10-29 11:25:03 | 2024-10-29 11:26:43 |
| <input type="checkbox"/> | Edit Copy Delete | 6 | Terrifier 3 | Movie | I've watch the movie in cinema and it was full of ... | 2024-10-29 16:29:03 | 2024-10-29 16:36:26 |

Figure 32

Deleting a product using DELETE. It will require a valid id in order to delete a product. Once deleted, a respond message is prompted that states that it has been deleted successfully:

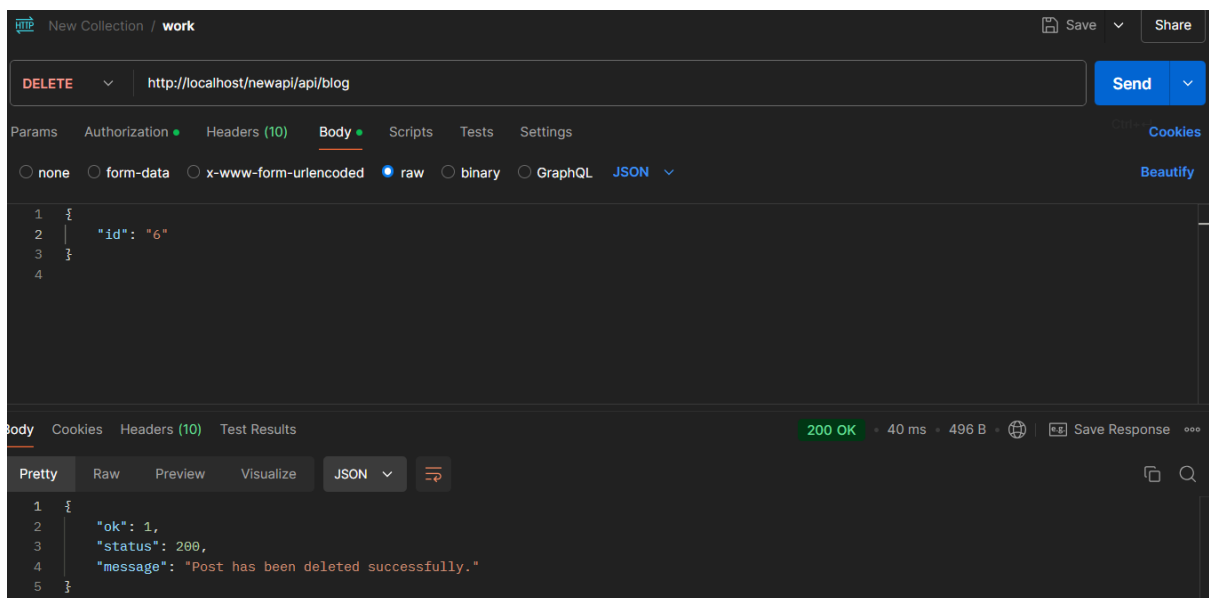


Figure 33

The record after deleting a blog in database:

| | | id | article_title | article_topic | article_info | created_at | updated_at |
|--------------------------|------------------|----|---------------|---------------|---|---------------------|---------------------|
| <input type="checkbox"/> | Edit Copy Delete | 1 | Venom 3 | movie | I have watch venom the last dance and it was inter... | 2012-06-01 02:12:30 | 2023-09-07 15:07:00 |
| <input type="checkbox"/> | Edit Copy Delete | 2 | Damso Concert | Music | I've attended his concert and it was very cool. | 2024-10-28 17:35:52 | 2024-10-28 17:42:15 |
| <input type="checkbox"/> | Edit Copy Delete | 4 | Candide | Book | This literature book was amazing. | 2024-10-29 11:25:03 | 2024-10-29 11:26:43 |

Figure 34

ERROR HANDLING AND SECURITY

Effective error handling enhances user experience and simplifies issue resolution, while robust security practices ensure the confidentiality, integrity, and availability of user data and the application itself. Combined, these measures contribute to a safer and more reliable e-commerce platform.

1. In users

Testing with empty data in database using GET:

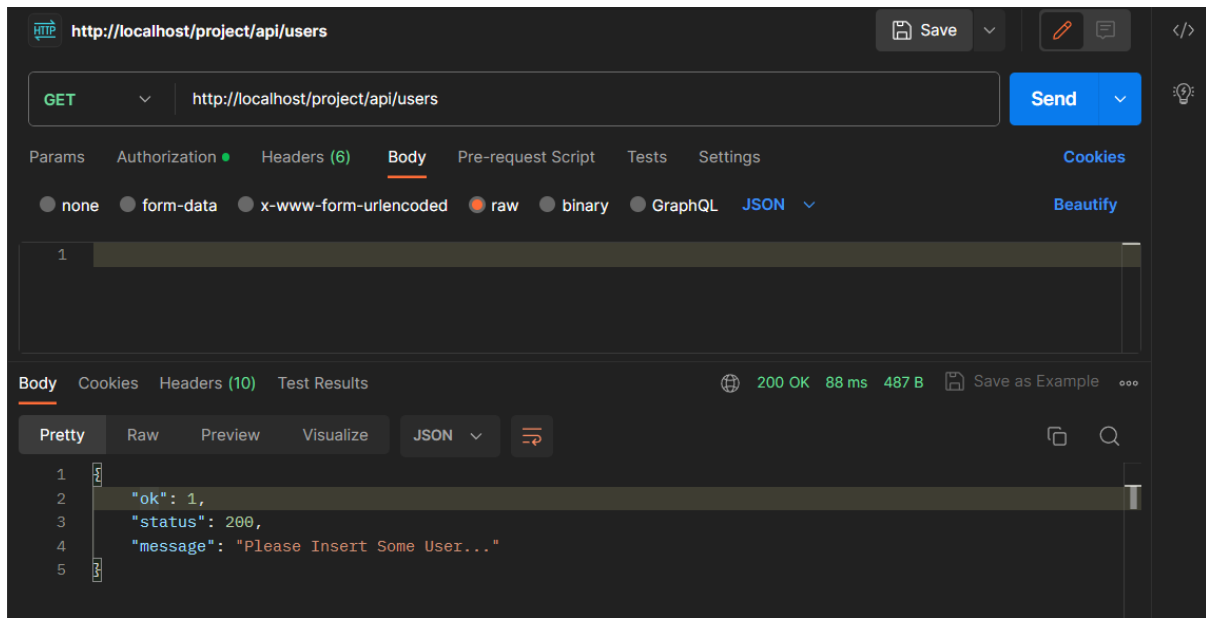


Figure 35

Testing with empty fields in body tab using POST:

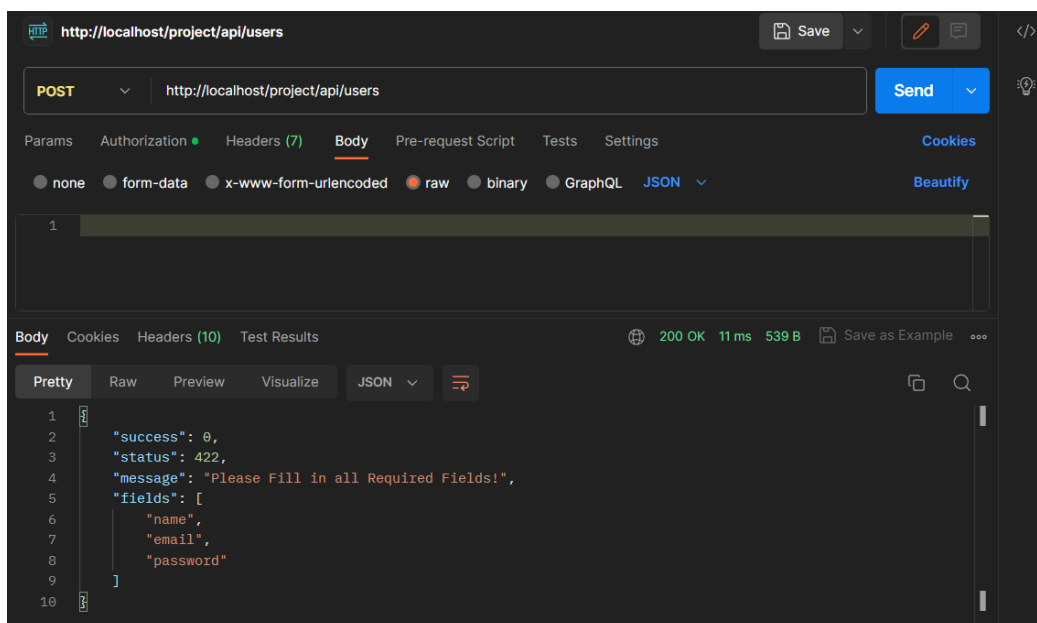


Figure 36

Testing with empty fields using PUT:

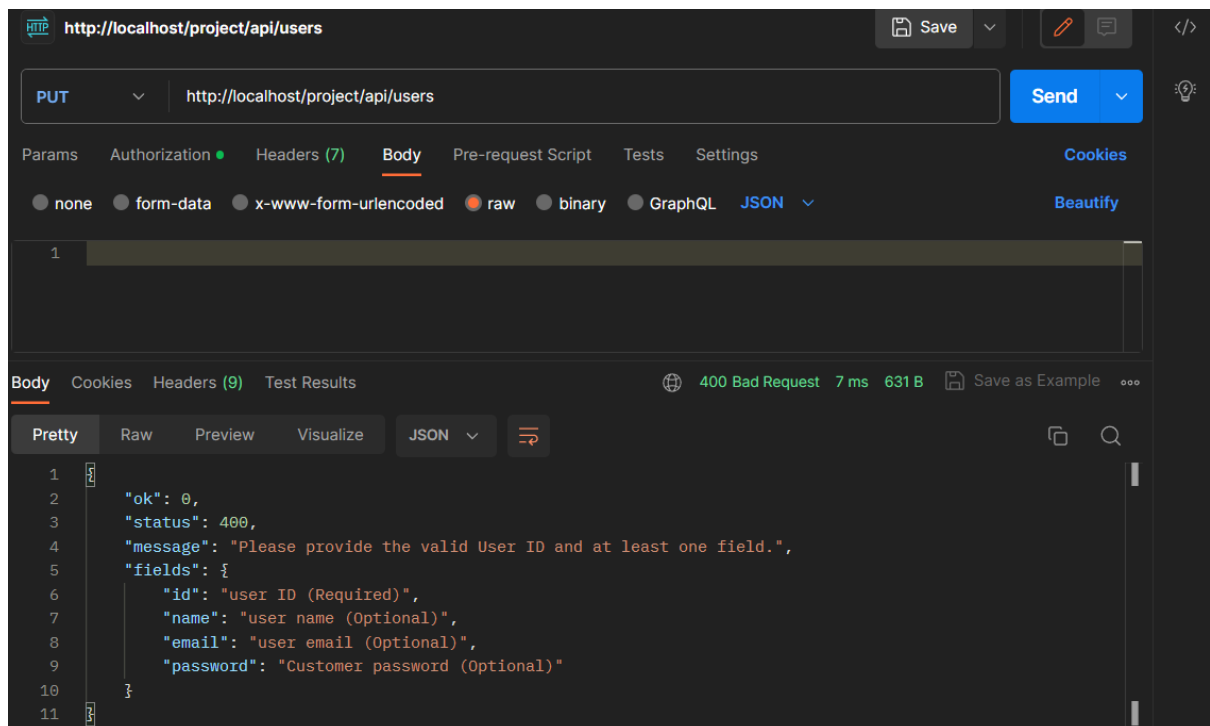


Figure 37

Testing with empty fields using DELETE:

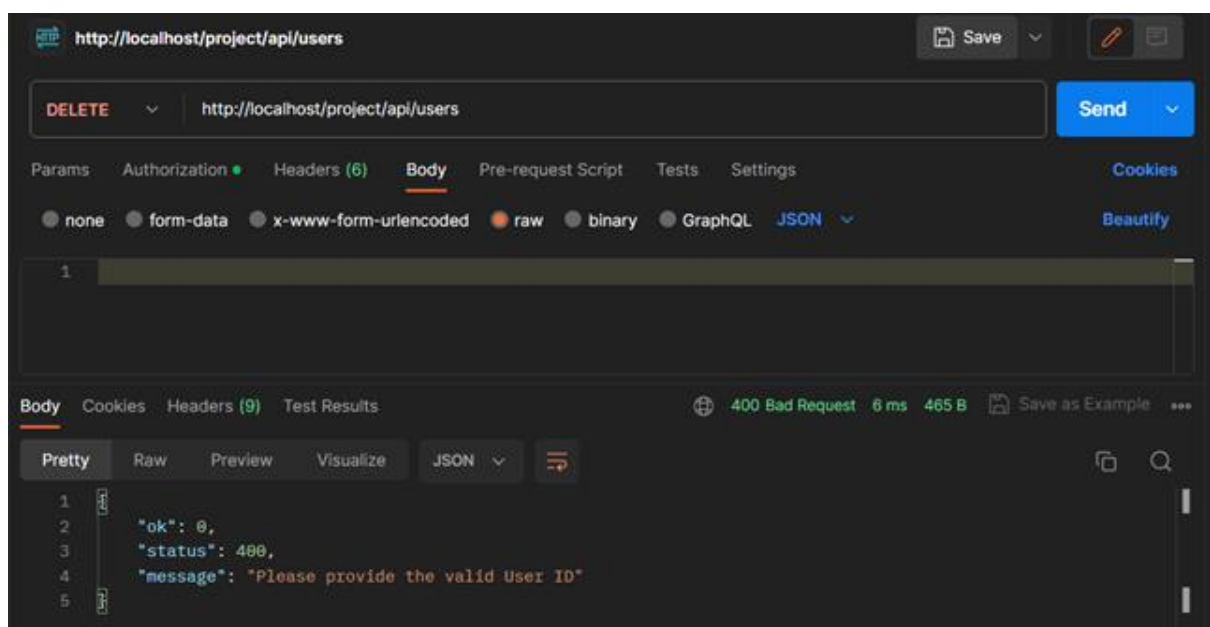


Figure 38

Table **users** in database “newapi”:

| | | id | name | email | password | created_at | updated_at |
|--------------------------|------|----|---------|-----------------|--|---------------------|---------------------|
| <input type="checkbox"/> | Edit | 1 | shameem | nubee@gmail.com | \$2y\$10\$PMGp9zRtJZCVPXxakudZZuuSspgIDC1UfKKFfEOmkUc... | 2023-09-08 11:23:58 | 2023-09-08 11:23:58 |
| <input type="checkbox"/> | Edit | 3 | Wick | john@mail.com | \$2y\$10\$jCeqE/3aO2lZtw3UE22w.AISuE6/c/exKHnjt.R9t... | 2024-10-28 12:55:56 | 2024-10-28 12:57:19 |
| <input type="checkbox"/> | Edit | 4 | Ben | ben@mail.com | \$2y\$10\$HwdNT3mDqQPMUquN9jD8FuHMr0md3atk5WnowOCZy7k... | 2024-10-29 11:23:29 | 2024-10-29 11:23:29 |

Figure 39

When creating a user with an email that already exists. It will verify the email ID and will not allow duplicated email ID. When this occur it will prompt a message that states that the E-mail is already used.

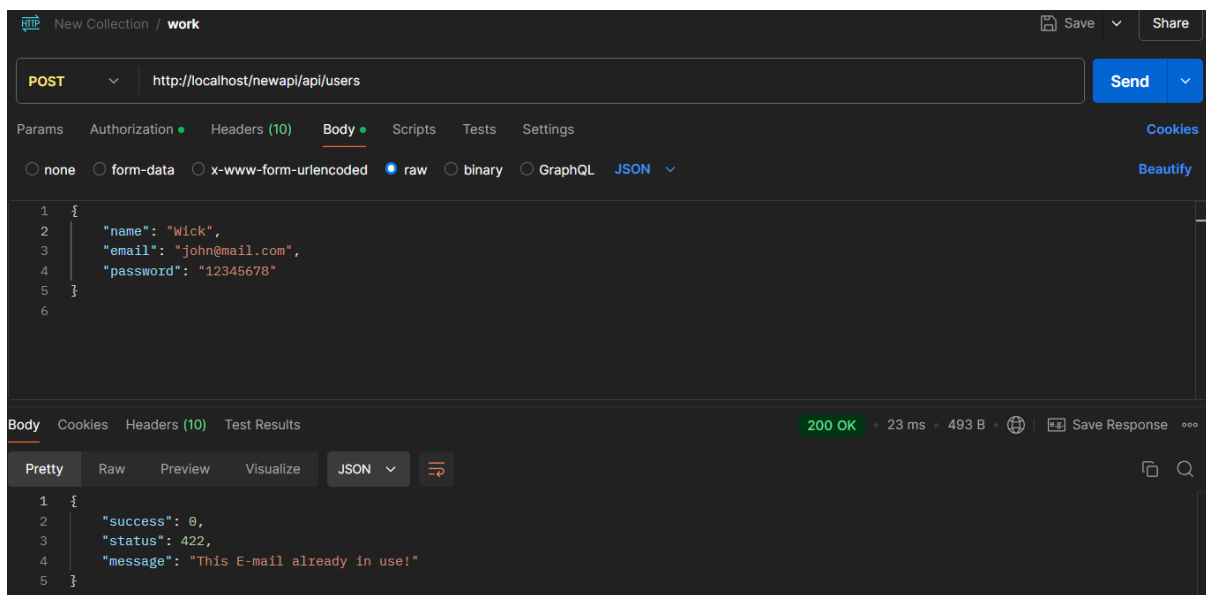


Figure 40

When deleting a user that does not exist, it will prompt a message that states that the user id is invalid.

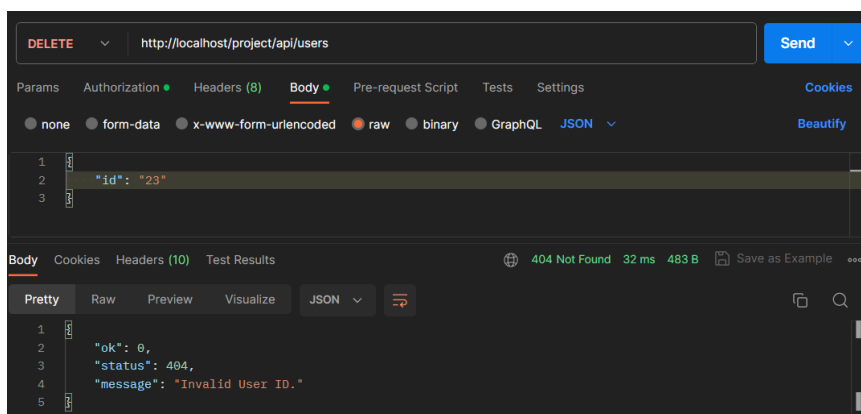


Figure 41

When reading a user that does not exist, it will prompt a message that states that the user id is not found:

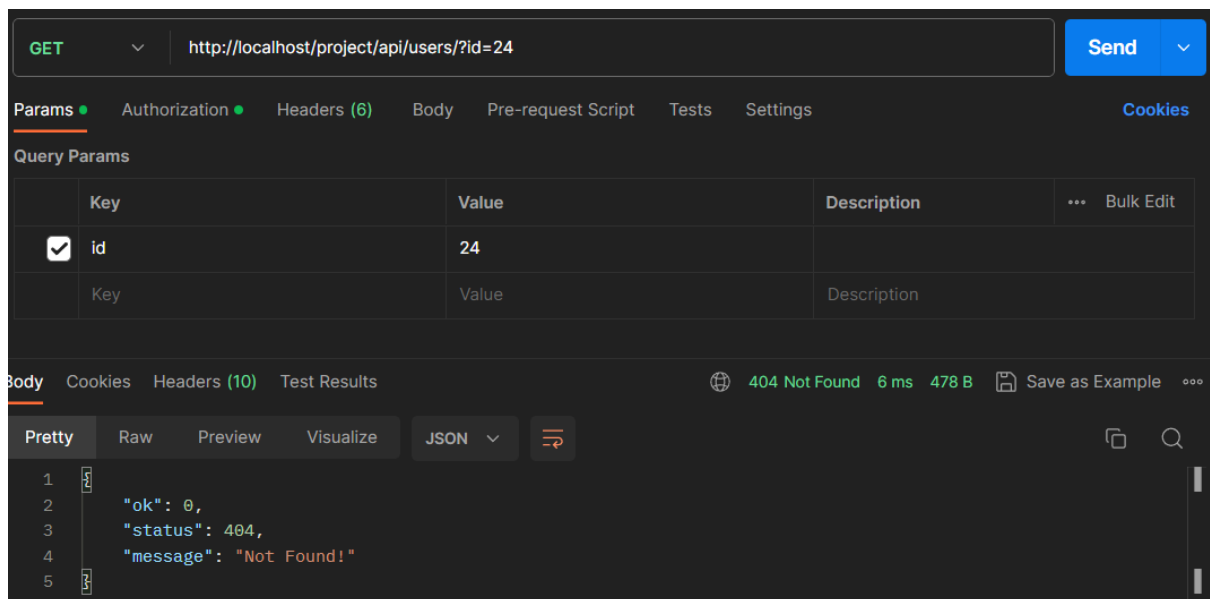


Figure 42

2. In login

Testing with empty data using POST:

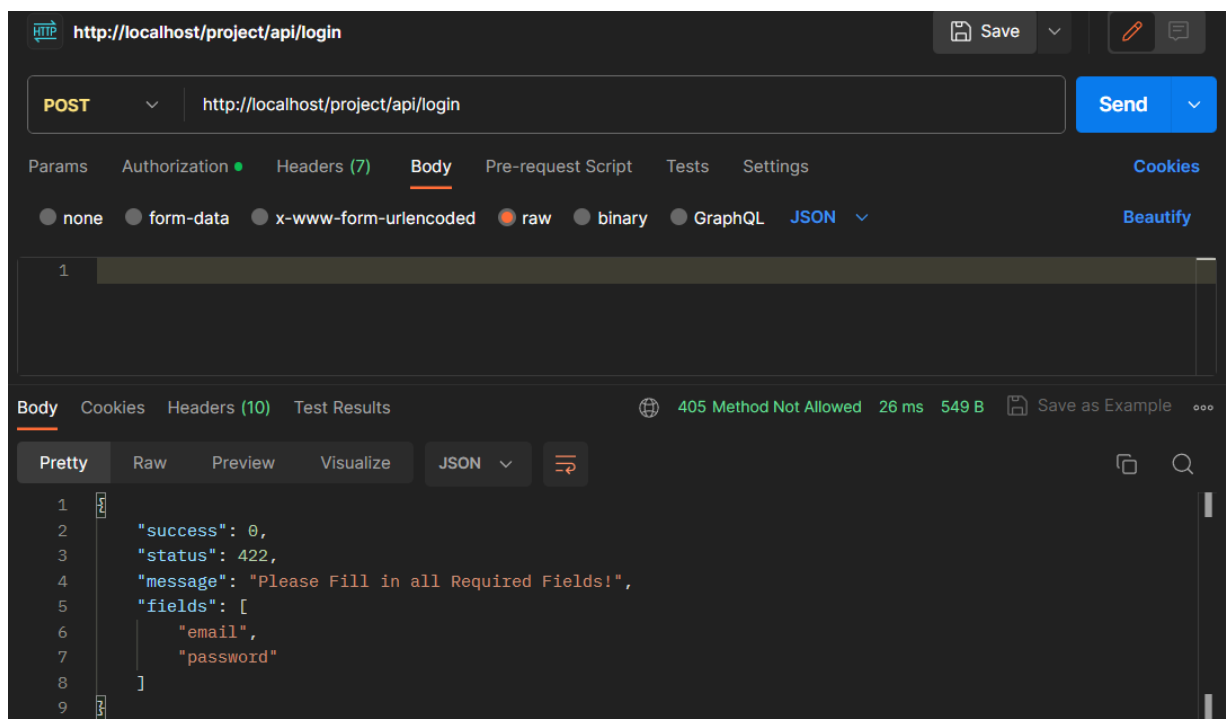


Figure 43

When a user tries to login but it has inserted a wrong password, it will prompt a message that states that the password is invalid.

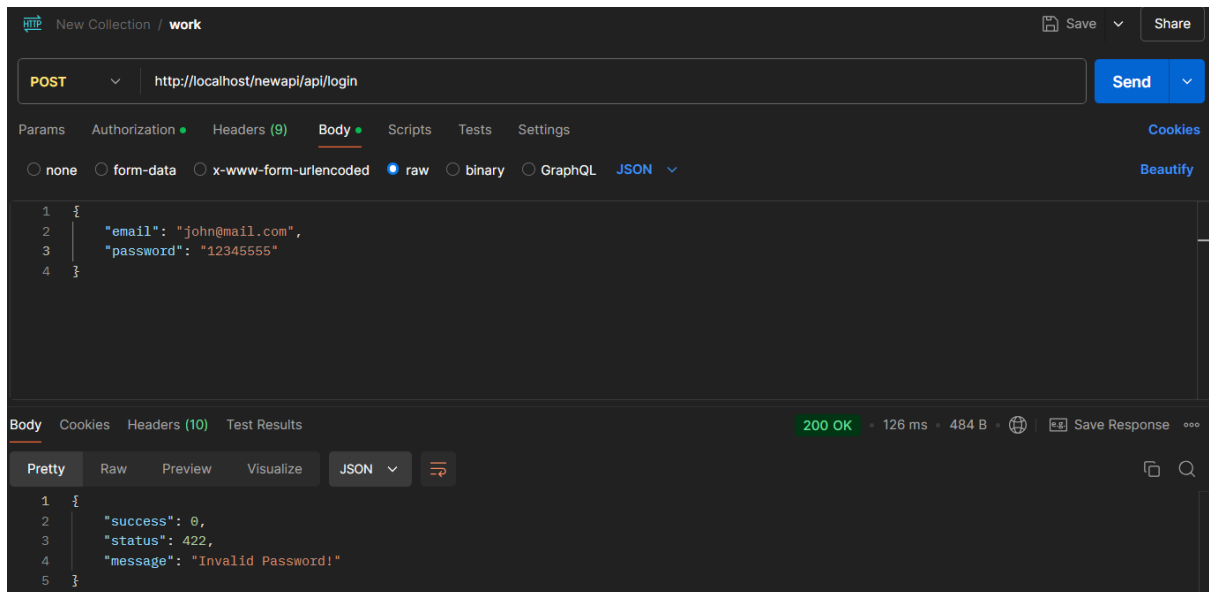


Figure 44

When a user tries to login but it has inserted a wrong email, it will prompt a message that states that the email is invalid.

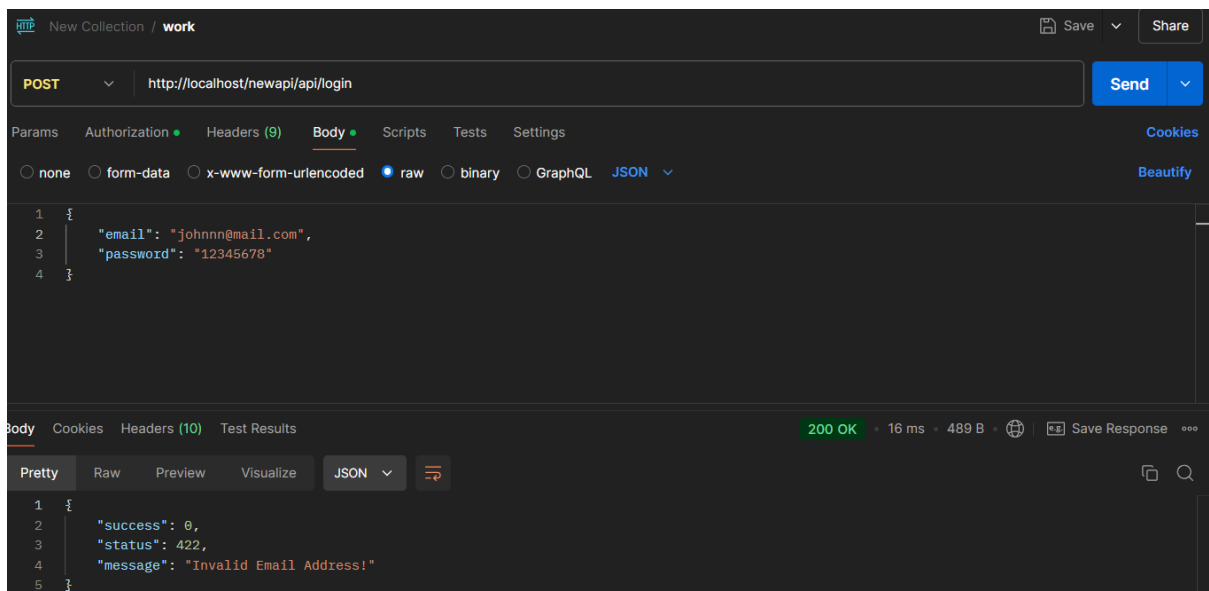


Figure 45

When a user tries to use GET in login but with empty data in token, it will prompt a message that states that the token is not found in request.

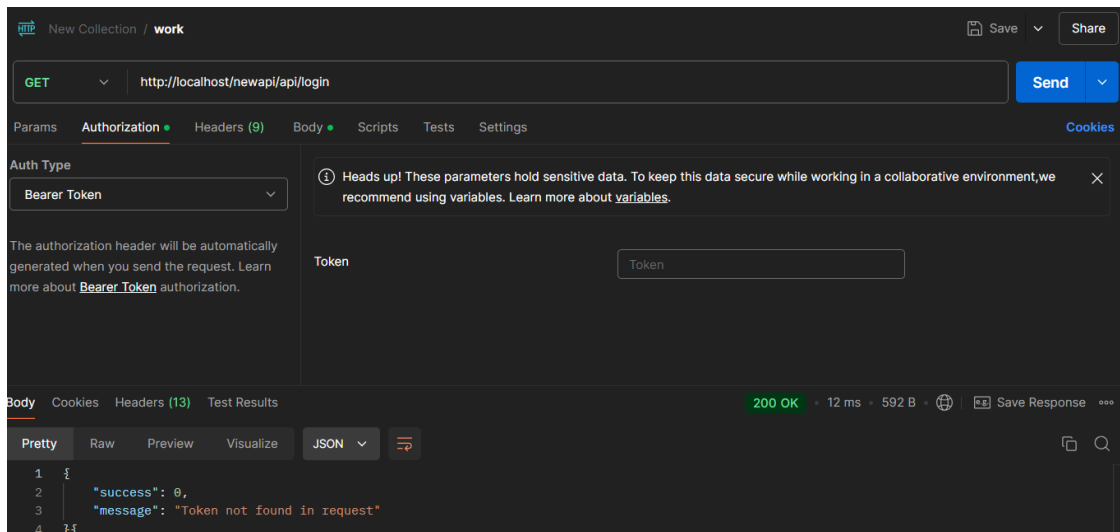


Figure 46

When a user tries to use GET in login but has inserted an invalid token, it will prompt a message that states that the signature verification has failed.



Figure 47

If the same token is inserted after one hour, it will prompt a message that states that the token has expired. Then it must be recreated to get a new valid token.

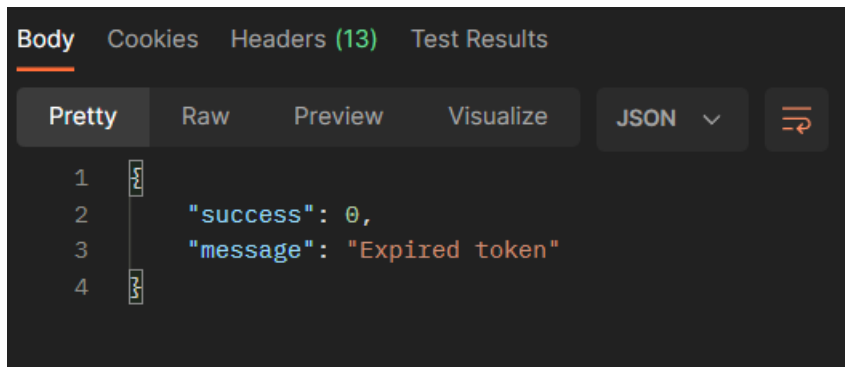


Figure 48

3. In Blog

Trying to use the token generated when a user log in. Only the token generated for users can be used else signature is invalid:

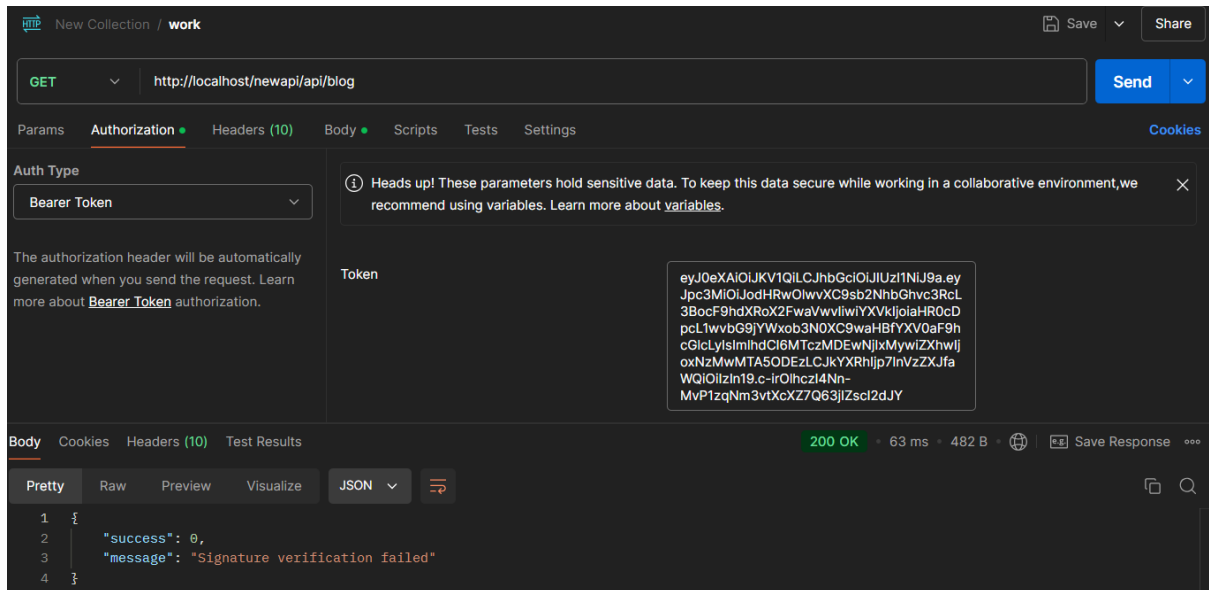


Figure 49

To perform any CRUD (GET, PUT, DELETE, POST) operation for blog it requires the token which will be generated when user logs in (only users) and is valid only for one hour:

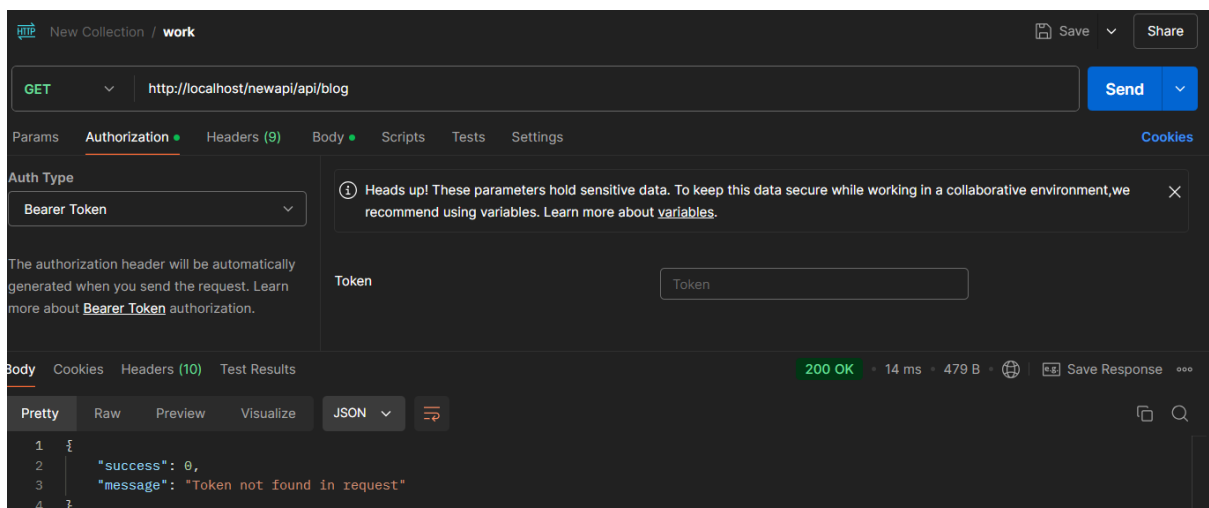


Figure 50 GET

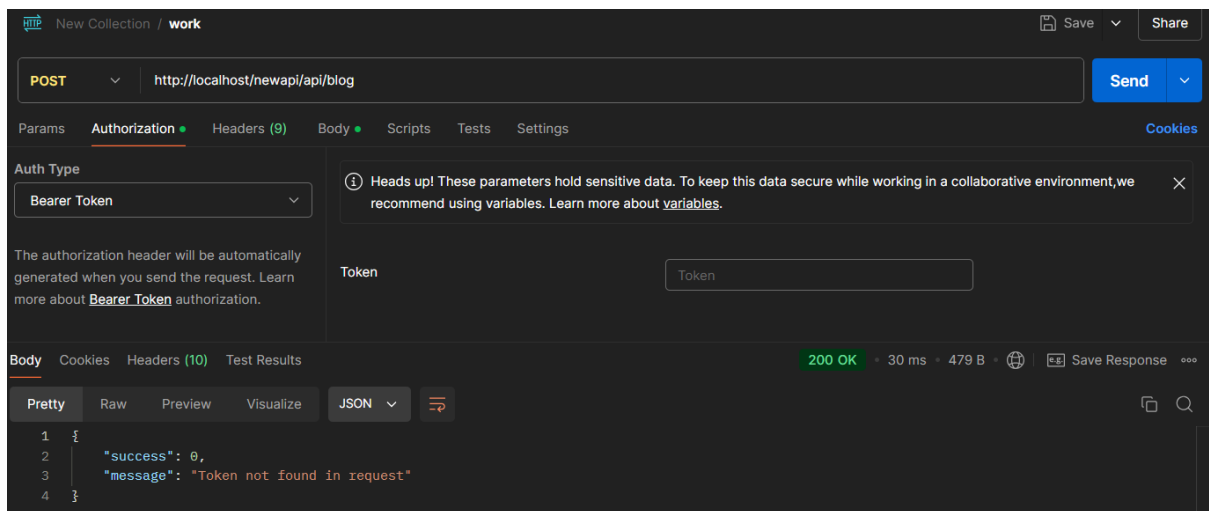


Figure 51 POST

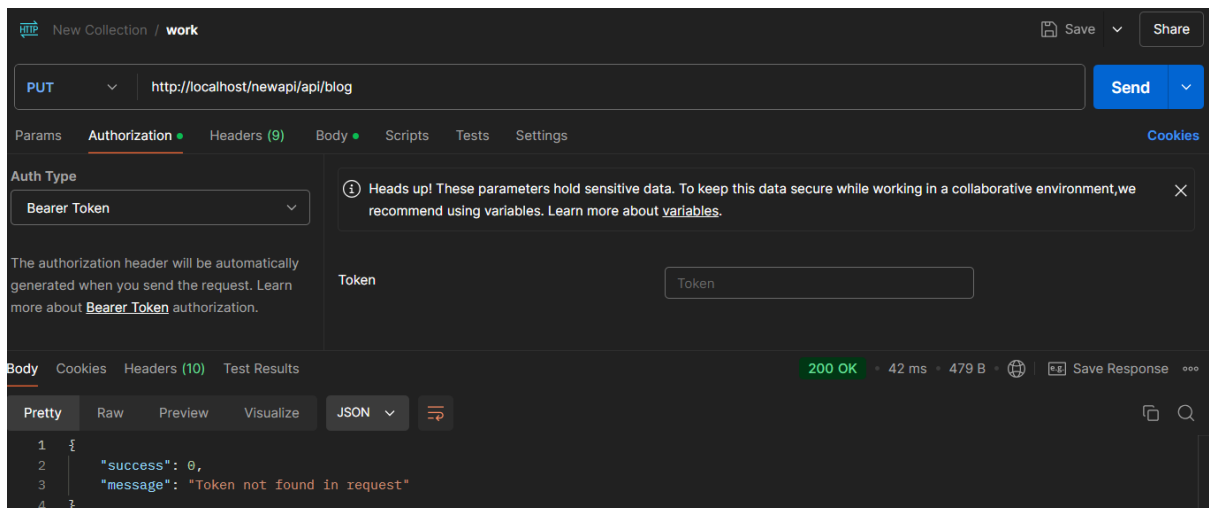


Figure 52 PUT

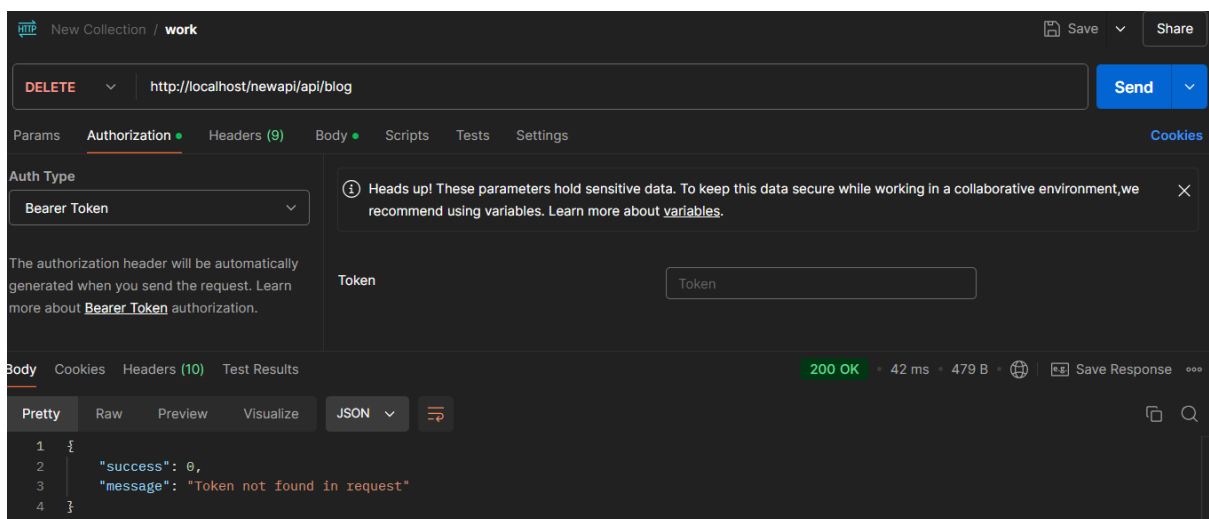


Figure 53 DELETE

Testing with empty fields using GET:

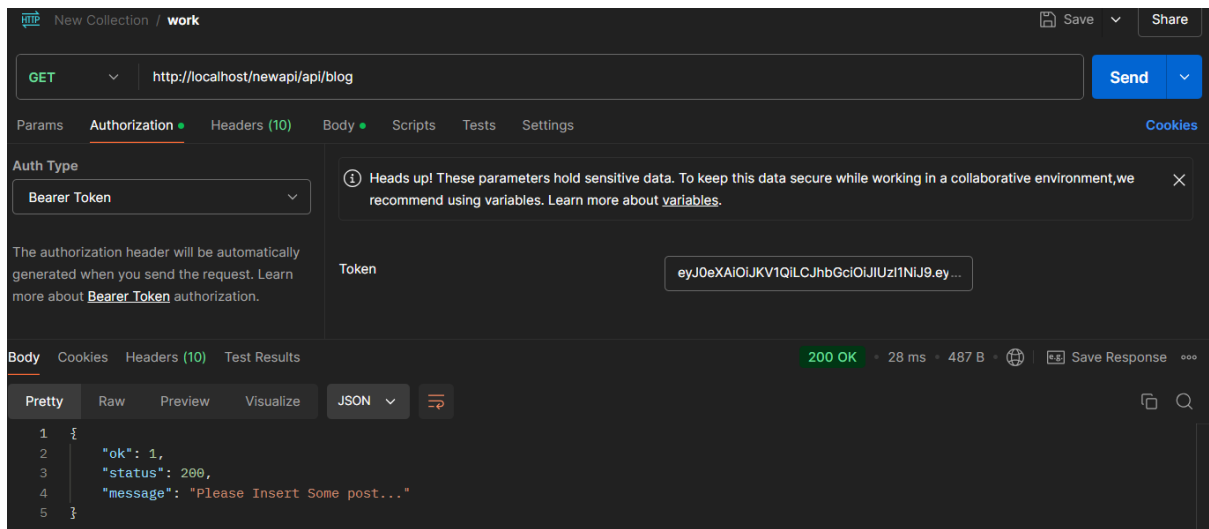


Figure 54

Testing with empty fields using POST:

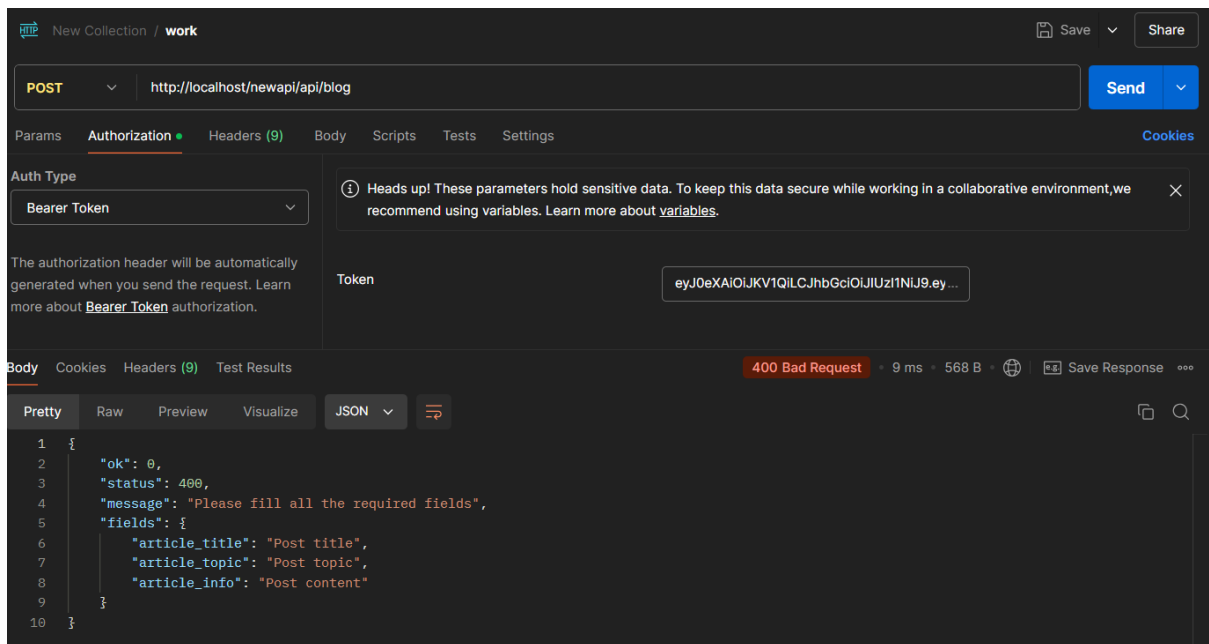


Figure 55

Testing with empty fields using PUT:

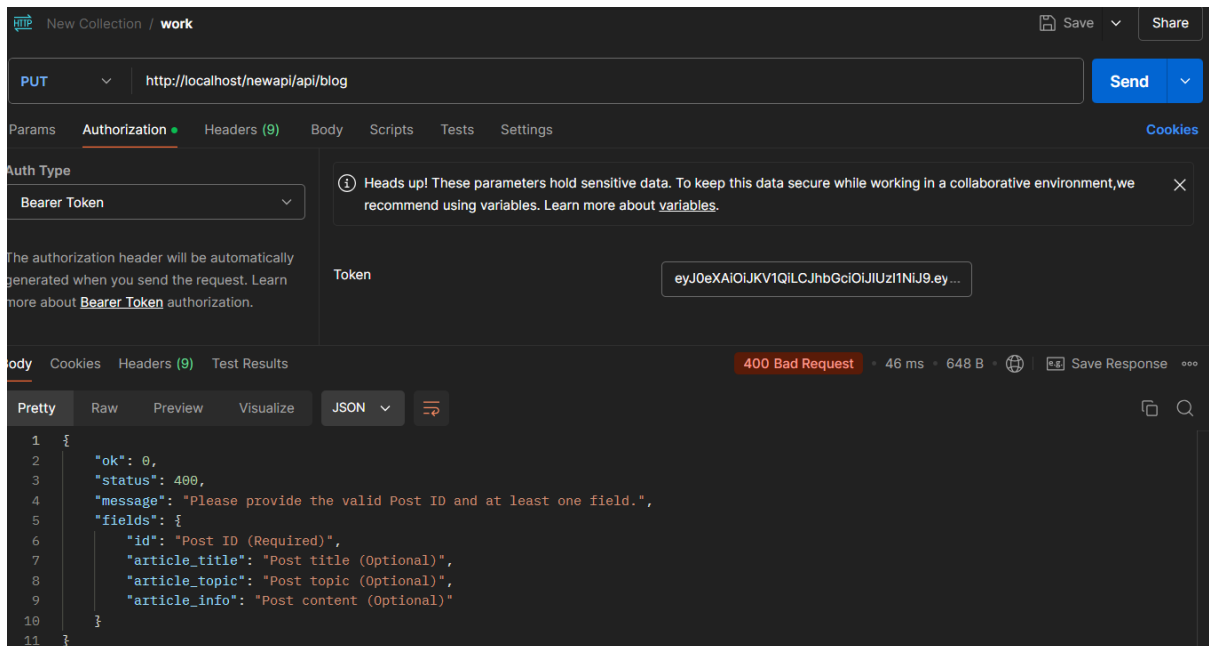


Figure 56

Testing with empty fields using DELETE:

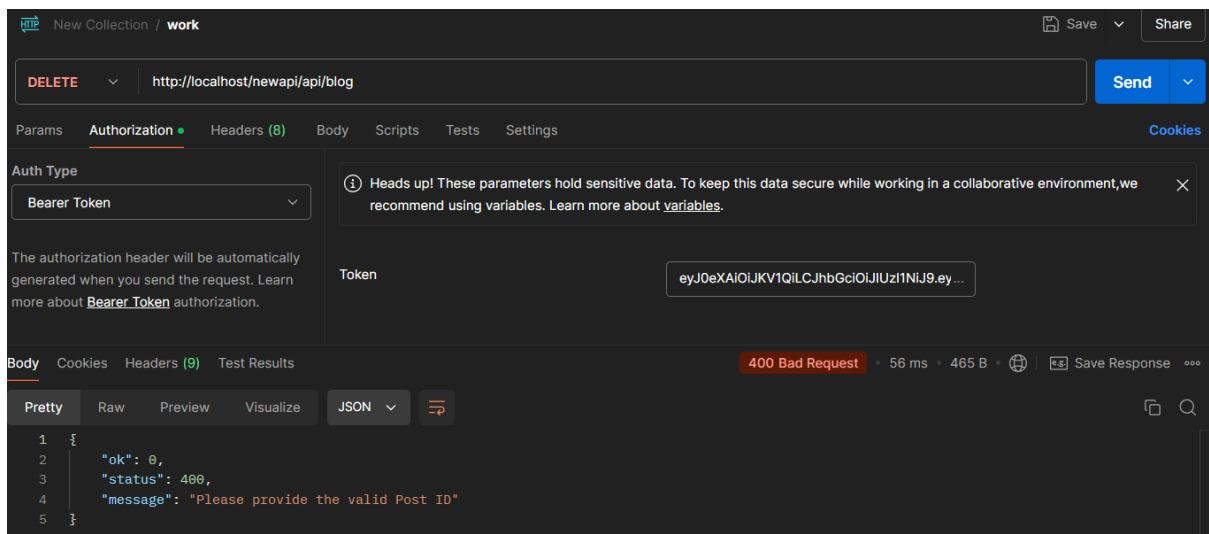


Figure 57

When reading a post that does not exist, it will prompt a message that states that the post is not found:

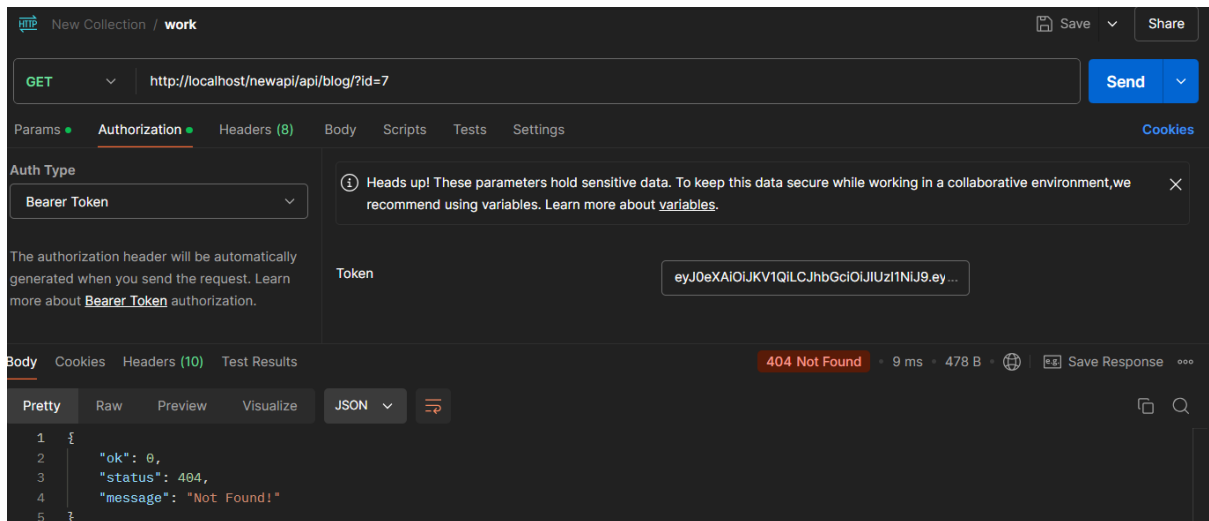


Figure 58

When using PUT to update a post, if the token is invalid, it will prompt a message that states that the token is not found in request:

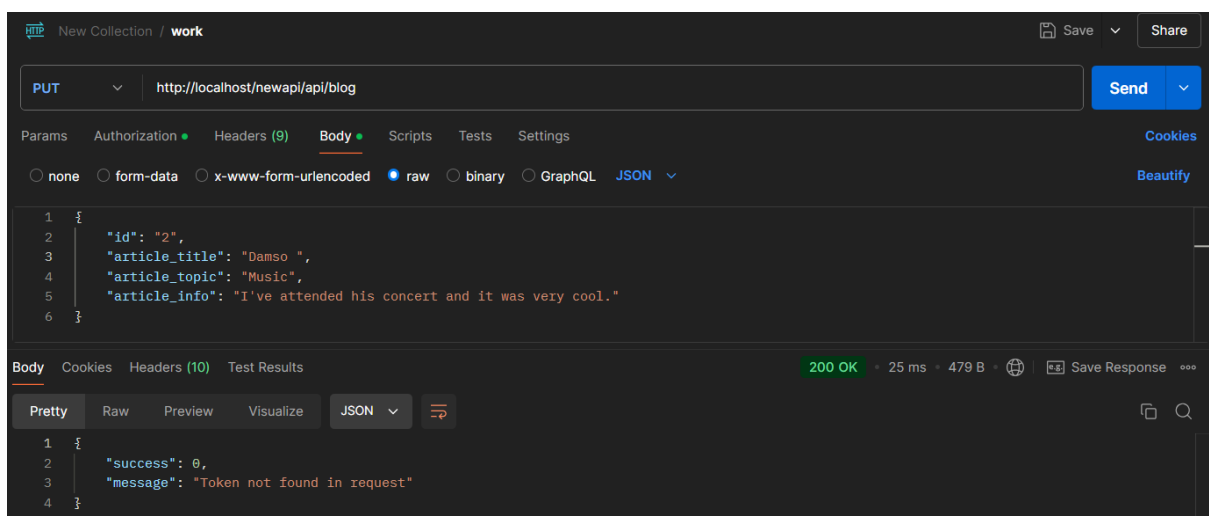


Figure 59

When deleting a post that does not exist, it will prompt a message that states that the post id is invalid.

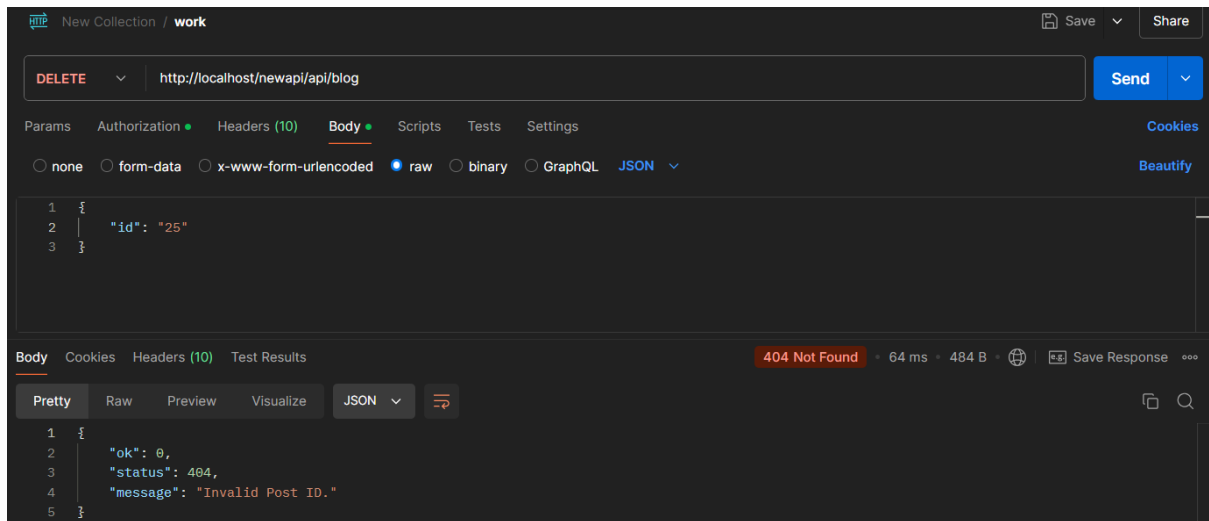


Figure 60