Project Synopsis

Simple Blog Website using Flask and SQLite

Author: Apoorv Malik (*apoorv.malik1@incedoinc.com*)

Objective:

The objective of this project is to design and develop a minimal web application that allows users to create, view, and delete blog posts. The project demonstrates the integration of a Python-based backend using Flask with a lightweight SQLite database for persistent storage, while maintaining a simple, user-friendly interface.

Motivation:

Even the simplest blog platform helps demonstrate full-stack web development fundamentals—handling user authentication, database operations, and frontend rendering. This project serves as a hands-on example for learning the core concepts of web frameworks, database connectivity, and client-server interaction.

Scope:

The website allows users to:

- Register and log in with basic credentials.
- Create, view, and delete personal blog posts.
- View all blogs posted by different users.
- Each blog entry includes author name, content, and timestamp of creation.

The system uses Flask for the backend logic and routing, SQLite for data storage, and HTML/CSS templates for the frontend display. It is lightweight and designed for small-scale, local deployments.

Technology Stack:

- Frontend: HTML, CSS, Jinja2 Templates
- Backend: Flask (Python Web Framework)
- Database: SQLite
- Tools Used: Postman for API testing, Python 3.10+ environment

---

Expected Outcome:

The final application provides a functional prototype of a blogging system, showcasing how Flask interacts with a relational database and template engine to deliver a simple CRD (Create, Read, Delete) web solution.

Project Report

Simple Blog Website using Flask and SQLite

---

1. Introduction

The Simple Blog Website is a lightweight web application that allows users to create, view, and manage blog posts.

It demonstrates the core principles of web development, including frontend-backend communication, database connectivity, and user session handling. The project focuses on simplicity and functionality, making it an ideal learning project for beginners in Python web development.

---

2. Objectives

The main objectives of the project are:

1. To build a simple, functional blogging platform using Flask.
2. To understand how web frameworks handle routing, sessions, and templates.
3. To implement database operations (Create, Read, Delete) using SQLite.
4. To develop a clean, minimal frontend interface for user interaction.
5. To connect frontend templates with backend logic and database queries.

---

3. System Design

3.1 Architecture Overview

The project follows a client-server architecture:

- The client (browser) sends HTTP requests to the Flask backend.
- The Flask server processes the requests, interacts with the SQLite database, and sends back dynamic HTML responses.

3.2 Components

➡ <u>Frontend (User Interface)</u>: Built using HTML, CSS, and Jinja2 templates.
Pages include:
- Home: Displays all blog posts.
- Create: Allows logged-in users to write new blogs.
- Login & Register: For user authentication.
- About: Describes the website's purpose and features.

➡ <u>Backend (Server Logic)</u>: Written in Flask. Handles:
- User registration and login.
- Creating, displaying, and deleting blogs.
- Flash messages for user feedback.
- Managing sessions for logged-in users.

➡ <u>Database (SQLite)</u>: Stores user information and blog entries.
Tables:
- users(id, username, password)
- blogs(id, user_id, title, content, timestamp)

The <u>*timestamp*</u> column uses SQLite's automatic timestamp.

4. Implementation

4.1 Tools and Technologies

- Programming Language: Python 3.x
- Framework: Flask
- Database: SQLite
- Testing Tool: Postman (for API testing)
- Template Engine: Jinja2

4.2 Backend Logic

The Flask app defines routes for major operations:

- POST /register — creates a new user.
- POST /login — verifies user credentials.
- POST /blogs — adds a new blog entry.
- GET /blogs — fetches all blogs from the database.
- DELETE /blogs/<id> — removes a blog by ID.

4.3 Frontend Integration

Templates extend a common base.html file for consistent layout.
The app uses Jinja2 syntax like:

```
{% if user %} ... {% endif %}
{{ b.title }}
```

to dynamically render data from the backend.

Flash messages are displayed to show success or error feedback (e.g., “Blog created successfully!” or “Invalid credentials”).

5. Results and Testing

The application successfully allows users to:

- Register and log in.
- Create new blog posts.
- View all existing blogs with author names and timestamps.
- Delete their own posts.

Postman was used to verify backend endpoints, ensuring proper communication between Flask routes and the SQLite database. The web interface is responsive, clean, and easy to navigate.

6. Conclusion

The Simple Blog Website project successfully demonstrates the integration of a web framework, a database, and HTML templating to form a complete full-stack application. It highlights how Flask and SQLite can be used to rapidly prototype dynamic web applications while keeping the codebase simple and understandable.

This project provides a strong foundation for future enhancements such as:

- Editing blog posts.
- Adding user profiles or comments.
- Deploying the app on a live server.

Overall, the project achieves its objective of showing the fundamental structure and workflow of a small-scale blog system using Flask and SQLite.