Project SECUREBLOG

auth\_user

|  |  |  |
| --- | --- | --- |
| id | username | email |
| 1 | alice | alice@email.com |

secureblog\_post

|  |  |  |
| --- | --- | --- |
| id | author\_id | title |
| 1 | 1 | Post |

Post.objects.create(author=request.user, title="Hello", content="World")

INSERT INTO secureblog\_post (author\_id, title, content) VALUES (1, 'Hello', 'World');

Model (Python class)

↓

ORM (Python query)

↓

Database Table (SQL query behind the scenes)

# Create a new post

Post.objects.create(author=request.user, title="First Post", content="This is a secure post.")

# Get all posts

Post.objects.all()

# Filter posts by current user

Post.objects.filter(author=request.user)

# Get a post by ID

Post.objects.get(id=1)

python manage.py dbshell

.tables

SELECT \* FROM secureblog\_post;

# SECURE BLOG - Project Documentation

## Table of Contents

1. Introduction

2. Project Overview

3. Features

4. Security Considerations

5. Role-Based Access Control (RBAC)

6. System Architecture

7. Database Design

8. Setup & Installation

9. Usage Guide

10. Future Enhancements

11. Credits

---

## 1. Introduction

Secure Blog (SECURE BLOG) is a Django-based web application designed to provide a secure and user-friendly platform for blogging. The system emphasizes security, including robust authentication, authorization, and role-based access control, ensuring that users interact with the platform appropriately according to their permissions.

---

## 2. Project Overview

The primary objective of SECURE BLOG is to create a blogging platform that is not only functional but also highly secure. The application enables users to create, read, update, and delete blog posts, while ensuring that sensitive operations are restricted to users with appropriate roles.

---

## 3. Features

- User registration and authentication

- Role-based access control (Admin, Editor, Author, User)

- Blog post creation, editing, and deletion

- Secure password management and password reset

- User profile management

- Admin dashboard for managing users and posts

- Responsive design for desktop and mobile devices

---

## 4. Security Considerations

- \*\*Authentication:\*\* Utilizes Django’s built-in authentication system with hashed passwords.

- \*\*Authorization:\*\* Enforces RBAC for all sensitive actions.

- \*\*Input Validation:\*\* All user inputs are validated and sanitized to prevent XSS and SQL Injection.

- \*\*Session Management:\*\* Secure cookie handling and session expiration.

- \*\*HTTPS Enforcement:\*\* Recommended for deployment to ensure encrypted data transmission.

---

## 5. Role-Based Access Control (RBAC)

### Roles:

- \*\*Admin:\*\* Full access to all resources and user management.

- \*\*Editor:\*\* Can manage (create, edit, delete) any blog post.

- \*\*Author:\*\* Can create and manage their own posts.

- \*\*User:\*\* Can read posts and comment, but cannot create or edit posts.

### Permissions Matrix:

| Feature | Admin | Editor | Author | User |

|-------------------|:-----:|:------:|:------:|:----:|

| Create Post | ✓ | ✓ | ✓ | ✗ |

| Edit Any Post | ✓ | ✓ | ✗ | ✗ |

| Edit Own Post | ✓ | ✓ | ✓ | ✗ |

| Delete Any Post | ✓ | ✓ | ✗ | ✗ |

| Delete Own Post | ✓ | ✓ | ✓ | ✗ |

| Manage Users | ✓ | ✗ | ✗ | ✗ |

| Comment | ✓ | ✓ | ✓ | ✓ |

| View Posts | ✓ | ✓ | ✓ | ✓ |

---

## 6. System Architecture

```

+---------------------+

| End User Browser |

+----------+----------+

|

v

+---------------------+

| Django Views |

+----------+----------+

|

v

+---------------------+

| Django Models |

+----------+----------+

|

v

+---------------------+

| SQLite DB |

+---------------------+

```

- \*\*Frontend:\*\* HTML templates, CSS, JavaScript (optional for interactivity)

- \*\*Backend:\*\* Django (Python)

- \*\*Database:\*\* SQLite (default, can be replaced by PostgreSQL/MySQL)

---

## 7. Database Design

\*\*Main Tables:\*\*

- `User`: Stores user credentials, role, and profile info.

- `BlogPost`: Stores blog content, author reference, timestamps.

- `Comment`: Stores comments, user reference, and post reference.

\*\*Sample BlogPost Table:\*\*

| Field | Type | Description |

|-------------|--------------|---------------------------|

| id | Integer | Primary Key |

| title | String | Title of the blog post |

| content | Text | Body of the blog post |

| author\_id | ForeignKey | Linked to User |

| created\_at | DateTime | Post creation timestamp |

| updated\_at | DateTime | Last update timestamp |

---

## 8. Setup & Installation

1. Clone the repository:

```

git clone https://github.com/A22051694/Secure-Blog.git

cd Secure-Blog

```

2. Install dependencies:

```

pip install -r requirements.txt

```

\*If `requirements.txt` is not present, install Django manually:\*

```

pip install django

```

3. Run migrations:

```

python manage.py migrate

```

4. Create a superuser (admin):

```

python manage.py createsuperuser

```

5. Run the development server:

```

python manage.py runserver

```

---

## 9. Usage Guide

- Open `http://127.0.0.1:8000/` in your browser.

- Register a new user or login as admin.

- Depending on your role, access the relevant features from the dashboard.

- Admins can access the admin panel at `/admin/`.

---

## 10. Future Enhancements

- Two-factor authentication (2FA)

- Email notifications

- Rich text editor for blog posts

- Improved comment moderation

- RESTful API for external integrations

- Dockerization for deployment

---

## 11. Credits

Developed by [A22051694](https://github.com/A22051694).

---

\*For any issues or contributions, please open a GitHub issue or pull request.\*