DRY Your Serializers

Organising relationships in
Django Rest Framework

whoami

Ian Clark

@evenicoulddoit

Lead Software Engineer

Housekeep (we’re hiring👌)

We’ll cover

- What is Django Rest Framework
- The problems we encountered
- How we solved them
- How you can too

Django Rest Framework - tl;dr

- Django framework
- Developing REST-based web APIs
- Built in support for permission checking, validation, throttling etc.
- Well documented
- Familiar design principles - forms & serializers
- Ease of use
- Great third party tools & community

Our Journey

Stage 1: Profit 😎

```
class PropertySerializer(serializers.ModelSerializer):
    class Meta:
        model = Property
        fields = (
            'account', 'line_1', 'line_2', 'city', 'postcode'
        )


class PropertyAPIView(generics.RetrieveAPIView):
    queryset = Property.objects.all()
    permission_classes = (...)
    serializer_class =  PropertySerializer
```

Stage 2: Organic development 😬

```
class BasePropertySerializer(serializers.ModelSerializer):
    class Meta:
        model = Property
        fields = (...)

class NewBookingPropertySerializer(serializers.ModelSerializer):
    class Meta:
        model = Property
        fields = (..., 'bedrooms', 'bathrooms', 'access')
```

Stage 3: Anarchy 😵

```
class BasePropertySerializer(serializers.ModelSerializer):
    ...

class NewBookingPropertySerializer(serializers.ModelSerializer):
    ...

class MemberPropertySerializer(serializers.ModelSerializer):
    ...

class OpsBasicPropertySerializer(serializers.ModelSerializer):
    ...
```

Taking back control

Our requirements

- Formalise serializer relationships
- Conditionally include fields & serializers
- Provide enough context for end user to make further calls later

Our solution - expandable fields (serializers)

```
class PropertySerializer(SerializerExtensionsMixin, serializers.ModelSerializer):
    class Meta:
        ...
        expandable_fields = dict(
            account=AccountSerializer,
            jobs=dict(
                serializer=JobSerializer,
                source='job_set',
                many=True
            ),
            access_information=dict(
                serializer=PropertyAccessInformationSerializer,
                source='*',
                id_source=False
            ),
        )
```

Our solution - expandable fields (views)

```
class PropertyAPIView(SerializerExtensionsAPIViewMixin, RetrieveAPIView):
    pass


class PropertyAPIViewWithDefaults(SerializerExtensionsAPIViewMixin, RetrieveAPIView):
    extensions_expand = {'job', 'account'}


class ImmutablePropertyAPIView(SerializerExtensionsAPIViewMixin, RetrieveAPIView):
    extensions_expand = {'job', 'account'}
    extensions_query_params_enabled = False
```

Varying the response (basics)

```
>>> GET /properties/x4Q/
{
    ...,
    "account_id": "kgD"
}
```

```
>>> GET /properties/?expand=jobs
{
    ...,
    "account_id": "kgD",
    "jobs": [...]
}
```

```
>>> GET /properties/x4Q/?expand=account
{
    ...,
    "account_id": "kgD",
    "account": {...}
}
```

```
>>> GET /properties/x4Q/?expand=access_information
{
    ...,
    "account_id": "kgD",
    "access_information": {...}
}
```

Varying the response (nested expansion)

```
>>> GET /properties/x4Q/?expand=account__customers
{
    ...,
    "account_id": "kgD",
    "account": {
      ...,
      "customers": [...]
    }
}
```

Varying the response (filtering fields)

```
>>> GET /properties/x4Q/?expand=account&only=line_1,account__name
{
    "line_1": "742 Evergreen Terrace",
    "account": {"name": "Homer Simpson"}
}


>>> GET /properties/x4Q/?expand=account&exclude=line_1,account__name
{
    "line_2": "Springfield",
    "account_id": "kgD",
    "account": {
      "active": true
    }
}
```

Feature Roundup

- ✓ A single serializer for all situations
- ✓ Individual views and end users can expand & filter fields conditionally
- ✓ Supports all Serializers
- ✓ Supports SerializerMethodFields
- ✓ Expand *-to-one and *-to-many relationships
- ✓ Optional reverse ForeignKey expansion
- ✓ ID-only expansion for many relationships
- ✓ BONUS: Supports HashIds
- ~ Optimized queries
- ✗ Setting *-to-many relationships

Alternative approaches

- Multiple serializers
- Similar DRF packages
  - drf-flex-fields
  - djangorestframework-queryfields
  - There's more....
- Non DRF solutions
  - Falcor - Created by Netflix - NodeJS based
  - Graphene - GraphQL for Python

Try it out!

- Installing

  ```
  $ pip install djangorestframework-serializer-extensions
  ```

- Full documentation

  django-rest-framework-serializer-extensions.readthedocs.io

- Contribute

  github.com/evenicoulddoit/django-rest-framework-serializer-extensions

Thanks

Ian Clark

@evenicoulddoit

Lead Software Engineer

Housekeep (we’re hiring👌)