## Project Tree Structure :

Project-1/

├── manage.py

├── Project-1/

│ ├── \_\_init\_\_.py

│ ├── asgi.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── registration/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── forms.py

│ ├── migrations/

│ │ └── \_\_init\_\_.py

│ ├── models.py

│ ├── tests.py

│ ├── urls.py

│ └── views.py

├── login\_logout/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── forms.py

│ ├── migrations/

│ │ └── \_\_init\_\_.py

│ ├── models.py

│ ├── tests.py

│ ├── urls.py

│ └── views.py

├── authentication/

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── apps.py

│ ├── forms.py

│ ├── migrations/

│ │ └── \_\_init\_\_.py

│ ├── models.py

│ ├── tests.py

│ ├── urls.py

│ └── views.py

├── templates/

│ ├── base.html

│ ├── registration/

│ │ └── registration.html

│ ├── login\_logout/

│ │ ├── login.html

│ │ └── logout.html

│ └── authentication/

│ └── dashboard.html

├── static/

│ ├── css/

│ │ └── styles.css

└── db.sqlite3

**Ubuntu and Python Libraries for Django Project**

This document lists the necessary Ubuntu packages and Python libraries required to set up a Django project using SQLite as the database.

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Package/Library | Purpose | Command to Install |
| Ubuntu Packages | python3 | Core Python interpreter | sudo apt install python3 |
| Ubuntu Packages | python3-pip | Python package manager (pip) | sudo apt install python3-pip |
| Ubuntu Packages | python3-venv | To create virtual environments | sudo apt install python3-venv |
| Ubuntu Packages | sqlite3 (optional) | SQLite command-line client | sudo apt install sqlite3 |
| Ubuntu Packages | libsqlite3-dev (optional) | Development headers for SQLite (if required) | sudo apt install libsqlite3-dev |
| Python Libraries | django | Django framework | pip install django |
| Python Libraries | djangorestframework (optional) | For REST API support | pip install djangorestframework |
| Python Libraries | django-crispy-forms | To improve form rendering | pip install django-crispy-forms |

**Tech Stack for Project-1**

|  |  |
| --- | --- |
| **Component** | **Technology/Tool** |
| Backend Framework | Django (Python) |
| Frontend Template Engine | Django Template Language (DTL) |
| Database | SQLite (default for development) |
| User Authentication | Django built-in auth system |
| CSS Styling | Custom styles via static/css/styles.css |
| HTTP Server | Django Development Server |
| Form Handling | Django built-in forms (AuthenticationForm, CustomUserCreationForm) |
| Security Features | Django CSRF protection and @login\_required decorator |
| Development Environment | Windows Subsystem for Linux (WSL) |
| IDE/Editor | Visual Studio Code (VSCode) |

**Commands:**

source venv/bin/activate

(venv) python manage.py check

(venv) python manage.py makemigrations

(venv) python manage.py migrate

(venv) python manage.py collectstatic

(venv) python manage.py createsuperuser

(venv) python manage.py shell

(venv) python manage.py runserver

**#browser**

http://127.0.0.1:8000/admin

**# DB commands (optional)**

(venv) python manage.py dbshell

sqlite> .tables

sqlite> SELECT \* FROM <table\_name>;

sqlite> SELECT \* FROM <table\_name>;

sqlite> .schema <table\_name>;

sqlite> pragma table\_info(chat\_app\_message);

**Workflow:**

Brief Workflow Explanation for Django Project

1. Overview

Your project implements user authentication, registration, and dashboard management with modular apps for:  
- Registration: Handles user sign-up.  
- Login/Logout: Manages user authentication sessions.  
- Authentication: Provides protected content, such as the dashboard, accessible only to logged-in users.

2. Workflow

The workflow covers the following routes and functionality:

Home (`/`)

- URL: Defined in `Project-1/urls.py`. Redirects to the Dashboard (`authentication:dashboard`) using the `dashboard\_view`.

User Registration (`/registration/register/`)

- View: `registration.views.register\_view`  
 - Handles GET requests to display the registration form (`CustomUserCreationForm`).  
 - Handles POST requests to validate and save the user and redirects to the login page.  
- Template: `templates/registration/registration.html` displays the form and validation errors.

User Login (`/auth/login/`)

- View: `login\_logout.views.login\_view`  
 - Handles GET requests to display the login form (`AuthenticationForm`).  
 - Handles POST requests to authenticate the user and redirects to the dashboard on success.  
- Template: `templates/login\_logout/login.html` displays the login form and handles validation errors.

User Logout (`/auth/logout/`)

- View: `login\_logout.views.logout\_view`  
 - Logs the user out and redirects to the login page.

Dashboard (`/authentication/dashboard/`)

- View: `authentication.views.dashboard\_view`  
 - Protected by `@login\_required`. Only accessible to logged-in users. Redirects unauthenticated users to the login page.  
- Template: `templates/authentication/dashboard.html` displays the user's username and a logout link.

3. Modular Components

- `registration`: Handles user registration using custom forms and views.  
- `login\_logout`: Manages user login and logout functionality.  
- `authentication`: Manages user-protected content like the dashboard.

4. Reusable Structure

- `templates/base.html`: Provides the base layout for all templates, ensuring consistent structure and styles.

5. Workflow Summary

1. A new user registers via `/registration/register/`.  
2. The user logs in via `/auth/login/`.  
3. Upon successful login, the user is redirected to `/authentication/dashboard/`.  
4. The user can log out via `/auth/logout/`.

6. Security Features

- CSRF Protection: All forms include `{% csrf\_token %}` to prevent CSRF attacks.  
- Login Required: Dashboard access is restricted to logged-in users using the `@login\_required` decorator.  
- Django's Built-In Authentication: Uses Django's secure, built-in methods for login, logout, and form handling.