Django Introduction

Django is an open source **Web Framework** written in **Python**. The goal of the framework is to leverage **Python** language fast development to a web framework, and to aim at a framework which build strong web applications with minimal development time. One of the biggest advantages of using **Django** is the very large community of contributors adding tons of quality packages that can easily be added to your **Django Application** . Those packages are really easy to install and are one of the reason developing applications with **Django** is so fast.
For example a popular package like **Django Rest Framework** easily turns your database tables to a fully configured REST Api.
Django is a web framework meaning it's very opinionated to how you should develop your web applications, so using it will probably mean that you need to develop apps the **Django** way, but don't worry cause they direct you to a really good way to build web applications which are scalable and easy to maintain.

Django Architecture

Django has their variation of a **Model View Controller architecture** .
The **View** part in **Django** is actually referred to as template. **Django** has a strong templating engine which allows us to separate the creation of the **HTML** page and embed into that page variables that will be send from the **Controller.** The aim of the templating engine is to be simple enough for non programmers to create the template. The **Controller** in **Django** is referred to as the **View** which allows us to pass variables that will be injected to the template.
The **Model** is actually referred in **Django** as **Model** and in **Django** there is a strong **Model** abstraction to our Database tables. **Django** works with the popular db engines among them are: MySql, Postgres, Oracle, MongoDb . And you can create a **Model** to represent a table in the database and this abstraction allows us to easily switch our database.

So the way it works is that **Django** uses a **URL** router to determine what view to send a request. The view get's the request and use the **Model** and **Template** to send a response.

Let's try and demonstrate this with our first **Django Application**. The application we will build is an extended hello world. Our page will have a form with an input of text requesting to write a greeting, when submitting the form the greeting is saved in an **sqlite** database. Below the form all our greetings that we saved are presented in a list.

virtualenv

In each project we want a clean **Python** environment that will be identical to the clean **Python** environment our servers will have. To achieve this we will create for each project a separated **Python** environment.
To install **virtualenv** you can install it globally on your system by typing:
> pip install virtualenv

this will install **virtualenv** globally and expose the **virtualenv** command in your terminal.
Let's open a new directory for our sample project, we will call this directory **django-architecture** cd into that directory and type in the terminal:
> virtualenv venv

This command will open a new clean virtual environment of **Python** located in the directory **venv** .
We need to tell our computer to use the **venv** environment and not use the default global environment, we want to install the packages in the virtual environment, so to activate the environment in your terminal type:

> source venv/bin/activate

The following steps for creating a virtual environment is highly recommended to do for each **Python** project you create and not only this one, and when your environment is activated you should see at the left of the terminal the name of the environment currently activated.

Install Django

To start a new **Django** project make sure you virtual environment is activated and type in the terminal:

> pip install django

Django is now installed in your virtual environment

django-admin

django-admin is a command line utility for administrative tasks that is available for us in our environment variable when we installed **Django**.
To check out what actions we can do with **django-admin** you can type in the terminal:
> django-admin help

This will show you the commands available before we created a new project (before the **settings.py** is available - we will talk about **settings.py** later)

One of the tasks that **django-admin** gives us is the ability to easily start a new project with the **startproject** command. In the terminal cd into the directory we created **django-architecture** and type:

> django-admin startproject django_architecture

The third argument is the name of the project.

In **Django** architecture web server application there is two concepts you should familiarize yourself with:

- **Project** - refers to the entire **web server application**
- **App -** a **Project** is made of one or more apps, the apps are like **modules** that have **urls, views** and **templates** that close certain section or logical part of your application

The separation between those two concepts gives **Django** application a lot of power and the ability to combine different apps even easily combine apps from the community and by doing so it's extremely easy to combine external packages to your application.

Project

Let's try and understand what files are created for us when we created our **Project**.
django-admin created a folder called **django_architecture** to hold all our project files and inside it there is the main **Project** package called **django_architecture** with the following files:

- **__init__.py -** This make **Python** treat the folder as a **Package** and the **init** file can contain **initialization** code when the package is imported, it can also be left empty.
- **settings.py -** Default settings file for our project, contains settings like: Database configuration, apps installed in our **Django Project**, templates configuration and static files configuration and other useful settings that we will cover in depth on the next lessons.
- **urls.py** - Contains the **urls** in our project, what urls are directing to what apps and views
- **wsgi.py -** WSGI is used as a middle man between the web server (for example **nginx**) and our **Python** application, **Django** initiates a default configuration to run our project with **WSGI**

Outside the package there is another file called **manage.py** this file contains common administartive tasks for the **Project** similar to **django-admin** only it's connected to the project settings file and can perform additional per project tasks.

Let's try and interact with **manage.py** and run the project we created so far with a development server. cd into **django_architecture** where your **manage.py** is located and type:

> python manage.py runserver

Now open your browser at **localhost:8000** and you should see your **Django** application is now working and there is a message **It Worked!**

Creating our App

The project represent the entire web application and we can divide our project into multiple apps according to what our web application contains. We can have a landing page app and a blog app and a store app and any other app you web application requires.
Open your **settings.py** file and you should see a section in that file with the following code:

```
INSTALLED_APPS = [
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
]
```

This section lists the current applications that are installed in your project. We see here that by default when we start a new **Django** project there are default apps already included in our project. We will go over some of those default apps on the next lessons.

Just a reminder of what we are about to do we want to create an app that will display a form with a text input to write a greeting message, all the greeting message will be saved in an **sqlite** database and display as a list below the form.

Let's create a new app, cd into the root directory of your project (the directory where you have your **manage.py** file) and in your terminal type:

> python manage.py startapp hello

the **startapp** command will start a new app and the last argument is how I want to call that app. We can see in our files that a new python package called **hello** with the following files are created for us:

- **admin.py** - connecting my app models to **django admin**, we will cover **django admin app** in future lesson

- **apps.py -** Configuration information about your app, django keeps refrence on the installed apps configuration, and mostly you wont have to touch this file.
- **models.py -** contains the business logic of you app and the models that help you communicate with the database tables this app needs
- **tests.py -** it's highly recommended to write tests for you app and it should be in this file
- **views.py -** this will hold the **controller** layer of our **MVC** architecture and the **URL's** set for the app will map to classes and functions that run on this file.

So we added a new app but we still need to modify the project **settings.py** to include this app in our installed apps. in the **settings.py** in the **INSTALLED_APPS** array add at the end the name of our new app **'hello'**

Our app in now connected to our project. You can run your project again by typing:
python manage.py runserver

you should see that nothing is changed… YET!
We will now add a url in out project that will map to the urls in our new app.

App URLS

We want to add urls in our application and connect them to the project urls.
But before we do that let's add a class that will handle the logic of the view of the app. Modify the **views.py** file with the following code:

```
from django.http import HttpResponse

class Hello(object):
  def hello_world(self, request):
     return HttpResponse(content='hello world')

site = Hello()
```

In this file we created a class with a function that send a response of **hello world** and we are placing an instance of the class in the variable called **site.**

In the app package **hello** create a file called **urls.py** with the following code:

```
from django.conf.urls import url
from hello.views import site

urlpatterns = [
  url(r'^$', site.hello_world, name='home')
```

```
]
```

This file will contain the URL's in the App hello and for now we simply created a single URL that directs to the **hello_world** method in the **Hello** class we created in the **views.py**. Notice that it's very common to have URL's for the Project and for the apps. The URL's for our web app are located at the project urls.py file an in it we can connect the apps urls with a certain prefix.
Modify the **urls.py** of the project that is located at the package **django_architecture**

```
from django.conf.urls import url, include
from django.contrib import admin
from hello import urls

urlpatterns = [
  url(r'^admin/', admin.site.urls),
  url(r'^$', include(urls))
]
```

notice that we placed the urls of the hello app without a prefix and on the root URL but if we placed it in the regex **r'^hello/$'** then to access the view we created in the hello package we would have to type the url **localhost:8000/hello/**
Try and run your app and go to the url: **localhost:8000** and you should see an hello world message.
So we saw here how we can create an app and also add a url and a view.
Let's continue to explore **Django** MVC

Database & Models

Most of the web application we will need to create will require us to use a database, connecting a database to a **Django** app is really easy and the framework supports the popular databases available. Also communicating with the database is really easy and you can achieve this by simply extending **Django Models.** With **Django** models each model represents a table in the database. In the application we are going to create we have a single table containing one string field of all the greetings submitted with the form. For this small app we will simply use an **sqlite** database but of course when you develop your app you will need to use a database more suitable for web applications like **Postgres.**
Open the **settings.py** file and notice there is a section in that file that should look like this:

```
DATABASES = {
  'default': {
     'ENGINE': 'django.db.backends.sqlite3',
     'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
  }
```

```
}
```

In this section we configure the database connection details, you can see that by default our app is connected to an **sqlite** database, we will keep the default settings at this time. Another thing you probably noticed when we created our hello application is that there is a folder in the app package called **migrations**, **migrations** in **Django** are in charge of keeping the database structure identical to our **Django Models**. Currently we have different apps installed in our project and to properly use them we will have to run their migrations so our database will include the proper tables for those apps models.
In your terminal type:
> python manage.py migrate

You should see a list of all the migrations that ran with their status.

Now let's create the proper model for our hello app. In the **hello** package modify the **models.py** file to this:

```
from django.db import models

class Greeting(models.Model):
  message = models.CharField(max_length=300, blank=False, null=False)

  def __unicode__(self):
     return self.message
```

We created a model that should reflect a database table called greeting with a single field of message.
We also create a **__unicode__** method for the class that will help us print a class description of the object that we will later see when we examine the admin interface.
To create the table in the database, in the terminal type:
> python manage.py makemigrations

this will create the file that will create the proper table now to run the file type:

> python manage.py migrate

This will run all the migrations. We just create our model and database table that will let us interact with the database. Let's create the final section of our application and complete the template and the view connected to it.

Template & View

In **Django** the **templates** represent the **view** of the MVC and basically they present the HTML that the user will interact with. The templates should be in a folder called **templates** in our app. So create a new folder in the **hello** package called **templates** and in that folder create a file called: **hello.html**

```
<form method="post">
  {% csrf_token %}
  <div class="form-row">
      <label for="message">Message</label>
      <input type="text" name="message" id="message">
  </div>
  <div class="form-row">
      <button type="submit">
          Submit
      </button>
  </div>
</form>

<ul>
  {% for message in greetings %}
      <li>
          {{ message }}
      </li>
  {% endfor %}
</ul>
```

We can easily use django template engine to write our html and embed variables in that html. In this simple view we created a form with **csrf** security tag to verify that the form is submitted from our site.
We also expect a list of greetings and we are iterating on that list. We will further examine what the template engine has to offer in the next lessons.

Now change the **views.py** file in the **hello** package:

```
from django.http import HttpResponseRedirect
from django.template.response import TemplateResponse
from models import Greeting
from django.urls.base import reverse

class Hello(object):
```

```
  def hello_world(self, request):
      if request.method == 'GET':
          return TemplateResponse(request, 'hello.html', context={
              'greetings': Greeting.objects.all()
          })
      if request.method == 'POST':
          greeting = Greeting.objects.create(message=request.POST.get('message'))
          greeting.save()
          return HttpResponseRedirect(reverse('home'))


site = Hello()
```

We are dealing in the same view with the post and the get actions, and in the post we are creating a new model instance that will be saved in the database, and then we are redirecting back to the regular get which simply renders the template and pass the list of the greeitng passed down to the template.

Summary

To summarize **Django Architecture**:

- we have our entire web application wrapped in a **django project**
- our project is divided to apps which are very modular and can easily be injected including apps from the community.
- Each app has a urls file which map url to views
- the views act as our controllers
- the view can render a template which is our view or deal with an action from the template