Software Requirement Specifications

Python Based Blog Website

Submitted By:

Name: Tehreem Fatima

Roll No: F22BINFT1M01138

Submitted To:

Ms. Sara Farid

Department of Information Technology

Faculty of Computing

The Islamia University of Bahawalpur

Sr no	Details	Date	Supervisor signature
1	Initial SRS Draft Submitted		
2	Flowchart and Use case Review		
3	Final SRS Approval		

Summary

This project is a fully functional Python–Django based Blog Management System designed to demonstrate the implementation of modern web development concepts using the Django framework. It provides a complete and secure user authentication module, enabling users to register, log in, manage their profiles, and access personalized blog functionalities. The system supports comprehensive CRUD operations for blog posts, allowing authenticated users to create, edit, update, and delete posts with ease. Each blog entry can include rich text content and image uploads, enhancing the visual presentation and readability of articles. The project incorporates a structured front-end developed using HTML, CSS, and JavaScript, providing a clean and intuitive user interface. On the backend, it utilizes SQLite as the default database for storing user data, blog content, and media files. Additionally, the built-in Django Admin Panel offers powerful administrative controls for managing users, posts, and system configurations. This repository serves as a robust foundation for academic projects and can be extended into a production-ready platform by integrating features such as comments, categories, search filters, and role-based access control. Overall, the project demonstrates best practices in Django development and is suitable for learning, customization, and real-world deployment

Table of contents

1. Introduction

A blog website is an online platform that allows users to share their ideas, information, articles, and knowledge. The purpose of developing this Django Blog Website is to create a simple, user-friendly, and secure system where users can easily create, edit, and delete blog posts. This system is specifically designed for users who want to start personal blogging or explore web development for learning purposes.

The primary goal of this project is to provide an environment where users can organize their content and make it accessible for others to read. Administrators are granted full control to manage the system effectively, including deleting posts, managing user accounts, and moderating inappropriate content.

The web application is developed using the Django framework, ensuring security, performance, and maintainability. The system can be accessed via web browsers, and it is designed to be intuitive so that even basic users can navigate and use the platform with ease.

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to define the requirements for a Python-based blog website developed using the Django framework. This document serves as a comprehensive guide for stakeholders, developers, and testers to ensure the system delivers a secure, user-friendly platform for blogging activities, including user authentication, content management, and administrative oversight. It emphasizes core functionalities like user sign-up, blog post creation with multimedia, and scalable design for personal or small-scale use.

1.2 Scope

The Django Blog Website is a full-featured content management and publishing platform designed for modern blogging needs. The system provides comprehensive tools for content creation, user management, social interaction, and administrative control.

- User authentication (sign-up, login, logout).
- CRUD operations for blog posts (create, read, update, delete) with image uploads.
- An interactive user interface for browsing and managing content.
- Django admin panel for global management.

1.3 Product Perspective

The Django Blog Website operates as an independent, self-contained web application following Django's Model-View-Template architectural pattern. The system is designed to be deployable on various hosting platforms while maintaining consistent functionality across environments.

1.4 User Characteristics

- **Guest / Visitor**: can read blog posts (no login required).
- **Registered User**: You can register, login, write blog posts, edit or delete their own posts, upload media, view/manage their own content.
- **Administrator**: can manage (create/read/update/delete) any blog post or user account; manage media; administer site settings (if implemented).

1.5 Similar Apps and Systems / Literature Review

Competitors Quick View:

WordPress - Giant but insecure, slow
Medium - Beautiful but you don't own your content
Ghost - Modern but small community
Blogger - Simple but outdated

Key Research Facts:

What Works:

- Fast loading (<3 seconds) = happy users
- Comments & notifications = more engagement
- Mobile-friendly = essential today
- Good SEO = more visitors
- Security = trust

What Users Want:

- Easy to use but powerful
- Own their content
- Fast & reliable
- Safe & private
- Customizable

Market Trends:

- Headless CMS growing fast
- Mobile-first design standard
- Security breaches expensive ($4M average)
- Social features increase engagement

Our Opportunity:

Problems We Solve:

1. **Security issues** of WordPress
2. **Platform lock-in** of Medium
3. **Outdated tech** of Blogger
4. **Limited features** of simple platforms

Our Advantages:

- **Modern tech** (Django/Python)
- **Security-focused** from start
- **You own everything**
- **Scalable** from day one
- **Developer-friendly**

Simple Strategy:

1. **Build safe** - Security first
2. **Keep simple** - Easy for beginners
3. **Make fast** - Performance matters
4. **Stay open** - No lock-in
5. **Grow smart** - Scale when needed

1.6 Assumptions and Dependencies

- Users have modern browsers and internet connections.

- For production deployment, appropriate static/media serving configured (not just Django development server).
- Administrators implement backup and security (since default doesn't enforce advanced security measures).

1.7 Proposed Technologies

Backend (Python)

- **Python 3.10+** - Latest stable version
- **Django 4.2 LTS** - Web framework (security + features)
- **Django REST Framework** - APIs if needed
- **PostgreSQL** - Production database
- **SQLite** - Development/testing

Frontend

- **HTML5/CSS3** - Basic structure
- **Bootstrap 5** - Responsive design
- **JavaScript** - Basic interactivity

2. Requirements

2.1 Functional Requirements

2.1.1 Sign Up

- **FR-1.1:** The system shall allow a new user to register an account with a unique username and password (optionally email).
- **FR-1.2:** The system shall validate that username is unique.
- **FR-1.3:** The system shall validate password strength (minimum length, complexity — optionally).
- **FR-1.4:** The system may (optionally) require email verification before enabling full account access.

2.1.2. Login / Logout

- **FR-2.1:** The system shall allow registered users to log in using valid credentials.
- **FR-2.2:** The system shall allow users to log out.
- **FR-2.3 (Optional):** The system may support password reset/recovery via email.

2.1.3. Create Blog Post

- **FR-3.1:** Authenticated users (authors) shall be able to create new blog posts, specifying at least: title, body/content.
- **FR-3.2:** The system shall support optional media (image) upload associated with a post.

2.1.4. View Blog Posts

- **FR-4.1:** Authors shall be able to edit their own posts (title, content, image).
- **FR-4.2:** Authors shall be able to delete their own posts.

2.1.5. Edit & Delete Own Post

- **FR-5.1:** The system shall display posts: title, content (rendered HTML or text), author name, date/time (created/updated), image (if any).
- **FR-5.2:** The system shall allow all visitors (authenticated or not) to view published blog posts.

2.1.6. Admin Management

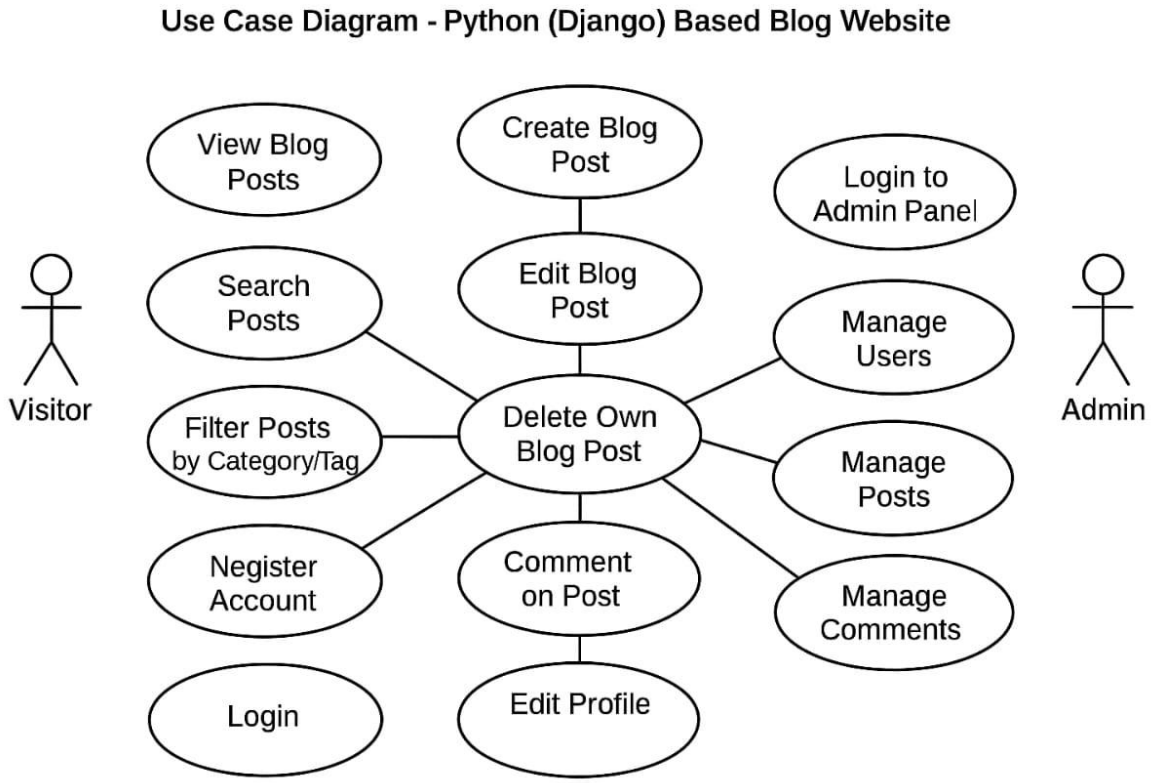
- **FR-6.1:** The system shall provide an admin interface where administrators can view all blog posts from all users.
- **FR-6.2:** The admin shall be able to edit or delete any post.
- **FR-6.3:** The admin shall be able to manage user accounts (delete/suspend users) if required.

2.2 Non-Functional Requirements

- **NFR-1 (Performance):** The system should load pages (listings, post detail) within acceptable time (e.g. < 2 seconds) under typical load for small to medium user base.
- **NFR-2 (Security):**
  • Passwords must be hashed (use Django's augh security).
  • CSRF protection enabled for all forms.
  • Input validation and sanitization to prevent XSS / injection.
  • Uploaded media files validated and stored securely (not executing as code).

- **NFR-3 (Reliability and Maintainability):**
  • The code should follow modular structure (separate Django apps where appropriate).
  • The system should handle errors gracefully (e.g. invalid URLs, missing media) and log exceptions.
- **NFR-4 (Usability):**
  • The UI should be clean and intuitive: simple navigation (home, log in/out, new post, edit, delete, view posts).
  • Forms should have proper validation and user-friendly error messages.
  • Templates should be responsive (work on various screen sizes).
- **NFR-5 (Extensibility):**
  • The design should allow for future enhancements (e.g. comments system, tags/categories, search, user profiles) without major rewrite.

3.Use Cases and Flow of Processes



3.1 Use Case – User Registration

Actors	Visitor (not logged in)
Precondition	Registration is enabled; user not already

	registered.
Postcondition	User account exists; user may log in.
Flow	1. Visitor navigates to the Registration page. 2. Visitor enters required details (username, password, confirm password; optionally email). 3. Visitor submits the form. 4. System validates inputs (uniqueness of username, password validity). 5. System creates user accounts. Optionally, send a verification email. User is redirected to Login page or dashboard (per design).

3.2. Use Case 2 – Login /Logout

Actor	Registered User
Flow	User navigates to login page → enters credentials → submits → system authenticates → if valid, user session created → user directed to home/dashboard. Logout: user clicks "Logout" → session ends → redirect to home or login page.

3.2. Use Case 3 – Create a New Blog Post

Actor	Authenticated User (Author
Flow	User clicks "New Post" → fills title, content/body, optionally uploads image → submits → system validates input and media → saves post (and image) → redirects to post detail or listing.

3.4 Use Case 4 – Edit / Delete Post

Actor	Author (for own posts) or Admin (for any post)
Flow (Edit):	User/Admin opens their post → clicks "Edit" → modifies title/body/image → submits → system validates and updates record/media → redirect to post.
Flow (Delete)	User/Admin clicks "Delete" → confirm → system deletes post record (and optionally media) → redirect to listing page.

3.5. Use Case 5 – View Post

Actor	Visitor / Any user
Flow	Navigate to home or posts listing → see list of posts (titles, excerpts) → click a post → view full content with image (if any), metadata (author, date). Possibly use pagination if many posts.

3.6. Use Case 6 – Admin Management

Actor	Administrator
Flow	Admin logs in → accesses admin panel → views list of all posts & users → selects user → performs edit or delete → confirms.

4. References

[1] Django Software Foundation, *Django Documentation*. Available: https://docs.djangoproject.com/

[2] Python Software Foundation, *Python Official Documentation*. Available: https://docs.python.org/

[3] Bootstrap Team, *Bootstrap Documentation*. Available: https://getbootstrap.com/

[4] W3Schools, *HTML, CSS, and JavaScript Tutorials*. Available: https://www.w3schools.com/