Dr. D.Y. Patil School of MCA

Charoli (BK), PUNE- 412105

SAVITRIBAI PHULE PUNE UNIVERSITY

MASTER OF COMPUTER APPLICATION

Project Report on

Blog Platform with CMS

BY

Student Name: Lokesh Patil

Roll No: 25234

Under The Guidance of

Prof. Dr. Zameer S Mulla

MCA-I(Sem-I)

2025-26

Dr. D. Y. Patil Educational Enterprises Charitable Trust's
Dr. D. Y. PATIL SCHOOL OF MCA
Charholi Bk., Via-Lohegaon, Dist-Pune-412105
Approved by AICTE, New Delhi Recognized by Govt of Maharashtra, Affiliated to Savitribai Phule Pune University
AISHE Code: C-45873 DTE Code: MC6201 SPPU PUN Code: IMMP019330

Date: 24/December/2025

Certificate

This is to certify that **Lokesh Raju Patil**, Roll No. **25234** a student of **Dr D Y Patil School of MCA**, has successfully completed the project entitled **Blog Platform With CMS** in partial fulfilment of the requirements for the **MCA**- Mini Project (Semester I) during the academic year **2025–2026**.

Prof. Dr. Zameer S Mulla	**Prof. Sapna Chavan**	**Dr E B Khedkar**
Project Guide	**HOD – MCA**	**Director**

ACKNOWLEDGEMENT

I acknowledge to all those who have been so helpful in my academic project work. Nevertheless, I have tried through this report to express my deepest gratitude to all those who have given their precious time, skill, knowledge, valuable advice and guidance and facilities.

At the very outset I take opportunity to express my deepest gratitude and thanks to **Director** of our College **Dr. E. B. Khedkar** and **Prof. Sapna Chavan**–HOD and other teaching and non-teaching staff for their useful guidance during the completion of this project report.

I am highly obliged to **Prof. Dr. Zameer S Mulla** - Project Guide - for his valuable guidance and encouragement given to complete the project. His guidance has certainly helped me to simplify the difficulties and finalize the system effectively.

Last but not the least I thank to my almighty God, Parents and friends for their constant support to me in all aspects during work.

Thank You,

Lokesh Patil

MCA – I (Sem-I)

Dr D Y Patil School of MCA

INDEX

Sr. No.	Title	Page No.
1	**Chapter 1: Introduction**	
1.1	Abstract	
1.2	Existing System and Need for System	
1.3	Scope of System	
1.4	Operating Environment - Hardware and Software Brief Description of Technology Used	
1.5	Operating systems used (Windows or Unix), RDBMS/No SQL used to build database (MySQL/oracle, Teradata, etc.)	
1.5.1	Operating systems used (Windows or Unix)	
2	**Chapter 2: Proposed System**	
2.1	Feasibility Study	
2.2	Objectives of Proposed System	
2.3	Users of System	
3	**Chapter 3: Analysis and Design**	
3.1	System Requirements (Functional and Non-Functional requirements)	
3.2	Entity Relationship Diagram (ERD)	
3.3	Table Structure	
3.4	Use Case Diagrams	
3.5	Class Diagram	
3.6	Activity Diagram	
3.7	Sequence Diagram	
3.8	Deployment Diagram & Module Hierarchy Diagram	
3.9	Sample Input and Output Screens.	
4	**Coding**	
4.1	Algorithms	
4.2	Code snippets	
5	**Testing**	
5.1	Test Strategy	
5.2	Unit Test Plan	
5.3	Acceptance Test Plan	
5.4	Test Case / Test Script	
5.5	Defect report/Test Log	
6	**Limitations of Proposed System**	
7	**Proposed Enhancements**	
8	**Conclusion**	
9	**Bibliography**	

1. Introduction

1.1 Abstract

This project aims to develop a Blog Platform with CMS, allowing users to register, create, edit, and delete blog posts seamlessly. The platform offers a rich content creation experience using a markdown editor and supports reader interaction through comments. The application emphasizes user-friendliness alongside SEO-friendly technology to improve content visibility and user engagement.

1.2 Existing System and Need for the System

With the increasing demand for personalized digital content, blogging platforms serve as essential tools for individuals and organizations to share ideas, stories, and expertise. Many existing platforms are either restrictive in content presentation or lack optimal SEO capabilities. This system aims to fill those gaps by delivering a flexible, SEO-optimized blogging platform built with modern technologies.

1.3 Scope of the System

The platform will provide the following features:

- User registration and authentication
- Creation, editing, and deletion of blog posts by authenticated users
- Rendering of posts written in markdown format
- Commenting functionality accessible to both visitors and authenticated users
- A responsive user interface compatible with desktops, tablets, and smartphones

Note: Administrative functionalities like post moderation and user management are not included in this phase.

1.4 Operating Environment

- **Hardware:** Any device with internet access and a modern web browser (e.g., desktops, laptops, tablets, smartphones)
- **Software:**
    - **Operating System:** Platform-independent (Windows, macOS, Linux/Unix) as the application is web-based
    - **Frontend:** Next.js (React framework) providing SEO-friendly server-side rendered pages
    - **Backend:** Python (Flask) serving RESTful APIs
    - **Database:** MongoDB, a NoSQL database offering flexible schema design

1.5 Brief Description of Technology Used

- **Next.js:** A React-based framework enabling server-side rendering and static site generation to enhance SEO and performance.
- **React Markdown:** Parses and safely renders markdown content in the frontend UI.
- **Python Web Framework (Flask):** Handles backend logic, API routing, and user authentication.
- **PyMongo:** Python libraries used to connect and interact with MongoDB.
- **MongoDB:** NoSQL database designed for storing JSON-like documents, ideal for flexible blog post and comment structures.

2. Proposed System

Study of Similar Systems

Popular blogging platforms such as WordPress and Medium offer extensive features but often come with heavy dependencies, limited customization, or proprietary restrictions. This system leverages modern frameworks like Next.js and Python-based backends with MongoDB to deliver a dynamic, user-authenticated blogging experience that is both SEO-friendly and highly customizable.

2.1 Feasibility Study

- **Technical Feasibility:** Utilizes mature, widely adopted technologies with strong community support, enabling straightforward development and deployment.
- **Economic Feasibility:** All core tools are open source, minimizing costs. Cloud hosting options for MongoDB (like Atlas) offer free tiers, keeping operational costs low.
- **Operational Feasibility:** The system's intuitive UI and clear functionalities ensure user acceptance and easy maintenance.

2.2 Objective of Proposed System

- Deliver a seamless, SEO-optimized blogging platform using modern frontend and backend technologies.
- Enable users to create formatted content efficiently via markdown.
- Foster community engagement through comments on posts.
- Securely manage user authentication and personalized content.

Module Specifications

1. **User Authentication Module:**
   Facilitates secure registration, login, and session management for users using Python backend solutions.
2. **Post Management Module:**
   Allows authenticated users to create, update, delete, and retrieve blog posts stored in MongoDB in markdown format.
3. **Markdown Rendering Module:**
   Converts markdown to HTML for frontend display using React Markdown.
4. **Commenting Module:**
   Permits users and visitors to add comments on blog posts, with data stored and managed in the database.
5. **Frontend UI Module:**
   Implements a responsive, accessible interface with Next.js and styling frameworks like Tailwind CSS.

2.3 Users of the System

- **Registered Users:** Can create, edit, delete their own blog posts and comment on any posts.
- **Visitors:** Can read posts and submit comments, with optional authentication for improved moderation in future updates.

3.1 System Requirements

System requirements define what the system should do and how well it should perform.

They are divided into Functional Requirements and Non-Functional Requirements.

3.1.1 Functional Requirements

Functional requirements describe the **core features and services** provided by the Blog Platform with CMS.

FR1: User Registration

- The system shall allow new users to create an account using valid credentials.
- Duplicate usernames or email IDs shall not be allowed.

FR2: User Login and Authentication

- The system shall authenticate users using secure login credentials.
- Only authenticated users shall be allowed to create, edit, or delete blog posts.

FR3: User Logout

- The system shall allow logged-in users to securely log out of the application.

FR4: Blog Post Creation

- The system shall allow authenticated users to create blog posts.
- Blog content shall be written in **Markdown format**.
- Each post shall include title, content, author, and creation date.

FR5: Blog Post Editing

- The system shall allow users to edit only their own blog posts.
- Updated content shall be saved and reflected immediately.

FR6: Blog Post Deletion

- The system shall allow users to delete their own blog posts.
- Deleted posts shall be removed from public view.

FR7: View Blog Posts

- The system shall allow visitors and registered users to view published blog posts.
- Blog posts shall be rendered in readable HTML format from Markdown.

FR8: Commenting System

- The system shall allow users and visitors to add comments on blog posts.
- Each comment shall store commenter name, comment text, and timestamp.

3.1.2 Non-Functional Requirements

Non-functional requirements describe **quality attributes**, performance, and constraints of the system.

NFR1: Performance

- The system shall load blog pages efficiently with minimal delay.
- API responses shall be optimized for fast data retrieval.

NFR2: Security

- User passwords shall be stored in encrypted or hashed format.
- Unauthorized access to protected resources shall be restricted.

NFR3: Scalability

- The system shall support an increasing number of users and blog posts without major performance degradation.

NFR4: Reliability

- The system shall ensure data consistency for posts and comments.
- In case of system failure, data shall not be lost.

NFR5: Usability

- The system shall provide a simple and intuitive user interface.
- Users with basic technical knowledge shall be able to operate the system easily.

Database Design Verification

Table 1: Users

Field Name	Data Type	Description
_id	ObjectId	Unique identifier for each user
username	String	Username of the user
email	String	Email address of the user
password	String	Encrypted password
reset_password_token	String	Token for password reset
reset_password_expires	Timestamp	Password reset expiry time
created_at	Timestamp	Account creation date

Table 2: Posts

Field Name	Data Type	Description
_id	ObjectId	Unique identifier for each post
title	String	Title of the blog post
body	Text	Blog content (Markdown)
author_id	ObjectId	Reference to Users table
tags	Array	List of tags
image	String	Featured image URL
created_at	Timestamp	Post creation time
updated_at	Timestamp	Last updated time

Table 3: Comments

Field Name	Data Type	Description
_id	ObjectId	Unique identifier for comment
body	Text	Comment content
post_id	ObjectId	Reference to Posts table
author_id	ObjectId	Reference to Users table
parent_id	ObjectId	Parent comment ID (for replies)
created_at	Timestamp	Comment creation time
updated_at	Timestamp	Last update time

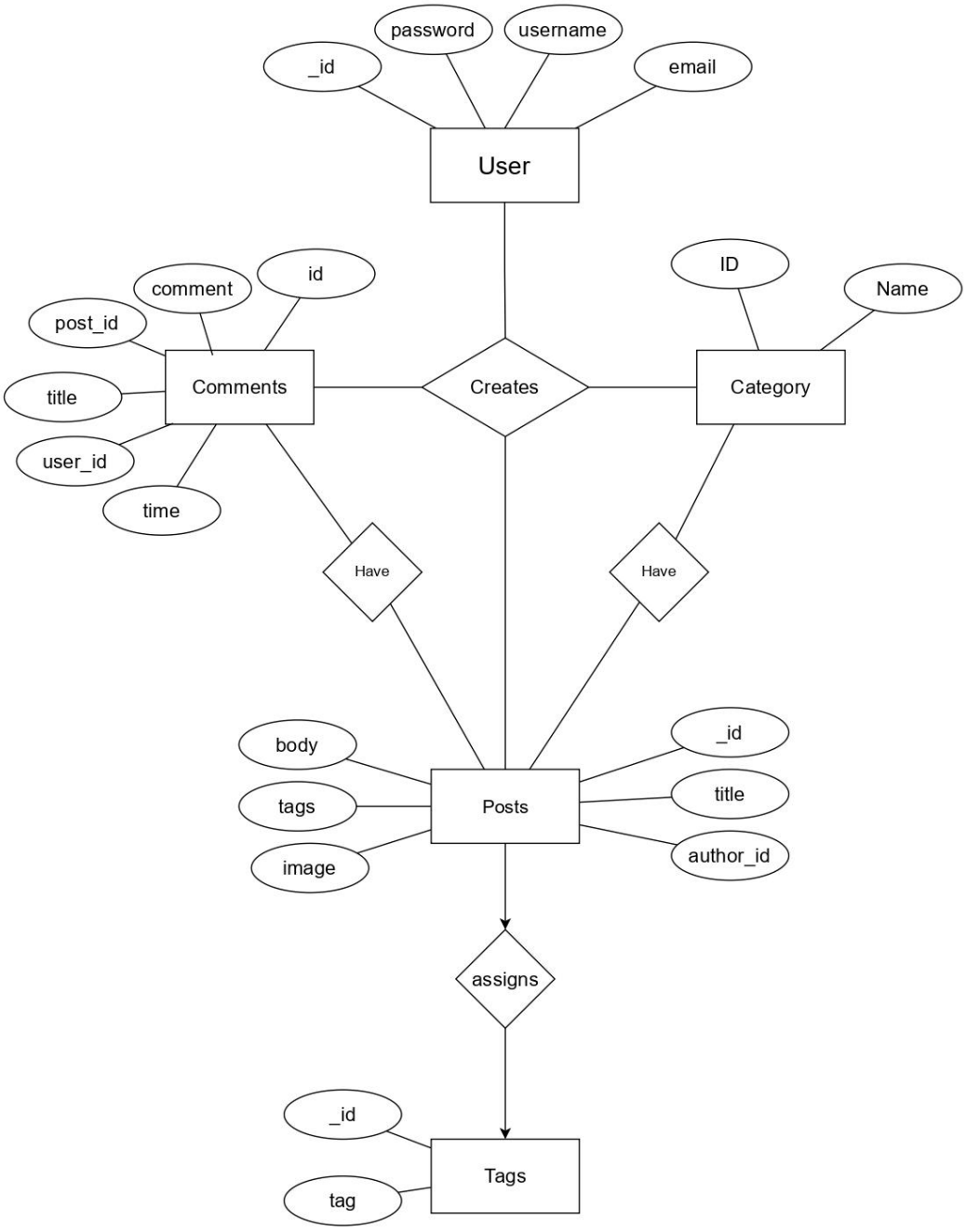
Table 4: Post_Likes

Field Name	Data Type	Description
`_id`	ObjectId	Unique identifier
`post_id`	ObjectId	Reference to Posts table
`user_id`	ObjectId	Reference to Users table
`created_at`	Timestamp	Like timestamp

Table 5: Follows

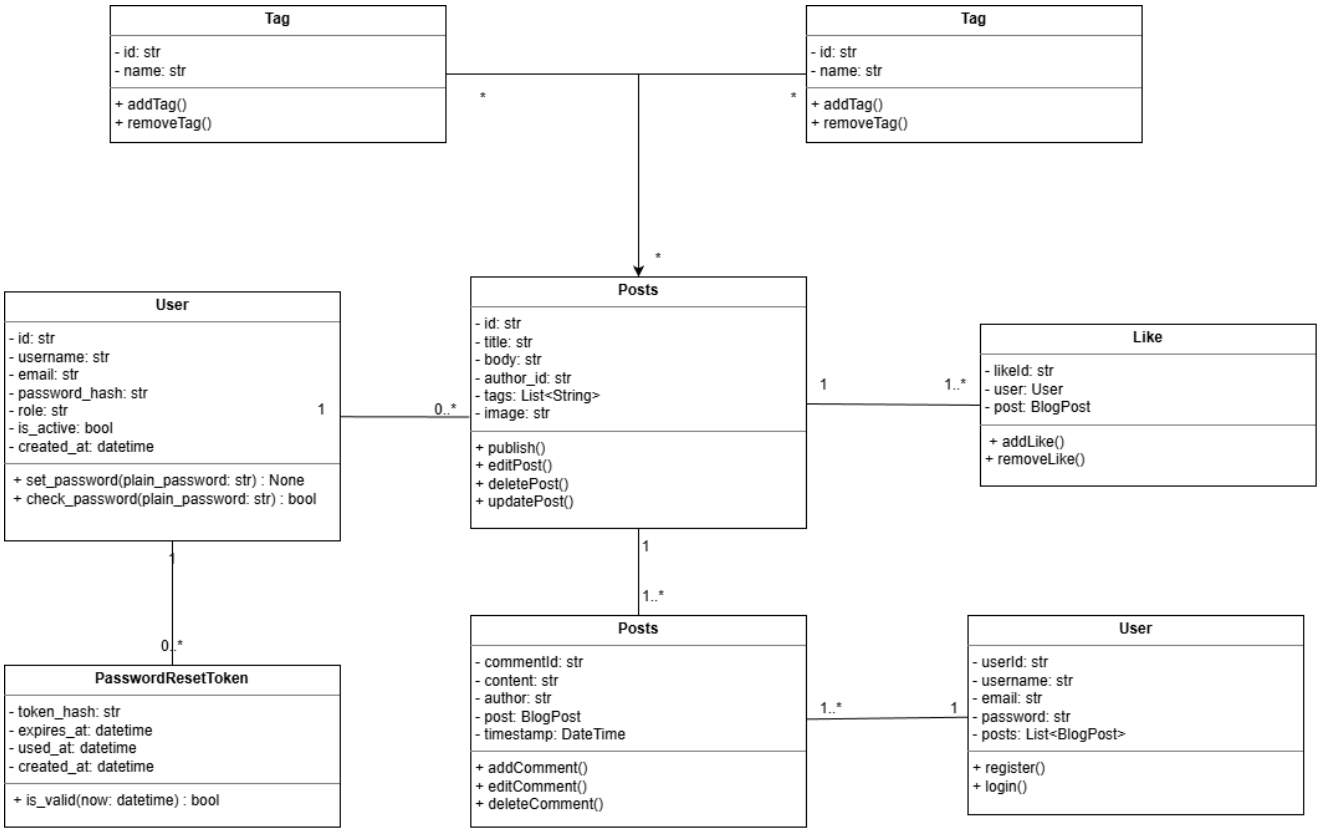
Field Name	Data Type	Description
`_id`	ObjectId	Unique identifier
`follower_id`	ObjectId	User who follows
`following_id`	ObjectId	User being followed
`created_at`	Timestamp	Follow date

ERD Diagram

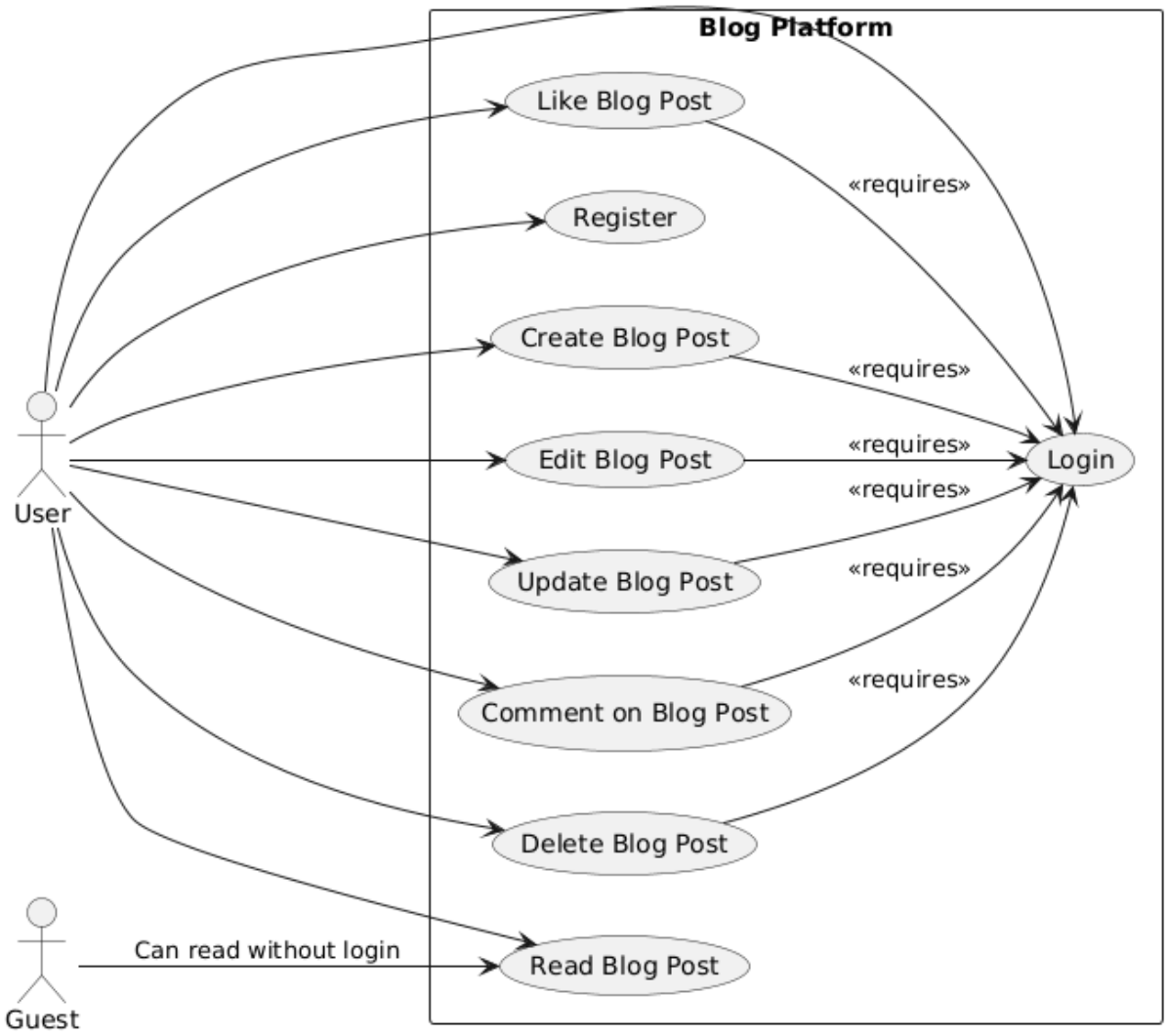

password
username
_id
email
User
ID
Name
comment
id
post_id
Comments
Creates
Category
title
user_id
time
Have
Have
body
_id
tags
Posts
title
image
author_id
assigns
_id
Tags
tag

UML Diagrams

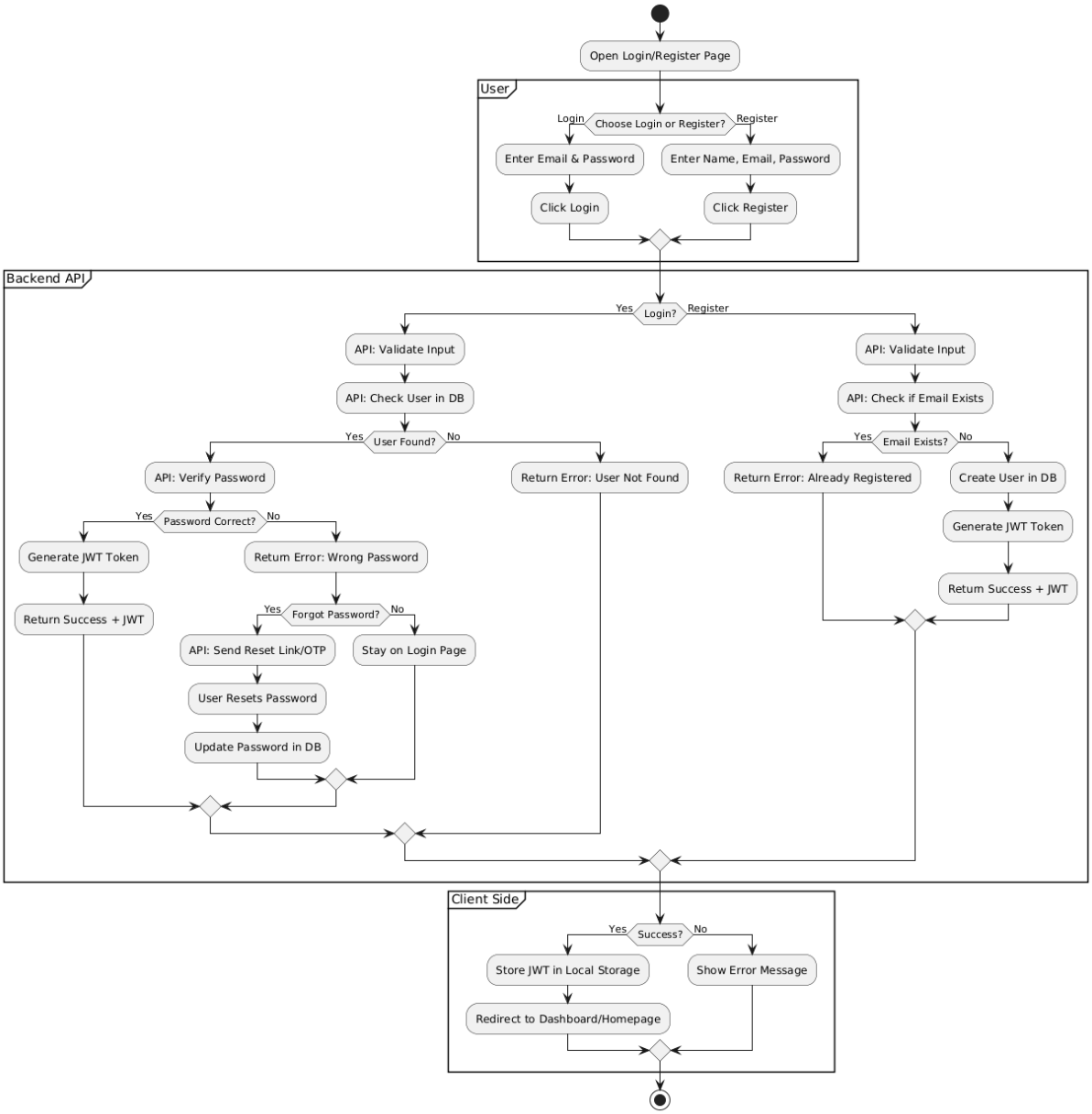
1. Class Diagram



2. Use Case Diagram

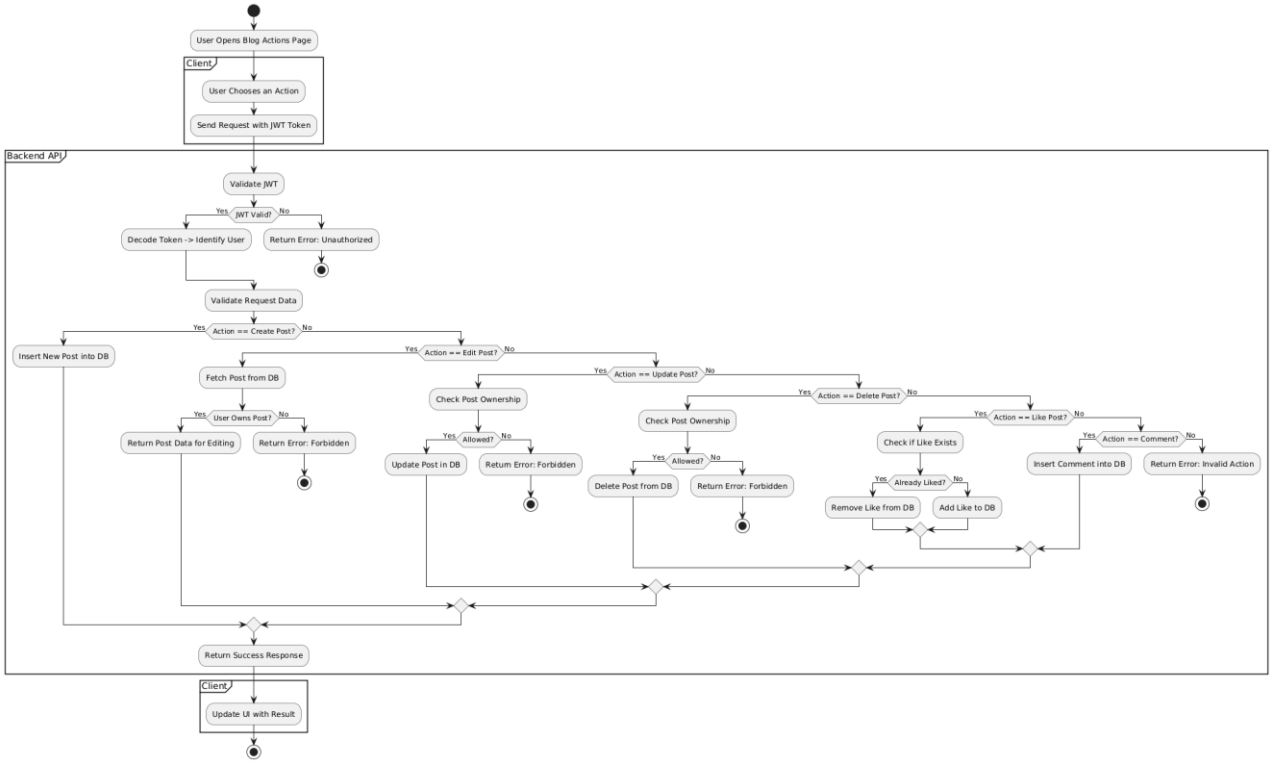


1. Activity Diagram

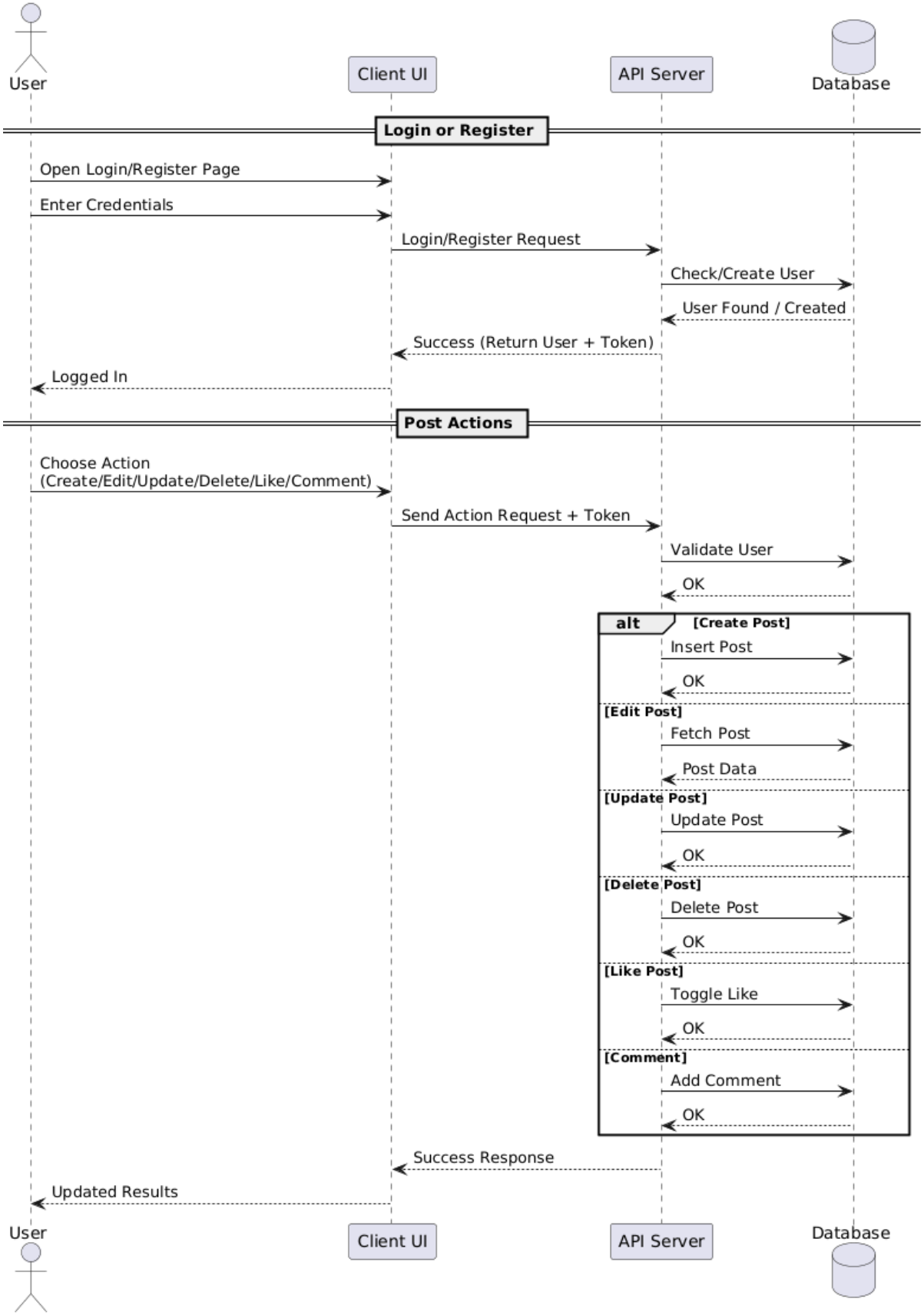


-Login Register

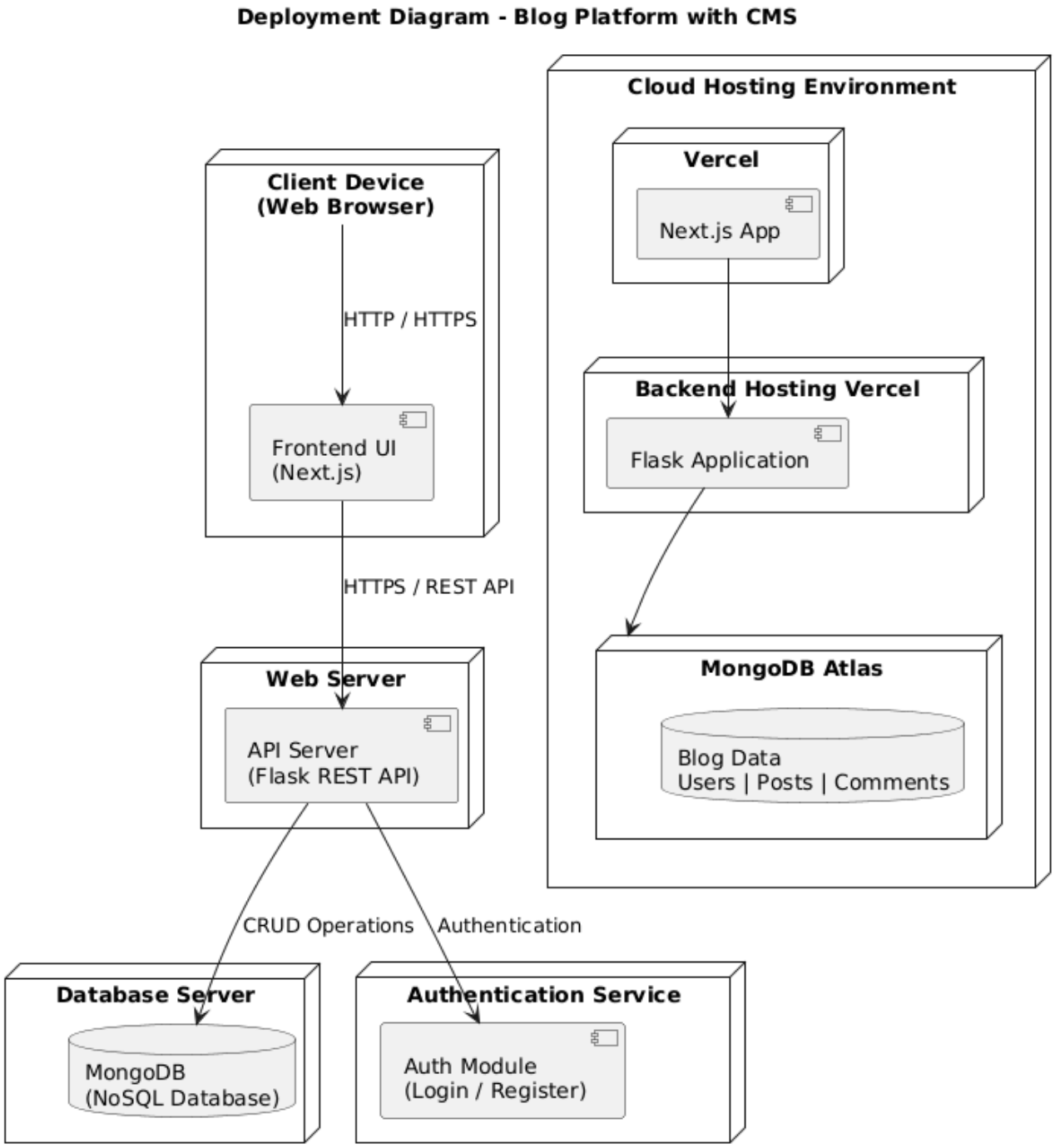
-Post Activity Diagram



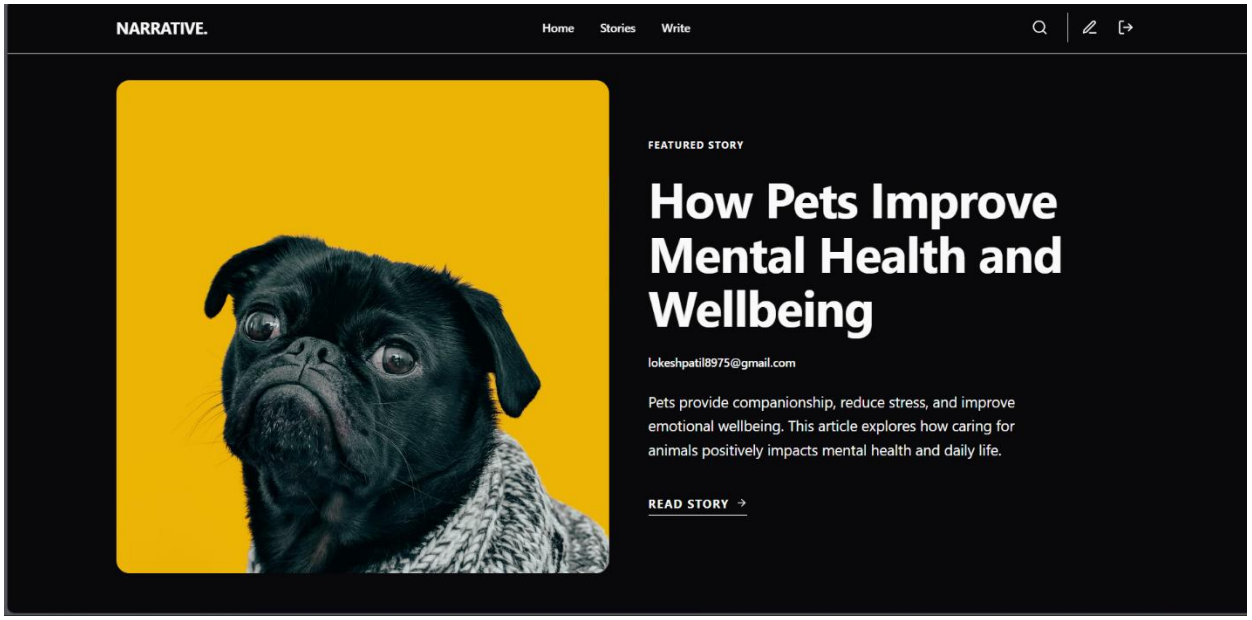
2. Sequence Diagram

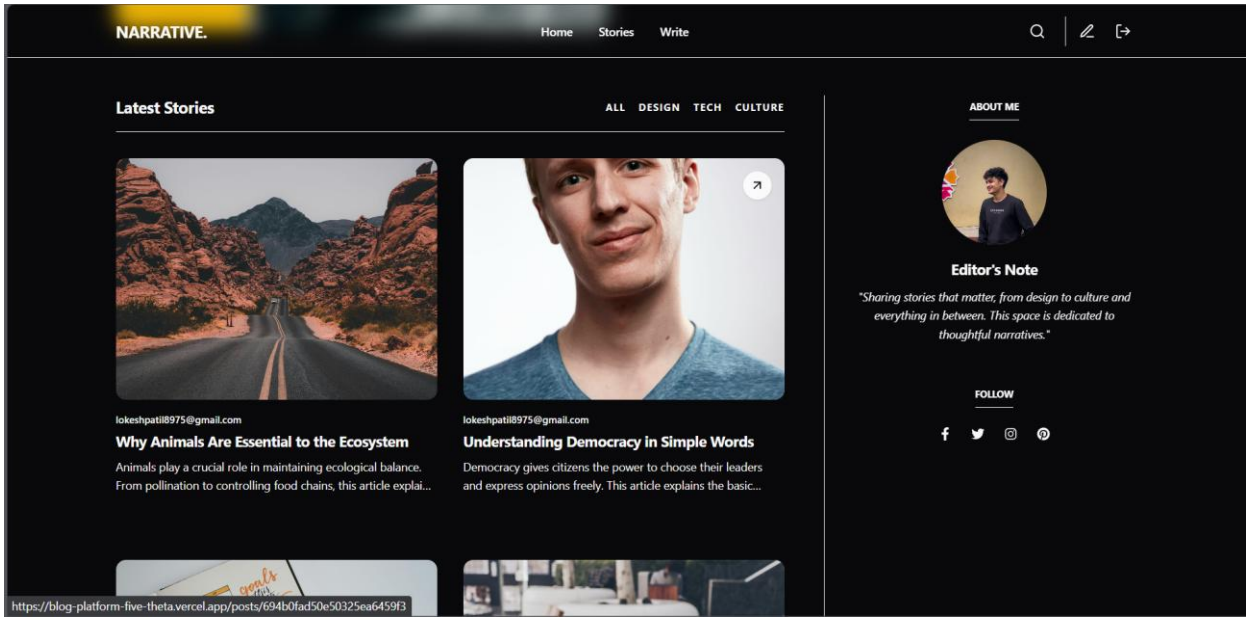


3. Deployment Diagram & Module Hierarchy Diagram

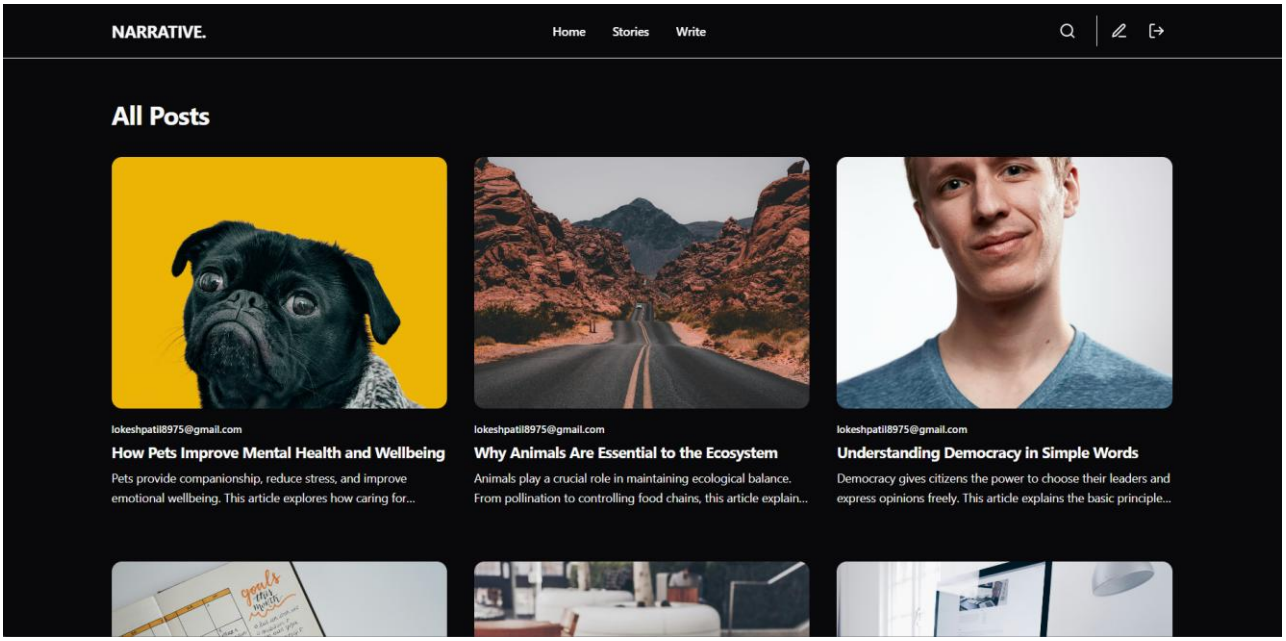


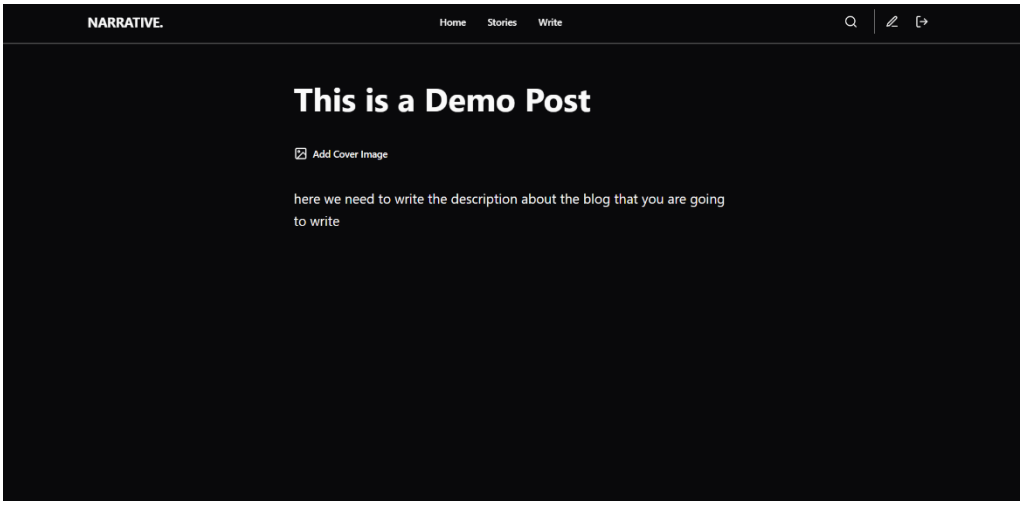
3.5 Sample Input and Output Screens






NARRATIVE.
Home
Stories
Write
Why Consistency Matters More Than Motivation
lokeshpatil8975@gmail.com
Just now • 5 min read
NOVEMBER


NARRATIVE.
Home
Stories
Write
All Posts
lokeshpatil8975@gmail.com
How Pets Improve Mental Health and Wellbeing
Pets provide companionship, reduce stress, and improve emotional wellbeing. This article explores how caring for...
lokeshpatil8975@gmail.com
Why Animals Are Essential to the Ecosystem
Animals play a crucial role in maintaining ecological balance. From pollination to controlling food chains, this article explain...
lokeshpatil8975@gmail.com
Understanding Democracy in Simple Words
Democracy gives citizens the power to choose their leaders and express opinions freely. This article explains the basic principle...


NARRATIVE.
Home
Stories
Write
This is a Demo Post
Add Cover Image
here we need to write the description about the blog that you are going to write

4. Coding

4.1 Alogorithm

Algorithm 1: User Registration

Input: Username, Email, Password

Output: User account created successfully or error message

Steps:

1. Start
2. Accept user registration details
3. Check whether email or username already exists
4. If exists, display error message and stop
5. Encrypt the user password
6. Store user details in the database
7. Display registration success message
8. End

Algorithm 2: User Login

Input: Email, Password

Output: Login success or failure

Steps:

1. Start
2. Accept login credentials
3. Retrieve user details from database
4. Compare entered password with stored encrypted password
5. If credentials match, generate authentication token
6. Allow user access to the system
7. Else display login error
8. End

Algorithm 3: Create Blog Post

Input: Post title, Content (Markdown), Author ID

Output: Blog post saved successfully

Steps:

1. Start
2. Verify user authentication
3. Accept post title and markdown content
4. Validate input fields
5. Store post data in database
6. Display post creation success message
7. End

Algorithm 4: Edit Blog Post

Input: Post ID, Updated content

Output: Post updated successfully

Steps:

1. Start
2. Verify user authentication
3. Fetch post using Post ID
4. Check whether user is the author
5. Update post content in database
6. Save updated timestamp
7. Display confirmation message
8. End

Algorithm 5: Delete Blog Post

Input: Post ID

Output: Blog post removed

Steps:

1. Start
2. Verify user authentication
3. Identify post using Post ID
4. Check user authorization
5. Delete post from database
6. Display deletion success message
7. End

Algorithm 6: View Blog Posts

Input: None

Output: List of published blog posts

Steps:

1. Start
2. Fetch all published posts from database
3. Convert markdown content to HTML
4. Display posts on user interface
5. End

4.2 Code Snippet

1. Login Code Snippet

```
const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setIsLoading(true);

    try {
      const { data } = await api.post("/auth/login", { email, password });
      login(data.access_token, data.user);
    } catch (error: any) {
      toast.error(error.response?.data?.msg || "Login failed");
    } finally {
      setIsLoading(false);
    }
  };
```

2. Reset Code Snippet

```python
def reset_password():
    data = request.get_json() or {}
    email = (data.get("email") or "").strip().lower()
    token = (data.get("token") or "").strip()
    new_password = data.get("password") or ""

    if not email or not token or not new_password:
        return jsonify({"msg": "email, token and password are required"}), 400

    users = mongo.db.users
    user = users.find_one({"email": email})
    if not user:
        return jsonify({"msg": "Invalid token or expired"}), 400

```

3. Token Generation

```python
if user:
        raw_token = _generate_token(32)   # 64 hex chars
        token_hash = _hash_token(raw_token)
        expires_at = _not_utc() + datetime.timedelta(hours=1)

        users.update_one({"_id": user["_id"]}, {"$set": {
            "reset_password_token": token_hash,
            "reset_password_expires": expires_at
        }})

        frontend_base = os.getenv("FRONTEND_URL", "http://localhost:3000").rstrip("/")
        reset_path = f"/reset-password?token={raw_token}&email={email}"
        reset_url = frontend_base + reset_path

        try:
            send_reset_email_sendgrid(email, reset_url)
        except Exception:
            # Log already handled inside helper
            pass

    return jsonify({"msg": "If an account exists, you will receive a reset email"}), 200
```

4. Submit Reset Token

```
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  if (!password || !confirmPassword || !token || !email) return;

  if (password !== confirmPassword) {
    setMessage({ type: "error", text: "Passwords do not match." });
    return;
  }

  setLoading(true);
  setMessage(null);

  try {
    const response = await api.post("/auth/reset-password", { email, token, password });

    if (response.status === 200) {
      setMessage({ type: "success", text: "Password successfully reset! Redirecting to login..." });
      setTimeout(() => {
        router.push("/login");
      }, 2000);
    }
  } catch (error: any) {
    console.error("Reset password error:", error);
    const msg = error.response?.data?.msg || "Failed to reset password. Token may be expired.";
    setMessage({ type: "error", text: msg });
  } finally {
    setLoading(false);
  }
};

```

5. Testing

Testing is performed to ensure that the Blog Platform with CMS works correctly and meets the specified requirements.

5.1 Test Strategy

- Manual testing is used for system validation.
- Functional and integration testing are performed.
- Both valid and invalid inputs are tested.
- Frontend and backend interaction is verified through APIs.

5.2 Unit Test Plan

- Individual modules are tested separately.
- User authentication, post management, and comment modules are tested.
- Each module is verified for correct input and output.

5.2 Unit Test Plan

- Individual modules are tested separately.
- User authentication, post management, and comment modules are tested.
- Each module is verified for correct input and output.

5.4 Test Case / Test Script

Test Case ID	Description	Expected Result	Status
TC_01	User Registration	Account created	Pass
TC_02	User Login	Login successful	Pass
TC_03	Create Post	Post created	Pass
TC_04	Edit Post	Post updated	Pass
TC_05	Delete Post	Post deleted	Pass
TC_06	Add Comment	Comment added	Pass

5.5 Defect Report / Test Log

Defect ID	Description	Status
D_01	API configuration issue	Fixed
D_02	CORS error	Fixed

6. Limitations of Proposed System

Although the Blog Platform with CMS fulfills the essential requirements of a modern blogging application, it has certain limitations due to project scope, time constraints, and academic focus.

1. **Limited Administrative Control**
   The current system does not include an admin panel for content moderation, user management, or comment approval. All users operate with the same level of access.
2. **Basic Authentication Mechanism**
   The authentication system is limited to basic login and registration. Advanced security features such as two-factor authentication, OAuth-based login, and account recovery mechanisms are not implemented.
3. **No Role-Based Access Control**
   The system does not differentiate users based on roles such as admin, editor, or moderator. Role-based permissions are considered out of scope for the current version.
4. **Limited Comment Moderation**
   Comments posted by visitors or users are not filtered or moderated automatically. This may allow inappropriate or spam content until manual intervention is introduced in future enhancements.
5. **Scalability Constraints**
   While MongoDB supports scalability, the current deployment and backend configuration are suitable only for small to medium traffic and may require optimization for large-scale usage.
6. **Limited Analytics and Reporting**
   The system does not provide analytics such as post views, user engagement statistics, or performance reports.

7. Proposed Enhancements

The Blog Platform with CMS can be further improved by incorporating additional features and advanced technologies in future versions. These enhancements aim to increase usability, security, scalability, and overall system efficiency.

1. **Administrative Panel**
   An admin dashboard can be introduced to manage users, blog posts, and comments. This would allow content moderation, user role management, and system monitoring.
2. **Role-Based Access Control**
   Different user roles such as admin, editor, and author can be implemented to provide controlled access to system functionalities.
3. **Advanced Authentication Mechanisms**
   Security can be enhanced by adding features such as two-factor authentication, email verification, password recovery, and OAuth-based login using third-party services.
4. **Improved Comment Moderation**
   Automated comment filtering and approval mechanisms can be added to prevent spam and inappropriate content.
5. **Advanced Search and Filtering**
   Features such as full-text search, tag-based filtering, category-wise browsing, and sorting of posts can be implemented for better content discovery.
6. **Analytics and Reporting Module**
   An analytics module can be developed to track post views, user engagement, comment activity, and system usage statistics.
7. **Performance Optimization and Caching**
   Caching mechanisms and performance optimization techniques can be applied to improve response time and handle higher traffic loads efficiently.
8. **Cloud Deployment and Scalability**
   The system can be deployed using scalable cloud infrastructure with load balancing to support large numbers of users and high availability.

8. Conclusion

The Blog Platform with CMS is a web-based application designed to provide a simple and efficient solution for creating, managing, and publishing blog content. The system successfully integrates modern frontend and backend technologies to deliver a user-friendly, responsive, and SEO-friendly blogging experience.

By using **Next.js** for the frontend and **Flask** for the backend, the application ensures fast page rendering, secure API communication, and smooth interaction between system components. The use of **MongoDB** as a NoSQL database provides flexibility in handling blog posts and comments with varying structures.

The platform enables registered users to create, edit, and delete blog posts using Markdown, while visitors can easily read posts and interact through comments. The modular architecture and clean separation of concerns make the system easy to understand, maintain, and extend.

Overall, the project meets its defined objectives and demonstrates the practical application of web development concepts, database design, and software engineering principles. The system serves as a strong foundation for future enhancements and can be further extended to support advanced features such as role-based access control, analytics, and large-scale deployment.

9. Bibliography

- MongoDB Documentation, https://www.mongodb.com/docs/
- Flask Official Documentation, https://flask.palletsprojects.com/
- Next.js Documentation, https://nextjs.org/docs
- React Documentation, https://react.dev/
- Tailwind CSS Documentation, https://tailwindcss.com/docs