**Web Development Documentation with Django**

* Django is a powerful and popular web framework for Python that follows the Model-View-Template (MVT) pattern.

**Steps to create a project**

**GENERAL REQUIREMENTS**

* Ensure you have Python installed on your system.
* Go to your web client/browser and in the URL and download it from the official Python website (<https://www.python.org/>).**python of version==3.11.4** which is the latest version.

1. **Configuration of the virtual environment**

* It's a good practice to create a virtual environment for your project.
* Create a folder where our entire Django project will be hosted.by a command **mkdir** then name of the folder.
* Install the virtual environment by the command” **pip install virtualenv”. The virtual environment enables us to separate projects from each other. This can sort a lot of confusion in your project management.**
* Create a sub folder within the main folder by a command **“mkdir"** then the name of the folder.
* You **cd** into the subfolder and then you specify the name of the virtual environment by a command **“virtualenv venv**
* You activate the **virtualenv** by a command **“venv\scripts\activate**
* if you want to deactivate you use a command **“venv\scripts\deactivate.bat”**

**2. Configuration of Django**

Install django by a command pip install django or **pip install django==version**

* create the project by the command **‘django-admin startproject** then name of the project eg Vanessa
* structure of your project
* **projectname/(vanessa),** this is the second folder of the same name of your folder created by django and it contains these files;

**1. manage.py,** this file is not editable and it is used in the command line to enable you interact with your project.

**2. settings.py,** allows you to configure settings of your project.

* **urls.py,** allows you define the url patterns of your project.
* asgi.py, enables you run your project using **ASGI** (**Asynchronous Server Gateway Interface**)server**. ASGI** provides an interface between asynchronous Python web servers and frameworks**.**
* **wsgi.py,** this also enables you run your project using **WSGI** (**Web Server Gateway Interface**)server**.** It is used to forward requests from a web server to a backend Python web application or framework (django). From there, responses are then passed back to the webserver to reply to the views(the requestor)**.**
* Then you **cd** into the project created eg Vanessa.
* Then you prompt **“python manage.py migrate”** this applies the default tables from django to your database.
* Then you create the superuser(admin) by **‘’python manage.py createsuperuser”**
* Then you create the app **“python manage.py startapp** “name of the app”eg beapp
* After appending the application go to the command line prompt **“python manage.py runserver”.** This will start the development server, and you should see output indicating that the server is running. By default, it listens on port 8000. You can access your project by navigating to **http://localhost:8000/** in your web client.
* Then you install crispy forms which help us create forms in our project by **“pip install django-crispy-forms.**
* We also install bootstrap4 for displaying crispy forms by **“pip install crispy-bootstrap4”**
* We then install bootstrap5 by **“pip install django-bootstrap-v5** ,helps us to display our bootstrap content.
* We then install filters for activating our search buttons by **“pip install django-filter”**
* We then install a library that will enable us upload images in our project by a command **“pip install Pillow”**
* We can also create our objects from the command line without using the admin interface to create them. we just have to import from the models created. Like from **>>>products. models import Product** .from that particular model where you want to create objects eg consider a created model ie class Product(models.Model)

title=models.CharField(max\_length=120)

* like**,>>>Product.objects.create(title=’new’)**,takes everything as a string
* We can also query for the objects created under a certain model eg >>>**Product.objects.all()**,it shows out all the objects you have so far created

**3. Settings configuration**

* In your project directory, locate the **settings.py** file inside the inner project folder (with the same name as your project). This file contains various configurations for your project, including installed apps, database settings, static files, and more
* Go to settings in the list of installed apps and you append the app you have created.
* Then we go to settings and append the downloads e.g. **crispy\_forms, crispy\_bootstrap4, bootstrap5, django\_filters, PIL.S**
* Go to settings and create a variable **STATIC\_ROOT** to define a path for the static files in the project.
* We create **STATICFILES\_URL** variable to define the path for static files in the application.

**4. Configuration of our templates and static folders**

* Create a **template folder** in the project folder.(they should be within the duplicated folder by django.) this stores the html files.
* Create a **static folder** in the project folder.(they should be within the duplicated folder by django.)
* Create a **static folder** in the application.(they should be within the duplicated folder by django.),where we put all the **static files like css files,images,videos** etc.

**5. Configuration of master urls (urls.py in the project)**

* We go to the list of urls patterns and create a path for the application to be accessed. NB: don't forget the **function include()** as your creating your path.

**6. Configuration of our application**

* Create a file called urls.py and then define our paths for the urls to be accessed by the views. This is done within the url patterns list you have created. The views are imported so as to access these patterns created.
* After downloading the filters and appending them in the settings in the installed apps, create a file called filters.py that activates the fields to be filtered.
* Then create the forms.py file which helps us to capture data that will be posted to the database.

**7. Configuration of our views/functions/methods**

* The views are functions that handle the http request and http response from the web client (web browser). eg **200(ok), 500(internal server), 404(error), 303(page has been temporarily moved), 503(service is unavailable).**
* The views are defined as functions/methods that handle the request and gives us the response.

**8. Configuration of models.py (models map to tables in django database)**

**Django provides an abstraction layer (model layer) for structuring and manipulating the data of your web application. Here we borrow the knowledge of OOP(Object Oriented Programing) which consists of Abstraction for creation of classes, Inheritance that creates the ability for the instances of classes(object) to inherit the properties of the class(blue print).**

* We import some models from django
* Create class eg class Country (**models.Model**, this enables us to inherit the features of the already existing model in django.) and give it attributes e.g. name. These attributes/properties/variables are validated basing on the nature of their datatype. These attributes appear as the columns on the interface and the fields that will capture data.
* We then define a function **“str”** to enable other classes we create below to access attributes in that class. And these classes access these attributes using a **Foreign key** (creates a relationship btn classes or between models created (tables)).
* We then make migrations in the command line by “**python manage.py makemigrations** to apply the changes to the database and then migrate by “**python manage.py migrate** “respectively”.
* Endeavor to make migrations whenever you make changes to your models to apply them.

**9. Configuration of admin**

* from .models import \*(all models created) though you can specify the particular model
* the attributes in the created model specify the fields for the captured data.
* Go to admin.py and register the model created that will be reflected on the interface.

10. **Accessing your Application**

* Open your web browser and visit <http://127.0.0.1:8000/> to view your Django project. If you want to access the admin interface, go to <http://127.0.0.1:8000/admin/> and log in with the superuser credentials you created earlier.

**NB: A web browser/client and server cannot communicate directly to the database. A web browser/client is a piece of software that sends a request and receives a response from a web server. Web server is a piece of software that receives a request and sends a response to the web client.**

**And the server cannot communicate directly to the database because it’s dumb.**

**So the backend language such as django (python) creates a bridge btn the database and the server to communicate.**

**A browser cannot post data directly to the database so it uses a form to post data.**

**Django project can have different applications. In such a scenario each application requires a specific path for defined in the master urls (urls of the project). However incases of a single application, one path is enough to handle the route for requests from the views.py file.**

## **Uploading Images in Django**

Django has two model fields that allow user uploads **FileField** and **ImageField** basically **ImageField** is a specialized version of  **FileField** that uses Pillow to confirm that a file is an image.

start by creating models.

from django.db import models

class Image(models.Model):

title = models.CharField(max\_length=200)

image = models.ImageField(upload\_to='images')

def \_\_str\_\_(self):

return self.title

The image column is an **ImageField** field that works with the Django's file storage API, which provides a way to store and retrieve files, as well as read and write them.

The **upload\_to** parameters specify the location where images will be stored which for this model is **MEDIA\_ROOT/images/**

Setting dynamic paths for the pictures is also possible.

**image = models.ImageField(upload\_to='users/%Y/%m/%d/', blank=True)**

This will store the images in date archives like **MEDIA\_ROOT/users/2020/04/12**

Now, Install Pillow by running the following command in your shell.

**pip install Pillow**

For Django to serve [media files](https://djangocentral.com/managing-media-files-in-django/) uploaded by users with the development server, add the following settings to the **settings.py** file of your project.

#### **settings.py**

**#Base url to serve media files**

**MEDIA\_URL = '/media/'**

**#Path where media is stored**

**MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media/')**

**MEDIA\_URL** is the URL that will serve the media files and **MEDIA\_ROOT** is the path to the root directory where the files are getting stored.

Now add the following configuration in the project's **urls.py** file.

#### **urls.py**

**from django.conf import settings**

**from django.conf.urls.static import staticurl**

**patterns = [**

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

**...]**

**if settings.DEBUG:**

**urlpatterns += static(settings.MEDIA\_URL,**

**document\_root=settings.MEDIA\_ROOT)**

With that Django's development server is capable of serving media files. Next, we need to create a model form for the Image model.

#### **forms.py**

**from django import forms**

**from .models import Image**

**class ImageForm(forms.ModelForm):**

**"""Form for the image model"""**

**class Meta:**

**model = Image**

**fields = ('title', 'image'**)

This will generate a form with fields title and image, which will be rendered in the templates. So let's create a barebone template for file uploading.

#### **index.html**

You must remember to include the **enctype** property in the form tag for the uploaded file to be attached to the request properly.

With that let's write views for handling the form.

#### **views.py**

**from django.shortcuts import render**

**from .forms import ImageForm**

**def image\_upload\_view(request):**

**"""Process images uploaded by users"""**

**if request.method == 'POST':**

**form = ImageForm(request.POST, request.FILES)**

**if form.is\_valid():**

**form.save()**

**# Get the current instance object to display in the template**

**img\_obj = form.instance**

**return render(request, 'index.html', {'form': form, 'img\_obj': img\_obj})**

**else:**

**form = ImageForm() return render(request, 'index.html', {'form': form})**

This is a very simple view since Django is doing all the work under the hood we are just validating the form and saving it on successful file upload. Now that we are done with the view let's map it to a URL.

#### **urls.py**

**urlpatterns = [**

**...... path('upload/', views.image\_upload\_view)**

**......]**

**Save all the files and run the server and navigate to the URL you should see the form in action.**