Test Document

for

Echo

Version 1.1

Prepared by

Group :18		Group Name: BitByBit
Ansh Adarsh	230157	ansha23@iitk.ac.in
Aryan Kumar	230215	aryank23@iitk.ac.in
Durbasmriti Saha	230393	durbasmrit23@iitk.ac.in
Gone Nishanth	230421	gnishanth23@iitk.ac.in
Govind Nayak Jarabala	230497	govindnj23@iitk.ac.in
Harsh Bhati	200408	harshb20@iitk.ac.in
Lavish Kanwa	230602	lavishk23@iitk.ac.in
Lokesh Kumar	230606	lokeshk23@iitk.ac.in
Someshwar Singh	231020	someshwars23@iitk.ac.in

Course:	CS253
Mentor TA:	Paras Ghodeshwar
Date:	28/03/2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
v1.1	All Group Members		06/04/2025

1 Introduction

Testing Strategy: -

We used manual testing to test our software.

Testing Period: -

The testing was mostly done after the implementation period. However, even during the implementation phase we did some manual testing of our code.

The Testers: -

Everyone in the team participated in testing. But most of the testing was done by few team members.

Coverage Criteria: -

We have mostly used Decision coverage (type of functional coverage) for testing.

Tools used for testing: -

We have used Postman to test our software. Postman is a popular API testing tool that lets its users to test and debug HTTP requests to APIs.

Postman has some key advantages which includes:

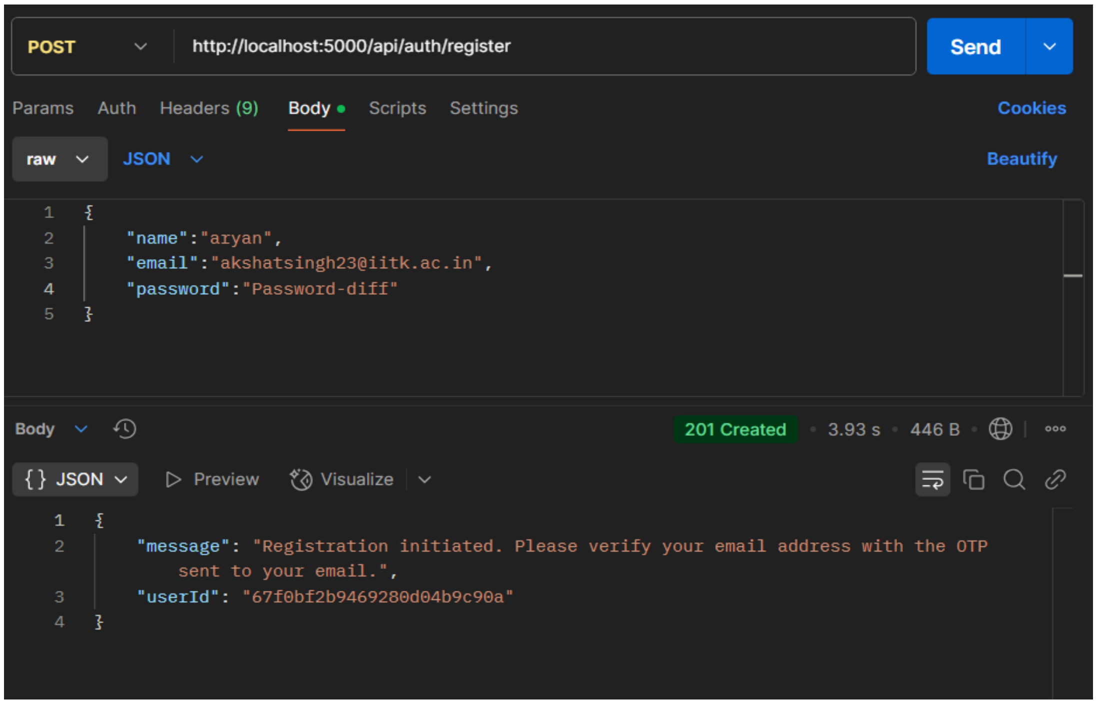
1. Easy to use.
2. Quick Responses.
3. User friendly Interface.
4. Free plans.
5. Robust feature sets.

2 Unit Testing

1. Sign-Up / Register

a. Register: - Generating the message and OTP

Unit Details: Check the functioning of signup/register
Test Owner: Aryan Kumar
Test Date: 04/01/2025
Test Results: User can send his details to receive OTP.
Structural Coverage: (Decision Coverage) We covered cases where user already exists or where user's email doesn't end with ".iitk.ac.in".

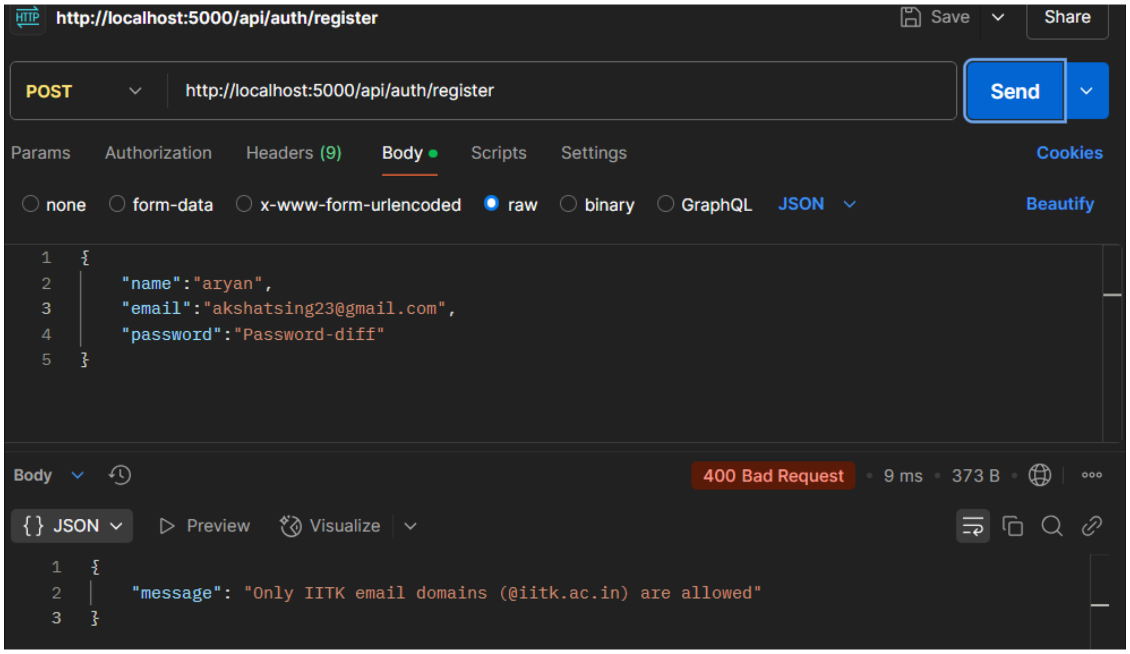


```
http://localhost:5000/api/auth/register
```

```
{
    "name":"aryan",
    "email":"akshatsingh23@iitk.ac.in",
    "password":"Password-diff"
}
```

```
{
    "message": "Registration initiated. Please verify your email address with the OTP
        sent to your email.",
    "userId": "67f0bf2b9469280d04b9c90a"
}
```

If user's email doesn't end with iitk.ac.in then the server sends and error.

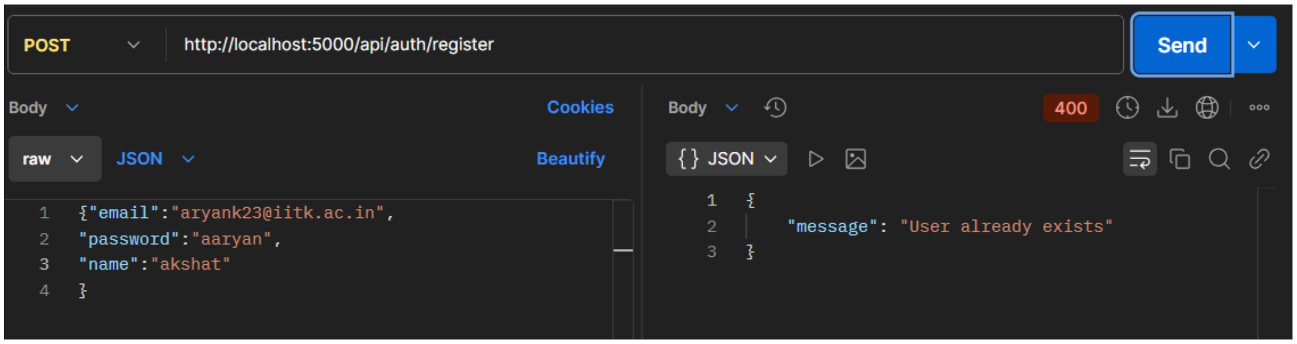


POST http://localhost:5000/api/auth/register

```
{
    "name":"aryan",
    "email":"akshatsing23@gmail.com",
    "password":"Password-diff"
}
```

```
{
    "message": "Only IITK email domains (@iitk.ac.in) are allowed"
}
```

If user's account already exists, then the server shows an error.



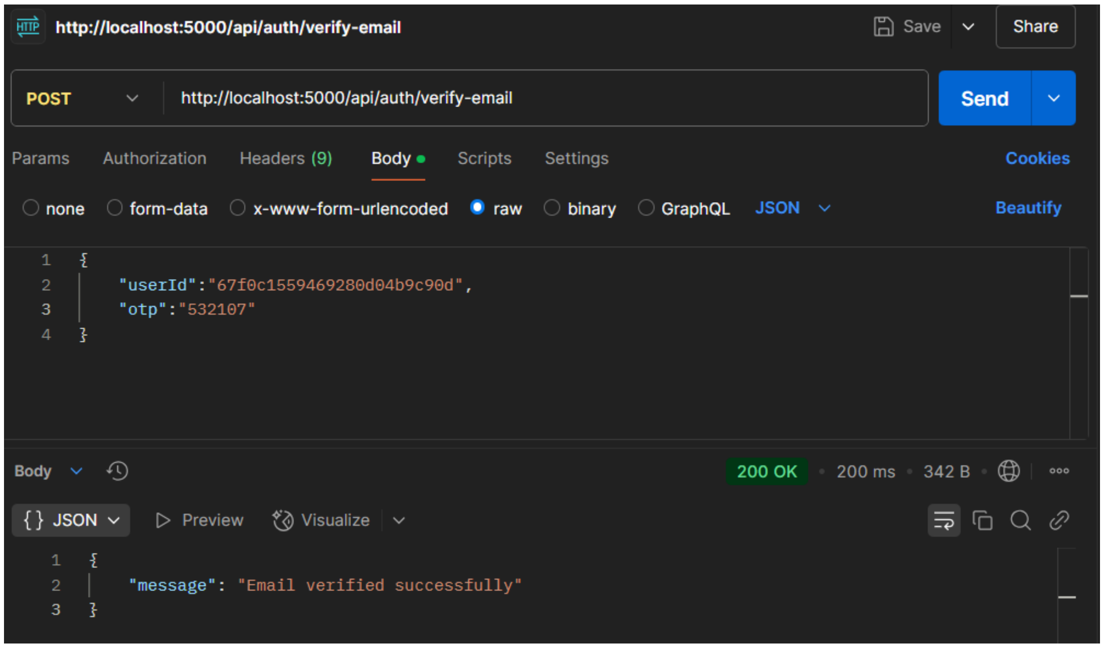
POST http://localhost:5000/api/auth/register

```
{"email":"aryank23@iitk.ac.in",
"password":"aaryan",
"name":"akshat"
}
```

```
{
    "message": "User already exists"
}
```

b. Verify Email: - Verifying the email with OTP.

Unit Details: Check the functioning of verify-email
Test Owner: Aryan Kumar
Test Date: 04/01/2025
Test Results: User is able to send OTP to verify his account.
Structural Coverage: (Decision Coverage) We covered cases where user sends the right and wrong OTP.



POST http://localhost:5000/api/auth/verify-email

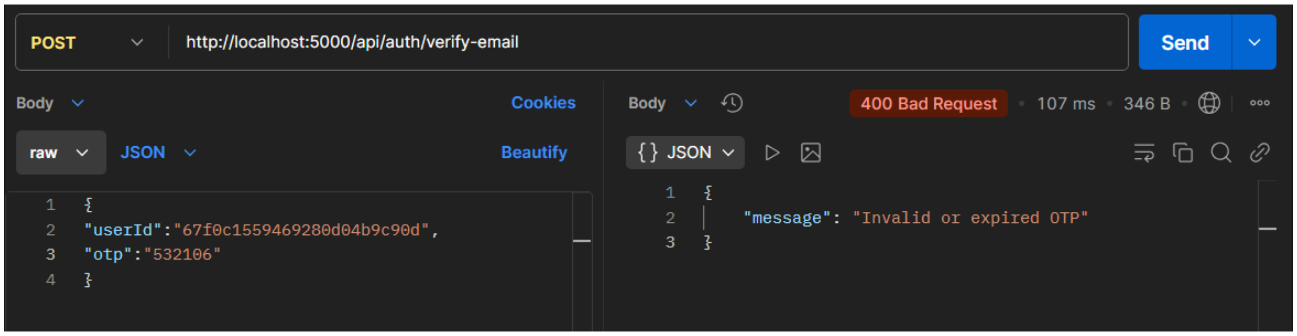
Body: none, form-data, x-www-form-urlencoded, raw [✓], binary, GraphQL — JSON

```
{
    "userId":"67f0c1559469280d04b9c90d",
    "otp":"532107"
}
```

200 OK • 200 ms • 342 B

```
{
    "message": "Email verified successfully"
}
```

If the user gives wrong OTP, the server sends an error.



POST http://localhost:5000/api/auth/verify-email

raw JSON

```
{
"userId":"67f0c1559469280d04b9c90d",
"otp":"532106"
}
```

400 Bad Request • 107 ms • 346 B

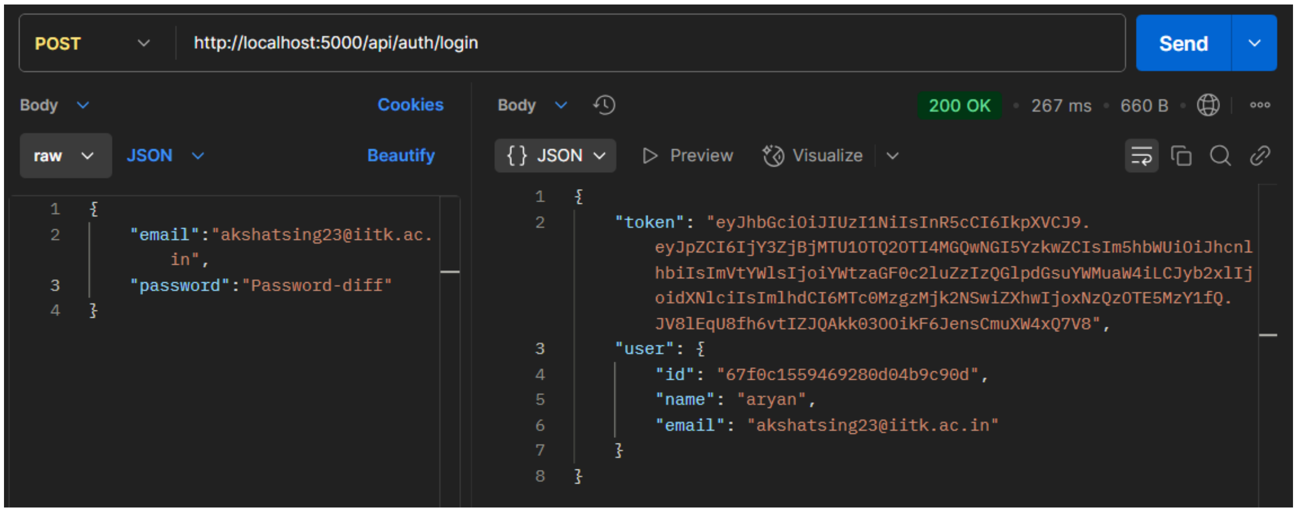
```
{
    "message": "Invalid or expired OTP"
}
```

2. Login

a. Login

Unit Details: Check the functioning of verify-email
Test Owner: Aryan Kumar
Test Date: 04/01/2025
Test Results: Works fine when email and password are correct. It generates the JWT token for the browser for a session.

Structural Coverage: (Decision Coverage) We covered cases where user sends the right and wrong OTP.



POST http://localhost:5000/api/auth/login

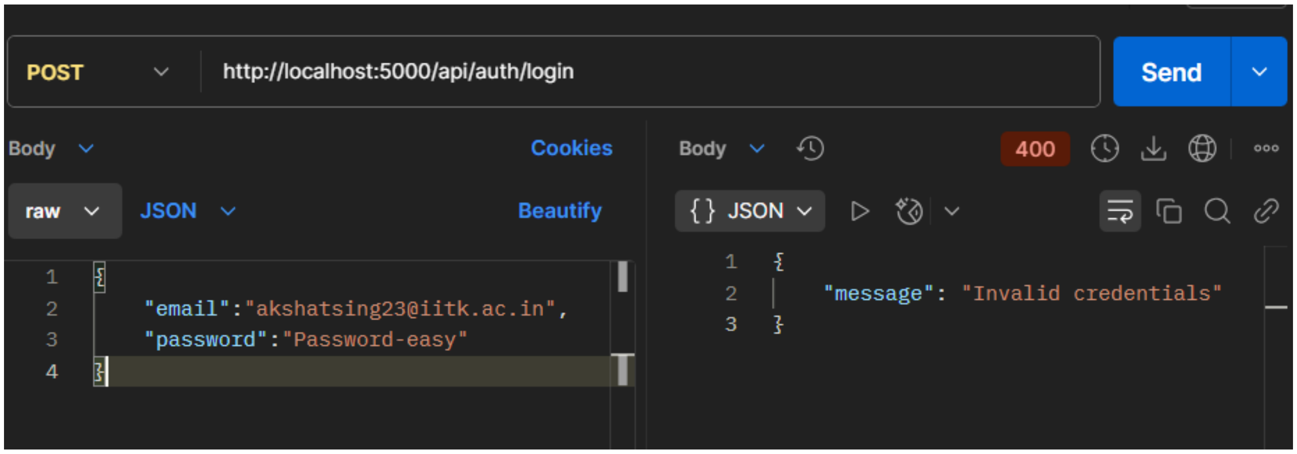
Body raw JSON

```
{
    "email":"akshatsing23@iitk.ac.in",
    "password":"Password-diff"
}
```

200 OK • 267 ms • 660 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJpZCI6IjY3ZjBjMTU1OTQ2OTI4MGQwNGI5YzkwZCIsIm5hbWUiOiJhcnl
        hbiIsImVtYWlsIjoiYWtzaGF0c2luZzIzQGlpdGsuYWMuaW4iLCJyb2xlIj
        oidXNlciIsImlhdCI6MTc0MzgzMjk2NSwiZXhwIjoxNzQzOTE5MzY1fQ.
        JV8lEqU8fh6vtIZJQAkk03OOikF6JensCmuXW4xQ7V8",
    "user": {
        "id": "67f0c1559469280d04b9c90d",
        "name": "aryan",
        "email": "akshatsing23@iitk.ac.in"
    }
}
```

When the user gives a wrong password



POST http://localhost:5000/api/auth/login

Body raw JSON

```
{
    "email":"akshatsing23@iitk.ac.in",
    "password":"Password-easy"
}
```

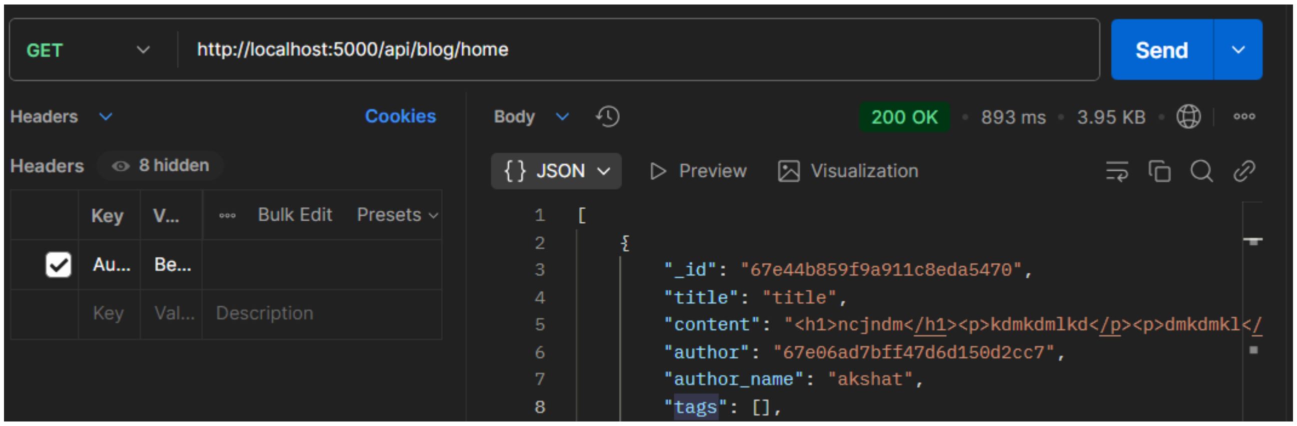
400

```
{
    "message": "Invalid credentials"
}
```

3. Fetching Blogs, Followed Blogs and Saved Blogs for a user

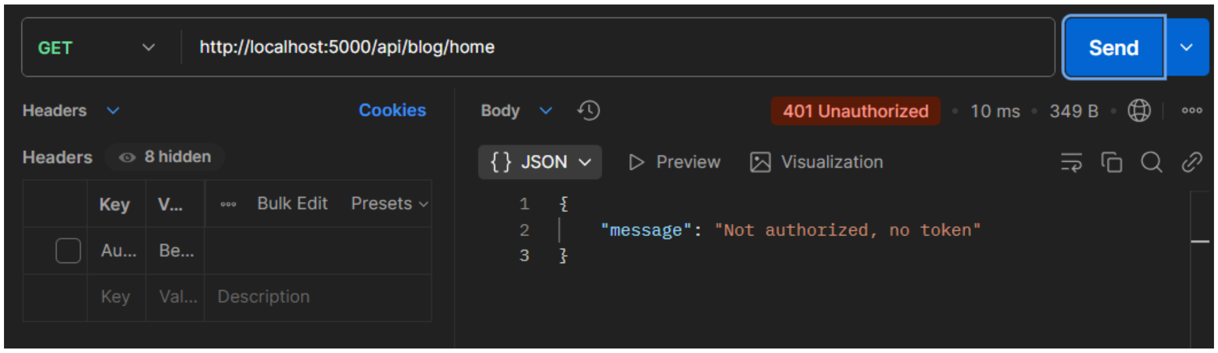
a. Fetching recommended blogs for the user. JWT token is used in the headers for auth.

Unit Details: Check the functioning of fetching blogs.
Test Owner: Aryan Kumar
Test Date: 04/01/2025
Test Results: Works fine the headers are correct.
Structural Coverage: (Decision Coverage) We covered cases where the headers are wrong. Headers are present at every API call for authorisation.



```
[
    {
        "_id": "67e44b859f9a911c8eda5470",
        "title": "title",
        "content": "<h1>ncjndm</h1><p>kdmkdmlkd</p><p>dmkdmkl</
        "author": "67e06ad7bff47d6d150d2cc7",
        "author_name": "akshat",
        "tags": [],
```

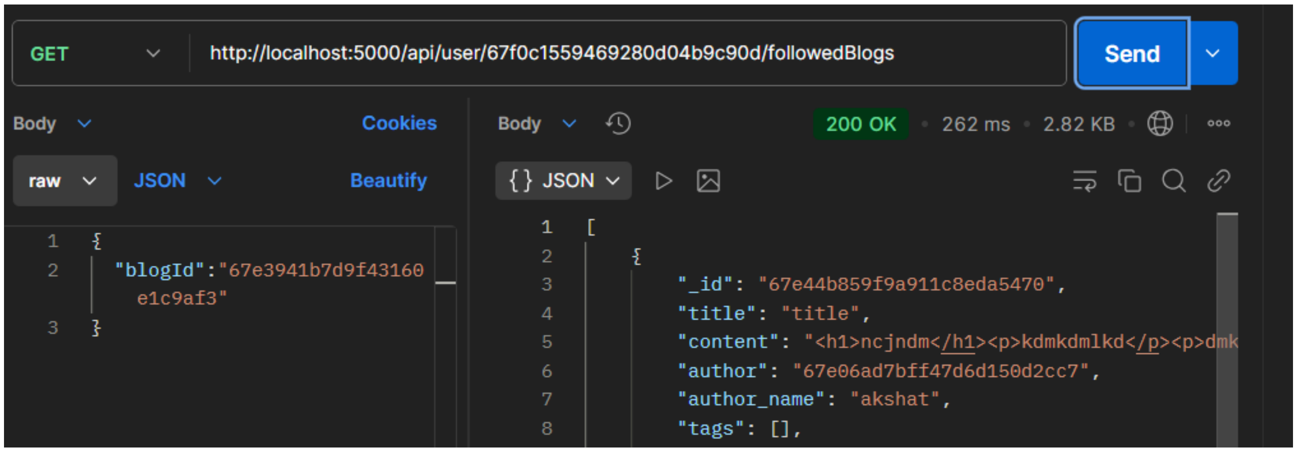
If the token is not present the server returns an error.



```
{
    "message": "Not authorized, no token"
}
```

b. Blogs for the following and saved tabs.

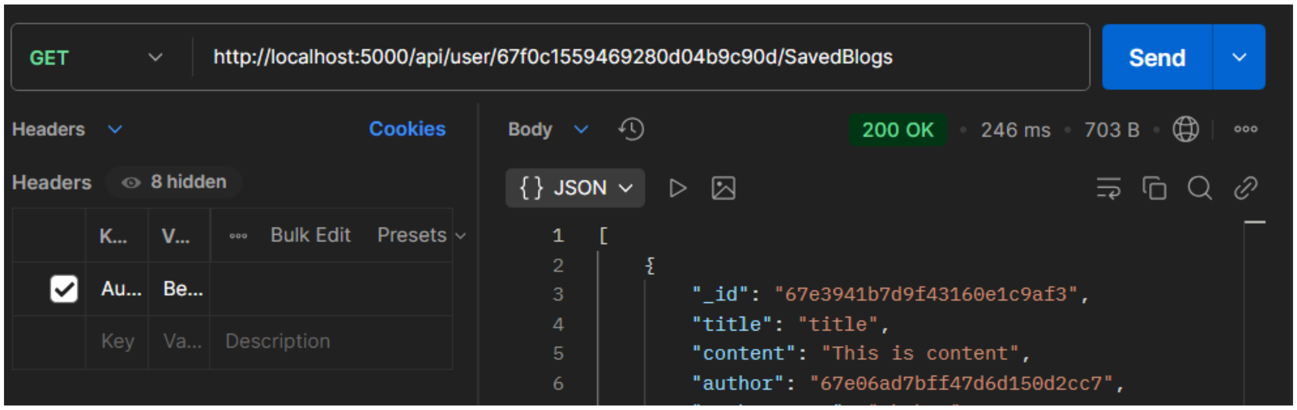
Unit Details: Check the functioning of fetching saved/bookmarked blogs and followed blogs.
Test Owner: Aryan Kumar
Test Date: 04/01/2025
Test Results: Works fine the headers are correct.
Structural Coverage: (Decision Coverage) We covered cases where the headers are wrong. Headers are present at every API call for authorisation.



http://localhost:5000/api/user/67f0c1559469280d04b9c90d/followedBlogs

```
{
  "blogId":"67e3941b7d9f43160
  e1c9af3"
}
```

```
[
  {
    "_id": "67e44b859f9a911c8eda5470",
    "title": "title",
    "content": "<h1>ncjndm</h1><p>kdmkdmlkd</p><p>dmk
    "author": "67e06ad7bff47d6d150d2cc7",
    "author_name": "akshat",
    "tags": [],
```



http://localhost:5000/api/user/67f0c1559469280d04b9c90d/SavedBlogs

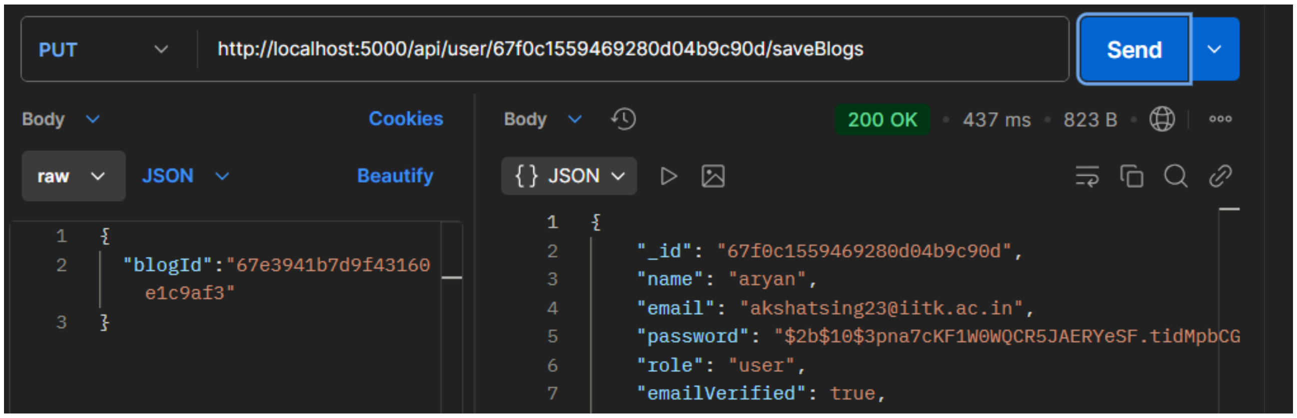
	K...	V...	Description
✓	Au...	Be...	
	Key	Va...	Description

```
[
  {
    "_id": "67e3941b7d9f43160e1c9af3",
    "title": "title",
    "content": "This is content",
    "author": "67e06ad7bff47d6d150d2cc7",
```

4. Interacting with blogs. (Upvoting, Saving/Bookmarking and Commenting on the blog)

a. Saving the blog

Unit Details: Check the functioning of saving/bookmarking the blog.
Test Owner: Someshwar Singh
Test Date: 04/01/2025
Test Results: Works fine the headers are correct and blogId is correct.
Structural Coverage: (Decision Coverage) We covered cases where the blogId is incorrect.

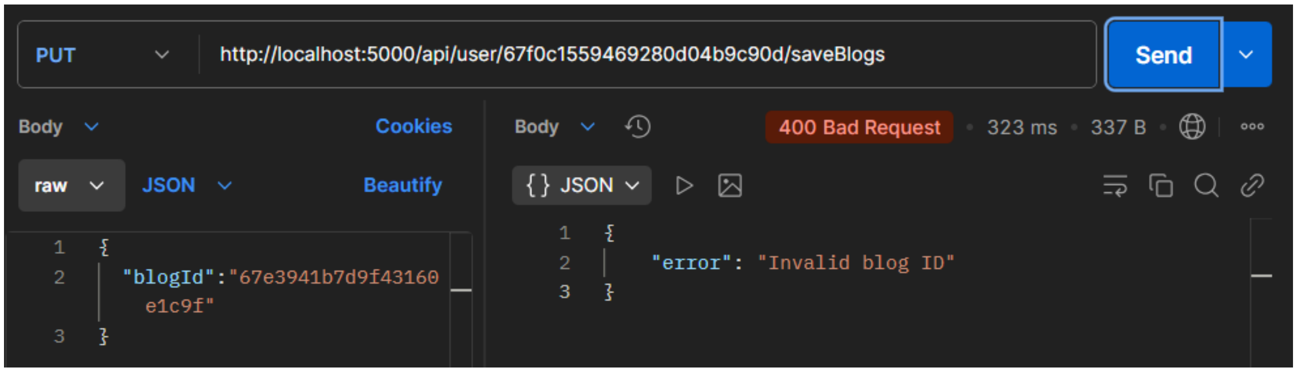


PUT http://localhost:5000/api/user/67f0c1559469280d04b9c90d/saveBlogs

Request body (raw JSON):
```
{
  "blogId":"67e3941b7d9f43160
  e1c9af3"
}
```

Response: 200 OK • 437 ms • 823 B
```
{
    "_id": "67f0c1559469280d04b9c90d",
    "name": "aryan",
    "email": "akshatsing23@iitk.ac.in",
    "password": "$2b$10$3pna7cKF1W0WQCR5JAERYeSF.tidMpbCG
    "role": "user",
    "emailVerified": true,
```

If the blogId provided is incorrect then the sever shows an error message of "Blog Not Found"



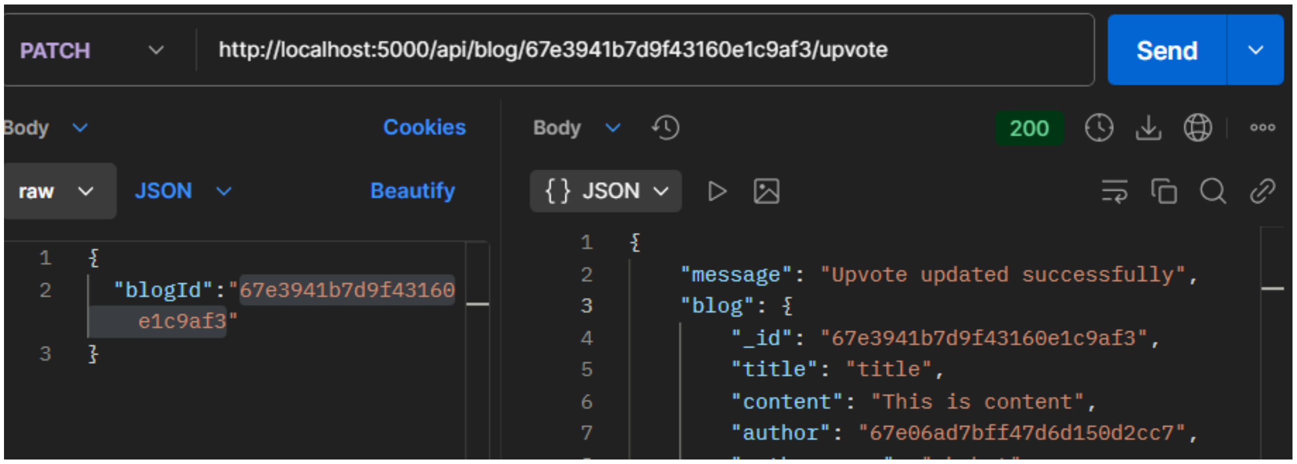
PUT http://localhost:5000/api/user/67f0c1559469280d04b9c90d/saveBlogs

Request body (raw JSON):
```
{
  "blogId":"67e3941b7d9f43160
  e1c9f"
}
```

Response: 400 Bad Request • 323 ms • 337 B
```
{
    "error": "Invalid blog ID"
}
```

b. Upvoting and Downvoting Blogs

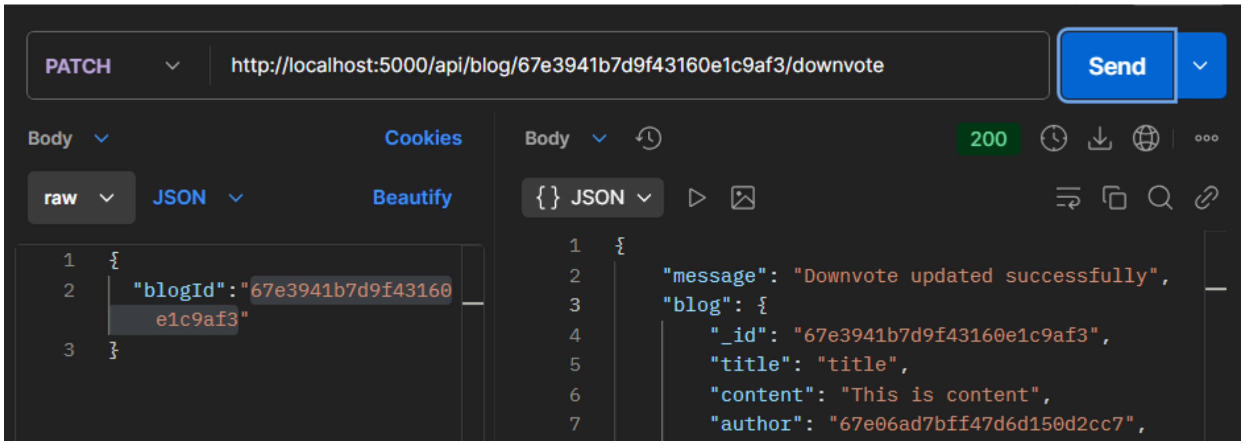
Unit Details: Check the functioning of upvoting and downvoting blogs.
Test Owner: Someshwar Singh
Test Date: 04/02/2025
Test Results: Works fine the headers are correct and blogId is correct.
Structural Coverage: (Decision Coverage) We covered cases where the blogId is incorrect.



```
PATCH http://localhost:5000/api/blog/67e3941b7d9f43160e1c9af3/upvote
```

```
{
  "blogId":"67e3941b7d9f43160e1c9af3"
}
```

```
{
    "message": "Upvote updated successfully",
    "blog": {
        "_id": "67e3941b7d9f43160e1c9af3",
        "title": "title",
        "content": "This is content",
        "author": "67e06ad7bff47d6d150d2cc7",
```



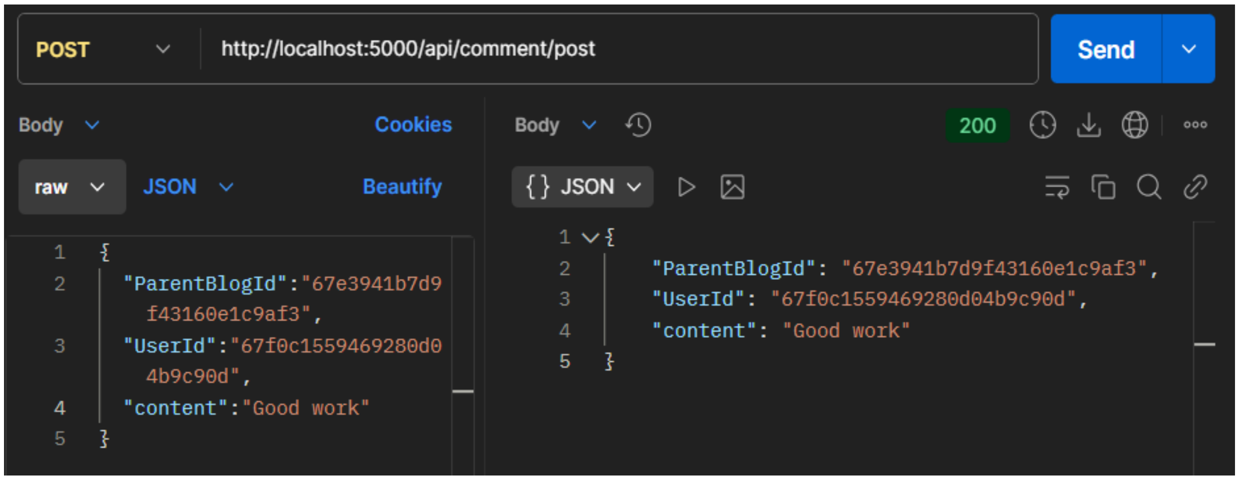
```
PATCH http://localhost:5000/api/blog/67e3941b7d9f43160e1c9af3/downvote
```

```
{
  "blogId":"67e3941b7d9f43160e1c9af3"
}
```

```
{
    "message": "Downvote updated successfully",
    "blog": {
        "_id": "67e3941b7d9f43160e1c9af3",
        "title": "title",
        "content": "This is content",
        "author": "67e06ad7bff47d6d150d2cc7",
```

C. Commenting on the blog.

Unit Details: Check the functioning of creating a comment on the blog.
Test Owner: Someshwar Singh
Test Date: 03/**31/2025**
Test Results: Works fine the headers are correct and blogId is correct.
Structural Coverage: (Decision Coverage) We covered cases where the blogId is incorrect.

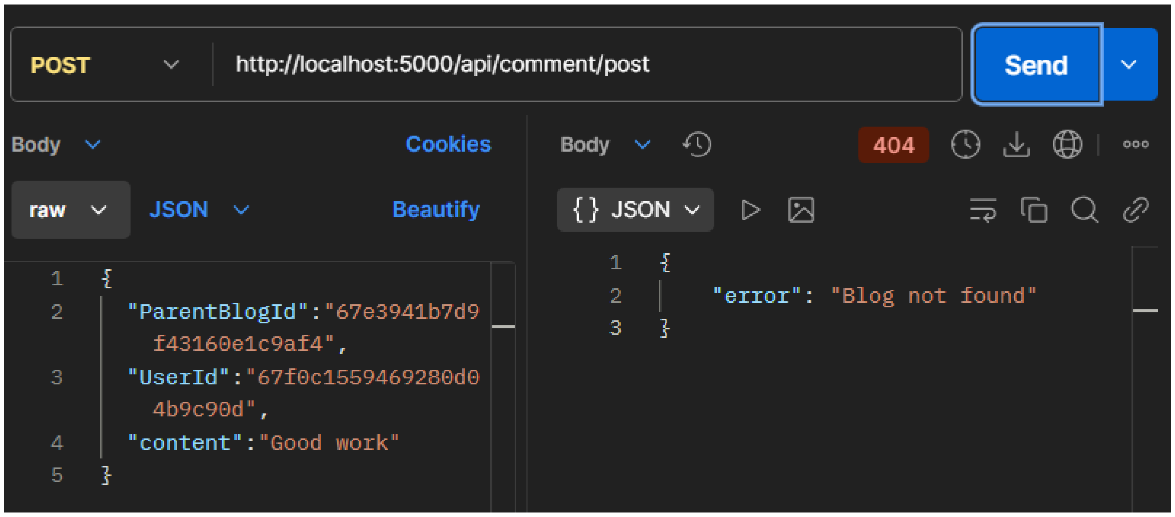

POST http://localhost:5000/api/comment/post

```
{
  "ParentBlogId":"67e3941b7d9f43160e1c9af3",
  "UserId":"67f0c1559469280d04b9c90d",
  "content":"Good work"
}
```

200

```
{
    "ParentBlogId": "67e3941b7d9f43160e1c9af3",
    "UserId": "67f0c1559469280d04b9c90d",
    "content": "Good work"
}
```

If the UserID and ParentBlogID is not correct, then the server shows error.


POST http://localhost:5000/api/comment/post

```
{
  "ParentBlogId":"67e3941b7d9f43160e1c9af4",
  "UserId":"67f0c1559469280d04b9c90d",
  "content":"Good work"
}
```

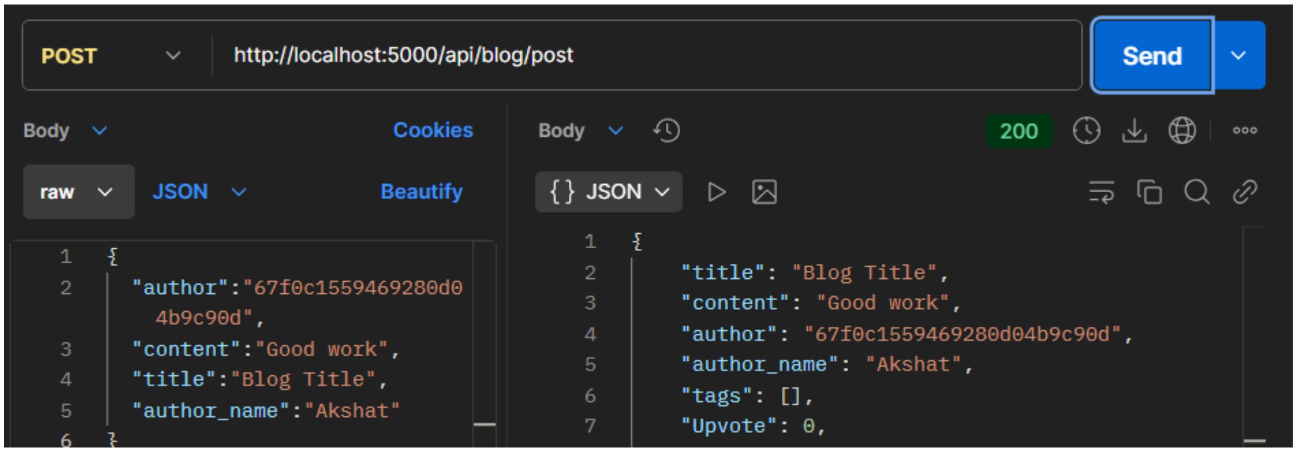
404

```
{
    "error": "Blog not found"
}
```

5. Blog Creation, Editing and Deletion.

a. Blog Creation

Unit Details: Check the functioning of creation of a blog.
Test Owner: Someshwar Singh
Test Date: 04/03/2025
Test Results: User is able to create the blog.
Structural Coverage: (Decision Coverage) We covered cases where one of the fields in the body of request went missing.


POST http://localhost:5000/api/blog/post

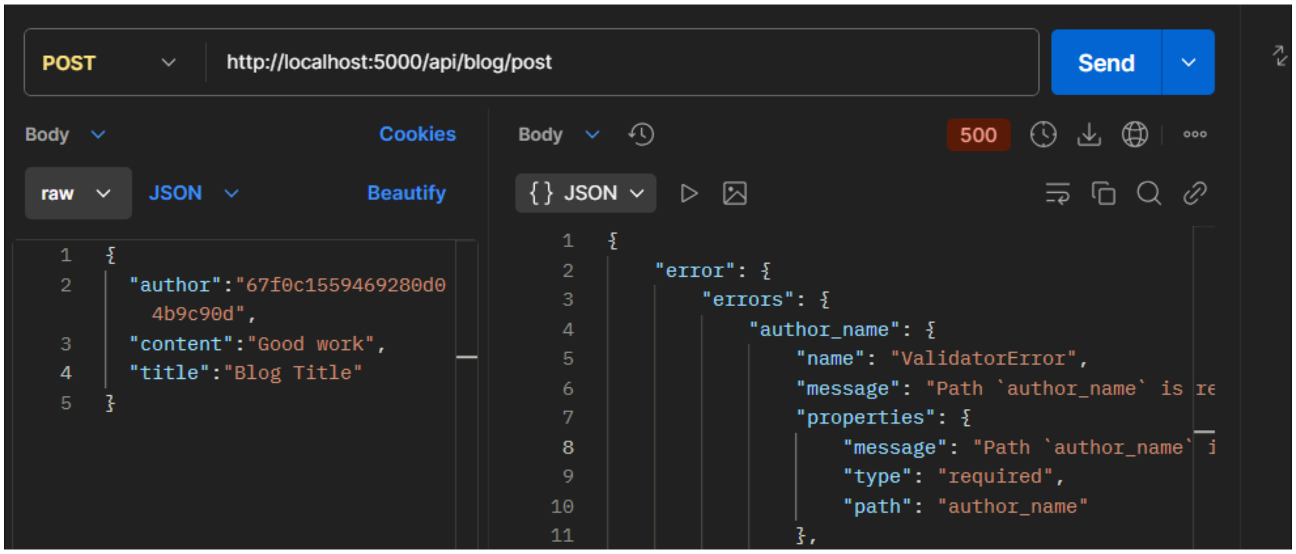
Body: raw JSON

```
{
  "author":"67f0c1559469280d0
   4b9c90d",
  "content":"Good work",
  "title":"Blog Title",
  "author_name":"Akshat"
}
```

Response: 200

```
{
    "title": "Blog Title",
    "content": "Good work",
    "author": "67f0c1559469280d04b9c90d",
    "author_name": "Akshat",
    "tags": [],
    "Upvote": 0,
```

If one of the fields is missing the server shows the Validator Error.


POST http://localhost:5000/api/blog/post

Body: raw JSON

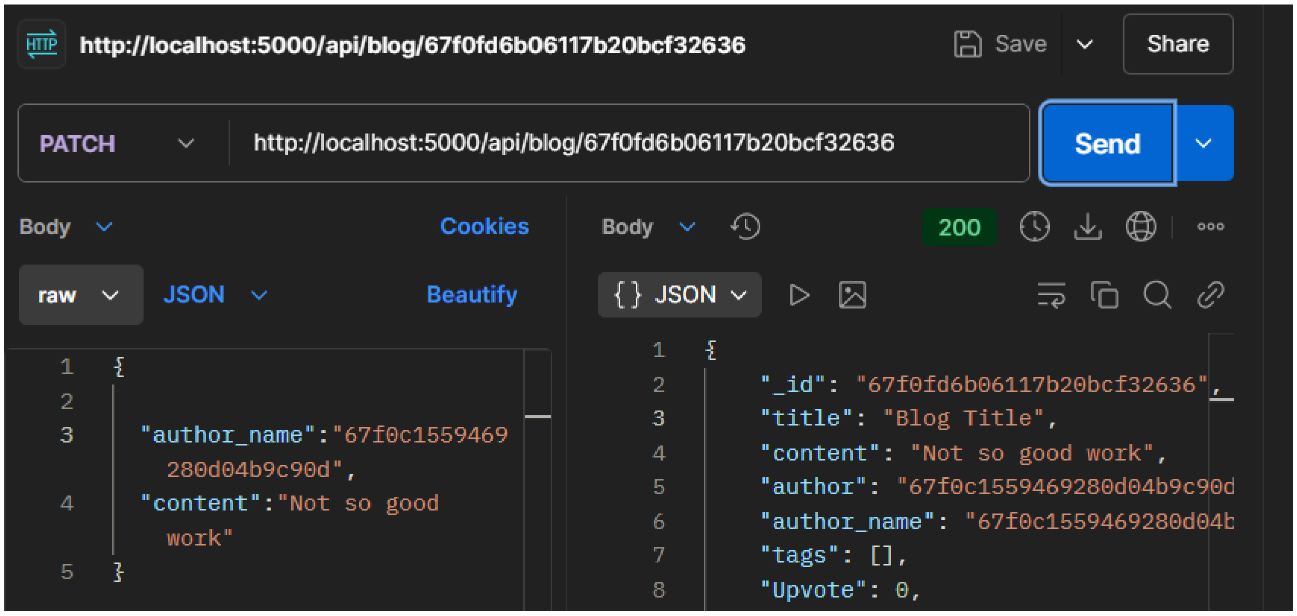
```
{
  "author":"67f0c1559469280d0
   4b9c90d",
  "content":"Good work",
  "title":"Blog Title"
}
```

Response: 500

```
{
    "error": {
        "errors": {
            "author_name": {
                "name": "ValidatorError",
                "message": "Path `author_name` is re
                "properties": {
                    "message": "Path `author_name` i
                    "type": "required",
                    "path": "author_name"
                },
```

b. Blog Editing

Unit Details: Check the functioning of updating a blog.
Test Owner: Someshwar Singh
Test Date: 04/03/2025
Test Results: User is able to edit the blog.
Structural Coverage: (Decision Coverage) We covered cases where one of the fields in the body of request went missing.


http://localhost:5000/api/blog/67f0fd6b06117b20bcf32636

PATCH http://localhost:5000/api/blog/67f0fd6b06117b20bcf32636

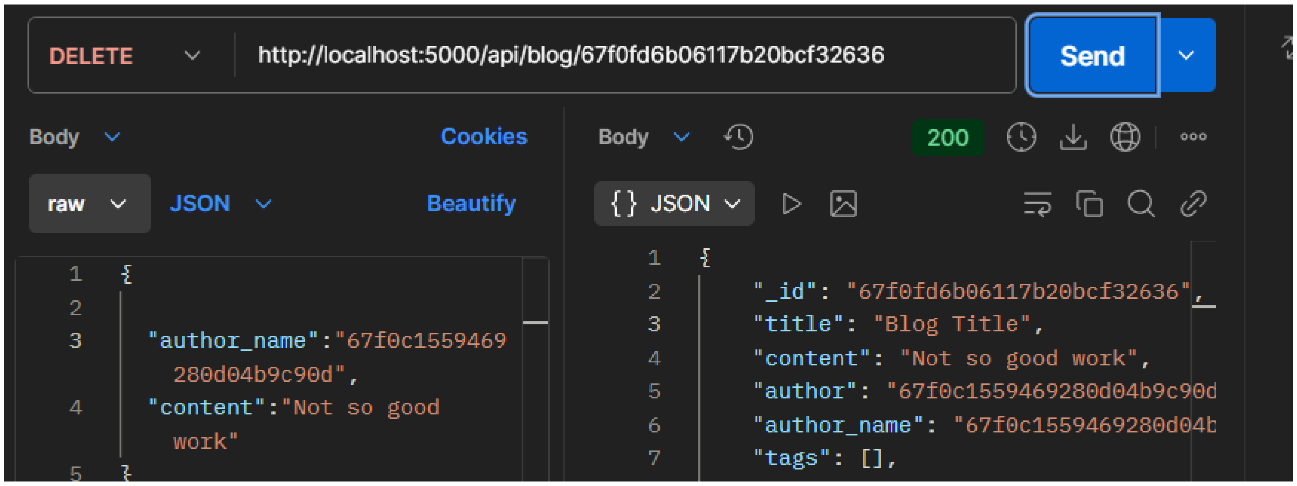
```
{

  "author_name":"67f0c1559469
    280d04b9c90d",
  "content":"Not so good
    work"
}
```

Response: 200

```
{
    "_id": "67f0fd6b06117b20bcf32636",
    "title": "Blog Title",
    "content": "Not so good work",
    "author": "67f0c1559469280d04b9c90d
    "author_name": "67f0c1559469280d04b
    "tags": [],
    "Upvote": 0,
```

c. Blog Deletion

Unit Details: Check the functioning of deletion of a blog.
Test Owner: Someshwar Singh
Test Date: 04/03/2025
Test Results: User is able to delete the blog.
Structural Coverage: (Decision Coverage) We covered cases where the user is not the author if the blog and is trying to delete the blog.


DELETE http://localhost:5000/api/blog/67f0fd6b06117b20bcf32636

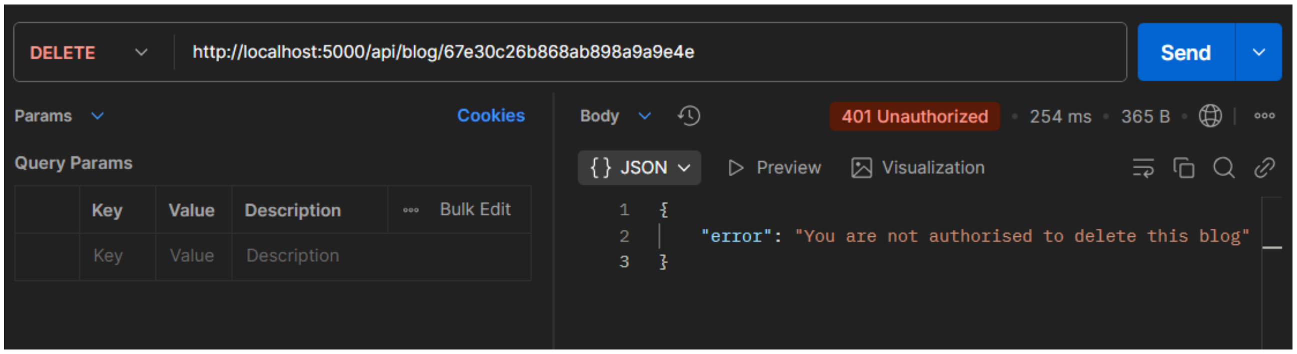
Body raw JSON

```
{

  "author_name":"67f0c1559469
    280d04b9c90d",
  "content":"Not so good
    work"
}
```

200

```
{
    "_id": "67f0fd6b06117b20bcf32636",
    "title": "Blog Title",
    "content": "Not so good work",
    "author": "67f0c1559469280d04b9c90d
    "author_name": "67f0c1559469280d04b
    "tags": [],
```

If the request sender is not the user, then he the server will send an error.


DELETE http://localhost:5000/api/blog/67e30c26b868ab898a9a9e4e

Params · Query Params

Key	Value	Description
Key	Value	Description

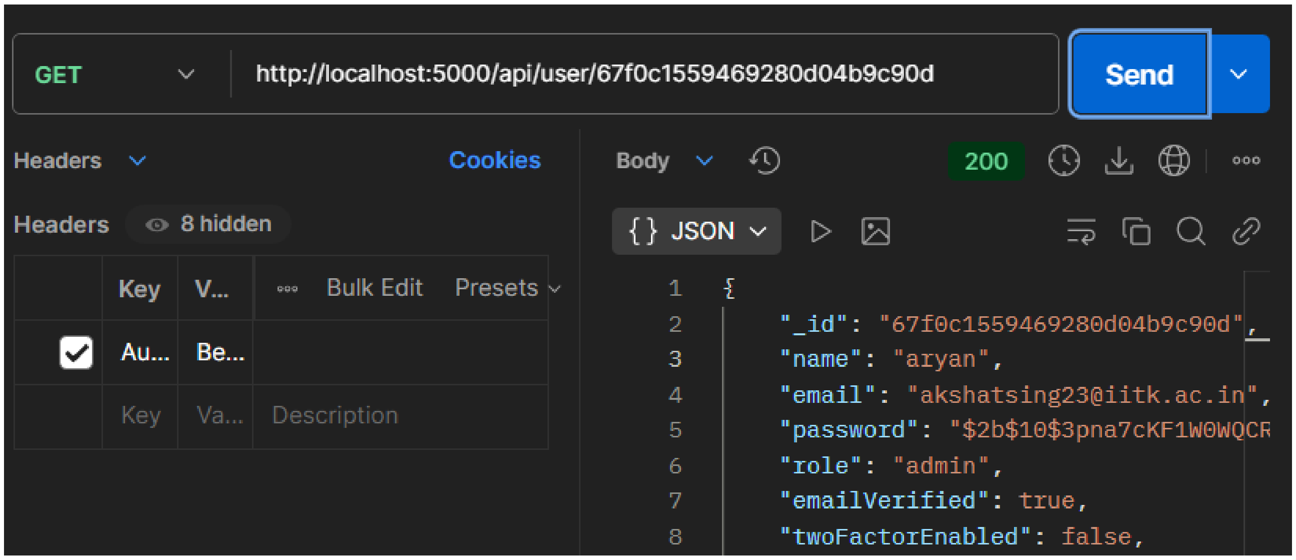
401 Unauthorized • 254 ms • 365 B

```
{
    "error": "You are not authorised to delete this blog"
}
```

6. User Profile Interactions (Editing information, fetching user's info, Following).

a. Fetching user's or any account information for the profile page.

Unit Details: Checking the functioning of fetching a account's information.
Test Owner: Someshwar Singh
Test Date: 04/01/2025
Test Results: User is able to fetch an account's info.
Structural Coverage: (Decision Coverage) We covered cases where the userId in the url is not correct.

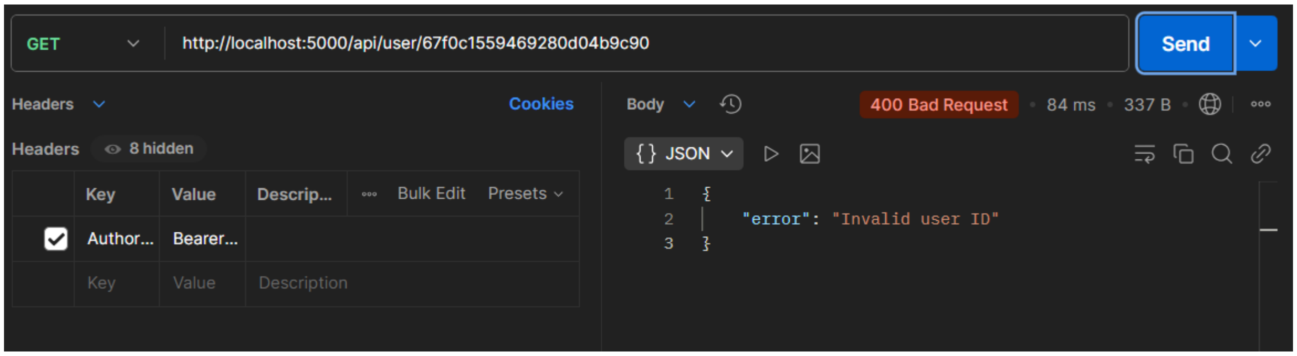


```
GET http://localhost:5000/api/user/67f0c1559469280d04b9c90d
```

	Key	V...
✓	Au...	Be...
	Key	Va...

```
{
    "_id": "67f0c1559469280d04b9c90d",
    "name": "aryan",
    "email": "akshatsing23@iitk.ac.in",
    "password": "$2b$10$3pna7cKF1W0WQCR
    "role": "admin",
    "emailVerified": true,
    "twoFactorEnabled": false,
```

When the userId is incorrect in the URL the server sends an error.



```
GET http://localhost:5000/api/user/67f0c1559469280d04b9c90
```

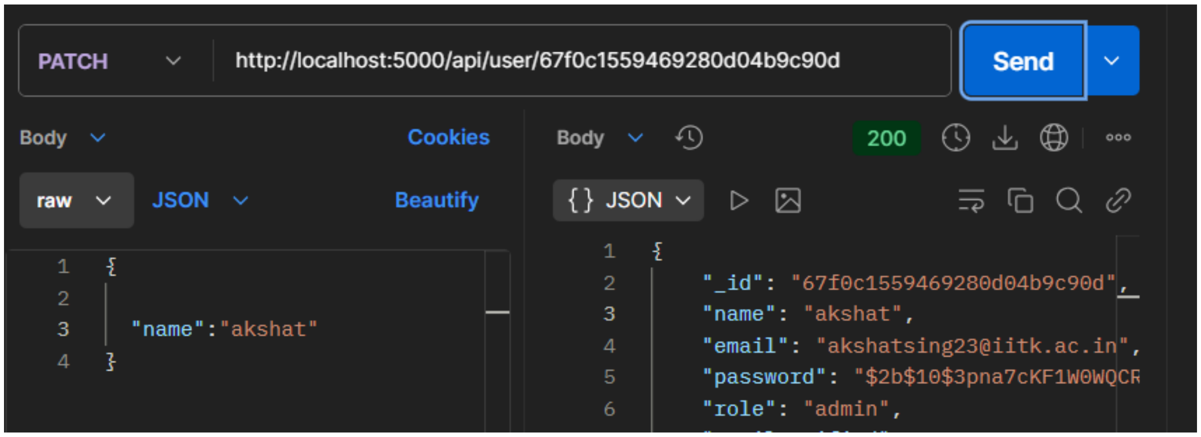
	Key	Value	Descrip...
✓	Author...	Bearer...	
	Key	Value	Description

```
{
    "error": "Invalid user ID"
}
```

b. Updating user's information

Unit Details: Checking the functioning of updating an account's information.

Test Owner: Someshwar Singh
Test Date: 04/01/2025
Test Results: User is able to update an account's information.
Structural Coverage: (Decision Coverage) We covered cases where the request sender doesn't own the account.



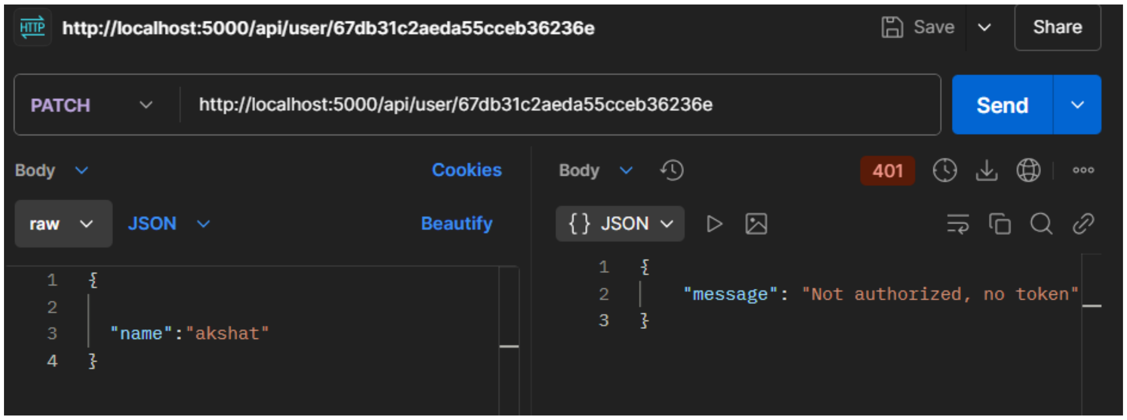
PATCH http://localhost:5000/api/user/67f0c1559469280d04b9c90d

```
{

  "name":"akshat"
}
```

200

```
{
    "_id": "67f0c1559469280d04b9c90d",
    "name": "akshat",
    "email": "akshatsing23@iitk.ac.in",
    "password": "$2b$10$3pna7cKF1W0WQCR
    "role": "admin",
```

If the user is trying to edit another person's account, the server will send an error.



http://localhost:5000/api/user/67db31c2aeda55cceb36236e

PATCH http://localhost:5000/api/user/67db31c2aeda55cceb36236e

```
{

  "name":"akshat"
}
```

401

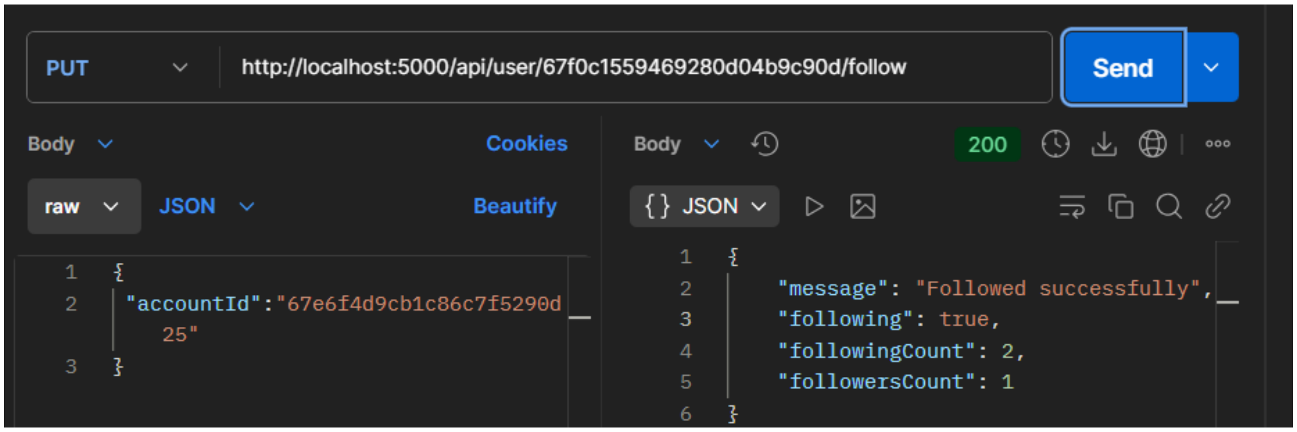
```
{
    "message": "Not authorized, no token"
}
```

c. **Following another user's account**

Unit Details: Checking the functioning of following another account.
Test Owner: Someshwar Singh
Test Date: 04/01/2025

Test Results: User is able to follow another account.

Structural Coverage: (Decision Coverage) We covered cases where the accountId in the body is not a viable ID.



PUT http://localhost:5000/api/user/67f0c1559469280d04b9c90d/follow

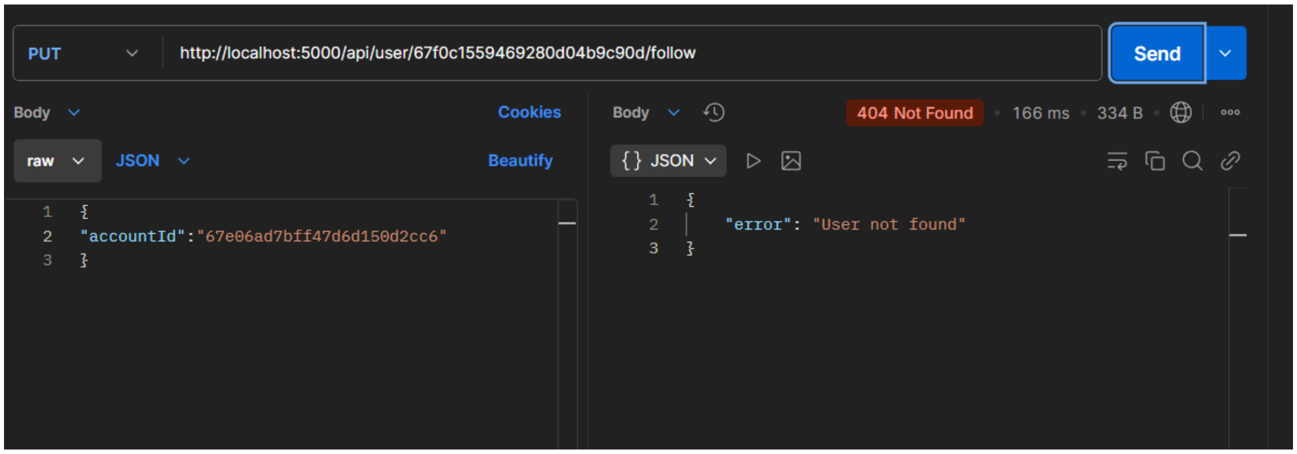
Body: raw JSON

```
{
"accountId":"67e6f4d9cb1c86c7f5290d25"
}
```

Response: 200

```
{
    "message": "Followed successfully",
    "following": true,
    "followingCount": 2,
    "followersCount": 1
}
```

When the accountId is not correct the server sends an error.



PUT http://localhost:5000/api/user/67f0c1559469280d04b9c90d/follow

Body: raw JSON

```
{
"accountId":"67e06ad7bff47d6d150d2cc6"
}
```

Response: 404 Not Found • 166 ms • 334 B

```
{
  "error": "User not found"
}
```

7. Searching Blogs and Tags

a. Searching query for blogs

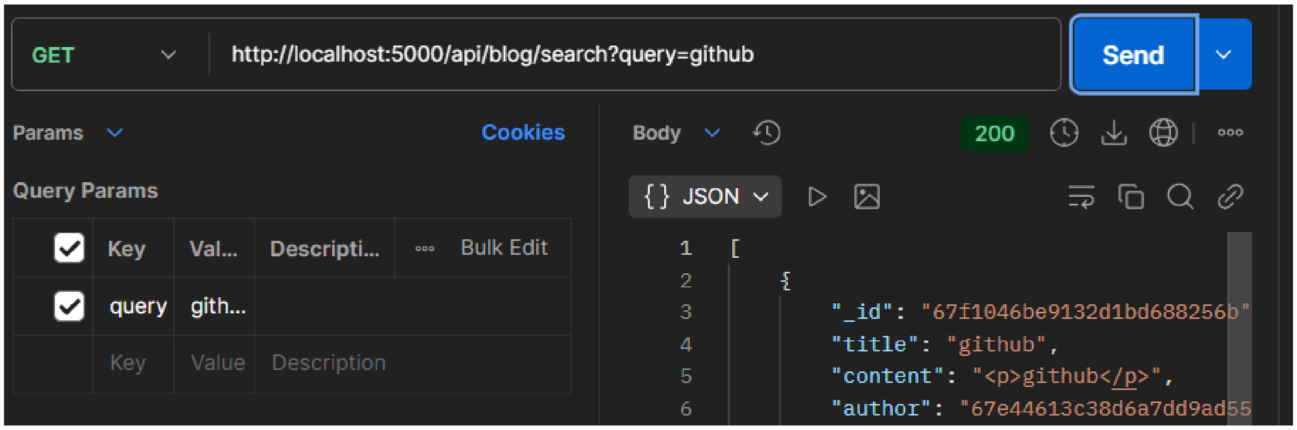
Unit Details: Checking the functioning of searching for blogs.

Test Owner: Someshwar Singh

Test Date: 04/03/2025

Test Results: User is able to search for blogs.

Structural Coverage: (Decision Coverage) We covered cases where the query is not preset in the URL.


GET http://localhost:5000/api/blog/search?query=github

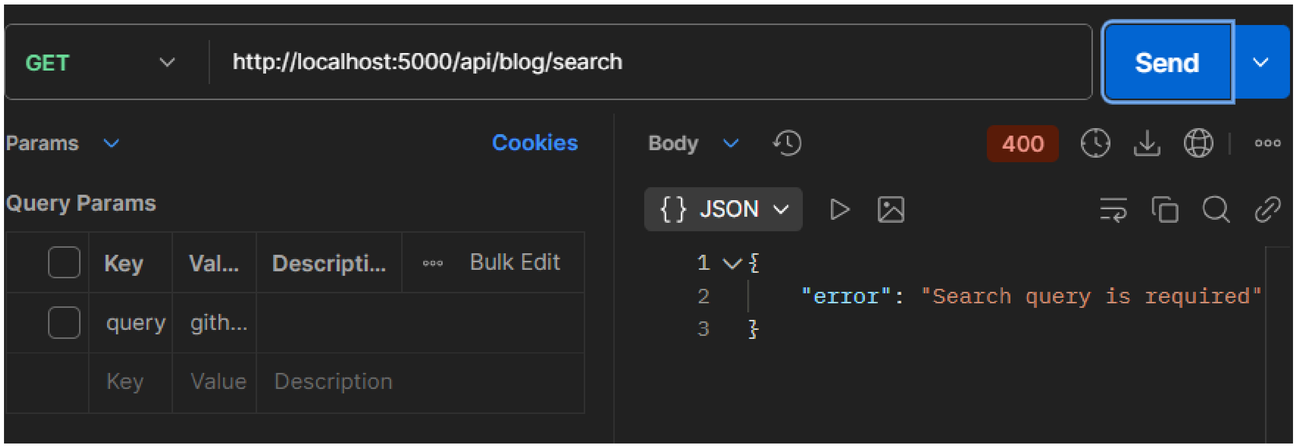
Query Params

	Key	Value	Description
✓	query	gith...	

200

```
[
    {
        "_id": "67f1046be9132d1bd688256b",
        "title": "github",
        "content": "<p>github</p>",
        "author": "67e44613c38d6a7dd9ad55
```

If query is not present.


GET http://localhost:5000/api/blog/search

Query Params

	Key	Value	Description
	query	gith...	

400

```
{
    "error": "Search query is required"
}
```

b. Searching tags with queries.

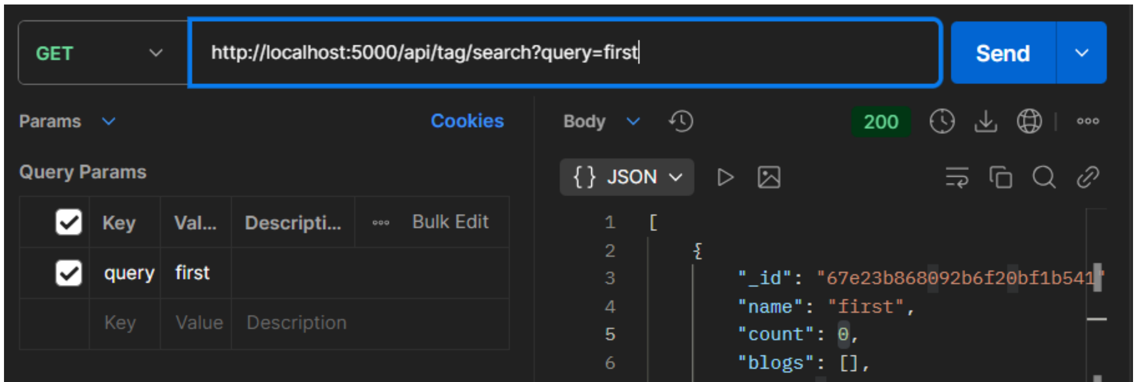
Unit Details: Checking the functioning of searching for tags.

Test Owner: Someshwar Singh

Test Date: 04/01/2025

Test Results: User is able to search for tags.

Structural Coverage: (Decision Coverage) We covered cases where the query is not present in the URL.


GET http://localhost:5000/api/tag/search?query=first

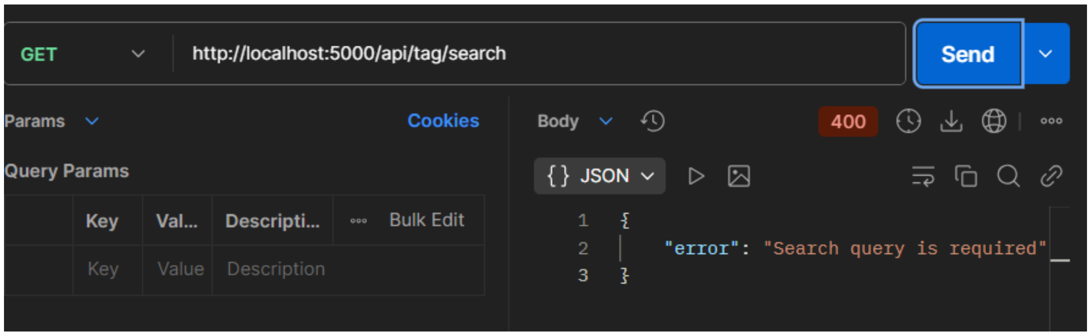
Query Params

	Key	Value	Description
✓	query	first	

200

```
[
    {
        "_id": "67e23b868092b6f20bf1b541",
        "name": "first",
        "count": 0,
        "blogs": [],
```

Similarly, if query is not present


GET http://localhost:5000/api/tag/search Send

Params Cookies | Body 400 | JSON

Query Params

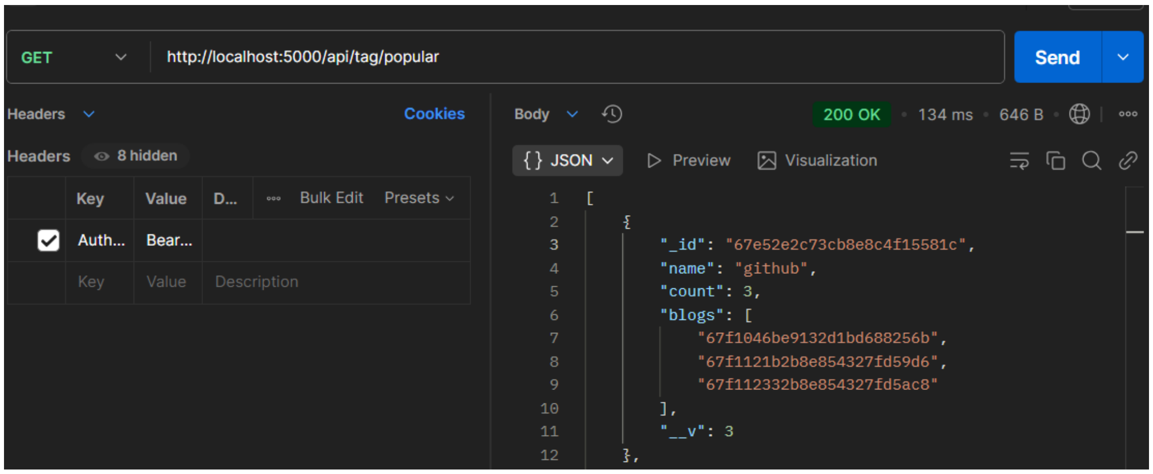
Key	Val...	Descripti...	Bulk Edit
Key	Value	Description	

```
{
    "error": "Search query is required"
}
```

8. Fetching Popular tags and users.

a. Fetching Popular Tags

Unit Details: Checking the functioning of fetching popular blogs.
Test Owner: Someshwar Singh
Test Date: 04/**01/2025**
Test Results: User is able to fetch popular blogs.
Structural Coverage: Statement Coverage was used.


GET http://localhost:5000/api/tag/popular Send

Headers Cookies | Body 200 OK 134 ms 646 B | JSON Preview Visualization

Headers 8 hidden

	Key	Value	D...	Bulk Edit Presets
✓	Auth...	Bear...		
	Key	Value	Description	

```
[
    {
        "_id": "67e52e2c73cb8e8c4f15581c",
        "name": "github",
        "count": 3,
        "blogs": [
            "67f1046be9132d1bd688256b",
            "67f1121b2b8e854327fd59d6",
            "67f112332b8e854327fd5ac8"
        ],
        "__v": 3
    },
```

b. Fetching top bloggers.

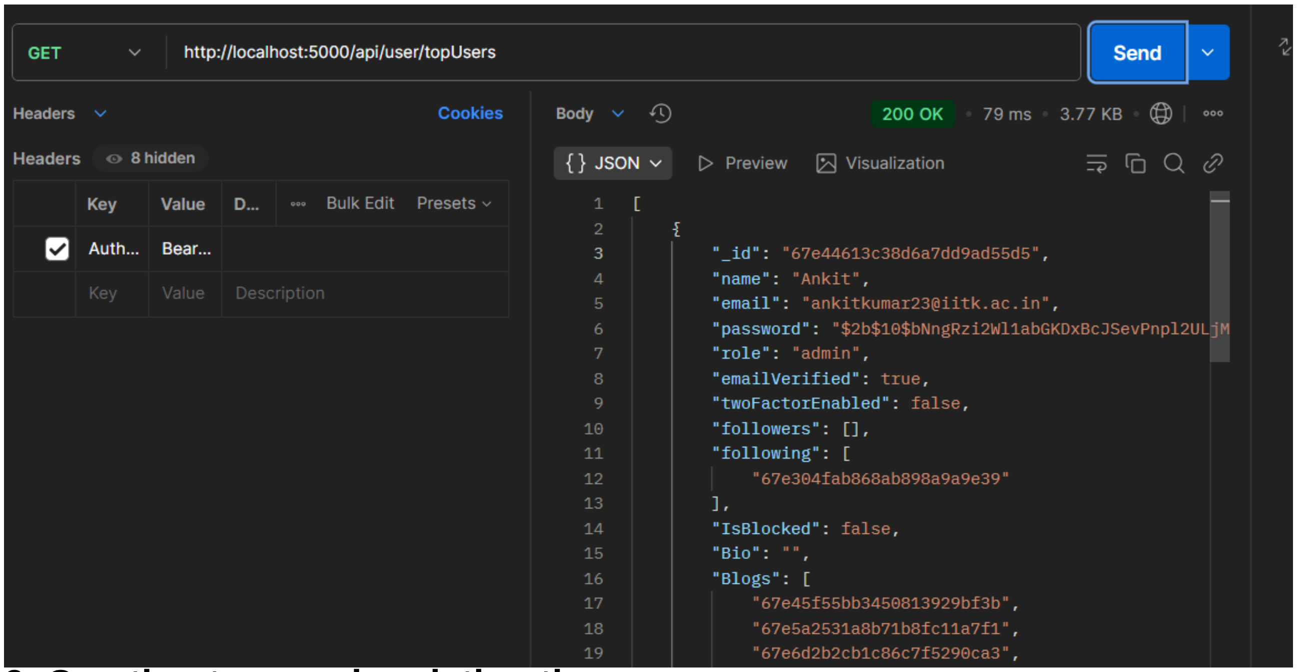
Unit Details: Checking the functioning of fetching top bloggers.

Test Owner: Ansh Adarsh

Test Date: 04/**01/2025**

Test Results: User is able to fetch top bloggers.

Structural Coverage: Statement Coverage was used.



GET http://localhost:5000/api/user/topUsers

Headers: Auth... Bear...

200 OK • 79 ms • 3.77 KB

```
[
    {
        "_id": "67e44613c38d6a7dd9ad55d5",
        "name": "Ankit",
        "email": "ankitkumar23@iitk.ac.in",
        "password": "$2b$10$bNngRzi2Wl1abGKDxBcJSevPnpl2ULjM
        "role": "admin",
        "emailVerified": true,
        "twoFactorEnabled": false,
        "followers": [],
        "following": [
            "67e304fab868ab898a9a9e39"
        ],
        "IsBlocked": false,
        "Bio": "",
        "Blogs": [
            "67e45f55bb3450813929bf3b",
            "67e5a2531a8b71b8fc11a7f1",
            "67e6d2b2cb1c86c7f5290ca3",
```

9. Creating tags and updating them

a. Tag Creation

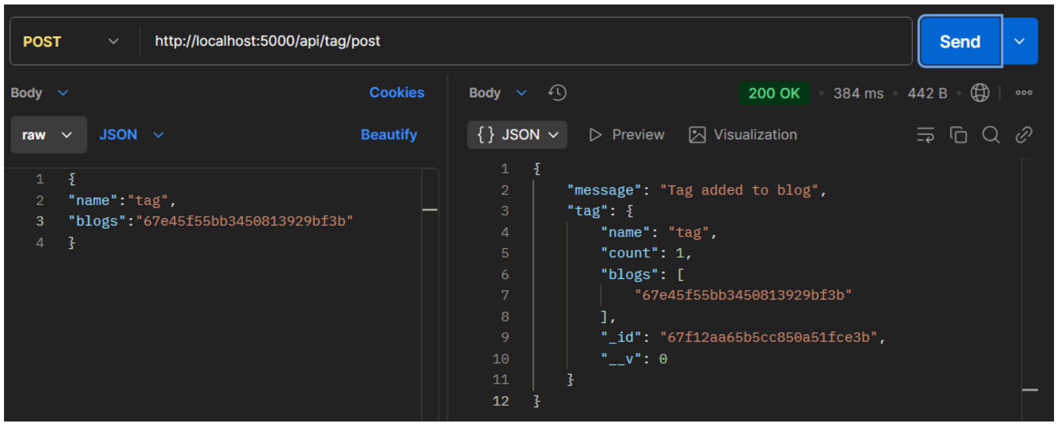
Unit Details: Checking the functioning of creating tags.

Test Owner: Durbasmriti Saha

Test Date: 04/**01/2025**

Test Results: User is able to create tags.

Structural Coverage: Statement Coverage was used.



POST http://localhost:5000/api/tag/post

Body raw JSON

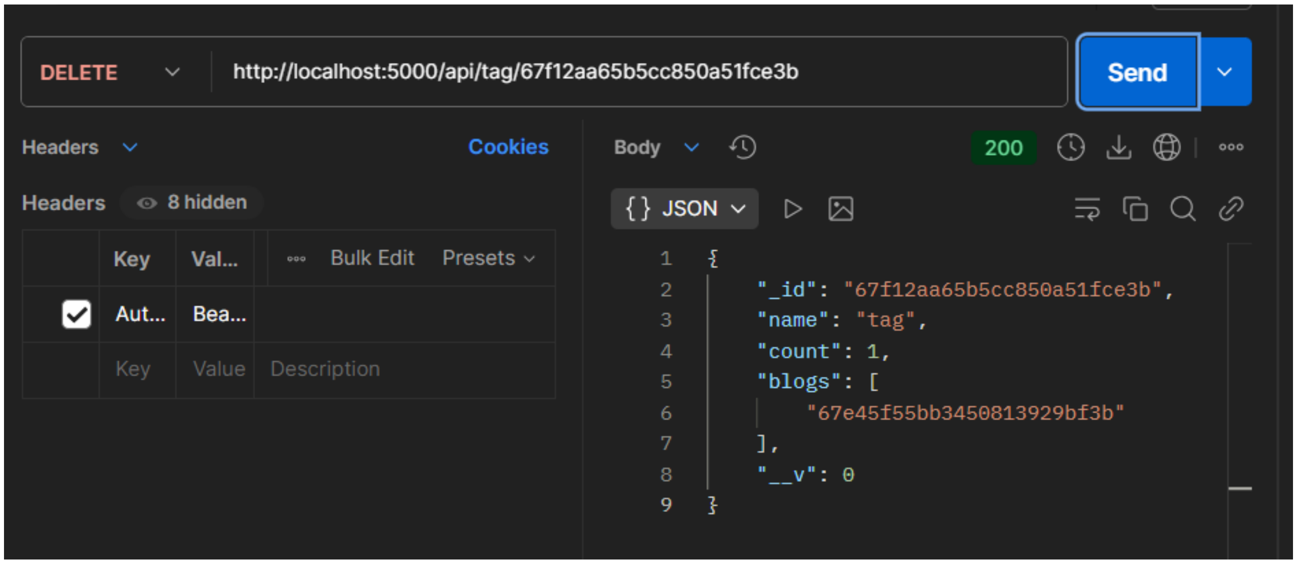
```
{
"name":"tag",
"blogs":"67e45f55bb3450813929bf3b"
}
```

200 OK • 384 ms • 442 B

```
{
    "message": "Tag added to blog",
    "tag": {
        "name": "tag",
        "count": 1,
        "blogs": [
            "67e45f55bb3450813929bf3b"
        ],
        "_id": "67f12aa65b5cc850a51fce3b",
        "__v": 0
    }
}
```

b. Tag Deletion

Unit Details: Checking the functioning of following another account.
Test Owner: Ansh Adarsh
Test Date: 04/02/2025
Test Results: User is able to delete tags.
Structural Coverage: Statement Coverage was used.



DELETE http://localhost:5000/api/tag/67f12aa65b5cc850a51fce3b

Key	Val...	Description
✓ Aut...	Bea...	
Key	Value	Description

```
{
    "_id": "67f12aa65b5cc850a51fce3b",
    "name": "tag",
    "count": 1,
    "blogs": [
        "67e45f55bb3450813929bf3b"
    ],
    "__v": 0
}
```

3 Integration Testing

User Authentication Flow

Integration testing verifies the interaction between different modules of the Echo application, ensuring they work together as expected. This section details the tests performed on key integration points.

3.1 Authentication

Module Details: This tests the complete user authentication lifecycle, integrating the Frontend signup/login forms, Backend authentication routes (/api/auth), controllers (authController.js), User model (User.js), database interactions, and Nodemailer for OTP/password reset emails.

Test Owner: Harsh Bhati

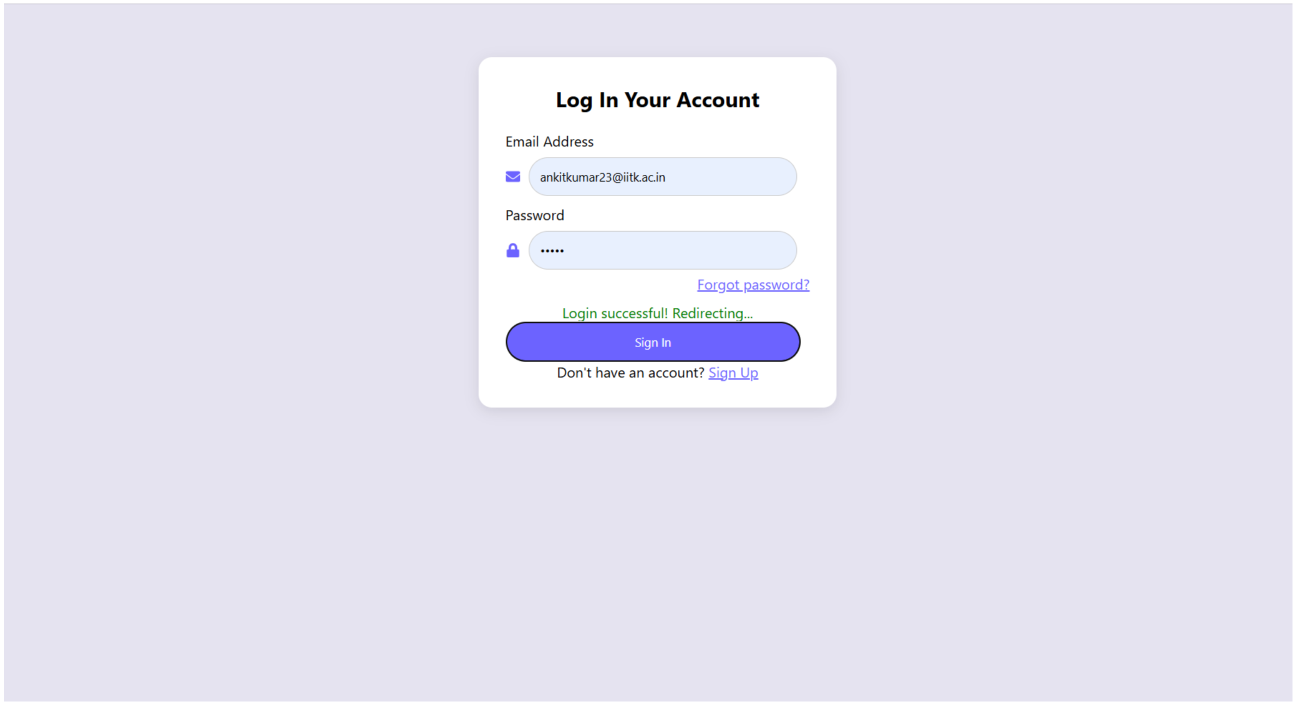
Test Date: 3/30/2025

- **Test Scenarios & Results:**
    - **Test Case 3.1.1: Successful New User Registration & Login:**
        - **Steps:**
            - Navigate to the Frontend Signup page (/signup).
            - Enter valid new user details (Name, IITK Email, Password).
            - Submit the signup form.
            - Verify backend receives data, creates a user record (unverified), generates OTP, and sends verification email via Nodemailer.
            - Navigate to the Email Verification page (/otp) using the userId passed from signup.
            - Enter the correct OTP received via email.
            - Submit OTP.
            - Verify backend validates OTP and marks the user as verified in the database.
            - Navigate to the Login page (/login).
            - Enter the registered email and password.
            - Submit login form.

Verify backend authenticates, generates JWT, and frontend receives token, storing it locally and redirecting to the Home page (/).

Test Results: The user is successfully registered, email verified, logged in, and redirected to the home page. JWT token is stored

Screenshot:


Log In Your Account

Email Address

ankitkumar23@iitk.ac.in

Password

•••••

Forgot password?

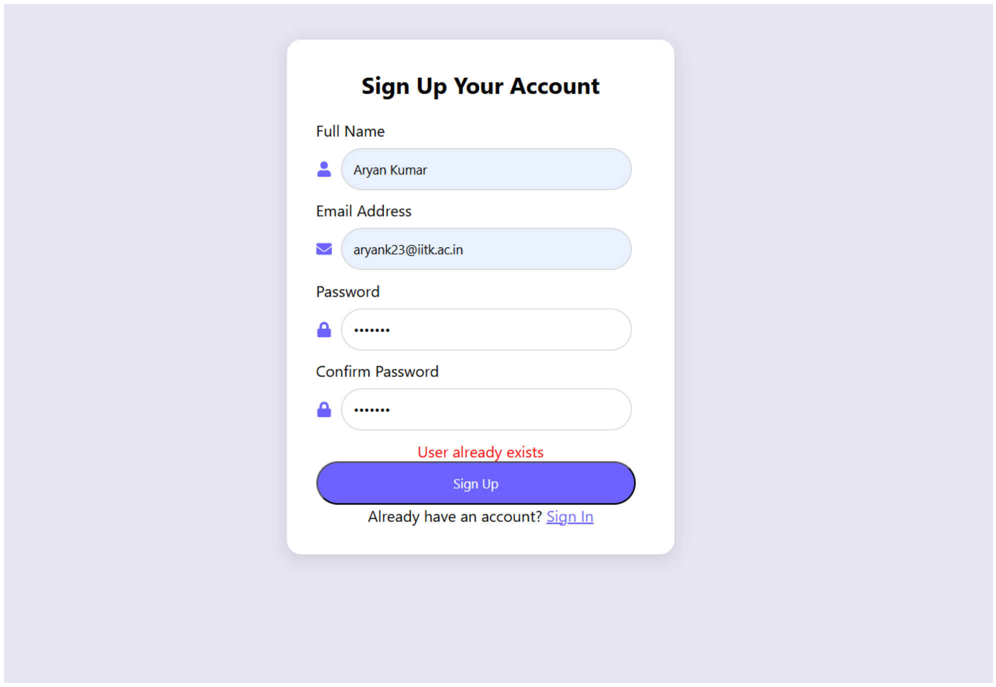
Login successful! Redirecting...

Sign In

Don't have an account? Sign Up

- **Test Case 3.1.2: Signup with Existing Email:**
  - **Steps:** Attempt signup with an email already present in the database.
  - **Result:** Frontend displays an error message "User already exists". Backend prevents duplicate user creation.

Screenshot:


Sign Up Your Account

Full Name

Aryan Kumar

Email Address

aryank23@iitk.ac.in

Password

•••••••

Confirm Password

•••••••

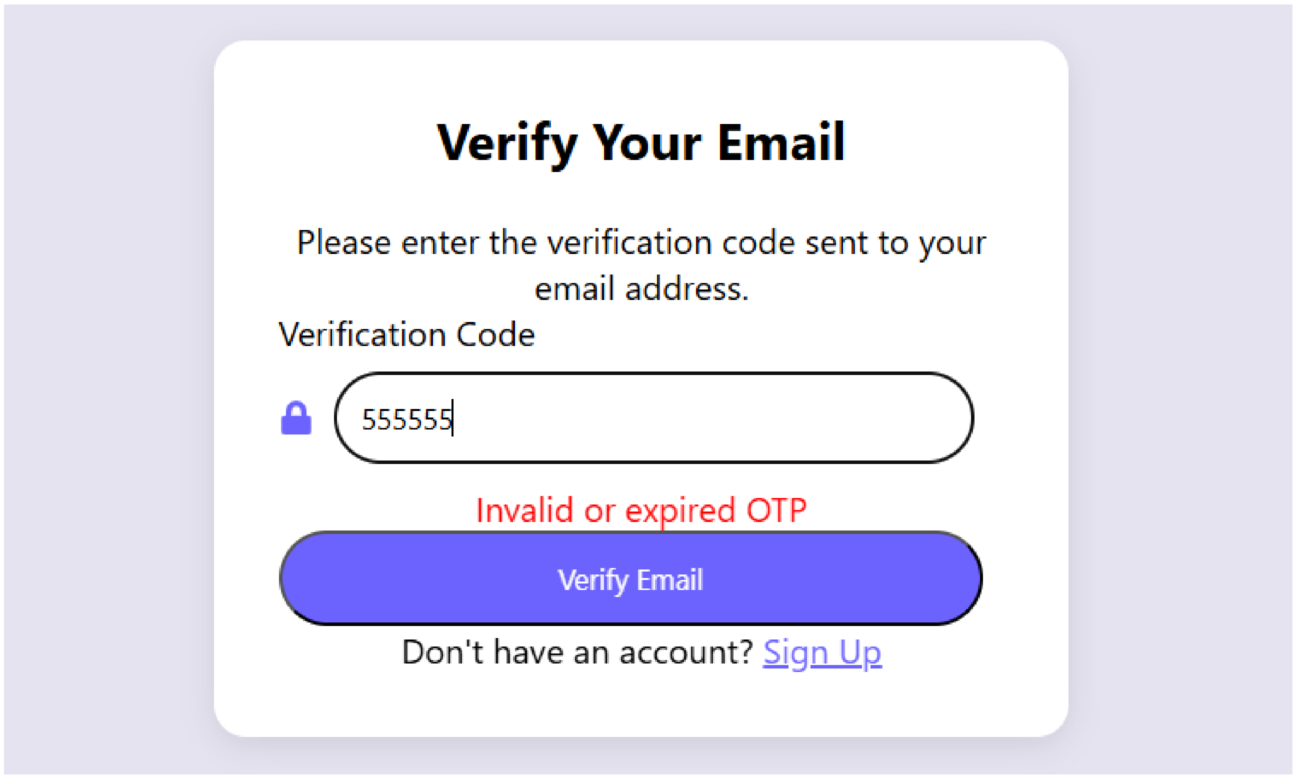
User already exists

Sign Up

Already have an account? Sign In

- **Test Case 3.1.3: Invalid OTP Verification:**
    - **Steps:** Enter an incorrect or expired OTP on the Email Verification page.
    - **Actual Result:** Frontend displays an error "Invalid or expired OTP". Backend rejects verification .

Screenshot:


Verify Your Email

Please enter the verification code sent to your email address.

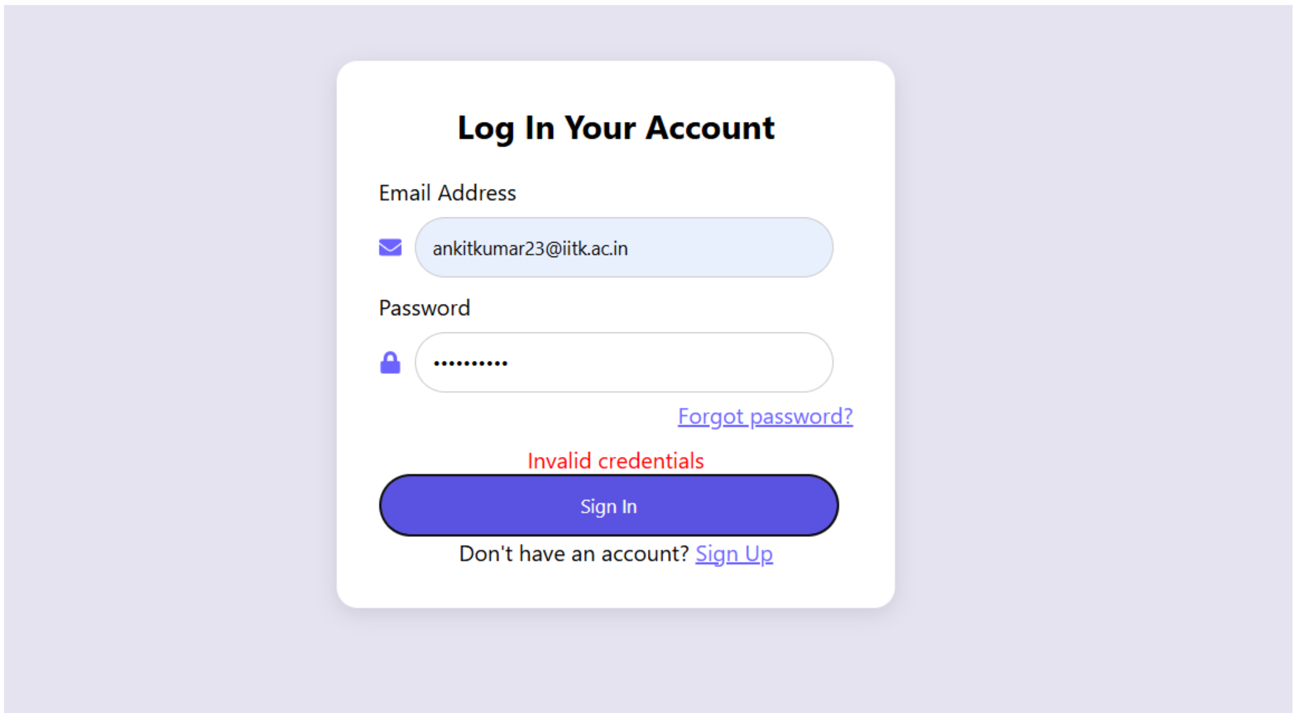
Verification Code

555555

Invalid or expired OTP

- **Test Case 3.1.4: Login with Invalid Credentials:**
    - **Steps:** Attempt login with incorrect password or non-existent email.
    - **Result:** Frontend displays error message "Invalid credentials". Backend denies login.

Screenshot:



Log In Your Account

Email Address

ankitkumar23@iitk.ac.in

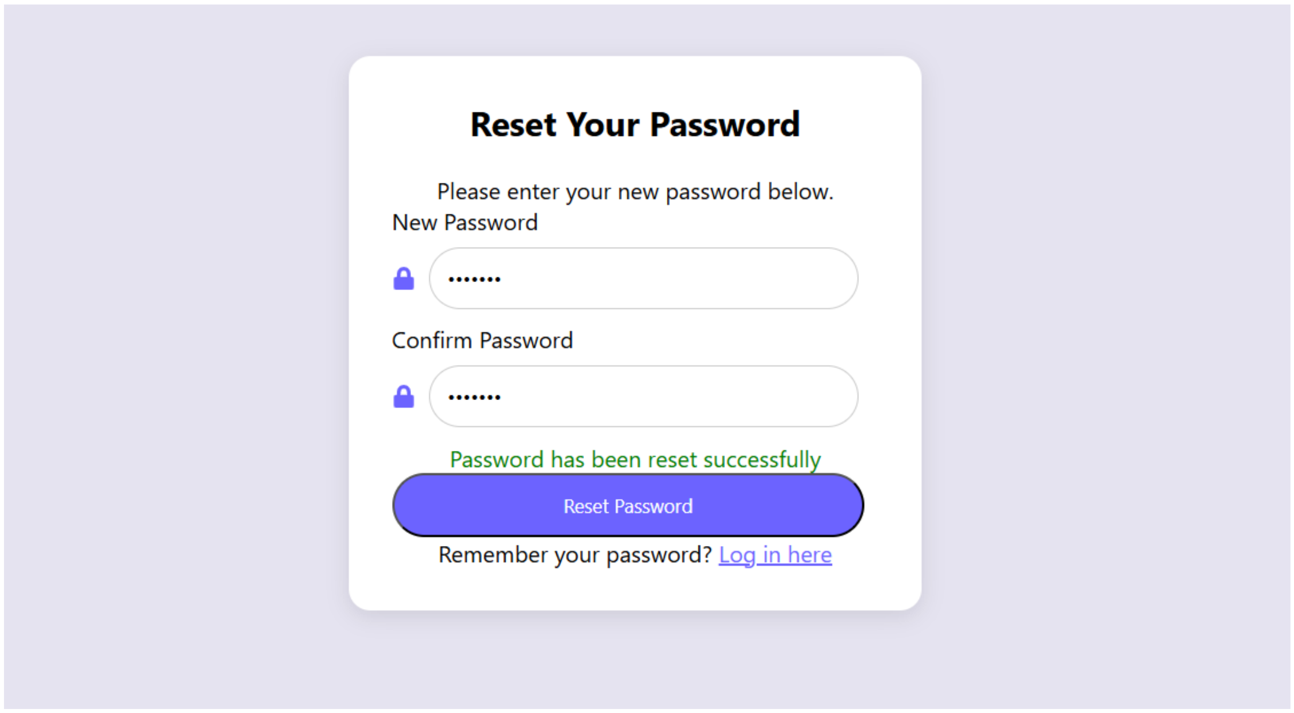
Password

••••••••••

Invalid credentials

- **Test Case 3.1.5: Forgot/Reset Password Flow:**
  - **Steps:**
    - Navigate to Forgot Password page (/forgot-password).
    - Enter a registered IITK email.
    - Submit the request.
    - Verify backend generates a reset token, stores its hash, and sends a password reset email via Nodemailer.
    - Click the reset link in the email (navigate to /reset-password?token=...).
    - Enter a new password and confirm it.
    - Submit the reset form.
    - Verify backend validates the token, updates the password hash in the database, and clears the reset token fields.
    - Attempt login with the new password.
  - **Actual Result:** Password reset email is sent, password is successfully updated, and user can log in with the new password.

Screenshot:



Reset Your Password

Please enter your new password below.

New Password

•••••••

Confirm Password

•••••••

Password has been reset successfully

Reset Password

Remember your password? Log in here

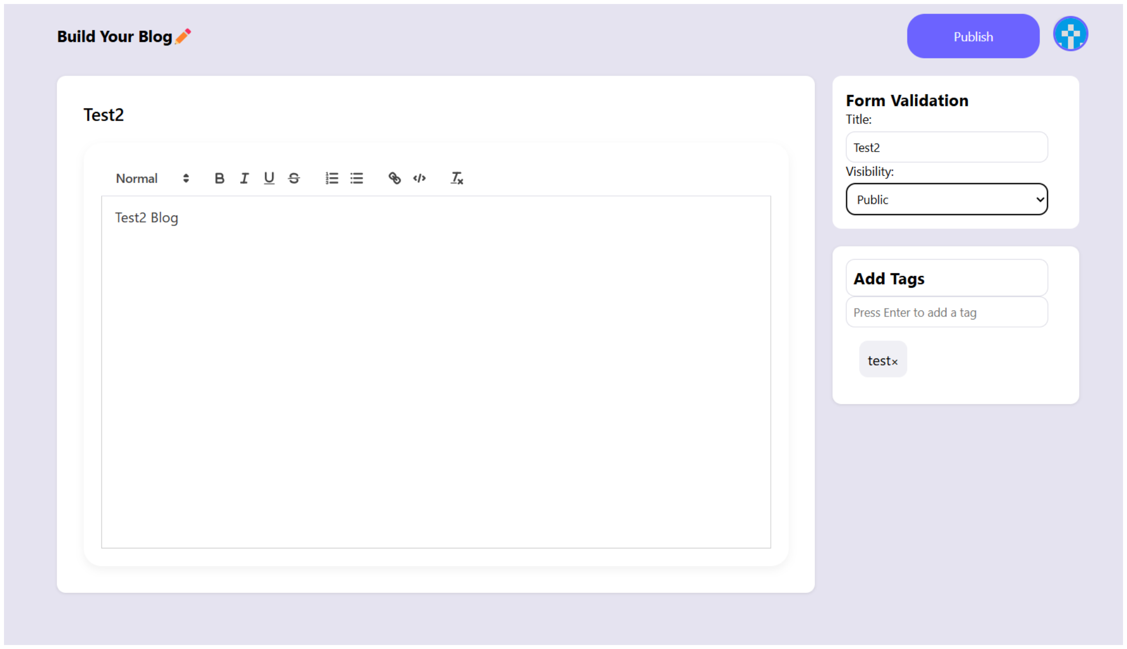
Blog management Flow

Module Details: Integration between Admin page and Blog components Tests the integration of Frontend components (Blog Writing, Blog Display, Blog Editor), Backend blog routes (/api/blog), controllers (blog.js), Blog model (Blog.js), User model (for author association, counts), Tag model (Tag.js), and database operations.
Test Owner: Harsh Bhati
Test Date: 03/31/2025

- **Test Scenarios & Results:**
  - **Test Case 3.2.1: Create and View Blog:**
    - **Steps:**
      - Log in as a user.
      - Navigate to the Blog Writing page (/write).
      - Enter Title, Content, and optionally select Visibility and add Tags.
      - Click Publish.
      - Verify frontend sends data to the backend /api/blog/post endpoint.
      - Verify backend creates a Blog document, associates it with the author (User document), updates user's blog count, creates/updates Tag documents, and returns the created blog data.
      - Navigate to the Home page (/) or the user's profile page.
      - Verify the newly created blog is displayed correctly with title, author, snippet, and tags.
      - Click on the blog to navigate to the Single Blog Page (/blog/:id).

- Verify the full blog content and details are displayed correctly.

▪ **Result:** Blog is created, stored in the database, associated with the author and tags, and displayed correctly on relevant frontend pages.
Screenshot:



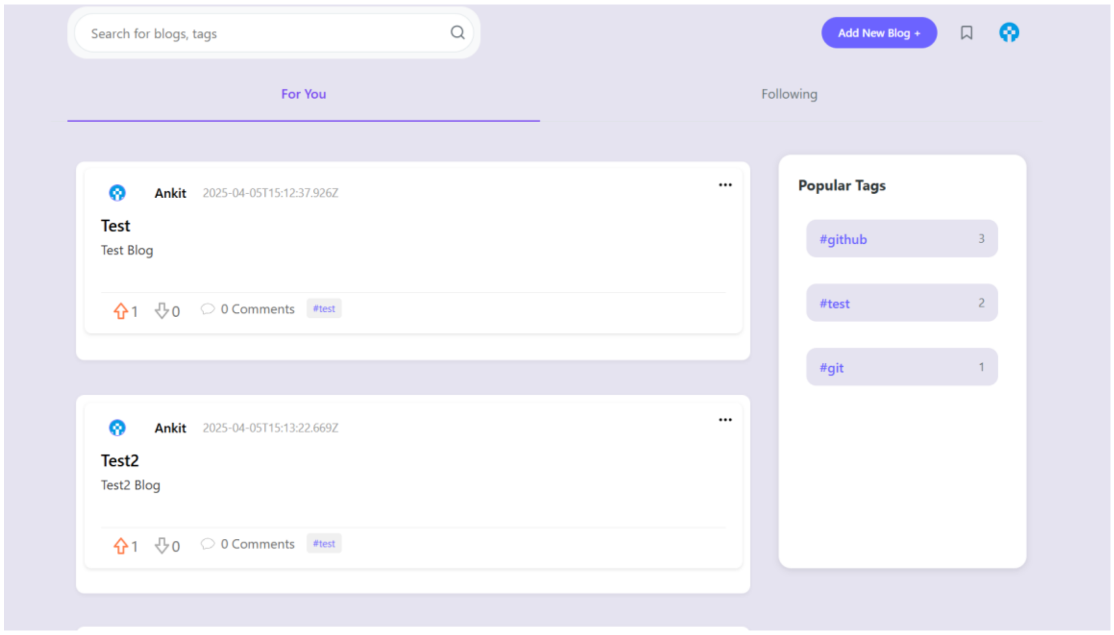
Test2

Test2 Blog

Title: Test2

Visibility: Public

test×



Ankit 2025-04-05T15:12:37.926Z

Test

Test Blog

1 0 0 Comments #test

Ankit 2025-04-05T15:13:22.669Z

Test2

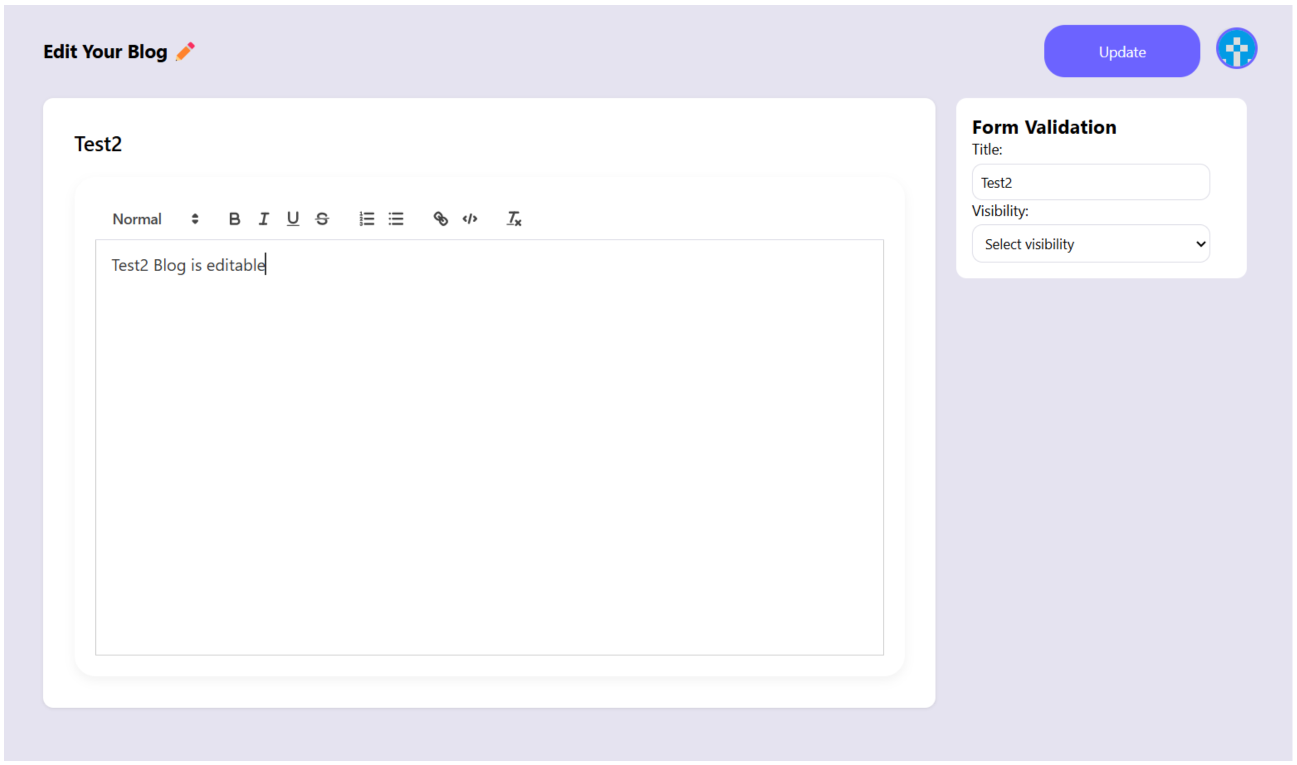
Test2 Blog

1 0 0 Comments #test

#github	3
#test	2
#git	1

- **Test Case 3.2.2: Edit Blog:**
  - **Steps:**
    - Log in as the blog's author.
    - Navigate to the blog post.
    - Click the "Edit" option (from the three-dot menu).
    - Navigate to the Blog Editor page (/edit-blog).
    - Modify the title and content.
    - Click "Update".
    - Verify frontend sends updated data to the backend /api/blog/:id (PATCH) endpoint.
    - Verify backend updates the Blog document in the database.
    - Verify the updated blog content is displayed on the Single Blog Page.
  - **Result:** Blog is successfully updated in the database and the changes are reflected on the frontend.

    Screenshot:



Test2

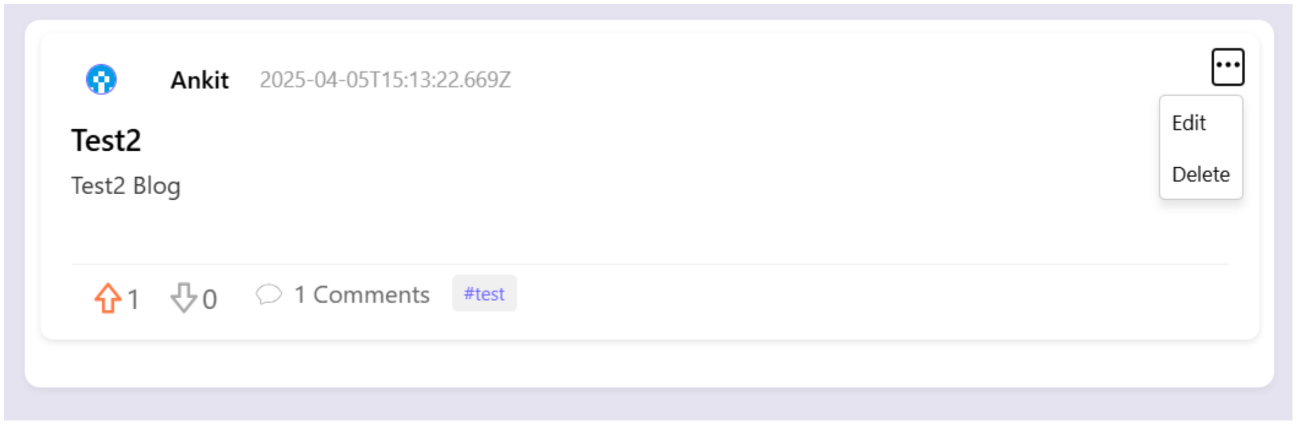
Test2 Blog is editable

Title: Test2

Visibility: Select visibility

- **Test Case 3.2.3: Delete Blog:**
  - **Steps:**
    - Log in as the blog's author or an admin.
    - Navigate to the blog post.
    - Click the "Delete" option (from the three-dot menu).
    - Confirm deletion.
    - Verify frontend sends request to the backend /api/blog/:id (SAVE) endpoint.
    - Verify backend deletes the Blog document, removes references from User (Blogs, SavedBlogs, likedBlogs), Tag documents, deletes associated Comment documents, and updates user's blog count.

        - Verify the blog is no longer visible on the frontend (Home, Profile, Search Results).
      - **Result:** Blog and associated data are removed from the database and frontend.



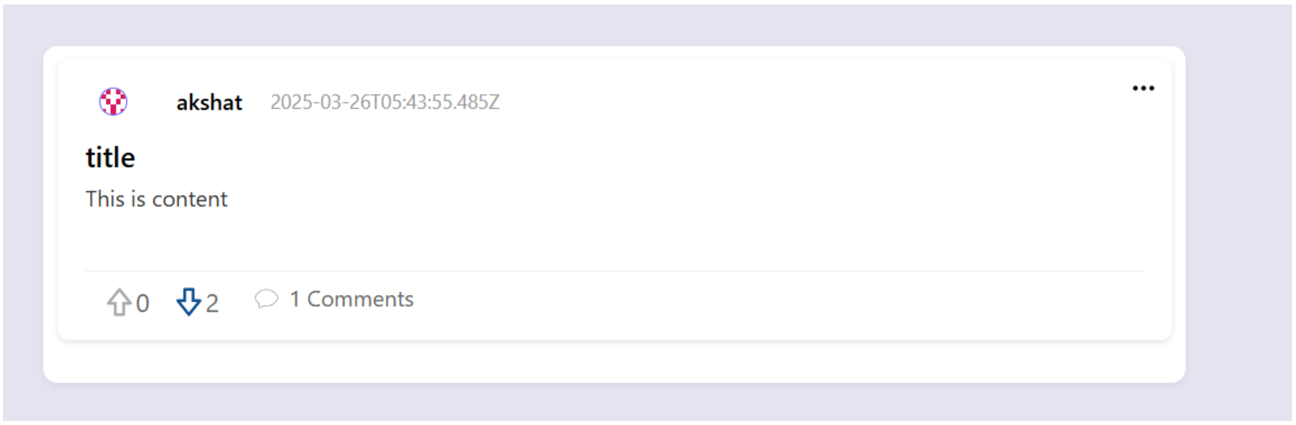
Blog Interaction Flow

Module Details: Tests interactions like voting, commenting, and saving, integrating Frontend components, Backend routes (/api/blog, /api/comment, /api/user), controllers, models (Blog, Comment, User), and database updates.
Test Owner: Harsh Bhati
Test Date: 03/31/2025

- **Test Scenarios & Results:**
  - **Test Case 3.3.1: Upvote/Downvote Blog:**
    - **Steps:**
      - Log in as user.
      - Navigate to a blog post.
      - Click the Upvote button.
      - Verify frontend sends request to /api/blog/:id/upvote.
      - Verify backend updates the Blog document (increments Upvote, adds user to upvoters, removes from downvoters if present), updates User's liked Blogs, and returns updated counts.
      - Verify frontend updates the vote counts and button state.
      - Click the Downvote button.
      - Verify frontend sends request to /api/blog/:id/downvote.
      - Verify backend updates the Blog document (increments Downvote, adds user to downvoters, removes from upvoters), updates User's liked Blogs, and returns updated counts.
      - Verify frontend updates the vote counts and button state.

- Click the same vote button again to undo the vote.
- Verify backend removes the vote and updates counts correctly.
- Verify frontend reflects the change.
- **Actual Result:** Votes are correctly registered, counts updated, user's liked list managed, and state reflected accurately on the frontend.

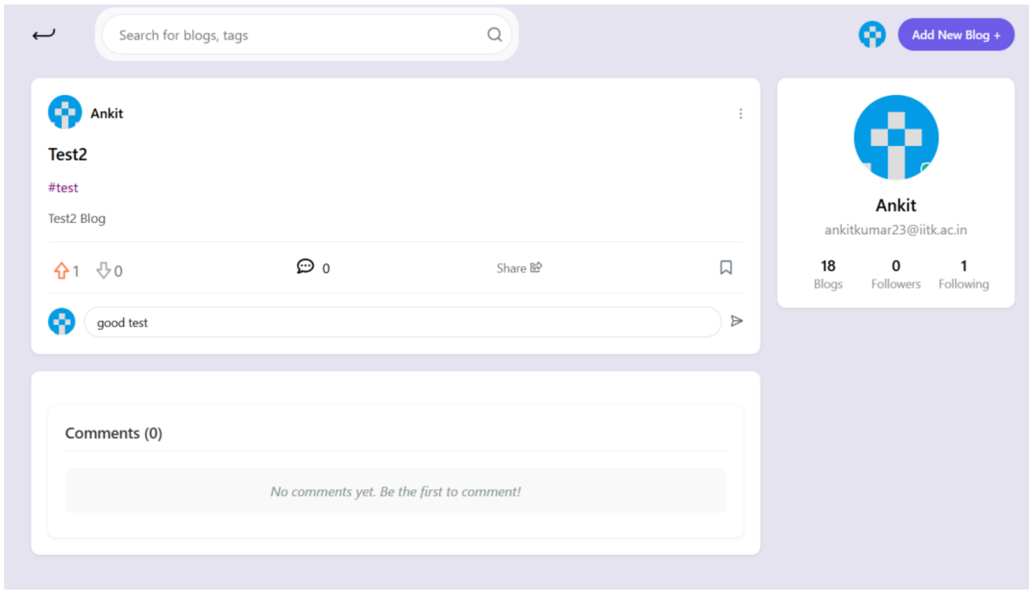
Screenshot:

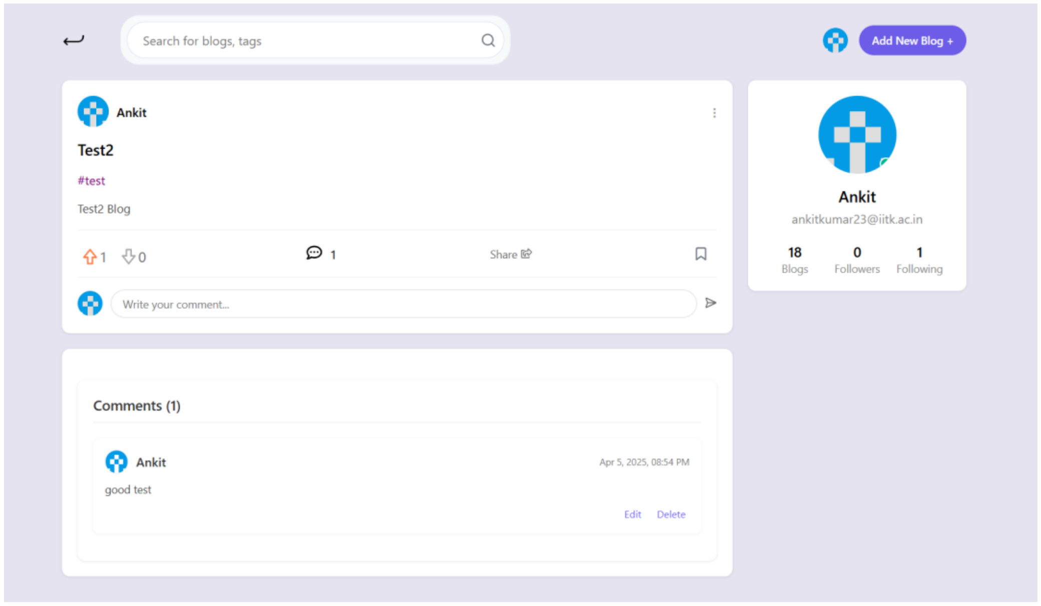

akshat 2025-03-26T05:43:55.485Z

title

This is content

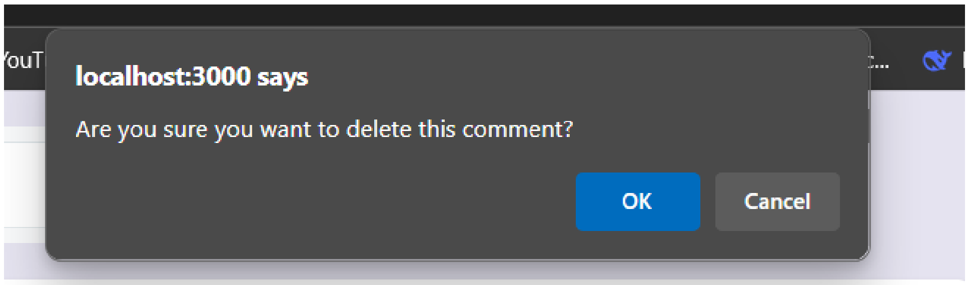
- **Test Case 3.3.2: Add Comment:**
  - **Steps:**
    - Log in as a user.
    - Navigate to a blog post (/blog/:id).
    - Enter text in the comment input field.
    - Click the Send button or press Enter.
    - Verify frontend sends comment data to /api/comment/post.
    - Verify backend creates a Comment document, associates it with the Blog and User, updates Blog's comments array, updates User's comments array, and returns the new comment.
    - Verify the new comment appears in the Comment List component on the frontend without a page refresh.
    - **Result:** Comment is saved to the database and displayed dynamically on the frontend.
      Screenshot:



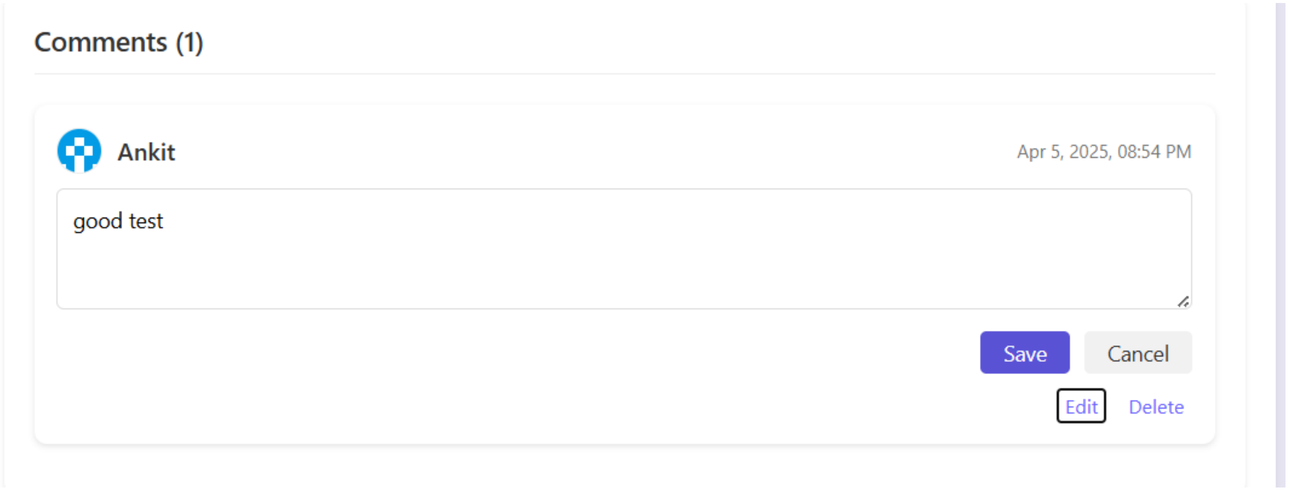


- **Test Case 3.3.3: Edit/Delete Own Comment:**
  - **Steps:**
    - Log in as the user who posted a comment.
    - Navigate to the blog post containing the comment.
    - Click "Edit" on the comment.
    - Modify the comment text and click "Save".
    - Verify frontend sends update request to /api/comment/:id (PATCH).
    - Verify backend updates the Comment document.
    - Verify the frontend displays the updated comment.
    - Click "Delete" on the comment and confirm.
    - Verify frontend sends delete request to /api/comment/:id (DELETE).

    - Verify backend deletes the Comment document and removes references from Blog and User.
    - Verify the comment disappears from the frontend list.
  - **Result:** User can edit and delete their own comments; changes are reflected in the database and frontend.
    Screenshot:


localhost:3000 says

Are you sure you want to delete this comment?

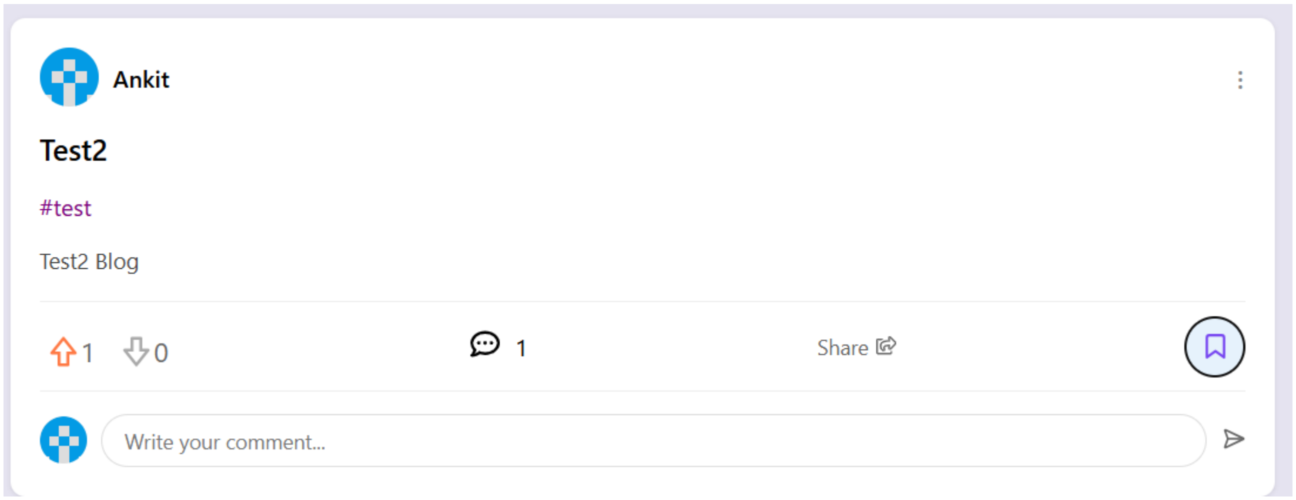

Comments (1)

Ankit Apr 5, 2025, 08:54 PM

good test

- **Test Case 3.3.4: Save/Unsave Blog:**
  - **Steps:**
    - Log in as a user.
    - Navigate to a blog post.
    - Click the Bookmark icon.
    - Verify frontend sends request to /api/user/:id/saveBlogs.
    - Verify backend updates the User's SavedBlogs array and the Blog's SavedBy array.
    - Verify the bookmark icon state changes on the frontend.
    - Navigate to the Saved Blogs view (e.g., via profile or header icon).
    - Verify the saved blog appears in the list.
    - Click the Bookmark icon again on the blog post to unsave.
    - Verify backend removes the blog from User's SavedBlogs and Blog's SavedBy.
    - Verify the icon state changes and the blog is removed from the Saved Blogs view.
  - **Result:** Blogs can be saved and unsaved, and the saved list is accurately maintained and displayed.

Screenshot:



Test2

#test

Test2 Blog

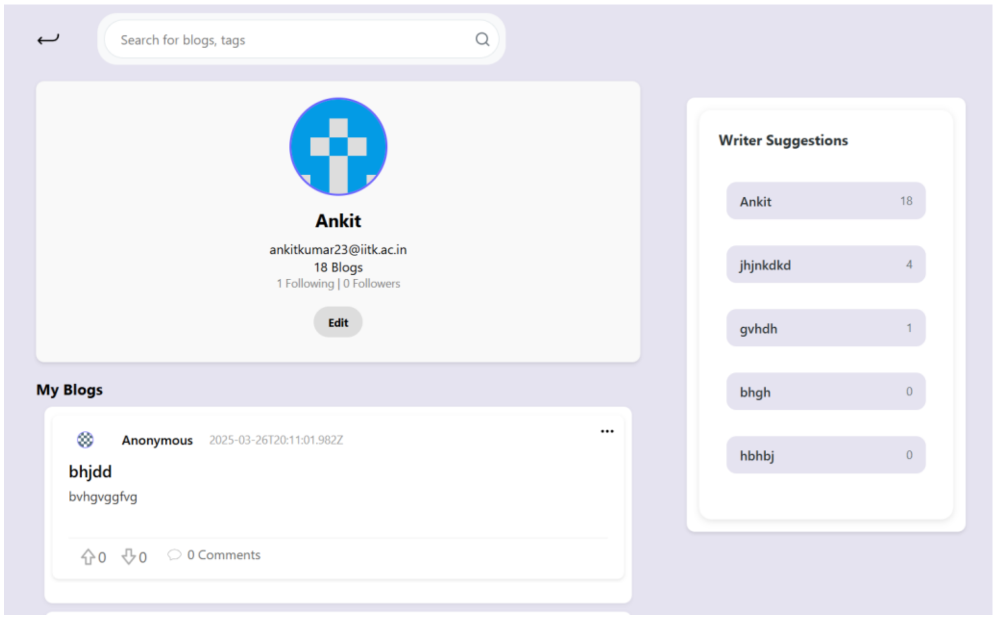
1 0 1

User Profile and Social Features

- **Module Details**: Tests viewing profiles and the follow/unfollow mechanism, integrating Frontend profile pages, Backend user routes (`/api/user`), controllers (`user.js`), User model, and database operations.
  Test Owner: Harsh Bhati
  Test Date: 04/01/2025

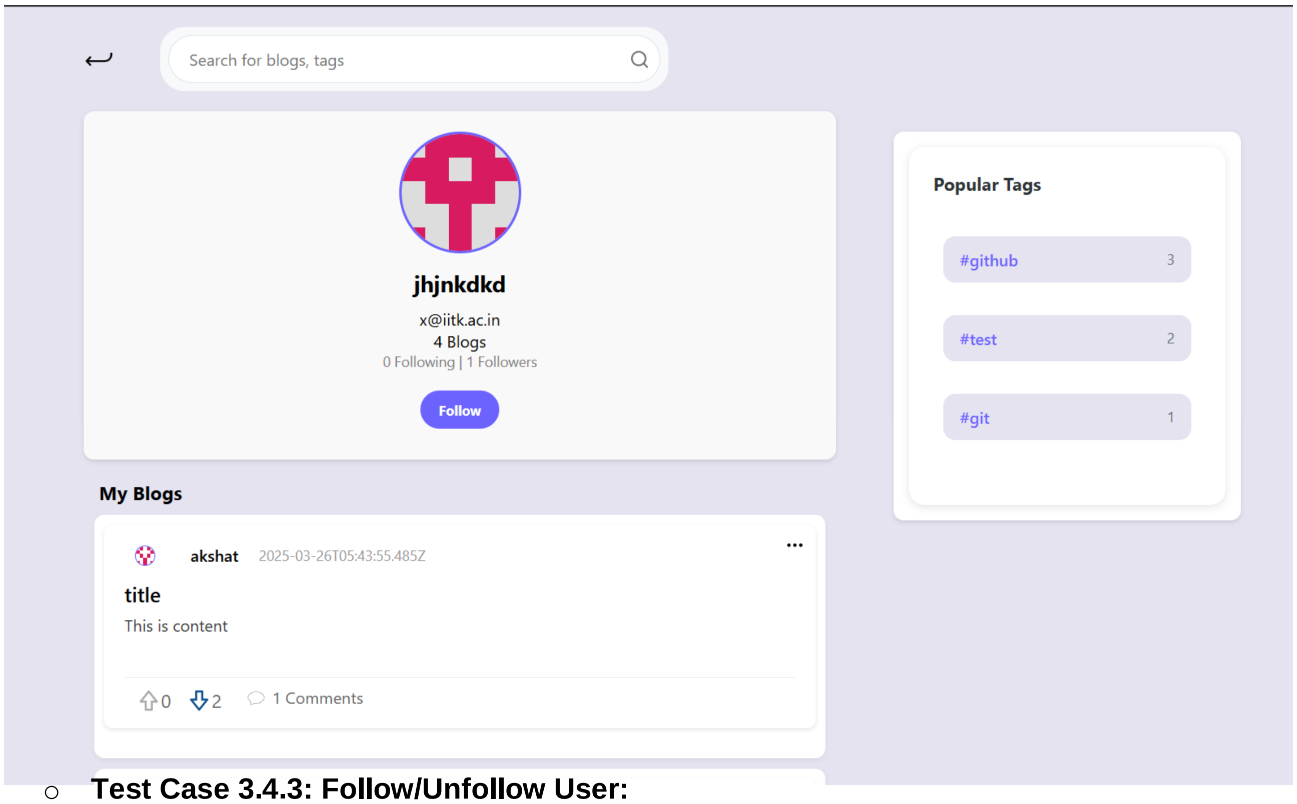
- **Test Scenarios & Results:**
  - **Test Case 3.4.1: View Own Profile:**
    - **Steps:**
      - Log in as a user.
      - Navigate to the User Profile page (/user-profile).
      - Verify frontend fetches data for the logged-in user from /api/user/:id.
      - Verify correct user details (Name, Email, Bio, Counts) are displayed.
      - Verify the user's own blogs are listed.
      - Verify the "Edit" button is present.
    - **Actual Result:** User can view their own profile information and blogs accurately.

Screenshot:



- **Test Case 3.4.2: View Other User's Profile:**
  - **Steps:**
    - Log in as User A.
    - Navigate to the profile page of User B (/account/:id).
    - Verify frontend fetches data for User B from /api/user/:id.
    - Verify User B's public details (Name, Email, Bio, Counts) are displayed.
    - Verify User B's public blogs are listed.
    - Verify the "Follow"/"Following" button is present.
  - **Actual Result:** User can view another user's public profile information and blogs.

Screenshot:



Search for blogs, tags

jhjnkdkd

x@iitk.ac.in

4 Blogs

0 Following | 1 Followers

Follow

Popular Tags

#github	3
#test	2
#git	1

My Blogs

akshat 2025-03-26T05:43:55.485Z

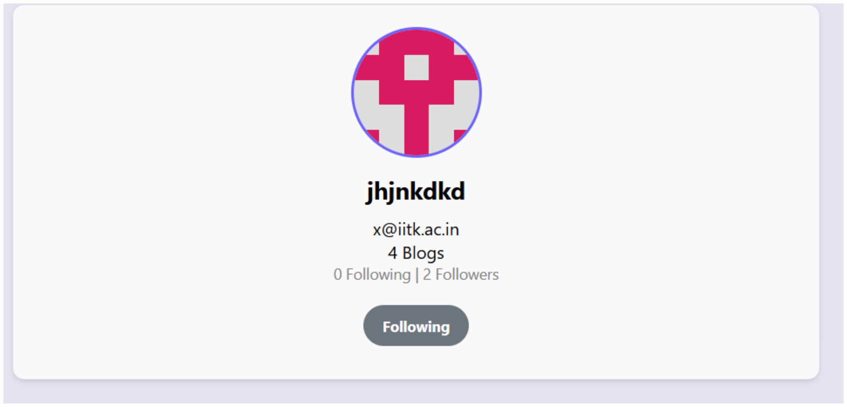
title

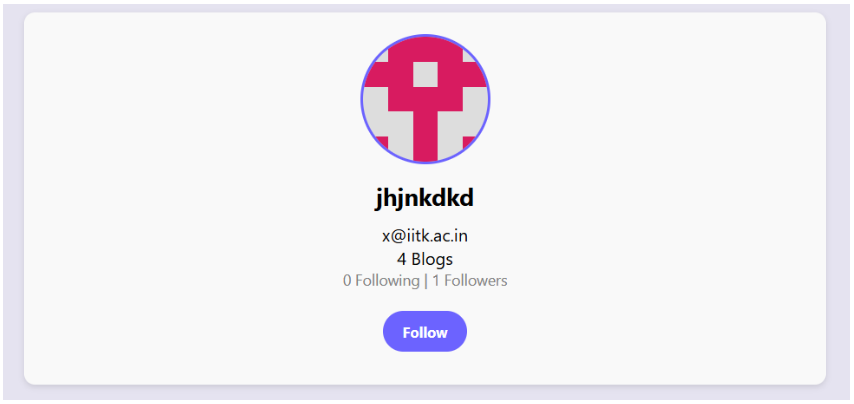
This is content

0 2 1 Comments

- **Test Case 3.4.3: Follow/Unfollow User:**
  - **Steps:**
    - Log in as User A.
    - Navigate to User B's profile page.
    - Click the "Follow" button.
    - Verify frontend sends request to /api/user/:id/follow (with User B's ID in the body).
    - Verify backend updates User A's following list/count and User B's follower's list/count.
    - Verify the button text changes to "Following" on the frontend.
    - Navigate to the Home page (/) and switch to the "Following" tab.
    - Verify User B's recent blogs appear in the feed.
    - Navigate back to User B's profile and click "Following".
    - Verify backend updates lists/counts for unfollow.
    - Verify button text changes back to "Follow".
    - Verify User B's blogs are removed from User A's "Following" feed.
  - **Expected Result:** Follow/unfollow actions update database correctly, button state changes, and the "Following" feed is updated.
  - **Actual Result:** Follow/unfollow actions update database correctly, button state changes, and the "Following" feed is updated.

    Screenshot:





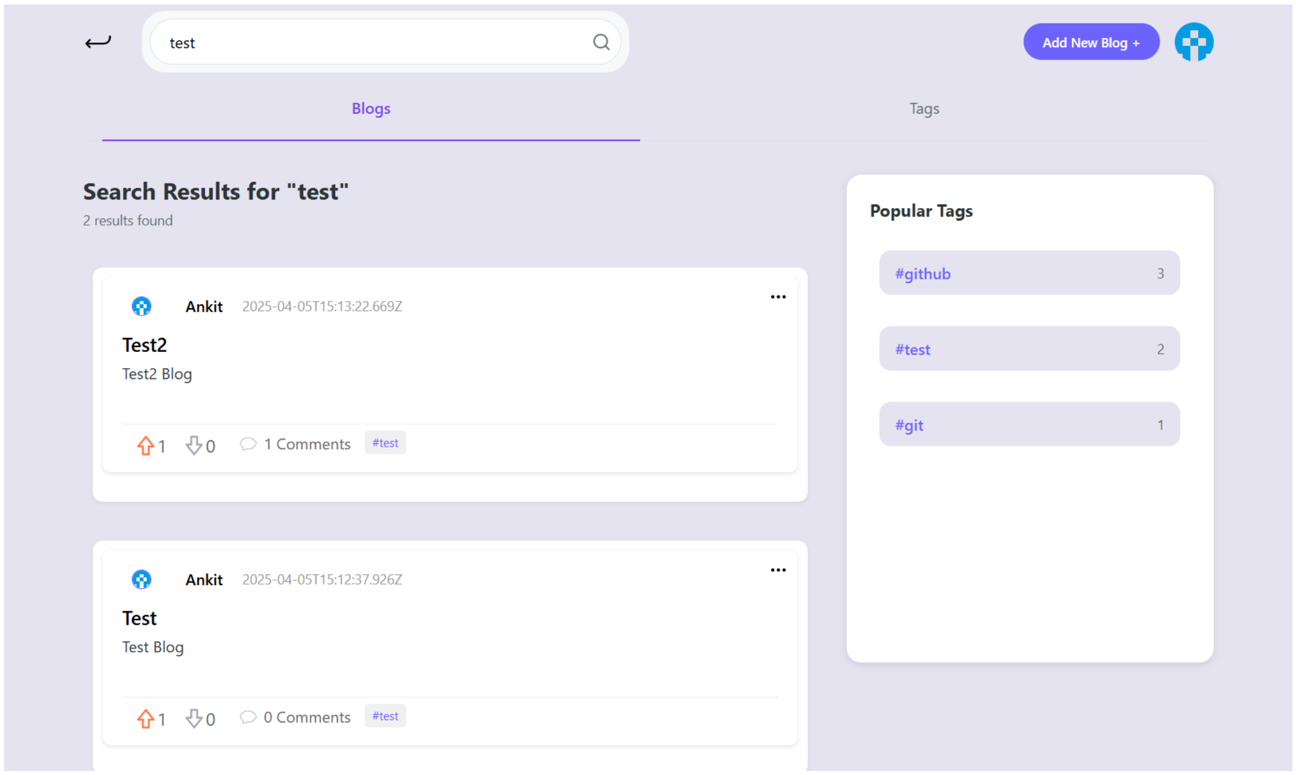
Search Functionality

- **Module Details**: Tests viewing profiles and the follow/unfollow mechanism, integrating Frontend profile pages, Backend user routes (`/api/user`), controllers (`user.js`), User model, and database operations.
  Test Owner: Harsh Bhati
  Test Date: 04/02/2025

- **Test Scenarios & Results:**
  Test Case 3.5.1: Search for Blogs:
  - **Steps:**
    - Log in as a user.
    - Enter a search term (matching blog title, content, or author name) in the header search bar.
    - Press Enter or click the search icon.
    - Verify frontend navigates to `/search?query=...` and sends request to `/api/blog/search`.

- Verify backend performs a search (e.g., text index search or regex on relevant fields) and returns matching blogs.
- Verify the Search Results page displays the correct matching blogs.

- **Expected Result:** Search returns relevant blogs based on the query, displayed on the results page.
- **Actual Result:** [Passed/Failed - Add details]
  Screenshot:


test

Add New Blog +

Blogs Tags

Search Results for "test"

2 results found

Ankit 2025-04-05T15:13:22.669Z

Test2

Test2 Blog

1 0 1 Comments #test

Ankit 2025-04-05T15:12:37.926Z

Test

Test Blog

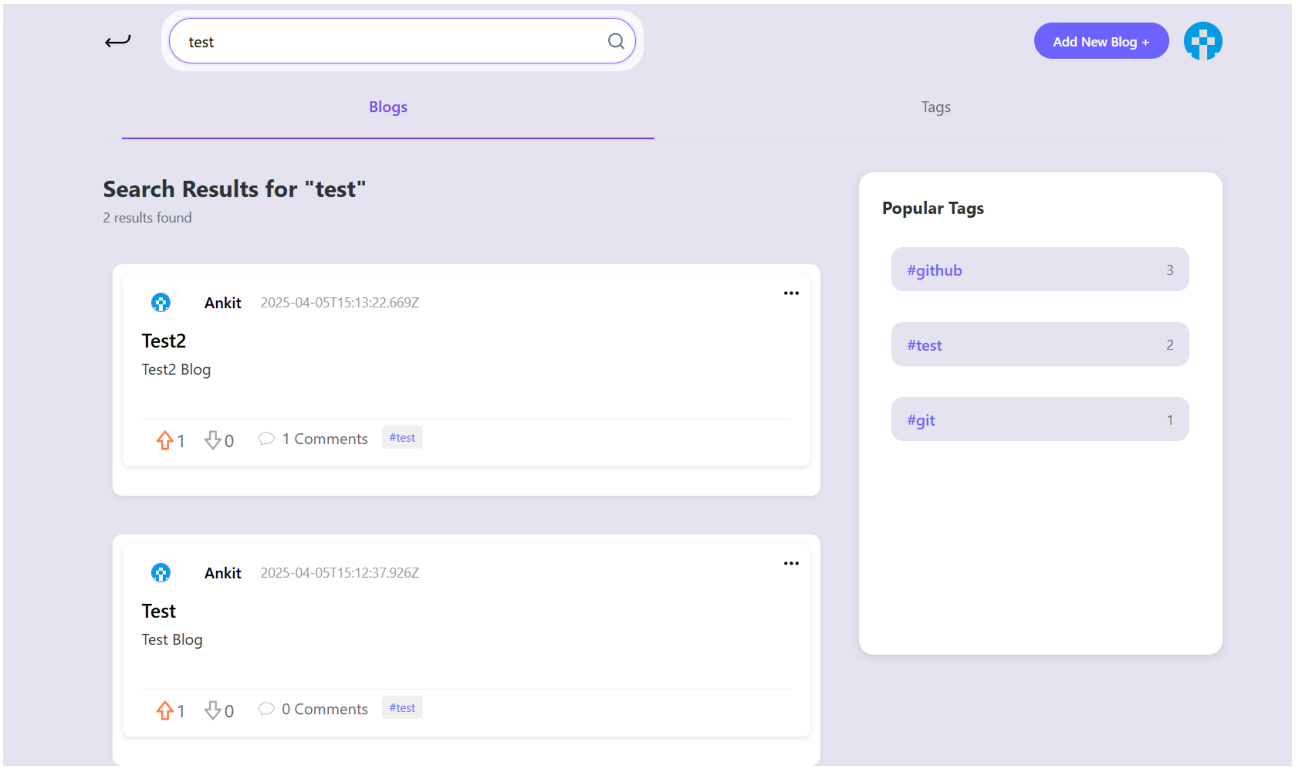
1 0 0 Comments #test

Popular Tags

#github	3
#test	2
#git	1

- **Test Case 3.5.2: Search for Tags:**
  - **Steps:**
    - Log in as a user.
    - Enter a tag name in the header search bar.
    - Press Enter or click the search icon.
    - On the Search Results page, switch to the "Tags" tab.
    - Verify frontend sends request to `/api/tag/search`.
    - Verify backend searches for tags matching the name and returns associated blogs.
    - Verify the Search Results page (Tags tab) displays blogs associated with the matching tag.
  - **Result:** Search returns blogs associated with the queried tag

Screenshot:



Search Results for "test"
2 results found

Ankit 2025-04-05T15:13:22.669Z
Test2
Test2 Blog
1 0 1 Comments #test

Ankit 2025-04-05T15:12:37.926Z
Test
Test Blog
1 0 0 Comments #test

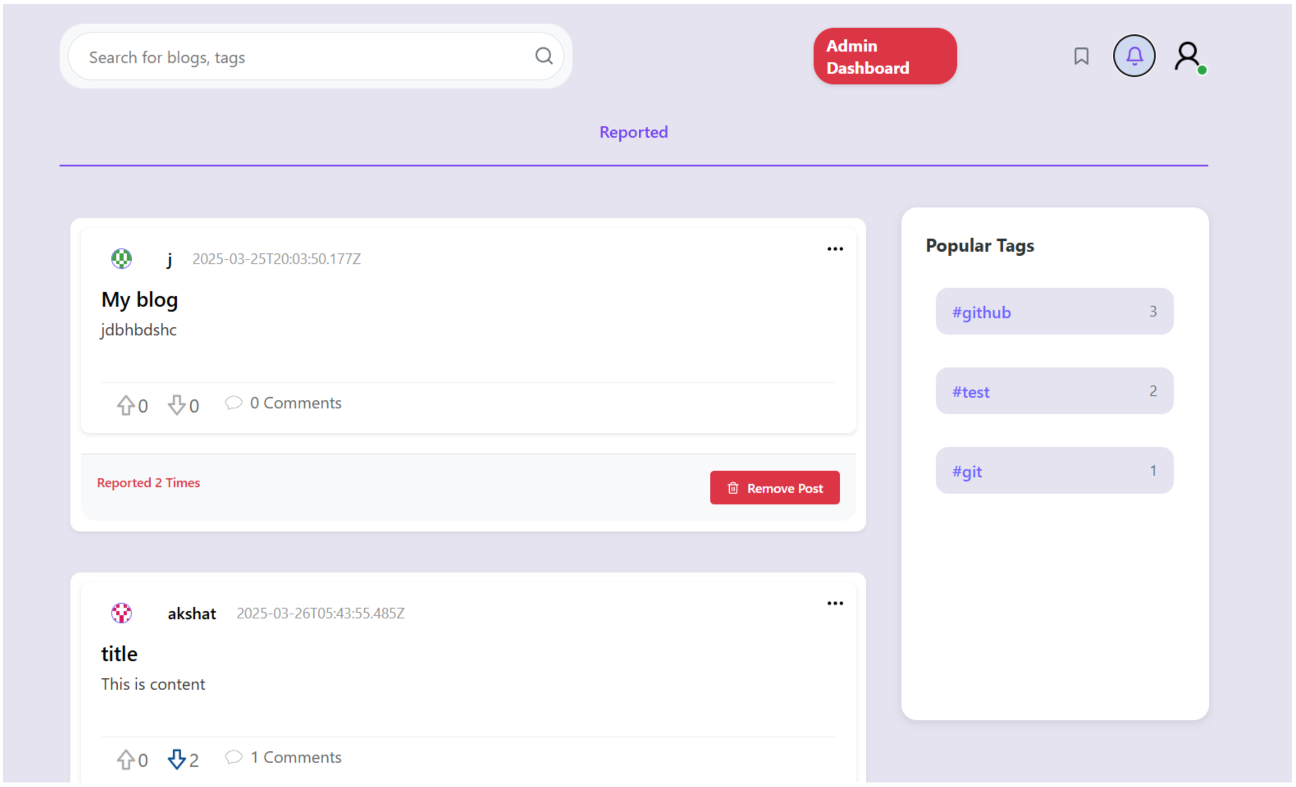
Popular Tags

#github	3
#test	2
#git	1

Admin functionality

- **Module Details**: Tests viewing profiles and the follow/unfollow mechanism, integrating Frontend profile pages, Backend user routes (`/api/user`), controllers (`user.js`), User model, and database operations.
  Test Owner: Durbasmriti Saha
  Test Date: 04/02/2025

- **Test Scenarios & Results:**
  - **Test Case 3.6.1: View Reported Blogs:**
    - **Steps:**
      - Log in as an admin user.
      - Navigate to the Admin Dashboard (`/admin`).
      - Access the "Reported" view (e.g., via header icon or tab).
      - Verify frontend sends request to `/api/blog/reported`.
      - Verify backend uses admin middleware, queries for blogs with `ReportCount > 0`, and returns them.
      - Verify the Admin Dashboard displays the reported blogs, potentially showing the report count.
    - **Result:** Admin can view a list of blogs reported by users.
    - **Screenshot:**



j 2025-03-25T20:03:50.177Z

My blog

jdbhbdshc

0 0 0 Comments

Reported 2 Times

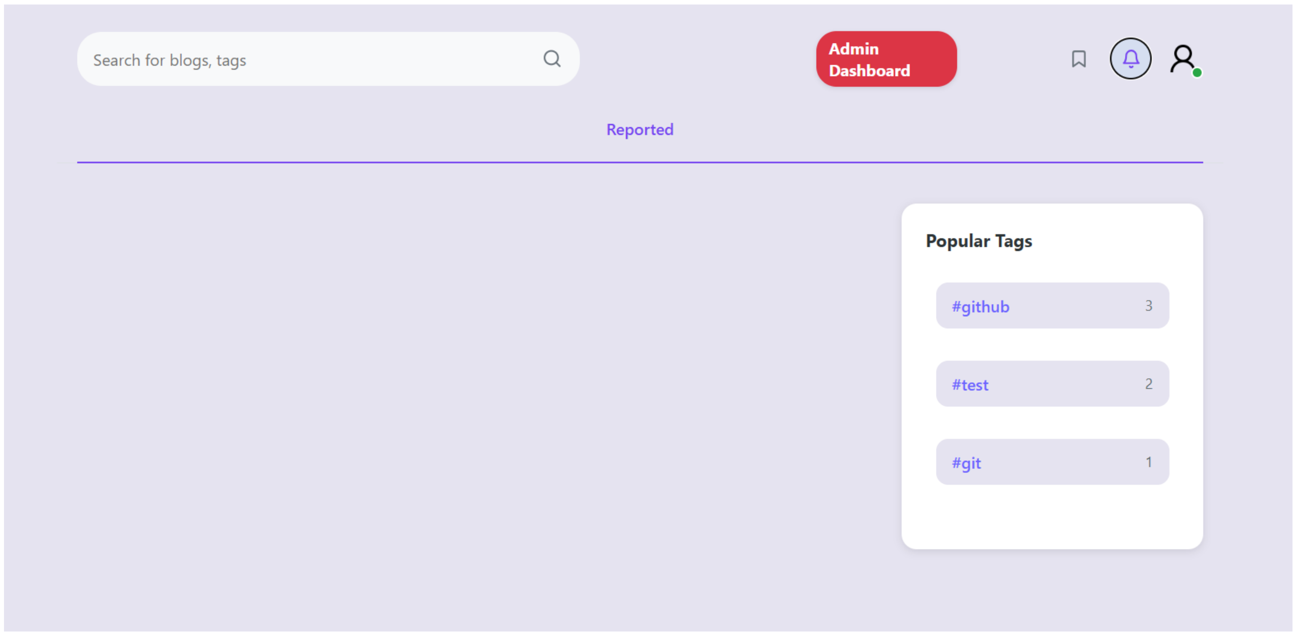
akshat 2025-03-26T05:43:55.485Z

title

This is content

0 2 1 Comments

- **Test Case 3.6.2: Admin Deletes Blog:**
  - **Steps:**
    - Log in as an admin user.
    - Navigate to the Admin Dashboard and view reported blogs (or any blog).
    - Click the "Remove Post" or similar delete button for a specific blog.
    - Confirm deletion.
    - Verify frontend sends DELETE request to `/api/blog/:id`.
    - Verify backend uses admin middleware and deletes the blog and associated data as described in Test Case 3.2.3.
    - Verify the blog is removed from the Admin Dashboard view and is inaccessible elsewhere.
  - **Result:** Admin can successfully delete any blog post.
    Screenshot:


Search for blogs, tags
Admin Dashboard
Reported
Popular Tags
#github 3
#test 2
#git 1

4 System Testing

1. Requirement: The user must have an IITK webmail to sign up and later login through the same credentials.

Both the sign-up and login functionality were tested and found to work properly. Various possible scenarios were created and tested completely to ensure their proper functioning. These test cases were tested manually, so all the test mentioned above were accurate and robust. We even intentionally put invalid inputs, credentials of non-registered user and incorrect credentials combination to see if the responses were as expected. The appropriate error message got displayed so that user know of what is going wrong.

Test Owner: Aryan Kumar
Test Date: 04/02/2025
Test Results: The sign-up and login page are working perfectly
Additional Comments: NA

2. Requirement: The user can reset their password through "Forgot password" functionality. Authentication shall be carried out through password reset mail on the IITK webmail that they used for signup.

The "Forgot password" functionality is very crucial in case user forgets their password or want to reset it for privacy concerns. The authentication is carried out through password reset mail. A test case is completely dedicated to see if this functionality is working properly. Additionally, functionality is only for already registered users otherwise it will show error.
Test Owner: Durbasmriti Saha
Test Date: 04/03/2025
Test Results: There is no error in sending the password reset mail and the password is being reset properly through that mail.
Additional Comments: NA

3. Requirement: The system allows users to choose tags according to their interest to search blogs related to that topic and at the time of writing blogs.
Both To make the search functionality more accessible and user friendly, the concept of choosing tags has been implemented. By this feature, instead of searching blogs by their title, the user can now search blogs of their interests also just by choosing tags from the sidebar. A dedicated test case has been executed to ensure that is working properly. This feature is extended for writers too. The writers can assign tags to their written blogs which will make easier for other users to search up for the blogs.

Test Owner: Harsh Bhati
Test Date: 04/03/2025

Test Results: Searching through tags and using tags during writing blog are working completely fine.
Additional Comments: NA

4. Requirement: The user can interact with other users through comments and upvotes and downvotes under blogs in order to share their thoughts.

We already tested the working of “comment’, “upvotes” and “downvotes’” under every blogs. The system also counts the number of comments, upvotes and downvotes under each blog correctly which helps the writer know about its reach and popularity. The others can also reply to someone else comments in threaded format. The users can also delete their comments. Overall, this feature is working perfectly fine.

Test Owner: Durbasmriti Saha

Test Date: 04/03/2025

Test Results: The users can comment, upvote, downvote to any blog.

5. Requirement: The system allows users to create and manage their blog posts using a rich text editor.

The blog post creation feature was tested thoroughly. Posts with formatted text, and links were created and verified. Draft saving, editing, and deletion features were also tested. The visibility of published posts across other user accounts was confirmed. Invalid file formats and empty submissions were tested to ensure proper error handling.

Test Owner: Ansh Adarsh

Test Date: 04/03/2025
Test Results: The blog post editor and management features are working as expected
Additional Comments: NA

6. Requirement: Users can choose their fields of interest (tags) and see related posts.

Users selected different tags during profile setup and post creation. The system was tested to ensure posts tagged with user-selected fields appeared on their feed. Users were able to successfully tag new posts and see relevant questions based on interests.

Test Owner: Lavish Kanwa

Test Date: 04/02/2025
Test Results: The user can now choose their fields of interest (tags).
Additional Comments: NA

7. Requirement: The Admin can review the reported blogs reported by some user. This feature makes sure that anyone's sentiment is not hurt, and blogs follow basic human values.

The system gives the power to the Admin to review any blogs/posts and delete if felt necessary. This ensures that no user is offended by any post. The admin can also see how many times a post was reported which is an essential information to take the final decision. We have executed test cases for both review and delete functionality and ensured that they are working quite perfectly. The number of reporting is also being counted correctly.

Test Owner: Lokesh Kumar

Test Date: 04/02/2025
Test Results: The admin can view the reported blogs perfectly fine and can delete the post if he/she thinks so.
Additional Comments: NA

8. Requirement: The system allows users to search for posts and other users.

Users tested the search functionality with full and partial keywords for blog titles, tags, and usernames. The results were relevant and ranked well. Scenarios with no matches returned appropriate messages. Case sensitivity and special character handling were tested.

Test Owner: Gone Nishanth

Test Date: 04/02/2025
Test Results: Search functionality is working correctly and is responsive.
Additional Comments: NA

9. Requirement: The users can follow other users making the website more personalized.

One user can follow other users. Every users have their "Following" and "Followers" list. The test cases mentioned above ensures that those features are working completely fine. The counting is also accurate.

Test Owner: Aryan Kumar

Test Date: 04/04/2025
Test Results: The user can view their "Following" and "Followers" list by visiting their profile.
Additional Comments: NA

10. Requirement: The writers/users can edit their blogs after the blog has already been posted.

This feature is very helpful as it allows to rectify any mistakes made during writing but still got posted. The writer can edit the blog they have already posted. Test case has been executed to see its proper working. The writer has to click on the ‘three dot’ options to select ‘edit’ and afetr editing has been done, the writer can ‘update’ the changes made.

Test Owner: Durbasmriti Saha

Test Date: 04/04/2025
Test Results:
Additional Comments: The writers have no choice to ‘draft’ the edited blog if it was posted earlier. They have to post it after editing.

11. Requirement: The users can Save a blog for future reference.

The user may want to look on a blog later. This feature is useful in that case. The saved blogs can be accessed from “Saved Blogs” section. One test case has been executed to ensure its proper functionality.

Test Owner: Ansh Adarsh

Test Date: 04/04/2025
Test Results: The blogs are saved perfectly and can be accessed from ‘save’ section in top right.
Additional Comments: NA

Conclusion

How Effective and exhaustive was the testing?

Testing was comprehensive as we ensured all major functionalities of ECHO worked as intended. Our testing covered the complete user journey from blog viewing to posting, and administrative controls. Through unit testing of individual components, integration testing of feature interactions, and system testing of complete workflows, we validated both frontend interfaces and backend API functionality.

Which components have not been tested adequately?

Most components in ECHO have been tested thoroughly, including user authentication, blog operations, and admin functionalities. Only minor aspects like error handling for failed API requests and some edge cases in the admin warning system could benefit from additional testing coverage.

What difficulties have you faced during testing?

The main challenges included testing user authentication flows and ensuring proper state management across different user roles. Integration testing between frontend and backend components was time-consuming due to the need to simulate various user scenarios. Testing features that required specific user states or conditions (like reported posts or saved blogs) needed complex test setup and environment configuration.

How could the testing process be improved?

Implementing a comprehensive automated testing pipeline would significantly enhance our testing process. Adding end-to-end testing tools would better validate complete user journeys. Creating a dedicated testing environment with mock data would make integration testing more efficient. Better documentation of test cases and expected behaviors across all components would help standardize the testing process. Additionally, implementing continuous integration testing would ensure new features don't break existing functionality.

Appendix A - Group Log

DATE	TIME	MEMBERS PRESENT	DESCRIPTION
28 March 2025	6:30 P.M.	Ansh, Aryan, Nishanth, Harsh, Govind, Lavish, Lokesh, Durba, Someshwar	Team gathered to define the testing scope and assign roles for the documents.
30 March 2025	10 P.M.	Ansh, Aryan, Nishanth, Harsh, Govind, Lokesh, Durba, Someshwar	Developers presented fixes for previously reported bugs. Team retested features and noted improved application stability.
2 April 2025	3 P.M.	Ansh, Aryan, Nishanth, Harsh, Govind, Lavish, Lokesh, Durba	Checked the progress of testing and identified the remaining testing phases to be completed and reviewed the status of testing documentation.
4 April 2025	6:30 P.M.	Ansh, Aryan, Nishanth, Harsh, Govind, Lavish, Lokesh, Durba, Someshwar	Conducted the final round of testing, and confirmed the application is ready for deployment.