Blogging Platform Documentation

1.Project Overview

Objective: This blogging platform allows users to create, manage, and interact with blog posts. It employs **Role-Based Access Control (RBAC)** to manage access and permissions for different user roles: **User**, **Moderator**, and **Admin**.

Features

- **User Registration** with hashed passwords and JWT authentication.
- **Role-Based Access Control** to assign different permissions to users.
- **JWT Tokens** for authorization and authentication.
- **APIs** to manage blog posts and user data.
- **MySQL Database** to store user and blog data.

2.Technology Stack

Backend Framework: Node.js with Express

Database: MySQL

Authentication and Authorization: JWT (JSON Web Tokens) with Role based

Password Hashing: bcrypt.js

API: RESTful

Version **Control**: Git and Github

3.Setup Instructions for Blogging Platform

Prerequisites

Before you begin setting up the blogging platform, ensure you have the following installed and set up:

1. Node.js (v14.x or later)

Download and install Node.js from the official website: https://nodejs.org.

You can verify the installation by running the following commands in your terminal:

node -v

2. npm (v6.x or later)

npm (Node Package Manager) is bundled with Node.js, so it should be installed automatically.

Verify the version of npm by running: npm -v

3. Git

- Install Git from https://git-scm.com/
- Verify Git installation by running: git –version

4.Setting Up the Blogging Platform

- **Step 1: Clone the Repository**

  Start by cloning the project repository to your local machine. If you already have a GitHub repository for the project, you can clone it using:

  git clone https://github.com/yourusername/blogging-platform.git

  cd blogging-platform

- **Step 2: Install Dependencies**

  In the project root directory, run the following command to install all required dependencies listed in package.json:

  npm install

  This will install all the required Node.js packages for the project, including Express, bcrypt, JWT, MySQL, etc.

- **Step 3: Configure Environment Variables**

  Create a .env file in the root directory of the project to store your sensitive credentials and configuration details.

  Add the following environment variables:

  JWT_SECRET=your_jwt_secret_key # Secret key for JWT generation

  You can also adjust the JWT_SECRET key as you needed.

- **Step 4: Setup MySQL Database**

  Create a MySQL database: In your MySQL server, create a database (e.g.,work) and two tables: users and blogs.

```sql
create database Work;
use Work;
CREATE TABLE users (
  user_id INT AUTO_INCREMENT NOT NULL,
  name VARCHAR(255) NOT NULL,

  email VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL,
  role ENUM('Admin', 'User', 'Moderator') NOT NULL, -- Assuming only these roles are allowed
  blocked BOOLEAN NOT NULL DEFAULT FALSE,
  PRIMARY KEY (user_id)
);
CREATE TABLE blogs (
  post_id INT NOT NULL AUTO_INCREMENT,
  user_id INT NOT NULL,
  title VARCHAR(255) NOT NULL,
  content_description TEXT NOT NULL,
  validated BOOLEAN NOT NULL DEFAULT FALSE,
  PRIMARY KEY (post_id),
  FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

Step 5: Run the Application

Once your environment is set up and all dependencies are installed, you can start the application.

Run the application:

- npm start – to run app normally
- npm run dev – to run app in development mode
- By default, the application will start on http://localhost:3001.

Once the application is running:

Test the APIs using tools like **Postman**.

Register a new user (it can be user/moderator/admin)

Login it to obtain a JWT token for authentication

Use the token to access protected routes to manage the blog posts and users.

5. JWT Authentication

- **User Registration**: When a user registers, their password is hashed using **bcrypt.js** before storing it in the database.
- **Login**: The user logs in with their email and password. If the credentials are valid, a **JWT token** is generated and returned to the user. This token is required for accessing protected routes.

- **JWT Structure**: The token contains the user's id, role for verification and authorization.

6. Authorization

- **Role-Based Access Control (RBAC)** is implemented to restrict access to resources based on the user's role.
- Roles include:
    - **User**: Can create, view their own blog posts.
    - **Moderator**: Can validate any blog post.
    - **Admin**: can assign any user as moderator and has all controls over users.

7. Middleware

- **verifyJWT:** Verifies JWT token to authenticate the user.
- **checkRole(requiredRole):** Checks if the logged-in user is an admin/user/moderator.

8. Roles and Permissions

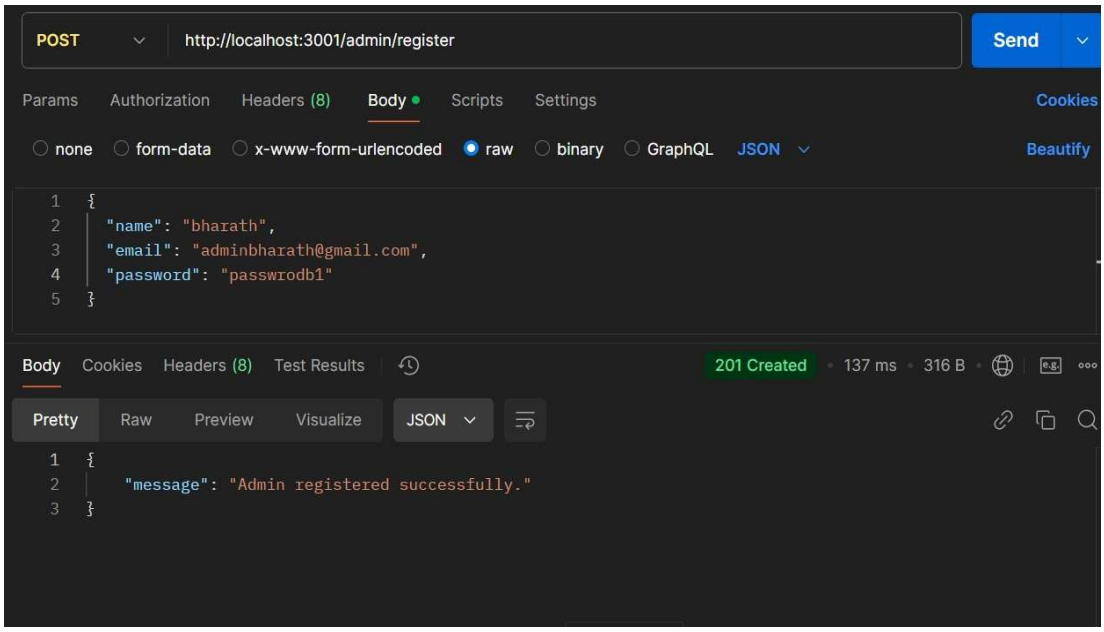
- **User:**
    - Can create their own blog posts.
    - Can view the blog posts.
- **Moderator:**
    - Can validate/invalidate the blog posts
- **Admin:**
    - Can manage users. (view, block)
    - Can assign users as moderators.

8. RESTful APIs

I. Admin Management

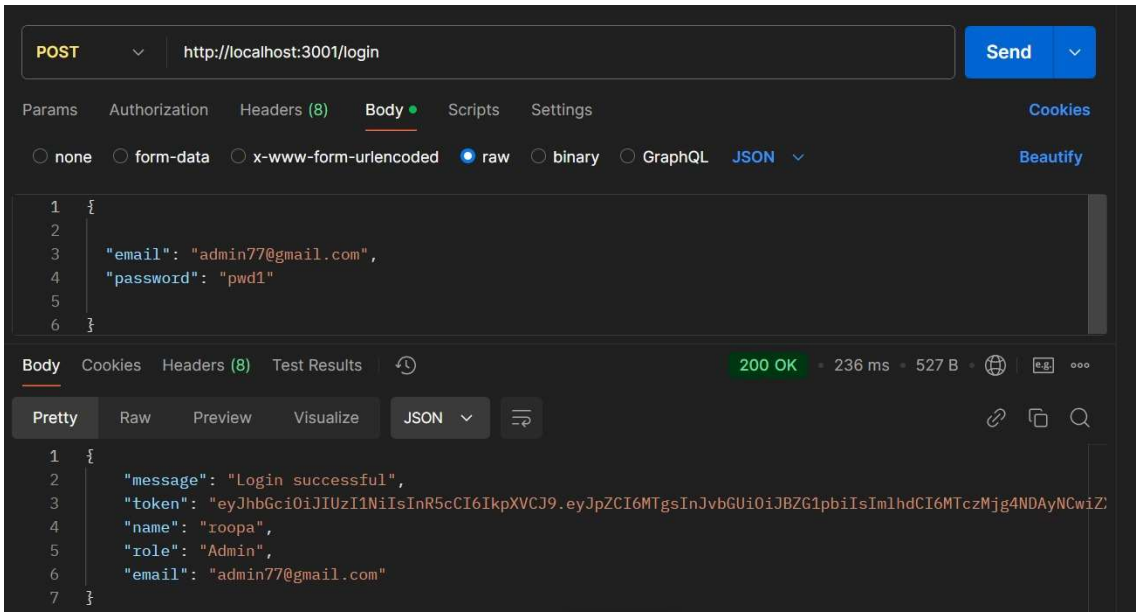
Register a new admin

- Endpoint: /admin/register
- Method: POST
- Description: Registers a new admin with name, email, password



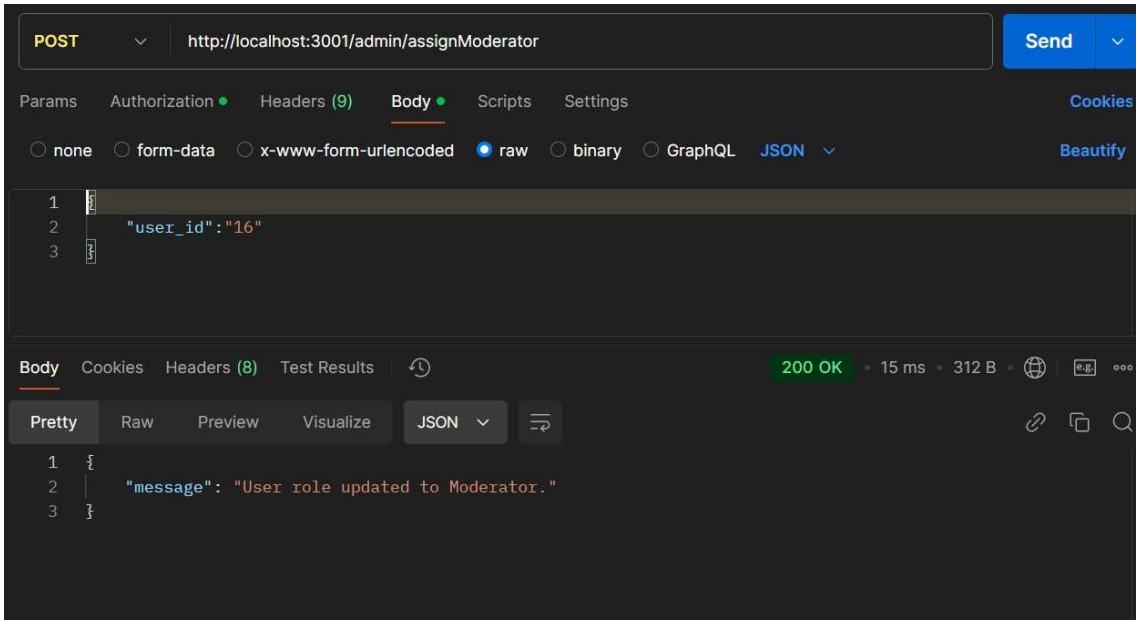
Login a admin

- Endpoint: /login
- Method: POST
- Description: Authenticates a user and returns a JWT token.
- Response: JWT token, name, role, email.



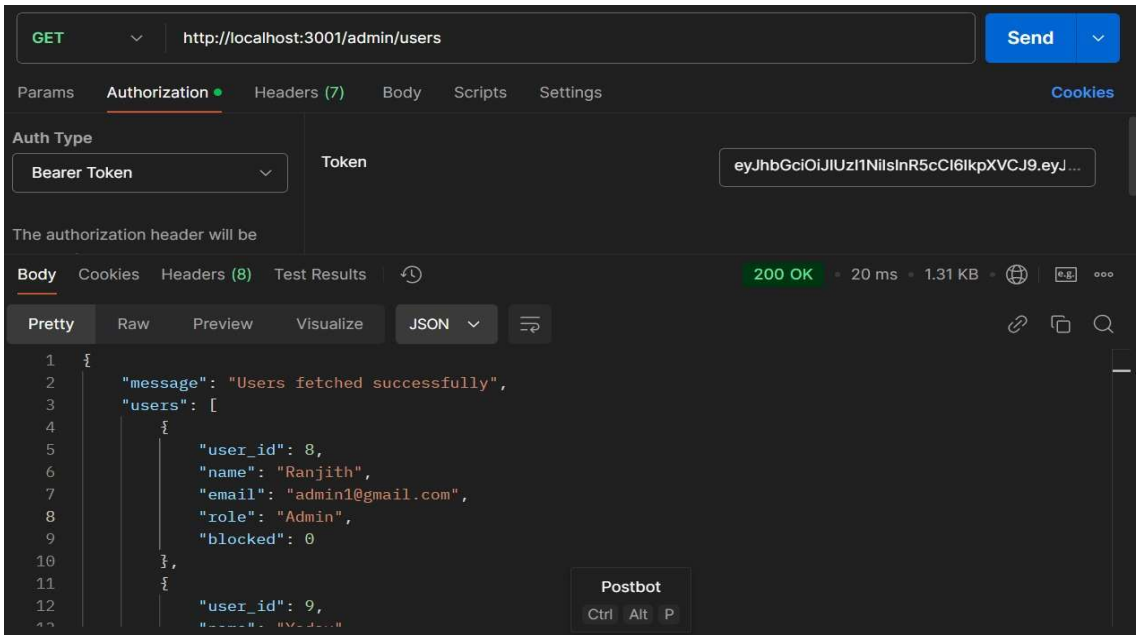
Admin assign moderator

- Endpoint: /admin/assignModerator
- Method: POST
- Description: Converts selected user into moderator.
- Headers: `Authorization: Bearer <JWT Token>`



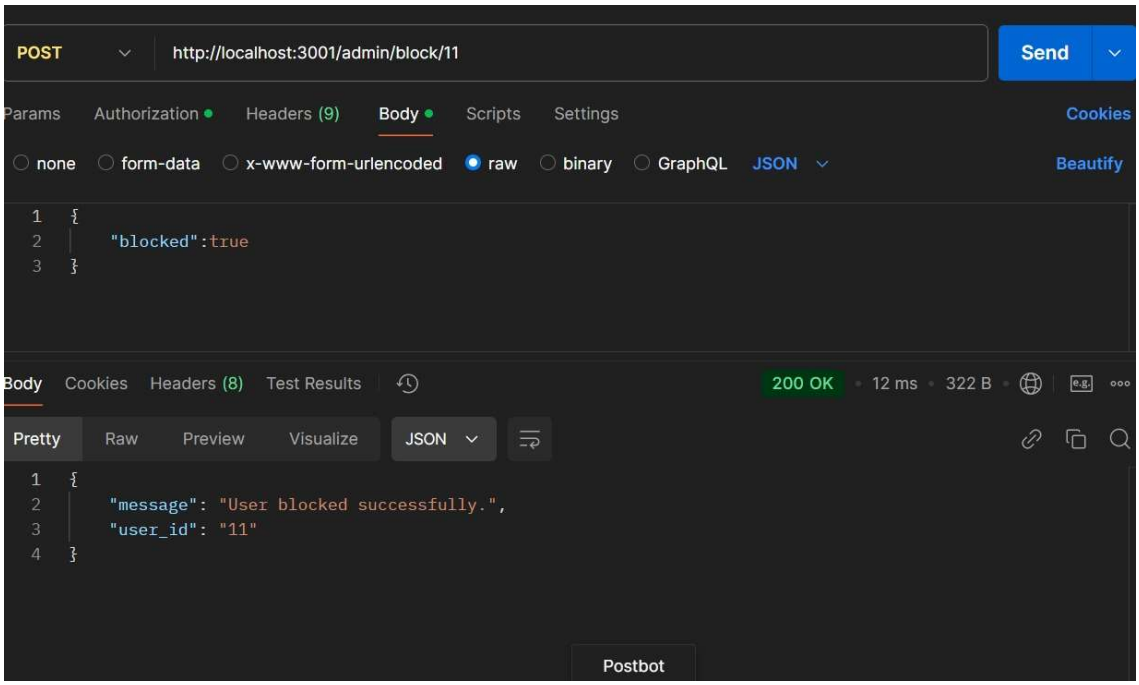
Admin Retrieve users

- Endpoint: /admin/users
- Method: GET
- Description: Authenticates a admin and retrieves users.
- Headers: `Authorization: Bearer <JWT Token>`



Admin Block users

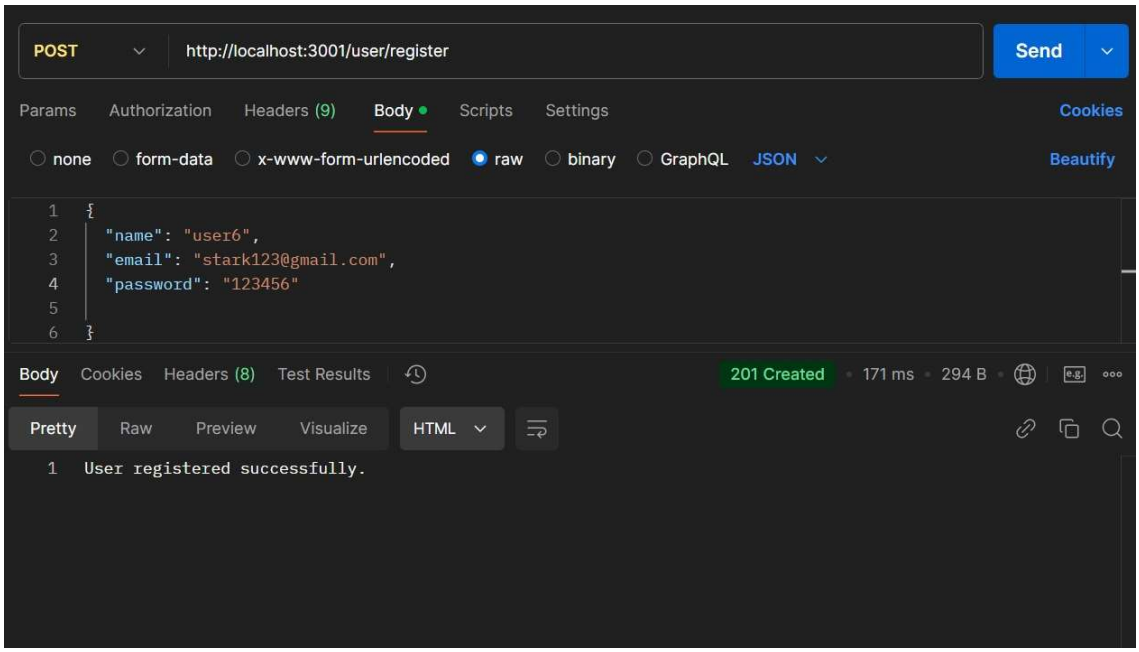
- Endpoint: /admin/block/:userId
- Method: POST
- Description: Authenticates a admin and blocks the users.
- Headers: `Authorization: Bearer <JWT Token>`



II. User Management

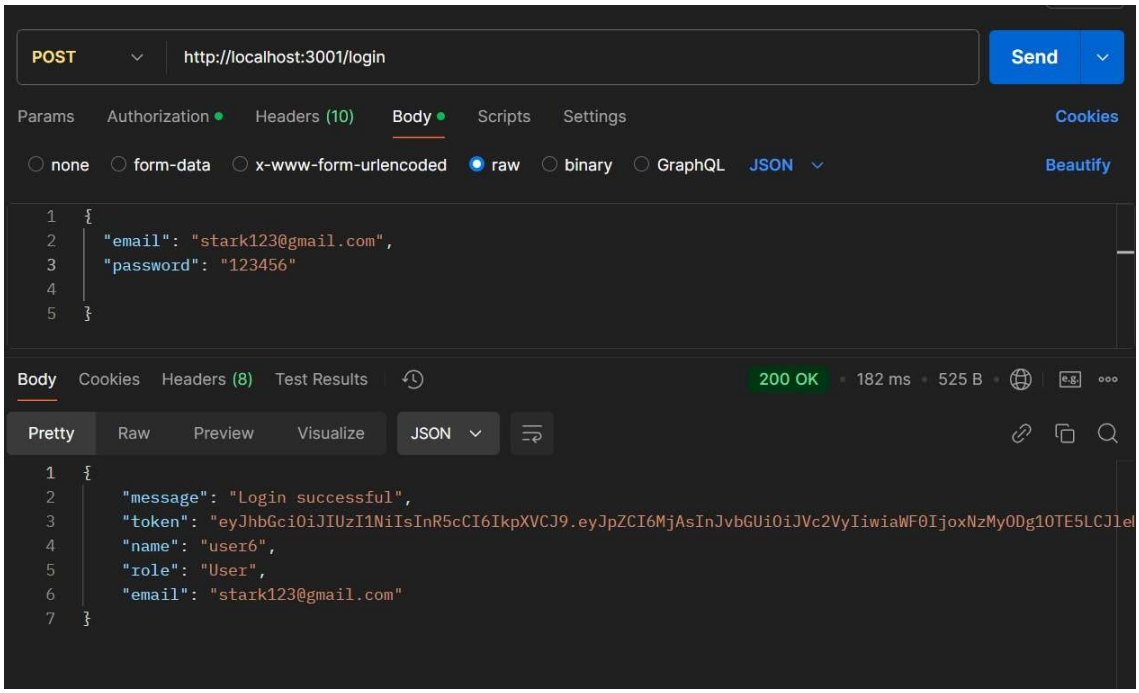
Register a new user

- Endpoint: /user/register
- Method: POST
- Description: Registers a new user with name, email, password.



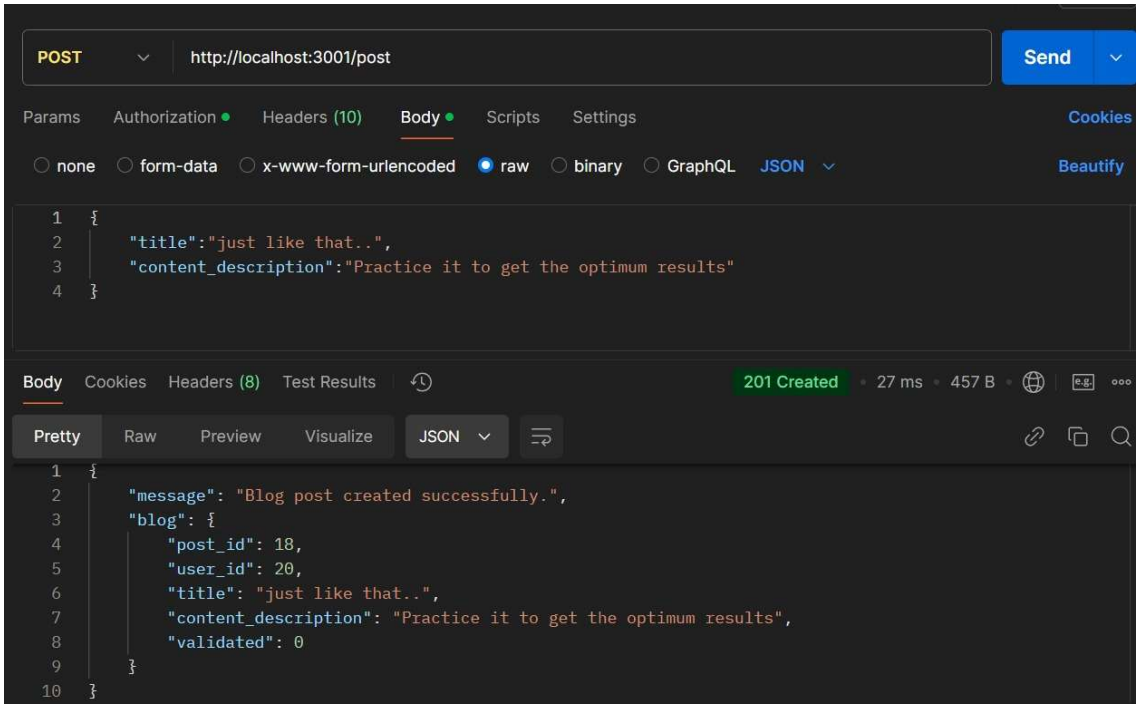
Login a user

- Endpoint: /login
- Method: POST
- Description: Authenticates a user and returns a JWT token.
- Response: JWT token, name, role, email.



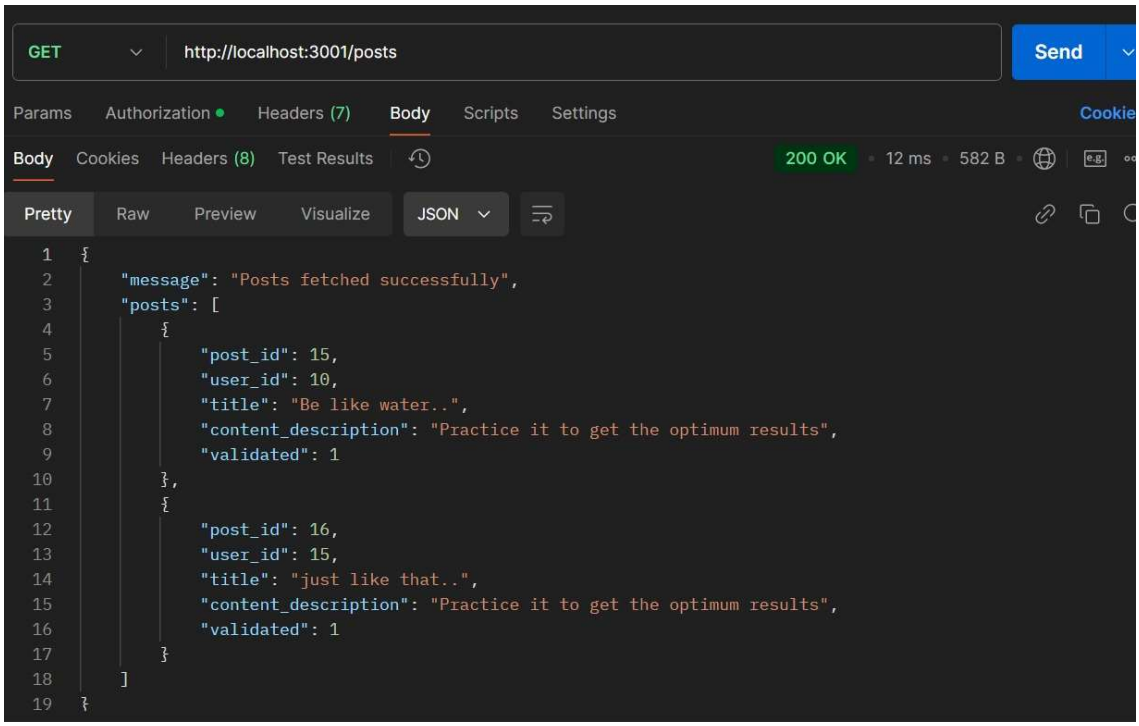
User create post

- Endpoint: /post
- Method: POST
- Description: Authenticates a user and creates a new post.
- Response: Returns new blog/post.



Retrieve All Posts (Validated)

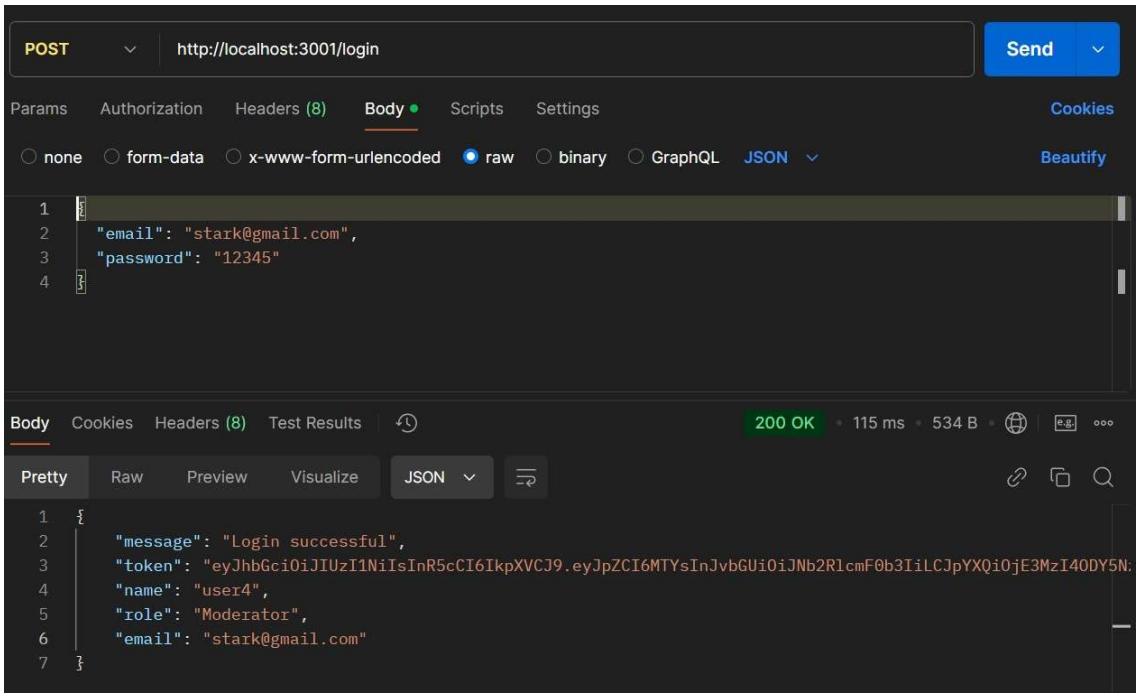
- Endpoint: /posts
- Method: GET
- Description: Authenticates token and retrieves the validated posts.
- Response: Returns validated posts.



III. Moderator Management

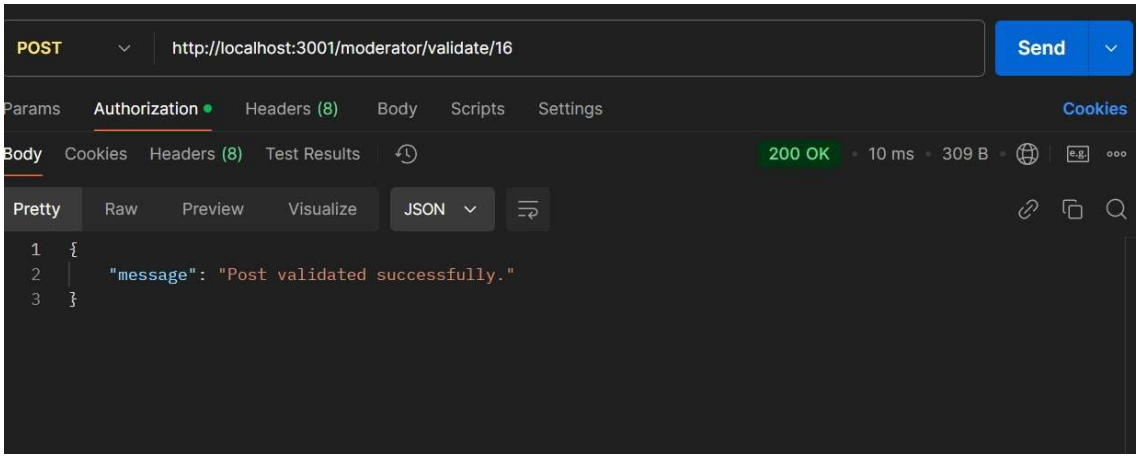
Login Moderator

- Endpoint: /login
- Method: POST
- Description: Authenticates a user and returns a JWT token.
- Response: JWT token, name, role, email.



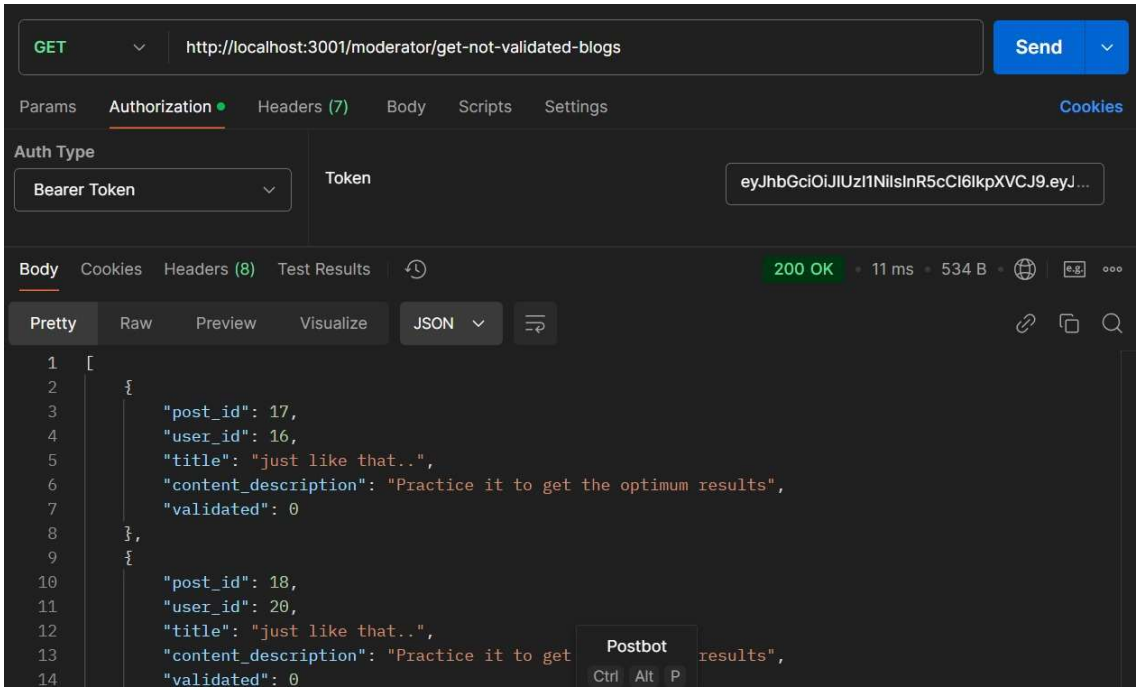
Validate the post/blog

- Endpoint: /moderator/validate/postId
- Method: POST
- Description: Authenticates a user and validates the post/blog.



Retrieve Invalidated posts/blogs

- Endpoint: /moderator/get-not-validated-blogs
- Method: GET
- Description: Authenticates a user and retrieves invalidated posts/blogs.



9. Additional Notes

- Error Handling: Implemented comprehensive error handling for all API endpoints, ensuring meaningful error messages and appropriate HTTP status codes.
- Security Considerations: Passwords are hashed using bcrypt before storage.

10. Conclusion

The Blogging Platform backend successfully implements a robust and scalable architecture using Node.js, Express, Mysql. Key achievements include:

- Comprehensive User Management: Secure registration and authentication with role-based access control (RBAC).
- Security: Implements best practices in authentication, authorization, and data protection.