



WebX Lab Mini Project

Blog Web Application with Flask and MongoDB

Submitted By :

Name: Kunal Punjabi

Roll No: 43

Division: D15A

Third Year (Semester-VI)

**Bachelor of Engineering in Information Technology
(Autonomous Program)**

Department of Information Technology

Vivekanand Education Society's Institute of Technology

**An Autonomous Institute Affiliated to University of
Mumbai**

Academic Year: 2023-2024

Certificate

This is to certify that I have completed the project report on the topic **Simple Blog Web Application using Flask and MongoDB** satisfactorily in partial fulfilment of the requirements for the award of **Mini Project in WebX** Lab of Third Year (Semester-VI) in Information Technology under the guidance of **Ms. Dipti Karani** during the academic year 2023–2024, as prescribed by An Autonomous Institute Affiliated to University of Mumbai.

Supervisor/ Examiner

Table of Contents

1. Problem Statement
2. Introduction
3. Methodology
4. Objective
5. Tools and Technologies Used
6. Features of the Project
7. Folder Structure of the Project
8. Working of the Project
9. Screenshots
10. Sample Code Snippets
11. Challenges Faced
12. Conclusion
13. Future Improvements
14. References
15. GitHub Link

1. Problem Statement

Create a simple and efficient blog web application where users can read, search, and manage blog posts, while providing a secure admin panel for authenticated users to create, update, and delete posts. The system should store data in a NoSQL database (MongoDB) and handle user authentication securely.

2. Introduction

This project is a web-based blogging platform developed using **Flask** (a Python web framework) and **MongoDB** (a NoSQL database). It allows users to read and search blog posts, and lets admin users create, update, or delete content. This system works like a mini **Content Management System (CMS)** where only authenticated users can manage blog content. The design is simple, responsive, and beginner-friendly.

3. Methodology :

1. **Flask Setup:** Use Flask to handle routing, rendering templates, and managing requests.
2. **MongoDB Integration:** Connect Flask to MongoDB using pymongo for storing user data and blog posts.
3. **Authentication:** Implement user registration and login using sessions and password hashing.
4. **Admin Dashboard:** Only logged-in admins can access post management features (CRUD).
5. **Search Feature:** Allow users to search blog posts by keywords or titles.
6. **UI Design:** Build simple HTML templates with Bootstrap or similar for frontend pages.
7. **Testing & Debugging:** Test functionality and fix any errors to ensure smooth user experience.

4. Objective

The main goal of this project is to:

- Build a dynamic and easy-to-use blog system.
- Allow only registered users (admin) to create and manage blog posts.
- Store all data securely in MongoDB.
- Provide a search feature so users can find posts easily.
- Learn how to create real web apps using Flask.

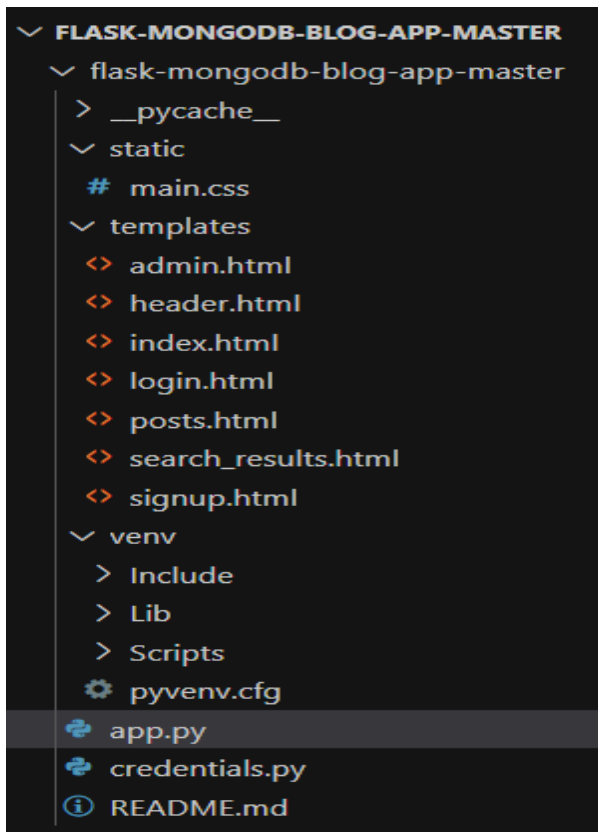
5. Tools and Technologies Used

Tools/Technology	Purpose
Python	Main Programming Language
Flask	Web framework to build the application
MongoDB	Database to store blog data
HTML/CSS	For designing the user interface
VS Code	Code editor
Jinja2	Templating engine in Flask

6. Features of the Project

Feature	Description
User Registration	New users can register on the platform.
Login System	Only registered users can log in.
Admin Dashboard	Logged-in users can manage blog posts.
Create/Edit/Delete Posts	Admin can add new blog posts or change existing ones.
Search Functionality	Users can search for blog posts by title.
Responsive UI	Clean and simple design

7. Folder Structure of the Project



Explanation :

flask-mongodb-blog-app-master

This is the main project directory that contains all the source code, templates, and configuration files.

pycache

- Stores compiled Python files for faster execution.
- Automatically generated by Python.

static

- Contains static assets such as CSS, JavaScript, and images.
- **main.css** – Defines the styling for the front-end.

templates

This folder contains all HTML templates used with Flask's Jinja2 templating engine.

- **admin.html** – Admin dashboard page
- **header.html** – Shared header/navigation bar for all pages
- **index.html** – Homepage of the blog application
- **login.html** – User login form
- **posts.html** – Displays blog posts
- **search_results.html** – Shows results of search queries
- **signup.html** – New user registration form

venv

- Python virtual environment folder containing project-specific dependencies.
- Internally includes:
 - **Include/, Lib/, Scripts/** – Required subfolders for virtual environment functionality
 - **pyvenv.cfg** – Configuration file for the virtual environment
 - **app.py**
- Main application file.
- Contains all route definitions, MongoDB database operations, and core Flask logic.

credentials.py

- Stores sensitive information such as MongoDB connection strings.
- Helps in keeping authentication and database access secure.

README.md

- Provides an overview of the project.
- Includes instructions on installation, setup, and usage.

8. Working of the Project

8.1 UML Diagrams

To understand the system architecture and flow, the following UML diagrams are used:

Use Case Diagram

The Use Case Diagram illustrates the interaction between the users (normal users and admin) and the application.

It highlights the actions like viewing, searching, creating, editing, and deleting blog posts based on the user's role.

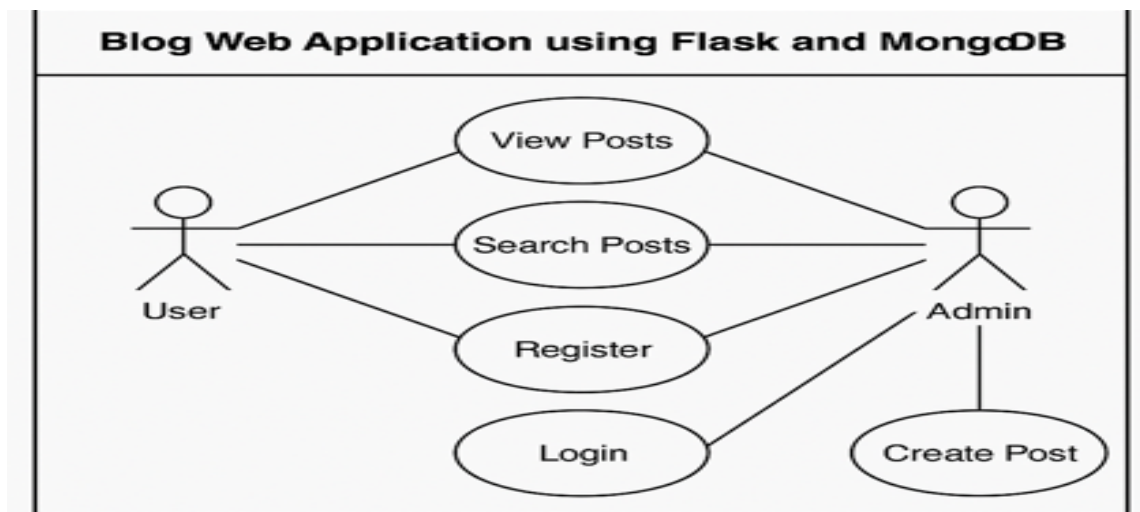


Figure 1 – Use Case Diagram of the Blog Web Application

Class Diagram

The Class Diagram explains how different classes or entities in the application (like User and Post) are structured. It shows attributes and operations related to each class and how they are related to each other.

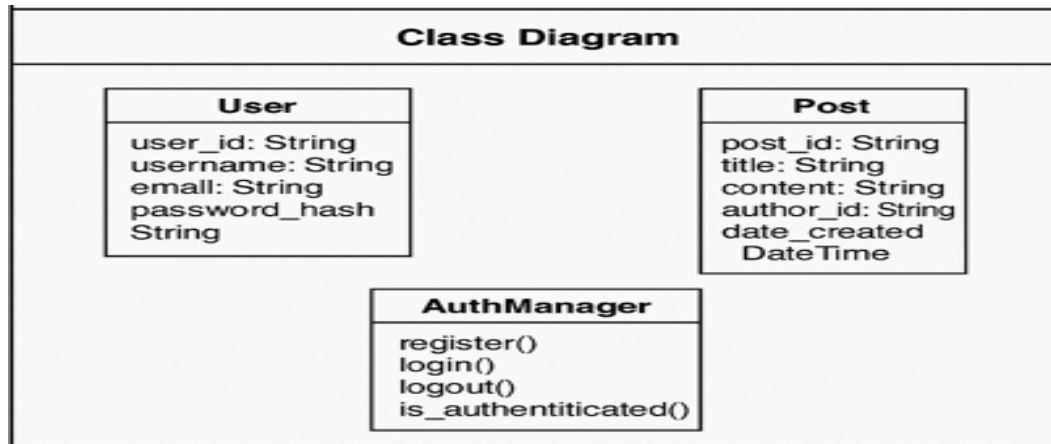


Figure 2 – Class Diagram of the Blog Web Application

7.2 Step-by-Step Working

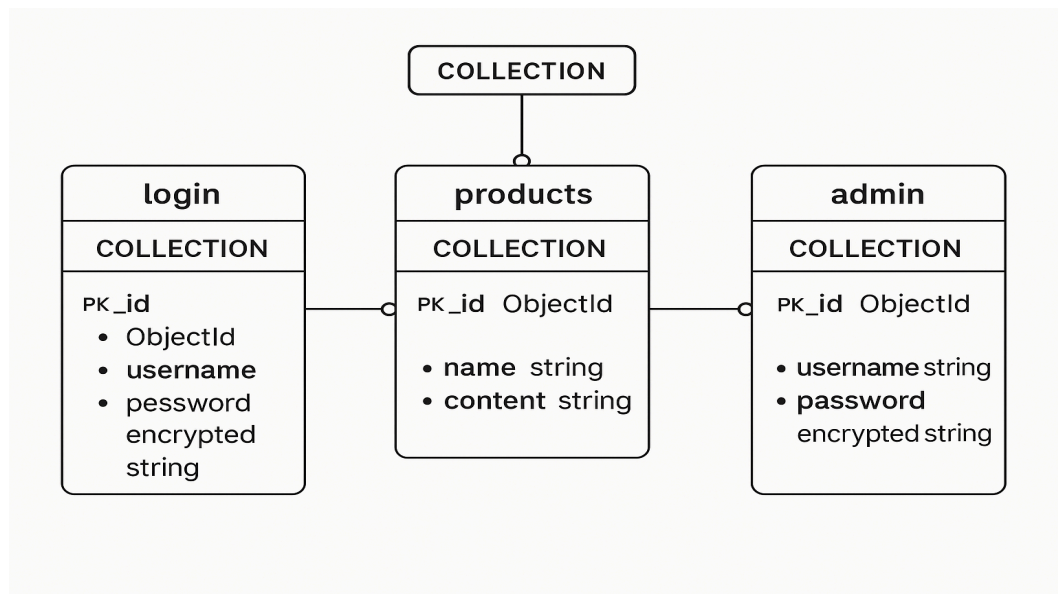
1. User opens the homepage of the blog application.
2. User can view and search for blog posts without logging in.
3. If the user wants to manage content, they need to **register** or **log in** as an admin.
4. Once logged in as admin, the user can:
 - **Create a new blog post.**
 - **Edit an existing blog post.**
 - **Delete a blog post.**
5. The system stores all data in **MongoDB**, including user credentials and blog content.
6. Admin can log out anytime, ending the session.

7.3 Database Representation Diagram

The below Entity-Relationship (ER) diagram represents the internal structure of the MongoDB database used in the Blog Web Application project.

There are **three collections**:

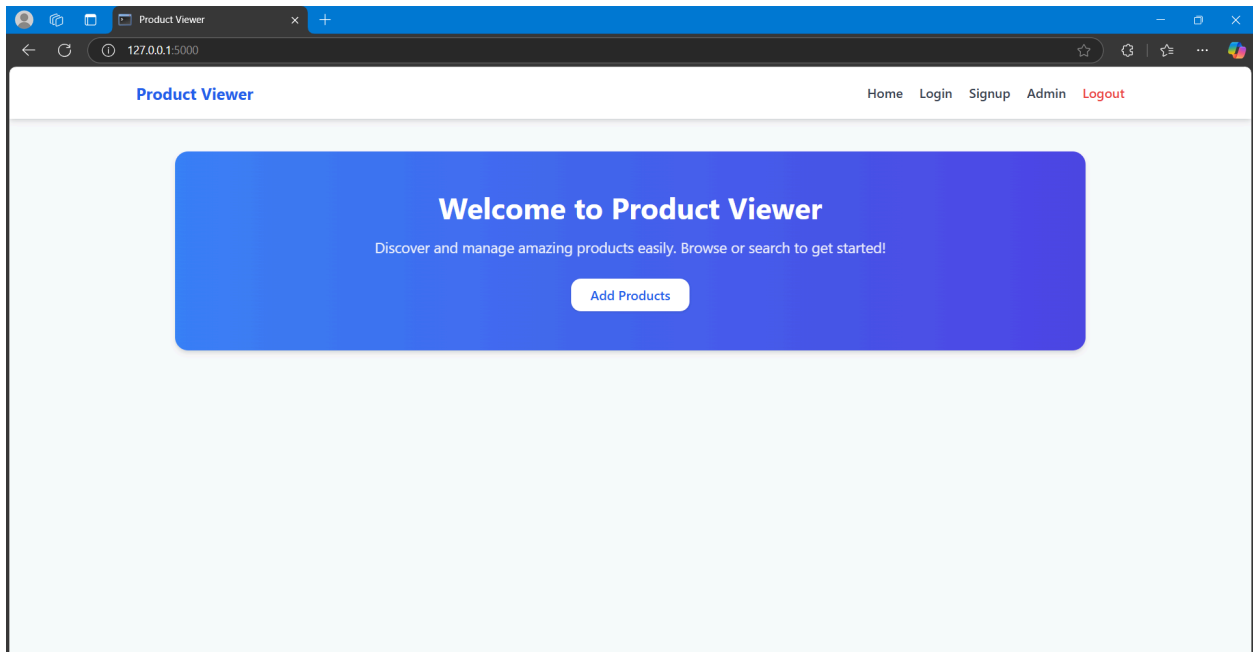
- **login** – stores user credentials including username and password.
- **products** – stores blog post-related data such as name and content.
- **admin** – stores admin login details and other related information.



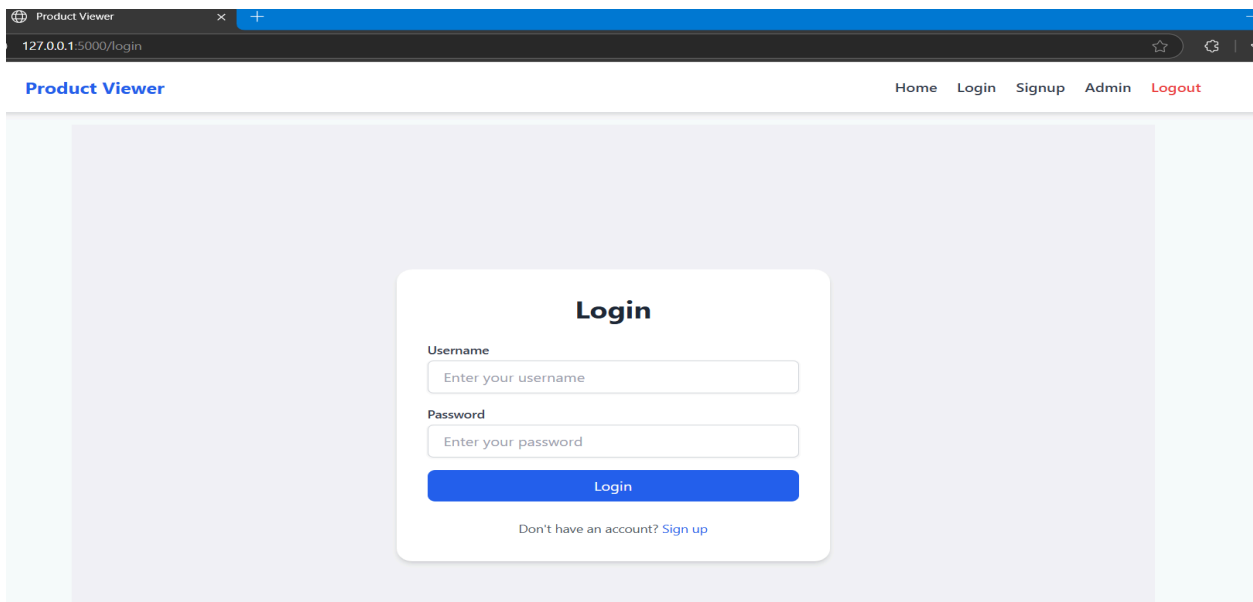
This diagram helps visualize how the data is organized and stored in MongoDB.

9. Screenshots:

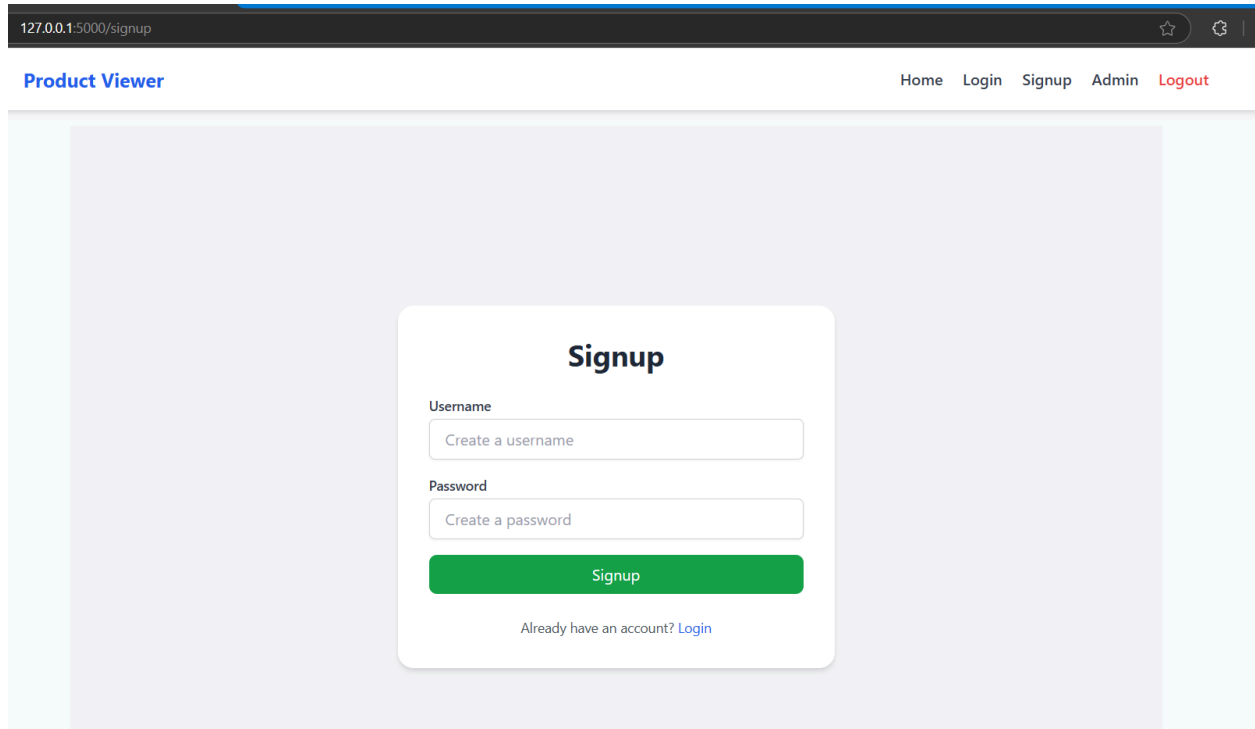
1. Home Page : Displays all published blog posts with a search option After login for users



2. Login Page : Allows registered users to log in and access the admin dashboard.

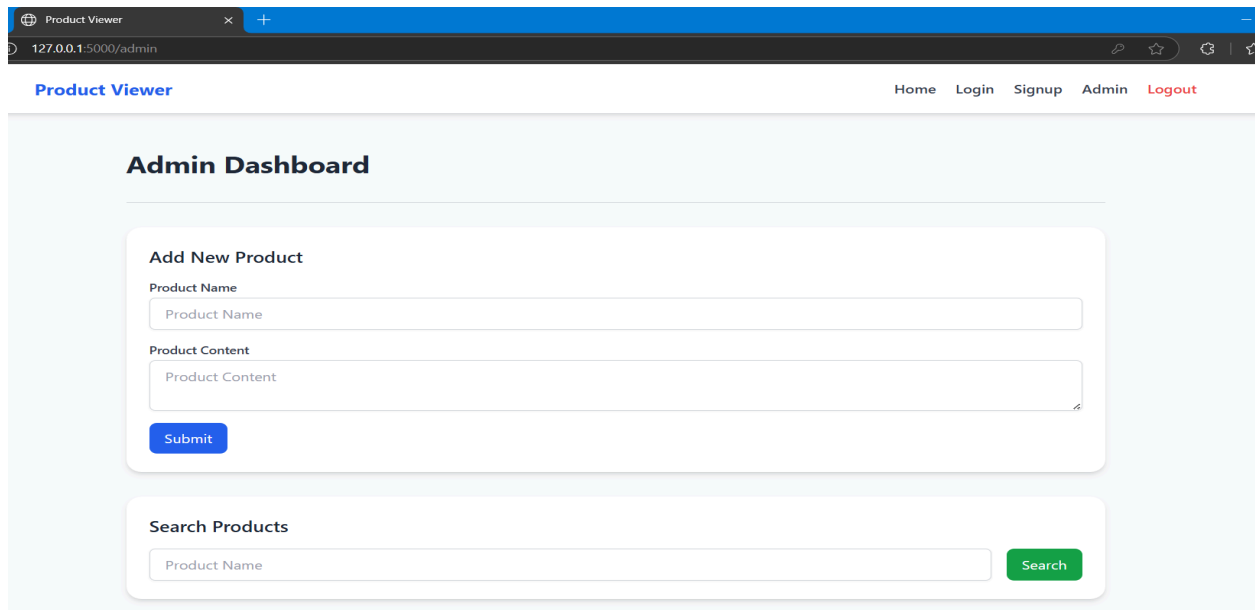


3. Signup Page : Enables new users to register for admin access.



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/signup". The page title is "Product Viewer". The navigation bar includes links for "Home", "Login", "Signup", "Admin", and "Logout". The main content area features a central "Signup" form. The form has two input fields: "Username" with the placeholder text "Create a username" and "Password" with the placeholder text "Create a password". Below these fields is a green "Signup" button. At the bottom of the form, there is a link that says "Already have an account? [Login](#)".

4. Admin Dashboard : Provides options to create, edit, and delete blog posts.



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/admin". The page title is "Product Viewer". The navigation bar includes links for "Home", "Login", "Signup", "Admin", and "Logout". The main content area features an "Admin Dashboard" section. This section contains two main forms. The first form is titled "Add New Product" and has two input fields: "Product Name" and "Product Content". Below these fields is a blue "Submit" button. The second form is titled "Search Products" and has a single input field labeled "Product Name" and a green "Search" button.

5. Create New Post : Lets admin add a new blog post with title and content.

Add New Product

Product Name

Product Content

Submit

After submit

All Products

Nothing 2a

This is a mobile phone brand

Delete

Edit

6. Search Page : Allows users to find blog posts by keywords.

Search Products

Search

After search

Search Results	
NAME	CONTENT
Nothing 2a	This is a mobile phone brand

10. Sample Code Snippets

MongoDB Connection (credentials.py)

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017/")
db = client['blog_app']
```

Flask Route for Creating Post (app.py)

```
@app.route('/create', methods=['GET', 'POST'])
def create_post():
    if request.method == 'POST':
        title = request.form['title']
        content = request.form['content']
        db.posts.insert_one({'title': title, 'content': content})
        return redirect(url_for('admin'))
    return render_template('create_post.html')
```

11. Challenges Faced

- Connecting Flask to MongoDB using Pymongo for the first time.
- Managing session-based login system securely.
- Displaying dynamic content on HTML pages using Jinja templates.
- Handling CRUD operations correctly with form validations.

12. Conclusion

This project helped me understand full-stack web development using Python. I learned how to build a real-world blog application using Flask, connect it with a NoSQL database like MongoDB, and create a login-protected admin dashboard.

Through this project, I gained hands-on experience with:

- Flask routing and templates
- MongoDB document storage
- Session-based login and logout
- Full web development workflow

13. Future Improvements

- Add comments feature under each blog post.
- Add image upload functionality in posts.
- Improve UI design using Bootstrap or Tailwind CSS.
- Add email verification for new users.
- Use JWT authentication for better security.

14. Reference:

- [MongoDB Documentation](#)
- [W3Schools HTML/CSS Reference](#)

15. [Github Link](#)