**Django Project Flow**

Project > App > urls > views > template

**Create Django project**

*django-admin startproject django\_demo* ***.***

django\_demo = project name

. = the period (.) at the end of the command so that it is installed in our current directory.

The .venv directory was created with our virtual environment but Django has added a django\_demo directory and a manage.py file. Within django\_demo are five new files:

* \_\_init\_\_.py indicates that the files in the folder are part of a Python package. Without this file, we cannot import files from another directory which we will be doing a lot of in Django!
* asgi.py allows for an optional [Asynchronous Server Gateway Interface](https://asgi.readthedocs.io/en/latest/specs/main.html) to be run
* settings.py controls our Django project’s overall settings
* urls.py tells Django which pages to build in response to a browser or URL request
* wsgi.py stands for [Web Server Gateway Interface](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface) which helps Django serve our eventual web pages.

The manage.py file is not part of django\_project but is used to execute various Django commands such as running the local web server or creating a new app.

**The abstract flow looks something like this:**

HTTP Request -> URL -> Django combines database, logic, styling -> HTTP Response

Django only loosely follows the traditional MVC approach with its own version often called **Model-View-Template (MVT)**.

MVTU would be a more accurate description.

* Model: Manages data and core business logic
* View: Describes which data is sent to the user but not its presentation
* Template: Presents the data as HTML with optional CSS, JavaScript, and Static Assets
* URL Configuration: Regular-expression components configured to a View

**The complete Django flow looks like this:**

HTTP Request -> URL -> View -> Model -> Template -> HTTP Response

Django uses the concept of projects and apps to keep code clean and readable. A single top-level Django project can contain multiple apps. Each app controls an isolated piece of functionality. For example, an e-commerce site might have one app for user authentication, another app for payments, and a third app to power item listing details. That’s three distinct apps that all live within one top-level project.

**Create an App**

*python manage.py startapp pages*

pages = app name

* admin.py is a configuration file for the built-in Django Admin app
* apps.py is a configuration file for the app itself
* migrations/ keeps track of any changes to our models.py file so it stays in sync with our database
* models.py is where we define our database models which Django automatically translates into database tables
* tests.py is for app-specific tests
* views.py is where we handle the request/response logic for our web app

Add it to the django\_project/settings.py file. In your text editor open the file up and scroll down to INSTALLED\_APPS

Add => pages.apps.PagesConfig

then in view of pages

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def HelloWorld(request):

return HttpResponse("Hello Saranj")

There are two types of views in Django:

* function-based views (FBVs)
* class-based views (CBVs).

Django originally started with only FBVs but over time added CBVs which allow for much greater code reusability, keeps things DRY (Don’t-Repeat-Yourself), and can be extended via mixins.

Add-on

* generic class based views (GCBVs)

Because web development quickly becomes repetitive Django also comes with a number of built-in generic class-based views (GCBVs) to handle common use cases such as creating a new object, forms, list views, pagination.

Create urls.py file

from django.urls import path

from .views import HelloWorld

urlpatterns = [

path('', HelloWorld, name="home"),

]

then update hello\_django urls file with

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('pages.urls')),

]

**Adding a templates**

create a templates folder under pages folder

create pages folder under it create HTML files

pages > templates > pages > demo.html

in settings.py inside TEMPLATES

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

In view.py

def demo(request):

return render(request, 'pages/demo.html')

In urls.py

urlpatterns = [

path('', hello, name='hello'),

path('demo', demo, name='demo')

]

To create a base.html using jinja template sytax

put it inside templates > base.html

base.html

<similar markup>

{% block content %}

{% endblock %}

<similar markup>

to import the base syntax

{% extends ‘base.html’ %}

{% block content %}

<different markup>

{% endblock%}

**Adding static file**

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

STATIC\_URL = 'static/'

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'real\_estate/static')

]

create a static folder under project

reference static file

{% load static %}

<link rel="stylesheet" href="{% static 'css/style.css' %}">

To create a block of code

{% include 'partials/\_topbar.html' %}

Linking

<a href="{% url 'index' %}">

to perform migration

*python manage.py migrate*

to create database with help of models

*python manage.py makemigrations*

create a super user to login in into admin area of django

*python manage.py createsuperuser*

to add listing model to django admin area

from django.contrib import admin

from .models import Listing

admin.site.register(Listing)