File Upload and Management System (Django Project)

Introduction:

Project Overview:

This project is a simple web-based **File Upload and Management System** built using the **Django framework**. It allows users to **create an account**, **log in**, and **upload their files** to the website.

Every user can decide whether to make their file **public** or **private**.

- **Public files** are visible to everyone and can be downloaded by any visitor.
- **Private files** are only visible to the person who uploaded them.

The idea behind this project was to create a small but functional platform where users can safely manage their files without needing any external storage or software.

Objectives:

The main goals of this project were:

- To build a working Django web app for file uploading.
- To use Django's authentication system for user login and signup.
- To give users control over file visibility (public or private).
- To allow downloads for public files.
- To keep the process simple and secure.

Scope:

This system can be useful for individuals or small teams who just need a basic file upload and sharing feature. It can later be expanded into a bigger platform by adding options like file sharing through links, folders, or cloud storage integration.

System Design and Functionality:

Design Overview:

The project is based on Django's **Model–View–Template (MVT)** structure:

- **Model:** Stores file details like file name, uploader, upload date, and visibility status

View: Handles user requests such as uploading, deleting, or changing file visibility.

Template: Displays everything to the user in a clean, easy-to-use interface.

How It Works:

A new user signs up and logs into the system.

Once logged in, they can upload a new file from their dashboard.

While uploading, they choose whether the file should be public or private.

Uploaded files are stored in Django's **media folder**, and file info goes into the **database**.

Public files can be seen and downloaded by anyone.

Private files are shown only to the user who uploaded them.

The user can delete any of their own files at any time.

Tools and Technologies:

Frontend: HTML, CSS

Backend: Python (Django Framework)

Database: SQLite

Editor: Visual Studio Code

Version Control: Git

Other: Django's built-in authentication and media file handling

Key Features:

User registration and login

File upload and deletion

Public and private file options

Download option for public files

Clean and simple user interface

Results and Conclusion:

Project Outcome:

The system works smoothly and fulfills its purpose. Users can sign up, log in, upload, delete, and manage their files without any issues. Public files can be easily downloaded, while private ones remain secure.

The website's layout is simple, which makes it easy for users to understand and use even if they aren't technical.

Challenges Faced:

During development, a few small issues came up, such as:

Configuring Django's media settings properly.

Handling file access permissions.

Displaying uploaded files neatly on the dashboard.

After some debugging and learning, these issues were successfully solved.

Future Improvements:

If the project is developed further, the following features could be added:

File sharing through generated links.

User profile management with profile pictures.

Better design using Bootstrap or Tailwind CSS.

Integration with cloud storage like AWS or Firebase.

Conclusion:

Overall, this project helped me understand how Django handles **user authentication, file uploads, and media storage**. It's a complete small-scale system that can be improved further with more advanced features.

The **File Upload and Management System** turned out to be a good learning experience, and it gave me a solid understanding of how to build practical web apps using Django.