# Blogging WebApp

## Abstract of project:

A Blog application with Django that allows users to create, edit, and delete posts. The homepage will list all blog posts, and there will be a dedicated detail page for each individual post(Profile).

It will have an authentication system with functionalites like Login/ Logout, Register (for new users), resetting the password in case the user of the account forgets it. Using the user's email the user would be able to recover his/her account and reset a new password.

An Admin page with GUI to view backend data.

Django is capable of making more advanced stuff but making a blog is an excellent first step to get a good grasp over the framework. The purpose of this project is to get a general idea about the working of Django.

## Technology used:

### Django:

The Django web framework is a free, open source framework that can speed up development of a web application being built in the Python programming language. Django facilitates “rapid development and clean, pragmatic design.” The Django web framework, deployed on a web server, can help developers quickly produce a web frontend that’s feature-rich, secure and scalable.

Starting with the Django web framework is more efficient way to build a web app than starting from scratch, which requires building the backend, APIs, javascript and sitemaps. With the Django web framework, web developers can focus on creating a unique application and benefit from greater flexibility than using a web development tool.

**SQLite:**

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures.

**Python:**

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

It is rich when it comes to it's documentations, community support and it's wide range of highly effective frameworks, libraries and modules.

Python's features include −

Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read − Python code is more clearly defined and visible to the eyes.

Easy-to-maintain − Python's source code is fairly easy-to-maintain.

A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases – Python provides interfaces to all major commercial databases.

GUI Programming – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable – Python provides a better structure and support for large programs than shell scripting.

## Database Model:

### Django's ORM:

Makes database management more Python-like

Starting a Django project allows you to build your application's entire data model in Python without needing to use SQL. Using an object-relational mapper (ORM), Django converts traditional database structure into Python classes to make it easier to work within a fully Python environment. Django-MySQL supports the JSON data type and related functions.

In Django, your database tables become Python classes. Web applications access and manage data through Django models. The fields of the database are simply converted into class attributes. If you're familiar with class attribute definition in Python, you can easily design and manage a Django database.

Django Web Framework offers a shortcut to full integration with your application's database. It provides CRUD (create, read, update, delete) functionality, HttpResponse and cross-site scripting, supplies user management capabilities, offers software administration features and more. You import the packages, connect to your database and then get back to work developing the parts of your application that make your product unique.

### Creates dynamic pages with templates

Because Django is designed to be used for web app development, it needs a way to easily create dynamic HTML that displays your user's unique data. The

Django application produces that dynamic HTML with a built-in templating engine called the Django template language (DTL).

An HTML template allows Django developers to combine static elements (including design elements such as colors, logos, or text) with data (such as user names or locations) to create a new web page on the fly. With model-view-controller (MVC), if you want your application to greet a user by name when they log in, you can build a template that displays the static text ("Welcome to the site, X") then use a dynamic placeholder to automatically display the user's first name from your database. When the page renders, it will combine the dynamic elements with the static ones to create a seamless user experience.

**Enhanced security**

When answering, "What is Django", we must talk about what special features Django offers for security. Web apps are frequent targets of hackers, especially applications that store user login information or financial data. Django offers features to help protect your application and your users.

One of the biggest risks for sites that accept user-entered data is that a malicious user will inject code with their data that can have a disastrous effect on your system. To protect against attacks like these, Django templates automatically escape common HTML characters in any user-entered field. For example, it will automatically convert '<' to '&lt;' to make it difficult to inject malicious code into your program. Django protects from SQL injection in a similar way, reinterpreting unauthorized commands so that users can't sneak their own code into your database.

Web developers can also count on Django APIs to automatically use cross-site request forgery (CSRF) protection to insert user-specific secret tokens into POST requests. As a result, web developers can prevent malicious users from duplicating other POST requests to masquerade as authorized users.

The protection of Django goes beyond its explicit security features: security efforts are enhanced by the extensive experience and expertise of the Django user base. If you build your entire web app from scratch, you run the risk of accidentally introducing a security vulnerability into your module. Django packages are widely used, open source and well reviewed by web developers, so you can be more confident that they'll protect your data.

**SQLite:**

In the development phase of this project SQLite was used. But it can not be used in the deployment of this web app.

**PostgreSQL:**

It can be used during the deployment of this web app.