Django REST framework

Django REST framework es un conjunto de herramientas potente y flexible para crear API web. La API es navegable, tiene un marco y usabilidad web para nosotros los desarrolladores. También cuenta con políticas de autenticación. Usa "Serialización" que admite fuentes de datos ORM y no ORM. Es personalizable por completo: solo use las vistas regulares basadas en funciones si no necesita las funciones más potentes. Amplia documentación y gran apoyo de la comunidad. Utilizado y confiado por empresas reconocidas internacionalmente, incluidas Mozilla, Red Hat, Heroku y Eventbrite.

Empezamos

0) Requerimientos:

```
djangorestframework
markdown       # Markdown support for the browsable API.
django-filter    # Filtering support
```

1) Crear el proyecto

```
python django-admin startproject myproject
```

2) Crear la aplicación

```
python manage.py startapp servicio
```

3) Agregar nuestra aplicación en myproject/myproject/settings.py y agregar nuestra aplicación así:

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

4) Modelo

Los modelos de Django se refieren a las tablas de la base de datos, los atributos de ese modelo se convierten en las columnas de esa tabla, esos atributos reciben el nombre de "django fields" los cuales manejan automáticamente las conversiones de tipos de datos para la base de datos que estemos usando.

Una de las grandes características de Django es su ORM, gracias a él no tenemos que escribir ninguna consulta de base de datos, e incluso se recomienda NO escribir una al usar Django.

ORM convierte sus Django models y todas las operaciones que realiza con las consultas de base de datos correspondientes. Esto significa que toda la manipulación se hará ahora con "objetos" desde Python creados a partir de ese modelo("clase"), y todo el material de la base de datos subyacente será cuidado por el ORM de Django.
Vamos a definir modelo "Salida" (clase) con los campos: hora(CharField), ciudad(CharField), vuelo(CharField), puerta(IntegerField) y observaciones(TextField). Para ello, abrimos el archivo myapp/models.py y escribimos:

```python
from django.db import models

class Salida(models.Model):
    hora = models.CharField(max_length=10,null=True)
    ciudad = models.CharField(max_length=100,null=True)
    vuelo = models.CharField(max_length=50,null=True)
    puerta = models.IntegerField(null=True)
    observaciones = models.TextField(null=True)

    class Meta:
        verbose_name = "Salida"
        verbose_name_plural = "Salidas"

    def __str__(self):
        return self.vuelo
```

Tipos de campos en modelos:
https://docs.djangoproject.com/en/4.1/ref/models/fields/

5) Migraciones. Realizamos la migración.

Las migraciones son la forma en que Django propaga los cambios que realiza en sus modelos (agregar un campo, eliminar un modelo, etc.) en el esquema de su base de datos.

makemigrations, que se encarga de crear nuevas migraciones en función de los cambios que haya realizado en sus modelos.

migrate, sirve para aplicar migraciones.

6) Crear el archivo serializers.py en la aplicación. Este es un archivo necesario propuesto por el framework que vincula el modelo con nuestra api.

```python
from .models import Salida
from rest_framework import serializers


class SalidaSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Salida
        fields = ["id","hora","ciudad", "vuelo", 'puerta', 'observaciones']
```

7) En views.py (en la aplicación) vamos a estar creando una clase a diferencia de otras veces donde se usa el framework con la idea original WSGI(funciones que vinculan a direcciones).
La idea es que esta clase resuelva el funcionamiento del servicio.

```
from .models import Salida
from rest_framework import viewsets, permissions
from .serializers import SalidaSerializer

class SalidaViewSet(viewsets.ModelViewSet):
    queryset = Salida.objects.all()
    permission_classes = [permissions.AllowAny]
    serializer_class = SalidaSerializer
```

8) Rutas en url.py
Recordemos que este archivo debemos crearlo nosotros, ya que en la aplicación(servicio se llama la nuestra) no vienen por defecto.

```
from django.urls import URLPattern
from rest_framework import routers
from .views import SalidaViewSet

router = routers.DefaultRouter()

router.register('api/departure',SalidaViewSet,"salidas")

urlpatterns = router.urls
```

Fuente: https://www.django-rest-framework.org/tutorial/quickstart/#quickstart

9) También si queremos podemos definir los métodos de nuestro servicio que un cliente puede emplear con **`http_method_names`** en el **ViewSet**

Por ejemplo:

```
class AlumnosViewSet(viewsets.ModelViewSet):
    queryset = Alumnos.objects.all()
    permission_classes = [permissions.AllowAny]
    serializer_class = AlumnosSerializer
    http_method_names = ['get'] #Ejemplo solo GET
    #http_method_names = ['get', 'post', 'put', 'delete']
```

10) Implementar autentificación con token de manera sencilla (seguridad)
    A) Lo primero es modificar en settings.py y agregar en INSTALLED_APPS [….] `'rest_framework.authtoken'` como una aplicación mas.

B) Agregar en settings.py también el siguiente campo:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
}
```

C) Luego realizar las migraciones correspondientes

D) Crear un usuario para administrar:
`python manage.py createsuperuser`

E) Por último, pedir el Token:
`python manage.py drf_create_token "tu_usuario"`

Para finalizar en views.py modificar de la clase de nuestro ViewSet modificar:

```
permission_classes = [permissions.AllowAny]
```

Por:

```
permission_classes = [permissions.IsAuthenticated]
```

Recordar de utilizar el token como vimos en clase.

Tener en cuenta que esta es una manera muy básica de conseguir una autentificación. Existen mejores y más complejas soluciones.

Fin del apunte.

Adicional:

Los JWT (JSON Web Tokens) se han popularizado enormemente, incluso algunos las consideran un reemplazo de los clásicos Tokens que usan otros frameworks, tales como Django Rest Framework.

Usar JWT permite guardar toda la información de nuestra sesión directo en el token y además están firmados criptográficamente.

JWT es un estándar para la creación de tokens de acceso basado en JSON, para el intercambio de información entre dos partes. Estos tokens, y su contenido, pueden ser verificados porque están firmados digitalmente. Esta firma criptográfica garantiza que el contenido no ha sido alterado y que el emisor es quien dice ser.

Existen varios módulos y paquetes para hacer uso de esta tecnología, por ejemplo:

Simple JWT

Cualquier duda estamos en contacto.
Saludos y Éxitos.
CB