Django REST framework first Project.

# Introducción.

## \*/\*/\*/ REST \*/\*/\*/\*/\*/\*/\*/

REST no es un protocolo, ni un framework, no está atado a ningún lenguaje, es una lógica de restricciones y recomendaciones para la transmisión de datos en línea. Específicamente define cómo diseñar interfaces de programación de aplicaciones (API), las cuales permiten la comunicación de datos entre aplicaciones web.

La transferencia de estado representacional (REST) es un conjunto de principios arquitectónicos no es un protocolo oficial como SOAP, cuyo mantenimiento está a cargo del Consorcio World Wide Web (W3C).

* La principal diferencia es que SOAP es un protocolo, y REST no lo es. Por lo general, alguno de los dos regirá a las API, según el caso práctico y las preferencias del desarrollador.

Cuando se envía una solicitud de datos a una API de REST, se suele hacer a través de un protocolo de transferencia de hipertexto, comúnmente denominado HTTP. Una vez que reciben la solicitud, las API diseñadas para REST (conocidas como API o servicios web de RESTful) pueden devolver mensajes en distintos formatos: HTML, XML, texto sin formato y JSON. El formato preferido para los mensajes es la notación de objetos JavaScript (JSON), ya que, a pesar de su nombre, puede leerlo cualquier lenguaje de programación, es ligero ylo comprenden tanto las personas como las máquinas. De esta forma, las API de RESTful son más flexibles y se pueden configurar con mayor facilidad.

Se considera que una aplicación es RESTful si cumple con seis pautas arquitectónicas. Una aplicación de RESTful debe tener lo siguiente:

1. Una arquitectura cliente-servidor compuesta por clientes, servidores y recursos.

Es la separación de responsabilidades y la portabilidad. Cuanto menos conoce el servidor sobre el cliente más desacoplada está su interacción y más fácil resulta el cambio de componentes. La portabilidad se da dado que la aplicación puede desarrollarse independientemente del servidor, este puede expandirse sin problemas.

1. Una comunicación cliente-servidor sin estado (Stateless). Esto significa que el contenido de los clientes no se almacena en el servidor entre las solicitudes, sino que la información sobre el estado de la sesión queda en el cliente. Esto hace que no exista contexto entre servidor y cliente, por lo que una nueva petición es independiente y no depende de peticiones pasadas. En aplicaciones muy grandes el servidor consta de varias computadoras, y el stateless permite que se pueda realizar las peticiones del cliente, sin importar el computador que la resuelva.

Estas peticiones son mandadas en archivos XML, JSON, TextoPlano etc.

Aquí se puede denotar la multirepresentabilidad que ofrece REST, pues el cliente es capas de leer diferentes tipos de recursos con ayuda de MIME types y los reproductores que usualmente tenemos en el navegador (Lee más sobre esto en el tema de "MIME types" que se encuentra más adelante).

1. Datos que pueden almacenarse en caché para eliminar la necesidad de algunas interacciones cliente-servidor, conocido como Cacheability. El servidor se encarga de definir que interacciones pueden ser almacenadas, por cuanto tiempo, etc.
2. Una interfaz uniforme entre elementos para que la información se transfiera de forma estandarizada, en lugar de ser específica para las necesidades de cierta aplicación. Este principio dicta el estandar de comunicación entre el cliente y el servidor, con el uso de métodos:

\*/ HTTP (GET, POST, PUT, DELETE, conocidos también, como verbos HTTP).

HTTP provee un acceso fácil y único para identificar los recursos, "URI".

\*Ejemplos sobre como se convina la forma estandarizada de mandar mensajes por parte del cliente, y el acceso a los recursos al servidor.

Estandarizar la interfaz permite que cada capa del sistema evolucione independientemente, ya que la comunicación fue estandarizada, no tienes porque preocuparte como esta siendo desarrollado el servidor, si tú estas trabajando en el cliente.

URI: 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.

Roy Fielding, el creador de REST, lo describe como "la característica principal que distingue el estilo arquitectónico de REST de los demás estilos basados en la red".

1. Una restricción del sistema en capas, en el que las interacciones cliente-servidor pueden estar mediadas por capas jerárquicas, cada una única y bien definida. Estas capas solo deben poder comunicarse con capas aledañas, es decir que dos capas separadas no pueden comunicarce directamente. Además las capas deben ser independientes, esto nos permitiría sustituirlas sin problemas.
2. Código según se solicite, lo que permite que los servidores amplíen las funciones de un cliente al transferir el código ejecutable (esto también reduce la visibilidad, así que es una pauta opcional).

Para más información.

https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest

## 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.

## Django.

Django es un framework para el desarrollo de aplicaciones y páginas web.

## 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:

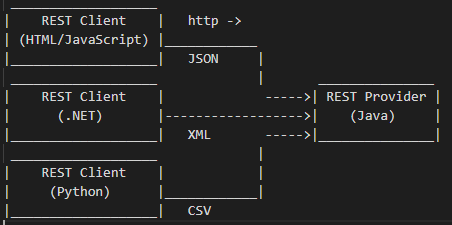
* Están asociados a información
* Permiten listar, crear, leer, actualizar y borrar información
* Para las operaciones anteriores necesitan una URL y un método HTTP para accederlas
* Usualmente regresan la información en formato JSON.
* Retornan códigos de respuesta HTML, por ejemplo 200, 201, 404, etc

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)

# Software setup

A continuación, se muestra una lista se programnas con los que deberás contar e instalar para poder continuar o

que te ayudarán en la realización de este curso.

## How to install Django? \*/\*/\*/

Coloca los siguiente en la línea de comando.

>> pip3 install django

Para saber la versión:

>> python3 -m django --version

## How to install Django REST Framework? \*/\*/\*/

Coloca los siguiente en la línea de comando.

>> pip3 install djangorestframework

Install MySQL y conector para python mysqlclient

Después de instalar MySQL server y MySQL workbench, instala lo siguiente:

>> pip3 install mysqlclient

Que te permite comunicarte deste Django.

## Install Postman

Su principal función es para hacer pruebas de testing REST, monitorizar, escribir pruebas automatizadas, documentarlas, simularlas, etc.

# REST in action

## Create de project.

Vamos a comenzar el proyecto, para ello vamos repetir algunos pasos que se hicieron en el proyecto de DjangoProjectBlog.

que se encuentra en el mismo directorio que este 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 este dcaso trabajaremos con MySQL)

1. No olvides crear tu fichero de "requirements"
2. Iniciamos uin 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

## Create a view.

Dirígete a ‘firstApp / views.py’ y agrega lo sig: (no quites cualquier otra cosa que ya esté escrito en el fichero).

from django.http import JsonResponse

# Create your views here.

def employee\_view(request):

    emp = {

        'id': 123,

        'name': 'John',

        'sal': 5000

    }

    return JsonResponse(emp)

‘JsonResponse’ nos ayuda a serializar el diccionario en un odato tipo json, en esta función, retornaremos una respuesta json con los datos del empleado “John”.

## Configure the URL and TEST.

Para que podamos ver esto en una ventana del navegador (por el momento una ventana de local host), nos dirigimos a ‘firstProject / urls.py’ y agregamos la función que acabamos de crear.

from firstApp import views

urlpatterns = [

    path('admin/', admin.site.urls),

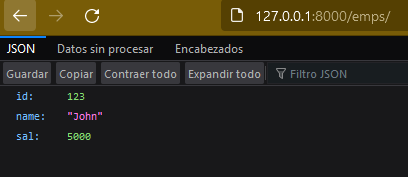
    path('emps/', views.employee\_view)

prueba el servidor con:

>> python manage.py migrate

>> python manage.py runserver

Verás algo así:



## Include.

Sucede que cuando nosotros trabajamos en el archivo ‘firstProject / urls.py’ y nos encontramos agregando rutas para ejecutar funciones en ellos, esta lista puede llegar a ser muy extensa si nuestro proyecto cuenta con varias aplicaciones o muchas funciones para ejecutar. Digamos por ejemplo que tenemos lo siguiente

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', hello),

    path('otra\_ruta/', otra\_ruta),

    path('otra\_ruta2/', otra\_ruta2),

    path('otra\_ruta3/', otra\_ruta3),

    ...

    path('otra\_rutan/', otra\_rutan)

]

Para evitar que se extienda demasiado, lo que haremos será ejecutar estas rutas desde cada una de las carpetas de cada aplicación que creemos. Para ello:

1. Creamos un archivo llamado ‘urls.py’ dentro de las aplicaciones que vayamos a llamar por una URL, en este caso lo creamos dentro en la carpeta ‘firstApp’
2. Dentro del nuevo archivo colocamos:

from django.urls import path

from . import views

urlpatterns = [

    path('emps/', views.employee\_view),

]

Es ahora aquí donde estamos colocando las rutas y las funciones que se ejecutarán

1. Por otro lado, en ‘firstProject / urls.py’, tendremos lo sig.

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

    path('admin/', admin.site.urls),

    path('firstApp/', include('firstApp.urls'))

]

Hemos importado la función **‘include’**, y hemos agregado ‘path('', include('firstApp.urls'))’.

De esta manera estamos incluyendo los path de las URL’s de la aplicación ‘firstApp’.

Si te diriges a <http://127.0.0.1:8000/firstApp/emps/> podrás ver que ahí se encuentra ‘employee\_view’.

## Create a model class.

En este momento podemos ver algunos datos dentro de un **objeto estático** (employee\_view) pero podemos tener estos datos en una tabla en una base de datos en su lugar.

Para ello vamos a crear una nueva clase modelo en ‘firstApp / models.py’:

class Employee(models.Model):

    id = models.IntegerField(primary\_key=True)

    name = models.CharField(max\_length=20)

    sal = models.DecimalField(max\_digits=10, decimal\_places=3)

    def \_\_str\_\_(self) -> str:

        return self.id + self.name + self.sal

primary\_key=True, significa que este campo se llenará automáticamente de 1 a n, por cada objeto que se instancie de la clase.

\_\_str\_\_ es un método especial, que nos permite retornar una cadena de texto cuando ponemos un objeto creado de la clase por una función print.

## Configurar las bases de datos y correr las migraciones.

En ‘firstProject / setting.py -> DATABASES’, borraremos lo que hay en el y colocaremos lo siguiente:

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'employeedb',

        'USER': 'root',

        'PASSWORD': 'test1234'

    }

}

‘USER’ y ‘PASSWORD’, deben de ser los que utilizaste cuado instalaste MySQL Worckbench.

Después ejecuta MySQL y creamos una base de datos con el nombre de ‘employeedb’.

Abrimos MySQL Workbench, y dentro escribimos las siguientes sentencias:

* create database employeedb;
* use emloyeedb;
* show tables; (podrás notar que no se visualiza ninguna tabla)

Nos dirigimos a nuestro proyecto en nuestro editor de código y hacemos las migraciones pertinentes, volvemos a ejecutar -show tables;- y podremos ver como ahora si se puede ver la tablas pertenecientes al proyecto, varias de ellas no las vamos a utilizar, ni siquiera las vamos a tocar, la que nos interesa es la que acabamos de crear, y para poder trabajar en ella lo hacemos de la siguiente manera:

* select \* from firstapp\_employee;

Listo ahora podemos ver su contenido, que por el momento es nada, aún hay que crear contenido.

## Use the model in the view and test.

Actualmente tenemos una función que contiene un dato estático en ‘views.py’ (def employee\_view), en lugar de hacer uso de el lo que queremos es hacer uso de muestro modelo creado recientemente.

Escribimos en ‘views.py’

from firstApp.models import Employee

y

data = Employee.objects.all()

response = {'employees': list(data.values('name', 'sal'))}

# Consume datos, como el caso del dicionario de arriba, para selializarlos en un dato tipo JSON.

return JsonResponse(response)

Notas personales: parece ser que tiene que ser una lista para por serializarlo, intente sin este elemento y no me permitía visualizar los datos.

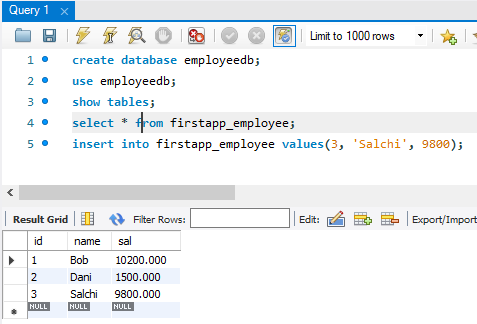
Data.values() -> te permite obtener todos los valores de data de forma iterable, aquí mismo puedes especificar que campos desear que se tomen.

Aun no tenemos ningún dato en este modelo. Dirijámonos a la interfaz de MySQL y escibamos:

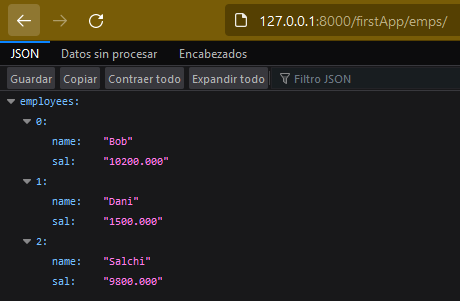
* insert into firstapp\_employee values(1, ‘Bob’, 10200)

Luego solo cambiemos los valores de ‘values()’ por más usuarios den banco y los ejecutemos a cada uno de ellos, asegúrate de crear al menos 3 datos diferentes de usuarios.

Hasta ahora tu pantalla de Query y tu tabla ala que le has insertado al menos 3 datos, deberían verse algo así:



Dirígete a tu terminal y corramos el servidor para ver los resultados.



# Serializadores