# **Flask Application Project Documentation**

**Overview of the Project:**

This project is a basic web application developed using the Flask framework. It implements user authentication, role-based access control, and the ability to post content to a blog-style home page. The application supports user login, logout, and posting functionality, and includes an admin dashboard for administrative tasks.

**Features:**

1.**User Authentication:**

•Users can register, login, and logout.

•Passwords are hashed using Werkzeug for security.

2.**Admin Dashboard:**

•Admin users have access to a dashboard where they can view all users and perform administrative tasks.

•Role-based access ensures that only users with admin privileges can access the dashboard.

3.**Post Creation and Display:**

•Logged-in users can create posts that are displayed on the homepage.

•Posts include a title, content, and a timestamp.

4.**Database Management:**

•All data, including users and posts, are stored in a SQLite database using SQLAlchemy.

•Database tables are created dynamically on application startup.

**Steps Taken During Development:**

1. Setting Up the Flask Application

• Installed Flask and its extensions using pip install flask flask\_sqlalchemy flask\_login.

• Initialized the Flask app, set up configurations, and created a SQLite database using SQLAlchemy.

2. User Authentication

• Defined a User model in SQLAlchemy, with attributes for the user ID, username, password hash, and admin status.

• Implemented login and logout functionality using Flask-Login.

• Passwords are securely hashed using Werkzeug’s password hashing functions.

3. Post Management

• Created a Post model that allows users to create blog posts, with attributes such as title, content, and timestamp.

• Users can create new posts through a form, and these posts are displayed on the homepage.

•The Home route fetches and displays all posts from the database.

4. Admin Dashboard

•Admin users have access to a dashboard where they can view all users and manage the application.

•Non-admin users attempting to access the dashboard are redirected with an “Access Denied” message.

•The dashboard route is protected by Flask-Login’s login\_required decorator and additional checks for admin status.

# Flask Configurations:

•SECRET\_KEY: A secret key was set to secure the application and prevent tampering with session data.

•SQLALCHEMY\_DATABASE\_URI: Configured the SQLite database URI to store data.

app.config['SECRET\_KEY'] = 'your\_secret\_key'

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///site.db'

5. Database:

•SQLAlchemy was used for database ORM, creating the necessary models for users, posts, and comments.

db = SQLAlchemy(app)

**Steps for Deployment:**

• Setting Up the Virtual Environment:

• Created and activated a virtual environment using python -m venv venv.

• Installed all dependencies using pip install -r requirements.txt.

• Database Initialization:

• The db.create\_all() function was called to create the necessary database tables for users and posts.

• Running the Flask Application:

• Ran the application in development mode with the command python app.py.

• The app runs on localhost:5000.

**Challenges Faced:**

**1. Role-Based Access Control:**

•Initially, all users could access the admin dashboard. This was corrected by implementing role-based checks using the current\_user.is\_admin attribute.

**2. Password Hashing:**

•Managing user authentication securely required implementing password hashing. The Werkzeug password hashing utilities were used to securely store passwords in the database.

**3. Database Migrations:**

•Initially, the database tables were created manually. However, as the project grew, it became necessary to automate database creation using SQLAlchemy’s db.create\_all().

**4. Handling Access Control for Non-Admin Users:**

•Ensuring that only admin users could access the dashboard required careful handling of Flask-Login’s decorators and conditional checks within routes.

**Example Scenarios:**

1. User Registration and Login:

• A new user navigates to the registration page and creates an account.

• After registration, they log in and are redirected to the homepage where they can see existing posts.

2. Admin Access:

• An admin user logs in and is redirected to their dashboard.

• They view all registered users and can perform administrative tasks.

3. Post Creation:

• A logged-in user navigates to the “Add Post” page and submits a new post.

• The post is immediately visible on the homepage, and other users can view it.

Technologies Used:

1. Flask: Web framework for building the application.

2. SQL Alchemy: ORM for database management.

3. Flask-Login: User session management and login/logout functionality.

4. Werkzeug: Secure password hashing.

**CONCLUSION:**

This documentation covers the development process for the Flask application, detailing the steps involved in setting up the app, the challenges encountered, and the implementation of various features. The project demonstrates the integration of user authentication, post management, and role-based access control in a Flask environment.