Django REST Framework Fazt.

<https://www.youtube.com/watch?v=GE0Q8YNKNgs&t=28s&ab_channel=Fazt>

# Introducción.

Este en un ejemplo práctico de Django REST Framework. Se trata de un framework encima de Django, que nos permite crear aplicaciones REST de forma sencilla.

En este ejemplo crearemos una aplicación CRUD, que nos permita crear, leer, actualizar y eliminar. Aprenderemos como desplegarlo de forma gratuita en render.com

## Qué es URI?

Identificador de recursos uniforme o URI —del inglés uniform resource identifier— es una cadena de caracteres que hace referencia a un recurso. Los más comunes son URLs, que identifican el recurso dando su ubicación en la Web. URNs (en-US), por el contrario, se refiere a un recurso por un nombre, en un espacio de nombres determinados, como el ISBN(International Standard Book Number) de un libro.

URI se emplea para todos los tipos de nombres y direcciones que se refieren a objetos internet tales como: páginas, imágenes, videos, etc.

Un URI es por tanto una cadena de caracteres que se utiliza para identificar un recurso o un nombre en internet. Su propósito es permitir la interacción entre diferentes recursos en Internet y otro tipo de red.

Aunque se acostumbra a llamar URL a todas las direcciones web, URI es un identificador más completo y por eso está recomendado su uso en lugar de la expresión URL.

## Que es un web service?

Es un programa diseñado para el intercambio de información máquina a máquina, sobre una red. Entonces esto hace que una computadora o programa pueda solicitar y recibir información de otra computadora o programa. A quien solicita la información se le llama cliente y a quien envía la información se le llama servidor.

## Qué es API?

La palabra viene de Application Programming Interface, y no es más que un programa, conocido como interfaz, con una capa de abstracción, que permite que 2 programas se comuniquen, por ejemplo Facebook. A diferencia de los web services, las API no necesariamente deben comunicarse entre una red, pueden usarse entre dos aplicaciones en una misma computadora.

## RESTful web service vs. RESTful api

La diferencia entre RESTful web service y RESTful api, es que el api no necesariamente se debe ejecutar en una red, puede ser en una misma computadora. Y son programas basados en REST.

## ¿REST vs RESTful?

RESTful web service o RESTful api, son programas web service o API basados en REST.

Pero muchas veces (REST y RESTful) se usan como sinónimos.

### ¿Que hace que un web service sea REST?

Usualmente los RESTful web service tienen estas características:

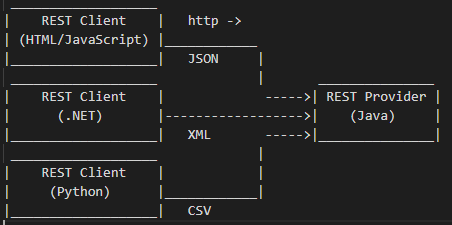
* Están asociados a información
* Permiten listar, crear, leer, actualizar y borrar información
* Para las operaciones anteriores necesitan una URL y un método HTTP para accederlas
* Usualmente regresan la información en formato JSON.
* Retornan códigos de respuesta HTML, por ejemplo 200, 201, 404, etc

IMPORTANTE.

Los servicios RESTful brindan interoperabilidad, lo que significa que cualquier aplicación que se ejecute en cualquier plataforma escrita en cualquier idioma puede ser consumida por cualquier otra aplicación que se ejecute en una plataforma completamente diferente y en un idioma diferente y en múltiples formatos de datos.

Esa es la clave de un servicio web RESTful

Ejemplo:



En este ejemplo podemos ver como todos los RESTful web services están escritos en lenguajes diferentes, tanto los clientes como el servidor. Además cada uno de los clientes está mandando sus peticiones en diferentes formatos. Esto es la INTEROPERABILIDAD y es posible gracias a MIME types (EE de ello más adelante).

## Métodos HTTP que se usan.

Cuando solicitamos una página web, podemos hacerlo por diferentes métodos, el más común es el GET, es el que usamos cuando digitamos una dirección en nuestro navegador, en ocasiones utilizamos POST, cuando enviamos un formulario con datos, pero las aplicaciones pueden usar otros métodos como PATCH, PUT, etc.

* Listar y leer: Usan el método GET
* Crear: Usan el método POST
* Actualizar: Usan el método PATCH para actualizar y PUT para reemplazar.
* Borrar: Usan el método DELETE

## Algunos códigos de estado HTTP.

Cuando se recibe una página HTML, también se recibe un código de estado HTTP (solo uno), en los web service RESTful, estos se usan para saber el estado de la ejecución del servicio, y estos son algunos de los usados:

* 200 (Ok), cuando una operación fue exitosa.
* 201 (Created), cuando se creó un registro (¿recuerdan? Con el método POST)
* 403 (Forbidden), cuando intentamos leer un registro para el que no tenemos acceso, por ejemplo los datos del perfil de otro usuario.
* 404 (Not found), cuando intentamos leer un registro que no existe.

## MIME types.

Los tipos MIME permiten a los navegadores reconocer el tipo de archivo que el servidor web ha enviado a través de HTTP. Los servidores insertan el encabezado MIME al comienzo de cualquier transmisión web.

Como resultado, el navegador (el cliente) puede elegir un método de visualización o reproductor adecuado.

Algunos de estos reproductores están integrados generalmente en el navegador (cliente) (por ejemplo, un reproductor de imágenes GIF y JPEG, así como la capacidad de manejar archivos HTML).

Los tipos MIME, también llamados a veces tipos de medios de Internet o tipos de contenido, describen el tipo de contenido de los medios, ya sea contenido en el correo electrónico o servido por servidores web o aplicaciones web, y están destinados a ayudar a guiar a un navegador web para que procese y muestre correctamente el contenido.

Ejemplos:

* text/html "for normal web pages"
* text/plain "for plain text"
* application/octet-stream meaning “download this file”
* application/x-java-applet "for Java™ applets"
* application/pdf "for Adobe® PDF documents."

Esto ayuda a que la arquitectura REST pueda leer multiplataforma.

Para conocer cuáles existen: <http://www.iana.org/assignments/media-types/media-types.xhtml>

## \*/\*/\*/ Django \*/\*/\*/

Django es un Framework web de código abierto escrito en python. Gracias a su repidez y flexibilidad puede crearse aplicaciones web en poco tiempo, partiendo de una arquitectura MVC (Modelo Vista Controlador).

Django pone énfasis en el re-uso, la conectividad, la extensibilidad de componentes, el desarrollo rápido y el pricipio de no te repitas (DRY).

Este framework te permite comunicar tus aplicaciones con bases de datos, utentificar a tus usuarios, crear formularios, manejar la seguridad de tu aplicación, crear interfaces html, tener un panel de administración para tu aplicación.

Django nos da las bases para crear aplicaciones web, pero este puede ser extendido con ayuda de otros frameworks, por ejemplo, Django REST Framework para crear aplicaciones conocidad como RESTfull API .

Es importante tener un poco de base en Frontend, es decir, conocer un poco de CSS, Html y JavaScrip, bases de datos (operaciones CRUD) y python.

## Django REST Framework

Django REST Framework es una extención de Django Framework, que permite a los desarrolladores el crear de forma fácil REST API, utilizando los conceptos de Django, como la generación de rutas, el panel de administrador, generar módulos y unirlos para crear una REST API.

Una REST API es una API con una estructura REST que permite que diferentes aplicaciones se comuniquen entre si, incluso cuando estas están escritas en códigos diferentes. Para que no haya ningún problema en la comunicación, se toma el protocolo http, y se usan los métodos http (GET, POST, PUT y DELETE), Con ellos el cliente realizará sus peticiones y las enviará a la REST API y REST define que para retornar información, será por medio de JSON, XML, etc, junto con un código de estado.

## Código de estado.

* 20X (Correcto)
* 30X (Cache - redireccionamiento)
* 40X (No encontrado)
* 50X (Error de servidor)

# REST in action

## Software Setup.

Vamos a comenzar el proyecto.

1. Creamos una carpeta que contendrá el proyecto.
2. Creamos un entorno virtual y lo activamos,
   1. Escribe en la consola >> pip install virtualenv, asegúrate de tener la ruta del programa ‘virtualenv’ dentro del ‘path’ de las variables de entorno.
   2. Escribe >> virtualenv venv

Con esto se creará una carpeta llamada ‘venv’, ahora nuestra carpeta (que represente al proyecto) ya será un entorno virtual, pero falta activarlo.

* 1. Para activarlo escribimos en consola >> .\venv\Scripts\activate
  2. Para activarlo presionamos ‘F1’ y escribimos <select interpreter>, seleccionamos aquella que sea <venv>.

1. Procedemos a instalar los elementos que vamos a utilizar.

>> pip3 install django

>> pip3 install djangorestframework

>> pip3 install mysqlclient (en el caso de que trabajaremos con MySQL)

1. No olvides crear tu fichero de "requirements"
2. Iniciamos un proyecto en django: django-admin startproject <Nombre del proyecto>
3. Iniciamos una app, dirigete con la terminal al interior de la carpeta del proyecto y escribe: python3 manage.py startapp <nombre de la app>
4. Dentro de setting.py en el apartado de ‘INSTALED\_APPS’, colocamos:

‘rest\_framework’

‘<El nombre de todas las apps que inicialicemos>’

1. Si gustas puedes a empezar a crear tu repositorio desde aquí

## Start Project.

1. En la terminal (venv) escribimos

>> django-admin startproject <nombre de proyecto> .

(no olvides el punto al final “ . ”). En <nombre de proyecto> se recomienda no usar nombres como ‘python’, ‘django’ o ‘test’, ya que son módulos que probablemente el framework utiliza y puede llegar a ocasionar errores más adelante. En este ejemplo se llamara “DRF\_CRUD”

1. Iniciamos una aplicación:

>> python manage.py startapp <nombre de la app>

En este caso lo nombraremos ‘projects’

Esta aplicación aun no es conocida por nuestro proyecto, por lo que hay que agregarlo a ‘DRF\_CRUD/settings.py -> INSTALLED\_APP’, al igual que la aplicación de ‘rest\_framework’.

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'rest\_framework',

    'projects',

]

# Modelo de datos.

Vamos a crear un modelo de proyecto para que nuestra API pueda hacer uso de nuestra tabla. Nos dirigimos a ‘projects/models.py’ y creamos un modelo.

from django.db import models

class Project(models.Model):

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

    description = models.TextField()

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

    created\_at = models.DateTimeField(auto\_now\_add=True)

Por último, debemos migrar los modelos a la base de datos para poder usarlas más adelante.

>> python manage.py makemigrations

>> python manage.py migrate

# API

Vamos a crear nuestra REST API, es momento de crear nuestros endpoints. Cuando nosotros ejecutamos nuestro módulo principal debemos crear un ‘ViewSet’, a través de Django podrá convertir los datos de python en objetos Json que luego podrán ser consultados por el cliente, también nos permite definir quién podrá acceder a estos datos. Para poder entender esto mejor continua la práctica.

Creamos un fichero en nuestra aplicación ‘projects/**serializers.py**’, aquí llamaremos un módulo especial de REST Framework (serializers).

from rest\_framework import serializers

Aquí indicaremos que esto está basado en un modelo que hemos creado anteriormente (class **Project**).

from .models import Project

Ahora hay que hacer que Django sepa que contestar cuando se haga alguna petición (Post, Put, Get, etc.)

# Esto convertirá un modelo en datos que puedan ser consultados

class ProjectSerializer(serializers.ModelSerializer):

    class Meta():

        # Nombre del modelo al que hacemos referencia.

        model = Project

        # Colocamos los campos que puedan ser consultados, es decir, que serán serializados.

        fields = ('id', 'title', 'description', 'technology', 'created\_at')

        # Definimos que campos son solo de lectura, es decir que no podrán ser modificados.

        read\_only\_fields = ('created\_at')

Con todo lo anterior ya tenemos a nuestro serializer. Ahora sí vamos a crear el ViewSet.

El **ViewSet** nos permite establecer quien puede consultar el **‘ProjectSerializer()’** del serializador.

Creamos un archivo ‘projects/api.py’ e importamos nuestro modelo.

from .models import Project

Importamos el módulo **Viewsets** y **permissions.**

from rest\_framework import viewsets, permissions

También vamos a importar el serializador que acabamos de crear hace poco para poder serializar los datos de ‘queryset’.

from .serializers import ProjectSerializer

Vamos a colocar lo siguiente en nuestro fichero actual, justo por debajo de nuestras importaciones.

# Qué consultas se podrán hacer.

class ProjectViewSet(viewsets.ModelViewSet):

    queryset = Project.objects.all()

    # Cualquier cliente o aplicación cliente podrá solicitar datos al servidor, más adelante puedes cambiarlo por 'IsAuthenticated' para que solo de acceso a los autentificados

    permission\_classes = [permissions.AllowAny]

    # A partir de que serializer se va serializar el queryset, es decir, cómo lo se va a convertir.

    serializer\_class = ProjectSerializer

Hasta aquí ya terminamos nuestra API, pero hace falta una URL que el cliente pueda consultar.

Creamos un fichero en nuestra app ‘projects/urls.py’

Podríamos crearlo manualmente como se ha hecho en otros ejemplos, donde hacemos uso de ‘urlpatterns = []’, y agregando las rutas una a una, pero con ayuda de DRF no hay necesidad de esto, pues contamos con la ayuda un módulo especial (**routers**) que nos ayudará a crear todas las rutas básicas (lo que se conoce como el CRUD).

from rest\_framework import routers

Este módulo router lo tenemos que ejecutar y con ayuda de ‘DefaultRouter’ vamos podremos crear el CRUD

router = routers.DefaultRouter()

Trabajaremos con el conjunto de datos que vienen del ViewSet que hicimos hace poco.

from .api import ProjectViewSet

Utilizando la variable creada ‘router’ (ya no ‘urlpatterns’), vamos a registrar el nombre de la ruta (‘api/projects’), indicaremos que está basado de ViewSet (ProjectViewSet) y por último nombraremos la ruta.

router.register('api/projects', ProjectViewSet, 'projects')

Y con esto ya tenemos una ruta creada con los verbos html (CRUD)