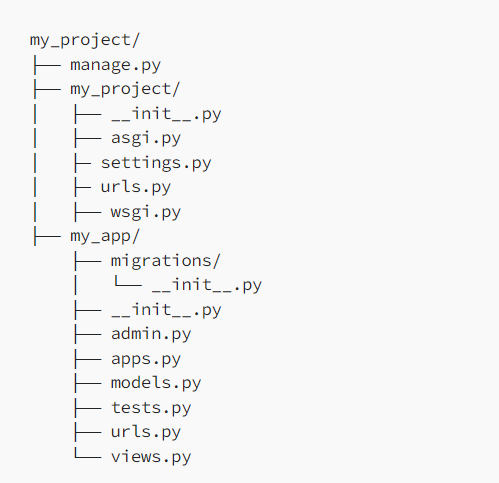
# Django Basic

* In web development it is used as a backend Python framework whose principle is based on modular approach (DRY – Don’t repeat yourself) and reusability.
* Involves orchestrating all components to work simultaneously. Other than its feature set, it also efficiently structures complex projects.
* It eases navigation, aids in comprehending the architecture, and fosters seamless teamwork, following the Model-View-Template (MVT) architecture. It separates models (databases), views (controlling functions) and templates (UI pages).
* A virtual env (venv) set up is recommended to ensure an isolated env fir the project.
* Difference b/w a project and its apps - A project represents the entire web application (collection of settings, configs, and other apps forming complete entity), whereas an app is a smaller, self-contained module within the project serving any standalone functionality.

# Initialization

Initiating a Django project - *django-admin startproject my\_project*

Starting an app - *python manage.py startapp my\_app*



# Project directory

* The main project directory in Django is from where all functionalities radiate, in the entire webapp.
* manage.py: Serves as the gateway to the development server, create apps, migrations, etc, guiding your project's activities.
* my\_project/settings.py: Store settings that configure your Django proj. i.e. database configs, middle wares, app’s functions. Blueprint shaping the behaviour of your proj.
* my\_project/urls.py: Contains URL dispatcher, which maps URLs to views to access them. Navigator through which users tour app’s pages.
* my\_project/wsgi.py (Web Server Gateway Interface): Serves as bridge connecting your application to the web server, enabling it to handle incoming requests, when deployed.
* my\_project/asgi.py (Server Gateway Interface): Entry point for web servers, asynchronously handling HTTP requests.
* my\_project/\_\_init\_\_.py: Converts a directory into a Python package, required for importing modules across your proj.

# Application directory

* models.py: Contains info about data structures used in the proj made using Django's Object-Relational Mapping. Each class and object represent a table and entries of database.
* views.py: Defines how your app interacts with user’s requests. Handles data processing, rendering templates and tangible responses.
* tests.py: Used to define unit tests to ensure your app’s components function as expected, supporting test-driven development.
* admin.py: Configures how your app’s models are shown in Django's admin interface. Allows administrators to manage data.
* migrations: Contains blueprint of all changes in your app’s models.
* utils.py: For utility functions or classes that are used in the project.
* Other Files: (forms.py) Houses form classes for data input, (urls.py) maps URLs to views and (apps.py) manages app-specific configs.

# Apps within Project (Benefits)

* Code Reusability
* Standardization
* Community Contributions
* Maintainability

# Best practice for project structure

A screen shot of a computer program

Description automatically generated

* project\_name: Root directory of your project.
* project\_name/project\_name: Holds core project settings and configuration.
* app1, app2: Individual apps you create within the project.
* Static: Houses static files like CSS, JavaScript, and images.
* Media: Stores user-uploaded files.
* Templates: Contains HTML templates.

# Naming Conventions

* Apps: my\_app
* Modules: my\_modules.py
* Classes: MyClass
* Functions and Vars: my\_functions

# Conclusions

Through this, we've grasped the pivotal role of “manage.py” managing tasks and the versatility of “settings.py” in configuring project behaviour. Moreover, the concept of reusable applications has been explored, with benefits of their modular design and autonomy. By embracing best proj. structure such as consistent naming conventions, a clear directory hierarchy, and modular code.