
Django REST Framework
lo mejor de Django
sin lo peor de Django

Algunas definiciones

API (Interfaz de programación de aplicaciones)

- Están pensadas para ser accedidas por otros programas.

- Ventajas de ofrecer una API como servicio:
  - Control de la información que se entrega.
  - Información actualizada.
  - Flexibilidad del manejo interno del servicio.
  - Volumen de datos.
  - Facilidad de filtrar información.
  - Datos normalizados.

- Desarrollo orientado a microservicios:
  - Equipos de desarrollo pequeños y especializados

Algunas definiciones

CRUD o ABM

- Se refiere a las operaciones básicas (“Crear, Leer, Actualizar y Eliminar”) de los objetos de nuestra base de datos.

REST

- Estilo de arquitectura de software para la creación de APIs.
- Métodos HTTP explícitos:
  - GET
  - POST
  - PUT
  - PATCH
  - DELETE

Django REST Framework

Serializers

- Convierten objetos de Python a formatos de datos más simples como JSON y XML (serialización) y viceversa (deserialización).
- Validan los datos que recibe la aplicación, como los Forms en Django.

Vistas (views) especializadas

- Vistas basadas en clases que se corresponden con los métodos de HTTP utilizados para CRUD:
  - CreateAPIView → POST
  - RetrieveAPIView y ListAPIView → GET
  - UpdateAPIView → PUT + PATCH
  - DestroyAPIView → DELETE

Serializers

```
class CommentSerializer(serializers.Serializer):
    email = serializers.EmailField()
    content = serializers.CharField(max_length=200)
    created = serializers.DateTimeField()

    def create(self, validated_data):
        return Comment(**validated_data)

    def update(self, instance, validated_data):
        instance.email = validated_data.get('email', instance.email)
        instance.content = validated_data.get('content', instance.content)
        instance.created = validated_data.get('created', instance.created)
        return instance
```

```
serializer = CommentSerializer(data={'email': 'foobar', 'content': 'baz'})
serializer.is_valid()
# False
serializer.errors
# {'email': [u'Enter a valid e-mail address.'], 'created': [u'This field is required.']}
```

ModelSerializer

```
class AccountSerializer(serializers.ModelSerializer):
    class Meta:
        model = Account
        fields = ('id', 'account_name', 'users', 'created')
```

- Basado en el modelo, genera automáticamente los campos y validaciones del serializer.
- Muy similar a ModelForm de Django.

Vistas basadas en funciones

```
from rest_framework.decorators import api_view


@api_view(['GET', 'POST'])
def hello_world(request):
    if request.method == 'POST':
        return Response({
            "message": "Got some data!",
            "data": request.data
        })
    return Response({
        "message": "Hello, world!"
    })
```

Vistas basadas en clases

```python
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import authentication, permissions
from django.contrib.auth.models import User


class ListUsers(APIView):
    """
    View to list all users in the system.

    * Requires token authentication.
    * Only admin users are able to access this view.
    """
    authentication_classes = (authentication.TokenAuthentication,)
    permission_classes = (permissions.IsAdminUser,)

    def get(self, request, format=None):
        """
        Return a list of all users.
        """
        usernames = [user.username for user in User.objects.all()]
        return Response(usernames)
```

Vistas genéricas

```
from django.contrib.auth.models import User
from myapp.serializers import UserSerializer
from rest_framework import generics
from rest_framework.permissions import IsAdminUser


class UserList(generics.ListCreateAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = (IsAdminUser,)
```

Viewsets

```
class UserViewSet(viewsets.ModelViewSet):
    """
    A viewset for viewing and editing user instances.
    """
    serializer_class = UserSerializer
    queryset = User.objects.all()
```

- Encapsulan la lógica de varias vistas relacionadas en una sola clase.
- Permiten utilizar actions y routers.
- GenericViewSet, ModelViewSet y ReadOnlyModelViewSet.

Actions

```
class UserViewSet(viewsets.ReadOnlyModelViewSet):
    """
    A viewset that provides the standard actions
    """
    queryset = User.objects.all()
    serializer_class = UserSerializer
    lookup_field = 'username'

    @action(detail=True)
    def group_names(self, request, pk=None):
        """
        Returns a list of all the group names that the given
        user belongs to.
        """
        user = self.get_object()
        groups = user.groups.all()
        return Response([group.name for group in groups])
```

Routers

- Generan automáticamente estructuras de URLs típicas.
- Si no se especifica el basename, se genera automáticamente en base al queryset de la viewset.

```
router = SimpleRouter()
router.register(r'users', UserViewSet, basename='user')
urlpatterns = router.urls
```

```
<URLPattern '^users/$' [name='user-list']>
<URLPattern '^users/(?P<username>[^/.]+)/$' [name='user-detail']>
<URLPattern '^users/(?P<username>[^/.]+)/group_names/$' [name='user-group-names']>
```

Filtering

```
class ProductList(generics.ListAPIView):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    filter_backends = (DjangoFilterBackend,)
    filter_fields = ('category', 'in_stock')
```

```
http://example.com/api/products?category=clothing&in_stock=True
```

Search

```
from rest_framework import filters

class UserListView(generics.ListAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    filter_backends = (filters.SearchFilter,)
    search_fields = ('username', 'email')
```

```
http://example.com/api/users?search=russell
```

Ordering

```
class UserListView(generics.ListAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    filter_backends = (filters.OrderingFilter,)
    ordering_fields = ('username', 'email')
```

```
http://example.com/api/users?ordering=-email,username
```

¡Gracias!

¿Preguntas?

- Documentación oficial: https://www.django-rest-framework.org