Mini Project Brief: Flask Blogging Website

Instructor: Giridhari Lal Gupta

---

1. Project Overview

Build a fully functional blogging platform using the Flask framework. This mini project will integrate core Flask concepts—routing, templates, forms, authentication, and static assets—with front-end technologies and database models to deliver a real-world web application.

1.1 Objectives

- Apply Flask fundamentals (routing, views, error handling).
- Implement role-based access control (Admin vs. Publisher vs. Visitor).
- Manage dynamic content with CRUD operations and draft workflows.
- Integrate a WYSIWYG or Markdown editor for rich-text formatting.
- Handle media uploads (images, audio, video) and static file serving.
- Enable user interactions: commenting, reactions, and sharing.
- Design search, filtering, and pagination features.
- Secure the app against common web vulnerabilities (CSRF, XSS, SQL injection).
- Deploy the finished app in a containerized environment.

---

2. User Roles & Permissions

- **Administrator**
- Manage users and roles
- Approve, publish, or delete any post
- Moderate comments and manage reports
- **Publisher (Content Creator)**
- Register & authenticate
- Create, edit, delete own posts
- Save drafts and publish when ready
- View personal publication history
- **Visitor (Unauthenticated)**
- Browse and read published posts
- Comment on and react to posts
- Filter and search content by author, tag, date
- Share posts via social links

---

3. Key Features & Functional Requirements

1. **Authentication & Authorization**

2. Email/password registration with verification
3. Password reset via secure tokens
4. Role enforcement using Flask-Login and Flask-Principal
5. **Post Management**
6. CRUD operations for posts
7. Drafts workflow (save without publishing)
8. Publish scheduling (optional)
9. Rich content: text, images, audio, video
10. Use a WYSIWYG (TinyMCE/CKEditor) or Markdown editor
11. Tags and categories for organization
12. **User Interaction**
13. Commenting system with moderation
14. Like/dislike reactions
15. Social sharing buttons
16. **Content Browsing**
17. Lists by author, tag, date, popularity
18. Full-text search and filter options
19. Pagination for large result sets
20. **Notifications**
21. Email alerts for comments and approvals
22. In-app notifications badge
23. **Admin Dashboard**
24. Overview of site metrics (posts, comments, users)
25. Manage reported content

---

4. Non-Functional Requirements

- **Performance:** Fast page loads (<2s), caching for hot content
- **Security:** CSRF protection, input sanitization, secure password storage
- **Responsiveness:** Mobile-first design with Bootstrap or Tailwind CSS
- **Maintainability:** Modular code with Blueprints, PEP8 compliance, unit tests
- **Deployment:** Docker-based setup, environment variable config, PostgreSQL in production

---

5. System Architecture & Tech Stack

- **Backend:** Flask, Flask-Login, Flask-WTF, SQLAlchemy, Flask-Migrate
- **Database:** SQLite (dev), PostgreSQL (prod)
- **Frontend:** Jinja2 templates, Bootstrap 5 or Tailwind, Vanilla JS
- **Media Storage:** Local (`/static/uploads`) or AWS S3 (optional)
- **Caching & Search:** Redis cache, PostgreSQL full-text search (optional Elasticsearch)
- **Deployment:** Docker, Gunicorn, Nginx

---

6. Milestones & Timeline

Day	Duration	Tasks
1	3 hrs	Project setup, environment configuration, user authentication
2	3 hrs	Post CRUD operations, draft workflow implementation
3	3 hrs	Media uploads setup, WYSIWYG/Markdown editor integration
4	3 hrs	Commenting system, reactions, notifications
5	3 hrs	Search/filters/pagination, admin dashboard, testing/deployment

---

7. Deliverables & Submission Guidelines

- **Code Repository:** Well-structured GitHub repo with clear README
- **Documentation:** Project report covering architecture, API endpoints, and usage
- **Demo Video (5–10 min):** Walkthrough of key features
- **Deployment Link:** Live site URL or Docker-compose instructions

---

8. Evaluation Criteria

- **Functionality:** All core features implemented and working
- **Code Quality:** Readable, maintainable, and well-documented
- **Security & Performance:** Proper protections and acceptable speed
- **User Experience:** Intuitive UI/UX and responsive design
- **Innovation:** Any additional creative features beyond requirements

---

9. Additional Suggestions

- Implement social login (OAuth)
- Add scheduled publishing of posts
- Build a RESTful API for mobile clients
- Integrate analytics (Google Analytics/Sentry)
- Provide multilingual support with Flask-Babel

Good luck! Reach out to **Giridhari Lal Gupta** with any questions or to schedule a review session.