Flask

Shweta Ajay Shinde
Masters in Data Analytics, San Jose State University
Data 230: Data Visualization
Instructor: Venkata Duvvuri
November 19, 2024

Introduction To Flask

Abstract

This report discusses the design and implementation of a Flask-based web application for managing blog posts. The project demonstrates key web development concepts, including CRUD operations, routing, and database management, using SQLite as the backend. The application offers functionalities such as creating, viewing, updating, and deleting posts, with a focus on simplicity and usability. The report highlights the architecture, implementation details, and outcomes of the project, as well as proposed future enhancements such as user authentication and responsive design.

Introduction

Web applications have become integral to modern technology, enabling interactive and user-friendly platforms for various purposes. This project, a Flask Blog Application, is designed to demonstrate foundational web development concepts using the Flask framework. The application allows users to perform CRUD operations on blog posts, integrating database management and user interaction through a simple and intuitive interface.
The objectives of the project are to:

1. Develop a functional web application using Flask.
2. Implement CRUD functionalities with SQLite.
3. Enhance understanding of web application routing and templating.

Application Purpose and Use Case

This app is designed as a simple blogging platform where users can create, edit, view, delete. It can be used for personal blogs, project documentation, or small content management systems.
This blogging application provides a platform for users to create, view, edit, and delete blog posts, allowing them to manage their content in a simple and efficient manner. It can be used by individuals for personal blogging or small teams for collaborative content management. The app enables seamless interaction with a database, making it easy to store and retrieve posts. Additionally, this type of platform is ideal for anyone looking to build a lightweight blogging system with basic features, without the complexity of larger CMS platforms.

Flask Framework Advantages

Flask was chosen for its simplicity and minimalism, making it a perfect choice for beginners and small projects. It allows developers to focus on the core features of the application rather than dealing with complex configurations. Flask’s modular design allows for easy integration with databases, templates, and static files. Moreover, its flexibility means developers can scale the application as needed or integrate advanced features without the overhead of larger frameworks.

Methodology

Application Design

The application follows a modular structure:

- **Flask Framework**: Handles routing, templating, and request handling.
- **SQLite Database**: Stores blog posts with fields for id, title, and content.
- **HTML Templates**: Used for rendering the user interface.

The application includes the following routes:

1. GET /: Displays all blog posts.
2. GET /<int:post_id>: Displays a single post.
3. POST /create: Handles new post creation.
4. POST /<int:id>/edit: Updates an existing post.

Detailed Explanation of Routes

1. GET /: Displays All Blog Posts

- **Purpose**: This route serves as the homepage of the application. It displays a list of all the blog posts stored in the database.
- **Process**:
    1. Connects to the database.
    2. Fetches all the posts from the posts table.
    3. Passes the retrieved data to the homepage template (Figure 1) for rendering.
- **Outcome**: Users can see all the blog posts in one place, usually with their titles and a short preview of the content.

Figure 1:
index.html

```
{% extends 'base.html' %}
{% block content %}
    <h1>{% block title %} Welcome to FlaskBlog {% endblock %}</h1>
    {% for post in posts %}
        <a href="{{ url_for('post', post_id=post['id']) }}">
            <h2>{{ post['title'] }}</h2>
        </a>
        <span class="badge badge-primary">{{ post['created'] }}</span>
        <a href="{{ url_for('edit', id=post['id']) }}">
            <span class="badge badge-warning">Edit</span>
        </a>
        <hr>
    {% endfor %}
{% endblock %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FlaskBlog</title>
    <link rel="stylesheet" href="{{ url_for('static', filename= 'css/style.css') }}">
</head>
<body>
  <h1>Welcome to FlaskBlog</h1>
</body>
</html>
```

2. GET /<int:post_id>: Displays a Single Post

- **Purpose**: This route is used to view the details of a specific blog post by its unique ID.
- **Process**:
    1. Extracts the post_id from the URL.
    2. Fetches the corresponding post from the database using the ID.
    3. Passes the post data to the (Figure 2) template for rendering.
- **Outcome**: Users can view the complete content of a selected post along with its title.

Figure2:
post.html

```
{% extends 'base.html' %}
{% block content %}
    <h2>{% block title %} {{ post['title'] }} {% endblock %}</h2>
    <span class="badge badge-primary">{{ post['created'] }}</span>
    <p>{{ post['content'] }}</p>
{% endblock %}
```

3. POST /create: Handles New Post Creation

- **Purpose**: This route enables users to create a new blog post by submitting a form.
- **Process**:
    1. Accepts input (title and content) from a form.
    2. Validates the input to ensure the title is not empty.
    3. Inserts the new post into the posts table in the database.
    4. Redirects the user back to the homepage to view the newly added post.
- **Outcome**: Users can add new blog entries that are saved in the database (Figure 3).

Figure 3:
create.html

```
{% extends 'base.html' %}
{% block content %}
<h1>{% block title %} Create a New Post {% endblock %}</h1>
<form method="post">
    <div class="form-group">
        <label for="title">Title</label>
        <input type="text" name="title"
               placeholder="Post title" class="form-control"
               value="{{ request.form['title'] }}">
    </div>
    <div class="form-group">
        <label for="content">Content</label>
        <textarea name="content" placeholder="Post content"
                  class="form-control">{{ request.form['content'] }}</textarea>
    </div>
    <div class="form-group">
        <button type="submit" class="btn btn-primary">Submit</button>
    </div>
</form>
{% endblock %}
```

4. POST /<int:id>/edit: Updates an Existing Post

- **Purpose**: This route allows users to edit an existing blog post.
- **Process**:
    1. Fetches the current data of the post to be edited and pre-fills the form.
    2. Accepts updated input from the form.
    3. Validates the new data and updates the corresponding row in the database.
    4. Redirects the user back to the homepage to reflect the changes.
- **Outcome**: Users can update the content or title of a blog post(Figure 4).

Figure 4:
edit.html

```
{% extends 'base.html' %}

{% block content %}
<h1>{% block title %} Edit "{{ post['title'] }}" {% endblock %}</h1>

<form method="post">
    <div class="form-group">
        <label for="title">Title</label>
        <input type="text" name="title" placeholder="Post title"
               class="form-control"
               value="{{ request.form['title'] or post['title'] }}">
        </input>
    </div>

    <div class="form-group">
        <label for="content">Content</label>
        <textarea name="content" placeholder="Post content"
                  class="form-control">{{ request.form['content'] or post['content'] }}</textarea>
    </div>
    <div class="form-group">
        <button type="submit" class="btn btn-primary">Submit</button>
    </div>
</form>
<hr>
<hr>

<form action="{{ url_for('delete', id=post['id']) }}" method="POST">
    <input type="submit" value="Delete Post"
           class="btn btn-danger btn-sm"
           onclick="return confirm('Are you sure you want to delete this post?')">
</form>
{% endblock %}
```

Database Schema

The application uses SQLite as the database to store blog posts. The schema for the database is simple and consists of a single table named posts.

Table: posts

Column	Data Type	Description
id	INTEGER	Primary Key, Auto-incremented, Unique identifier.
title	TEXT	Title of the blog post. Cannot be NULL.
content	TEXT	Main content of the blog post.

- **Example Rows**:
id	**title**	**content**	
1	My First Post	This is my first blog!
2	Flask Basics	Learning Flask is fun.

Flask

| 3 | This is my 1st Blog |Happy Happy |

How Routes and Database Interact

1. **Create**: A new row is added to the posts table when a user submits the "Create" form.
2. **Read**: Data from the posts table is retrieved to display posts on the homepage or in detailed views.
3. **Update**: Existing rows in the posts table are modified when a user edits a post.
4. **Delete**: A row is removed from the posts table when a user deletes a post.

Figure 5:
init_db.py

```
import sqlite3

# open a connection between python script and database.db to create it
connection = sqlite3.connect('database.db')

# open the schema.sql to read what inside it
with open('schema.sql') as f:
    connection.executescript(f.read())

# make the cursor to execute what inside the schema in database
cur = connection.cursor()

cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('First Post', 'Content for the first post')
            )

cur.execute("INSERT INTO posts (title, content) VALUES (?, ?)",
            ('Second Post', 'Content for the second post')
            )

connection.commit()
connection.close()
```

Error Handling

Error handling in this application is done using Flask's abort() function, which helps manage missing pages or invalid requests by triggering a 404 (Figure 6) error when a post is not found. This ensures the user experience remains smooth and prevents the app from breaking. It also improves usability by informing users when something goes wrong, such as when they attempt to access a non-existent post. Proper error handling ensures the application is robust, reliable, and user-friendly.

Figure 6:
Abort

```
def get_post(post_id):
    # open the connection to db
    conn = get_db_connection()
    # select the post base on it's id
    post = conn.execute('SELECT * FROM posts WHERE id = ?', (post_id,)).fetchone()
    # clos the connection
    conn.close()
    # checking if we already have the post or not
    if post is None:
        abort(404)
    return post
```

Flask

Flash Messaging

Flash messaging is used in the app to provide immediate feedback to users after they perform actions like creating, editing, or deleting posts. This feedback can include success messages (e.g., “Post created successfully!”) or error messages (e.g., “Title is required!”). Flash messages are temporary, disappearing after the user is redirected to another page, preventing unnecessary clutter. This feature enhances the user experience by keeping them informed of the actions they have taken.

Security Measures

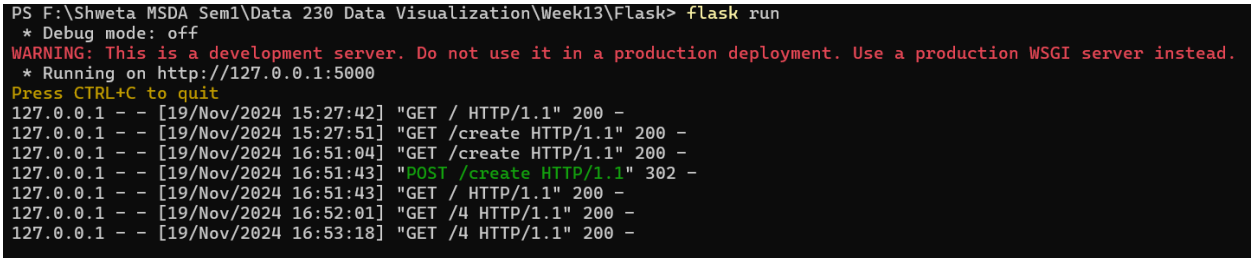
To protect the app from potential security vulnerabilities, Flask uses a SECRET_KEY for session management and protection against CSRF attacks. This ensures that data shared between the client and server remains secure. Additionally, input validation is employed to ensure that data entered by users (such as titles or content) is clean and properly formatted before being added to the database. These security measures prevent unauthorized access and maintain the integrity of the application.

Running the Flask Application

The Flask application is started using the flask run command in the project directory. It launches a development server on http://127.0.0.1:5000, which can be accessed in a browser to test the app. The terminal logs all incoming requests, showing routes accessed and actions performed (e.g., GET, POST). Flask's warning emphasizes that this server is for development only and not recommended for production. For deployment, a production-grade server should be used. Please refer to the Figure 7

Figure 7:

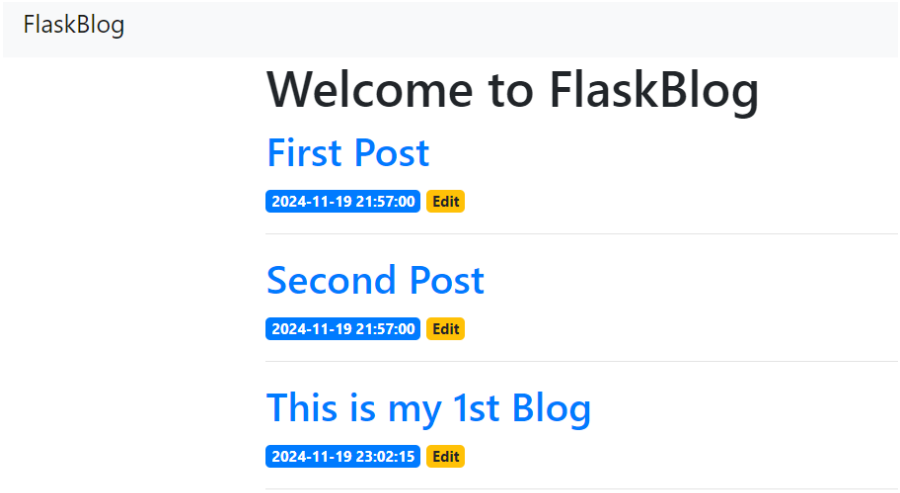
Flask Run



Testing

The application was manually tested to ensure that all features work as expected, including the creation, editing and deleting (Figure 8, 9, 10, 11, 12, 13) of posts. Special attention was given to edge cases, such as submitting empty titles or content to verify that the app responds with appropriate feedback. Routing was thoroughly tested to ensure users are directed to the correct pages, and database interactions were verified to confirm data was accurately added, modified, or removed. These tests ensure that the app functions correctly and delivers a reliable user experience.

Figure 8:
Flask



...

- New Post

Figure 9:
Add new

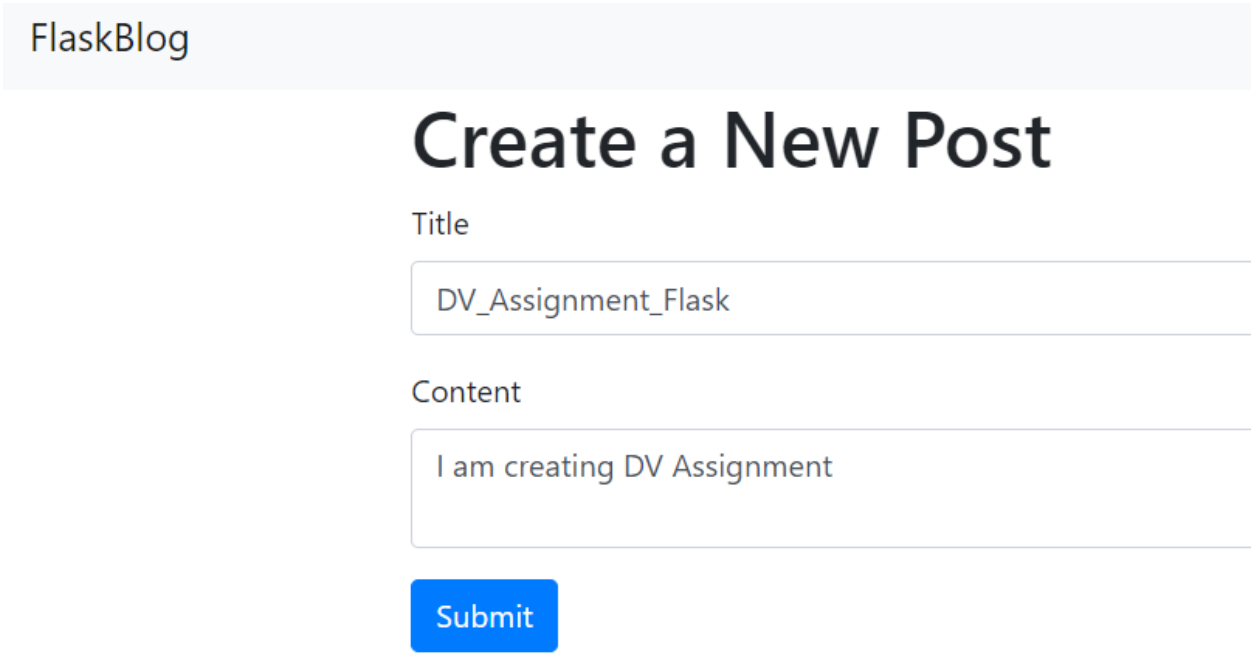


Figure 10:
Updated Flask

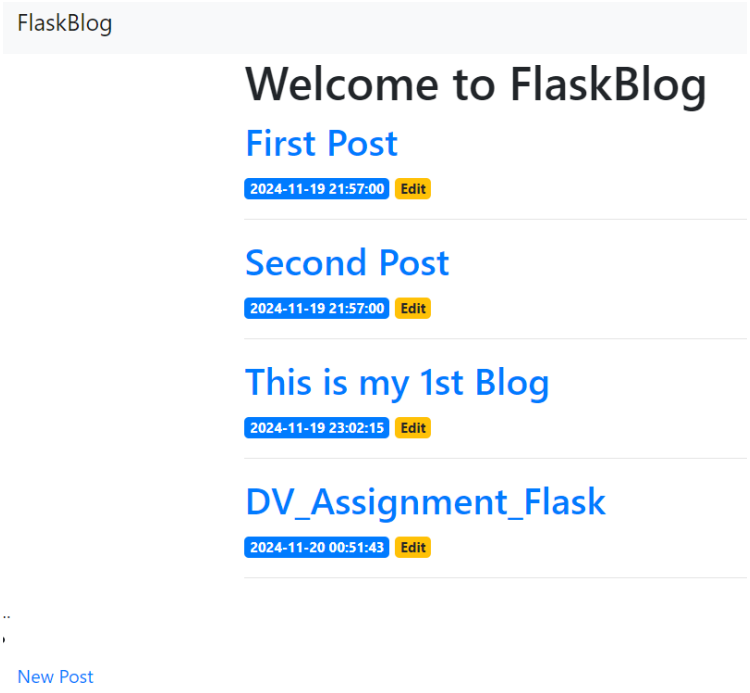
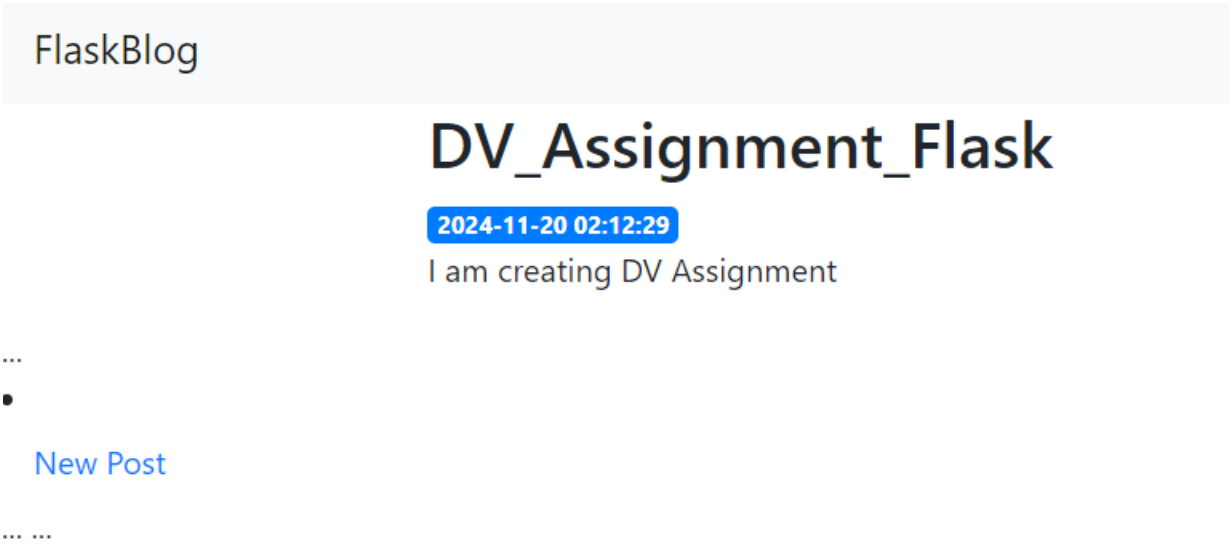


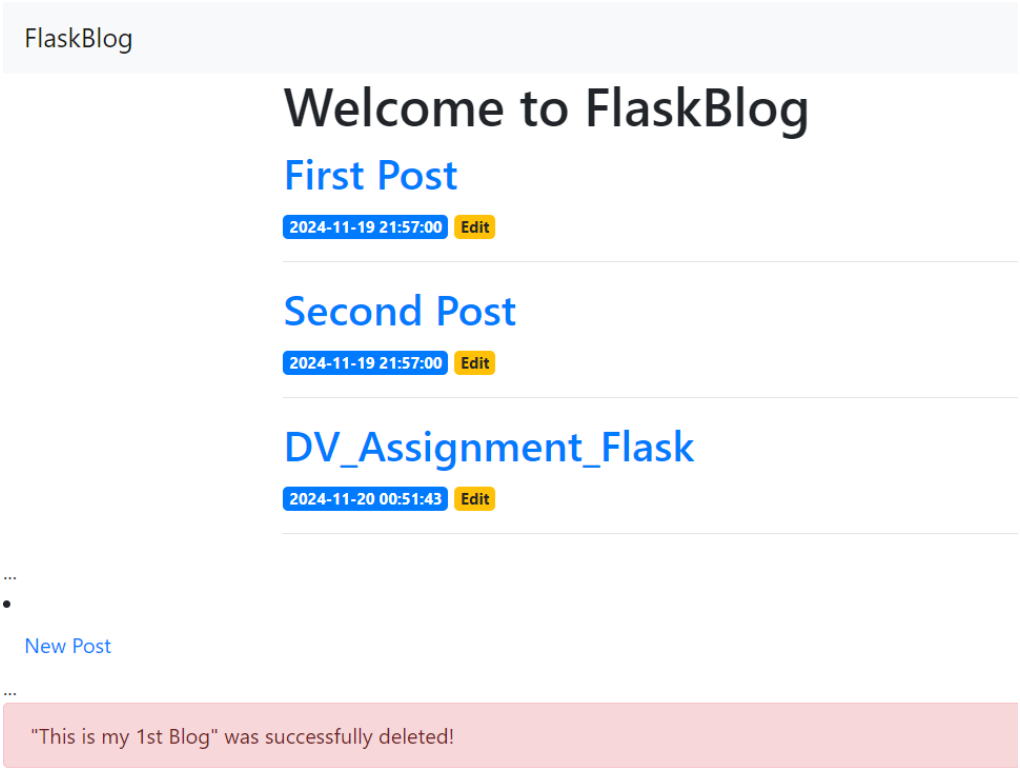
Figure 11:
Post Details



Flask

Figure 12:
Delete Post

Figure 13:
Updated Post



Potential Enhancements

Some potential enhancements to improve the app include adding user authentication, which would allow users to create and manage their own posts securely. Another potential feature is pagination, which would be useful for handling large numbers of posts, making the application more efficient and user-friendly. Supporting image uploads would also make the platform more versatile, allowing users to enrich their blog posts with visual content. These improvements could elevate the app to a more sophisticated level, making it better suited for diverse use cases.

Conclusion

In conclusion, this Flask-based blog application effectively demonstrates the use of basic web development concepts, such as CRUD operations, template rendering, and database integration. It serves as an excellent example for anyone looking to understand how to build a simple web application with Flask. While the app is functional and user-friendly, there are several avenues for future improvement, including scalability, security, and feature enhancements. Overall, this project showcases the potential of Flask for rapid development of lightweight web applications.

Additionally, the application highlights the importance of clean code structure and modular design, making it easy to maintain and expand. By leveraging Flask's simplicity and SQLite's efficiency, the app provides a strong foundation for learning and experimentation. Whether used for educational purposes or as a prototype for a larger project, this application underscores the value of open-source frameworks in building accessible and effective solutions.

References

Saber, N. (2024). *Building a Flask Blog: A Step-by-Step Guide for Beginners*. Medium. https://medium.com/@noransaber685/building-a-flask-blog-a-step-by-step-guide-for-beginners-8bffe925cd0e.

Shinde, S. K. (n.d.). *Flask*. GitHub. Retrieved November 19, 2024, from https://github.com/ShindeShwetaK/Flask