

**Renato Oliveira****09 Feb 2021**

5 min read

Associated topics

Django

Serializers

# Como usar Serializers de Django REST Framework

Serialização é o processo de transformar dados em um formato que pode ser armazenado ou transmitido e, então, reconstruído. Ele é usado em todas as partes do desenvolvimento de aplicações, ou quando estamos armazenando dados numa base de dados, na memória ou convertendo-os em arquivos.

Recentemente tive a oportunidade de ajudar dois desenvolvedores aqui da **Labcodes** a entender mais sobre serializers, e achei que seria uma boa compartilhar minha abordagem com outras pessoas.

Suponha que você está criando um software para um site de e-commerce e você tem um pedido que registra a compra de um único produto, por alguém, por um certo preço, em uma certa data:

```
class Order:
```



Agora, imagine que você quer armazenar e recuperar os dados desse pedido através de um banco de dados de chave-valor. Por sorte, sua interface aceita e retorna dicionários, então você precisa converter seu objeto Order em um dicionário:

```
def serialize_order(order):  
    return {  
        'product': order.product,  
        'customer': order.customer,  
        'price': order.price,  
        'date': order.date  
    }
```

E, se você quiser alguns dados desse banco de dados, você pode pegar os dados do dicionário e transformá-los no objeto do seu Pedido:

```
def deserialize_order(order_data):  
    return Order(  
        product=order_data['product'],  
        customer=order_data['customer'],  
        price=order_data['price'],  
        date=order_data['date'],  
    )
```

É um processo simples e direto quando estamos trabalhando com dados simples, mas quando temos que lidar com objetos de atributos complexos, essa abordagem não escala bem. Você também precisaria realizar a validação de diferentes tipos de campos, e isso seria trabalho demais para se fazer manualmente.

[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

**Django** vem com um módulo de serializers que permite que você “traduza” Models para outros formatos:

```
from django.core import serializers

serializers.serialize('json', Order.objects.all())
```

Ele cobre a maior parte dos casos para as aplicações web, como JSON, YAML e XML. Mas você também pode usar serializers de terceiros ou criar os seus próprios. Você só precisa cadastrá-los no seu arquivo de settings.py:

```
# settings.py
SERIALIZATION_MODULES = {
    'my_format': appname.serializers.MyFormatSerializer,
}
```

Para criar seu próprio **MyFormatSerializer**, você precisa implementar o método **serialize()** e aceitar um QuerySet e opções extras como parâmetros:

```
class MyFormatSerializer:
    def serialize(self, queryset, **options):
        # serialização muito complicada acontecendo
```

Agora, você pode transformar seu QuerySet em um dado de novo formato:

[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

Você pode usar parâmetros opcionais para definir o comportamento do seu serializer. Por exemplo, se você definir que quer trabalhar com a serialização em formato “nested” (em detrimento de “flat”), em situações de **ForeignKeys**, ou se você simplesmente quer que os dados retornem para suas chaves primárias, você pode criar um parâmetro **flat=True** como uma opção e lidar com isso dentro do método:

```
class MyFormatSerializer:

    def serializer(self, queryset, **options):
        if options.get('flat', False):
            # não serializar recursivamente
            # serializar recursivamente
```

Uma forma de usar a serialização Django é com os comandos de gestão **loaddata** e **dumpdata**.

## Serializers em Django REST Framework

Na comunidade de Django, o **Django REST framework** (DRF) fornece os serializers mais conhecidos. Apesar de você poder usar serializers de Django para construir o JSON ao qual você vai expor na sua API, o serializer do framework REST vem com boas funcionalidades que podem te ajudar a lidar com e validar dados complexos.

No exemplo do Pedido, você pode criar um serializer assim:

[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

```
from restframework import serializers

class OrderSerializer(serializers.Serializer):
    product = serializers.CharField(max_length=255)
    customer = serializers.CharField(max_length=255)
    price = serializers.DecimalField(max_digits=5, decimal_places=2)
    date = serializers.DateField()
```

E facilmente serializar seus dados:

```
order = Order('pen', 'renato', 10.50, date.today())
serializer = OrderSerializer(order)
serializer.data

# {'product': 'pen', 'customer': 'renato', 'price': '10.50', 'date': '2023-11-25'}
```

Para conseguir retornar uma instância de dados, você precisa implementar dois métodos—*create* e *update*:

[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

```
from rest_framework import serializers

class OrderSerializer(serializers.Serializer):
    product = serializers.CharField(max_length=255)
    customer = serializers.CharField(max_length=255)
    price = serializers.DecimalField(max_digits=5, decimal_places=2)
    date = serializers.DateField()

    def create(self, validated_data):
        # perform order creation
        return order

    def update(self, instance, validated_data):
        # perform instance update
        return instance
```

Depois disso, você vai poder criar e atualizar instâncias apenas chamando `is_valid()` para validar os dados e `save()` para criar ou atualizar uma instância:

```
serializer = OrderSerializer(**data)
### to validate data, mandatory before calling save
serializer.is_valid()
serializer.save()
```

## Serializers Model

Quando você está serializando dados, é comum ter que fazer isso

[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

```
from django.db import models

class Order(models.Model):
    product = models.CharField(max_length=255)
    customer = models.CharField(max_length=255)
    price = models.DecimalField(max_digits=5, decimal_places=2)
    date = models.DateField()
```

Você pode criar um serializer para ele desta forma:

```
from rest_framework import serializers

class OrderSerializer(serializers.ModelSerializer):
    class Meta:
        model = Order
        fields = '__all__'
```

Django cria automaticamente todos os campos do model no serializer, além de também criar os métodos **create** e **update**.

## Usando serializers em Class-based Views (CBVs)

Assim como os Forms com os CBVs de Django, serializers também se integram bem com DRFs. Você pode definir o atributo

**serializer\_class** para que o serializer fique disponível para a View:

[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

```
from rest_framework import generics

class OrderListCreateAPIView(generics.ListCreateAPIView):
    queryset = Order.objects.all()
    serializer_class = OrderSerializer
```

Você também pode definir o método `get_serializer_class()`:

```
from rest_framework import generics

class OrderListCreateAPIView(generics.ListCreateAPIView):
    queryset = Order.objects.all()

    def get_serializer_class(self):
        if is_free_order():
            return FreeOrderSerializer
        return OrderSerializer
```

Existem outros métodos nos CBVs que interagem com serializers. Por exemplo, `get_serializer()` retorna um serializer já instanciado, enquanto `get_serializer_context()` retorna os argumentos que você vai passar ao serializer quando estiver criando sua instância. Para Views que criam ou atualizam dados, existem ainda `create` e `update`, que validam dados a serem salvos com o método `is_valid`. Por fim, ainda temos o `perform_create` e o



[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

ERICSON **Classy Django REST Framework**. É uma versão "test

framework” do **Classy Class Based Views** que vai te dar uma análise em profundidade das classes que compõem o DRF. E é claro que a **documentação oficial** também é um recurso sem igual.

### SUBSCRIBE TO GET OUR ARTICLES FIRST!

Subscription implies consent to our privacy policy



## Related Posts

Development 7 min read

### Migração o código-fonte de um design system para...

Juliana Barros Lima (Jules) ● 06 Out 2023

Development 4 min read

### Simplificando a criação testes com model-bakery

Higor Vinicius ● 23 Mai 2023

#### ALSO ON LABCODES

Decoupling logic from react components

3 years ago • 1 comment

Como escrever microtextos para ...

3 years ago • 1 comment

Technical Debt: 'ruin your softwa

3 years ago • 3 comment

[All posts](#)[Conferences](#)[Business](#)[Development](#)[Design](#)

0 Comments

[1 Login](#)

G

[LOG IN WITH](#)[OR SIGN UP WITH DISQUS](#) ?[Share](#)[Best](#)[Newest](#)[Oldest](#)

Be the first to comment.

[Subscribe](#)[Privacy](#)[Do Not Sell My Data](#)

# Let's build something cool together!

[Talk about your project](#)

The content of our blog is licenced under  
Creative Commons Attribution 4.0 International.

Designed and coded with ❤ in Recife.

Directly from the  **MANGUEZ.AL**

[contact\(at\)labcodes.com.br](mailto:contact(at)labcodes.com.br)



**BLOG**



All posts

Conferences

Business

**Development**

Design