U P T

UNIVERSIDAD POLITÉCNICA DE TULANCINGO

ORTIZ OSORIO CRISTIAN URIEL
FRANCISCO MELO MARIO ETZAEL

Ingeniería en Sistemas Computacionales

Restful Python

Octavo Cuatrimestre

Tulancingo de Bravo, Hidalgo

Enero – Abril 2020.

Creación de entorno virtual de python

1.- Para poder ejecutar e instala django se recomienda crearlo en un entorno virtual de python el cuál funciona como un contendor para que las instalaciones de paquete con pip no estén instaladas en el SO pero sí en el entorno virtual.

```
python3 -m venv env
```

2.-Activaremos el entorno virtual creado para poder trabajar dentro de este e instalar paquetes y librerías

```
source env/bin/activate
```

3.- Instalaremos los paquetes o librerías requeridos para poder elaborar un api restful en python.

```
pip install django

pip install djangorestframework

pip install pygments
```

Creación de un proyecto y app de django

1.- Crearemos un nuevo proyecto de django para poder emepzar a trabajar.

```
django-admin startproject tutorial
```

2.- Posteriormente crearemos una app dentro del proyecto generado.

```
cd tutorial

python manage.py startapp snippets
```

Configuración del proyecto para utilizar frameworks y bases de datos

1.- Dentro del archivo settings.py ubicado en la carpeta raíz del proyecto agregaremos o modificaremos las siguientes instancias

1.1.- Agregaremos la dependecia del framework al proyecto y la dependecia de la configuración de la app creada anteriormente

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

1.2.- Para la configuración de este proyecto se utiliza mysql para su conexión debemos agregar las siguientes líneas

```
DATABASES = {
    'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': 'Rest',
            'USER': 'root',
            'PASSWORD': 'Esnafer19',
            'HOST': 'localhost',
            'PORT': '3306',
        }
}
```

2.-Posteriormente agregaremos la librería de mysql dentro del archivo __init__.py ubicado en la carpeta del proyecto (carpeta raíz).

```
import pymysql
pymysql.install_as_MySQLdb()
```

Modelos en python.

1.- Un modelo en Django es un tipo especial de objeto que se guarda en la base de datos Una base de datos es una colección de datos. Es un lugar en el cual almacenarás la información sobre usuarios, tus entradas de blog, etc. Los tipos de datos y acerca de su composición estarán en https://docs.djangoproject.com/en/3.0/topics/db/models/ la documentación oficial de django. Finalmente el modelo quedará de esta manera generando solo una tabla.

```
1  import uuid
2  from cassandra.cqlengine import columns
3  from django_cassandra_engine.models import DjangoCassandraModel
4  from django.db import models
5
6
7  class Empleado (models.Model):
8      nombreCompleto=models.CharField(max_length=35)
9      clave=models.CharField(max_length=8)
10     rol=models.CharField(max_length=35)
11     telefono=models.CharField(max_length=35)
12     edad=models.CharField(max_length=35)
13     genero=models.CharField(max_length=35)
14     salario=models.CharField(max_length=10)
```

2.- Crearemos las migraciones del modelo para generarlo dentro de la base de datos.

Nota: La base de datos ya debe de estar creada.

```
python manage.py makemigrations snippets

python manage.py migrate
```

Serializadores

Los serializadores nos permiten definir al detalle cómo serán las respuestas que devolverán nuestro API y cómo procesaremos el contenido de las peticiones que nos lleguen.

1.-Nuestro serializador solo contendrá una clase debido a que solo es una tabla y contendrá las columnas que se requieren de regreso de la información en formato json.

```
from rest_framework import serializers
from snippets.models import Snippet


class SnippetSerializer(serializers.ModelSerializer):
    class Meta:
        model = Snippet
        fields = ['id', 'title', 'code', 'linenos', 'language', 'style']
```

Para más información se recomienda visitar el sitio oficial https://www.django-rest-framework.org/tutorial/1-serialization/#working-with-serializers

Vistas genéricas basadas en clases.

Las vistas genéricas de Django brillan realmente cuando se trata de presentar vistas del contenido de tu base de datos. Ya que es una tarea tan común, Django viene con un puñado de vistas genéricas incluidas que hacen la generación de vistas de listado y detalle de objetos increíblemente fácil.

1.- Nuestra vista genérica se compone de dos clases la primera es una list en donde lo métodos de http se componen de get y post deacuerdo a la APIView de REST.

```
from snippets.models import Snippet
from snippets.serializers import SnippetSerializer
from rest_framework import generics


class SnippetList(generics.ListCreateAPIView):
    queryset = Snippet.objects.all()
    serializer_class = SnippetSerializer
```

2.-Y una vista detallada en donde se modifica el registro o se elimina de acuerdo a un id PK mostrándolo en la APIView de REST.

```
class SnippetDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Snippet.objects.all()
    serializer_class = SnippetSerializer
```

URLS

En todo el proyecto contamos con 1 archivos urls.py que se encuentra en el directorio raíz o urls proyecto. Tenemos que crear un archivo urls.py dentro de la app el cuál re direccionará a las views.py genéricas creadas

1.- Urls.py del proyecto o directorio raíz donde 'snippets' es nuestra aplicación:

```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('snippets.urls'))
]
```

2.- Urls.py de la app snipets re direccionando a las vistas genéricas en el archivo views.py

Nota: La segunda url recibe un parámetro una id del registro para modificar o eliminar.

```
from django.urls import path
from snippets import views

urlpatterns = [
    path('snippets/', views.SnippetList.as_view()),
    path('snippets/<int:pk>/', views.SnippetDetail.as_view()),
]
```

Servidor

Por ultimo ejecutaremos el servidor de la siguiente manera

```
python manage.py runserver
```

Y accederemos a la dirección que nos menciona en consola.

Conclusiones

Python es un lenguaje de programación muy útil y gracias a la gran variedad de frameworks que éste contiene es sencillo y rápido el desarrollo de software, aplicaciones o servicios dentro de esto. Django por su parte es un framework para la elaboración de páginas web en python el cual tiene una estructura bastante amable, cuenta con una gran cantidad de personas en su comunidad para ayuda y resolución de problemas dentro de foros oficiales. Y una gran documentación completa y sencilla. Pero por otro lado REST-FRAMEWORK es una framework de django que permite la transacción de información a través del protocolo http con sus respectivos métodos POST, GET, PUT, DELETE.