drf-toolbox Documentation

Release 0.1.7

FeedMagnet

November 30, 2014

Contents

This is drf-toolbox, a set of specialty classes that adds support for enhanced functionality on top of Django REST Framework.

DRF toolbox works with Django REST Framework and adds specialty classes to provide the following functionality:

- **Nested relationships for Foreign Keys**
  - Nesting of URLs of parent-child relationships within URL routing.
  - Display of full parent or child relationships within renderers.
  - Support for more robust display of API endpoints.
- Support for fields provided by django-pgfields.
- JSON rendering of `datetime` objects to Unix timestamps.

CHAPTER 1

Dependencies & Limitations

drf-toolbox depends on:

- Python 2.7+ or 3.3+ (Python 2.6 probably works, but is not explicitly tested against.)
- Django 1.6+
- Django REST Framework 2.3.10+
- dict.sorted 1.0.0+
- pytz 2013.9+
- six 1.4.1+

These dependencies are installed automatically when installing through pip.

CHAPTER 2

Quick Start

In order to use drf-toolbox in a project:

- **Installation**
  - `pip install drf-toolbox`
- **Usage**
  - Add DRF Toolbox' classes to your REST Framework settings, where appropriate.
  - Subclass DRF Toolbox' model serializer and model viewset.

CHAPTER 3

Getting Help

If you think you've found a bug in drf-toolbox itself, please post an issue on the Issue Tracker.

For usage help, you're free to e-mail the authors, who will provide help (on a best effort basis) if possible.

License

New BSD.

CHAPTER 5

Index

5.1 Nested Relationships

drf-toolbox adds robust support for nested relationships between models, both within serialization and within URL registration.

A **nested relationships** between models is permissible when models have a foreign key (one to many). Relationships can be displayed in either direction, and can be registered to nested URLs from parent to child only.

5.1.1 Displaying Nested Relationships

In order to setup display of full nested relationships, you will need to use the model serializer provided by DRF Toolbox as your default model serializer.

To do this, add it to your Django REST Framework settings:

```
REST_FRAMEWORK = {
    'DEFAULT_MODEL_SERIALIZER_CLASS': 'drf_toolbox.serializers.ModelSerializer',
}
```

Note: Remember that setting the DRF Toolbox `ModelSerializer` to your default serializer is not, in of itself, sufficient. You must also use it as the superclass for custom serializers you define.

DRF Toolbox's default model serializer will display included nested relationships as dictionaries, with the parent or child model shown in its entirety with default serialization (if no serializer is explicitly specified).

Posit two models:

```
from django.db import models

class Parent(models.Model):
    label = models.CharField(max_length=100)
    value = models.IntegerField()

class Child(models.Model):
    parent = models.ForeignKey(Parent)
    name = models.CharField(max_length=50)
```

The default Django REST Framework serialization on the child model would look something like this:

```
{
    "id": 24,
    "parent": 42,
    "name": "foo bar baz"
}
```

Using the serializer provided by DRF Toolbox, the parent value is expanded to a full model representation:

```
{
    "id": 24,
    "parent": {
        "id": 42,
        "label": "bacon",
        "value": "is yummy"
    },
    "name": "foo bar baz"
}
```

5.1.2 Controlling Fields

Django REST Framework's base model serializer implementation allows you to manually set fields to include or exclude using the serializer's `Meta` class:

```
from rest_framework import serializers

class ChildSerializer(serializers.ModelSerializer):
    class Meta:
        model = Child
        fields = ('id', 'parent', 'name')
```

The DRF Toolbox implementation maintains this, but additionally allows you to specify the fields on the related model. Doing so requires using a dictionary instead of a tuple, as follows:

```
from drf_toolbox import serializers

class ChildSerializer(serializers.ModelSerializer):
    class Meta:
        model = Child
        fields = {
            'self': ('id', 'parent', 'name'),
            'parent': ('id', 'label'),
        }
```

This will cause a child representation that looks like:

```
{
    "id": 24,
    "parent": {
        "id": 42,
        "label": "bacon",
    },
    "name": "foo bar baz"
}
```

The same syntax applies to the *exclude* option.

Note: If using the dictionary syntax for `fields` or `exclude`, and you want to specify the fields on the "main" model, use the string `'self'`.

5.1.3 Routing Nested Relationships

DRF Toolbox also adds the ability to understand nested relationships in *routing*, not just in display.

By default, Django REST Framework supports URL patterns such as:

```
^/parent/$
^/parent/(?P<pk>[\d]+)/$
^/child/$
^/child/(?P<pk>[\d]+)/$
```

DRF Toolbox adds support for nested URLs:

```
^/parent/(?P<pk>)/child/$
^/parent/(?P<parent__pk>[\d]+)/child/(?P<pk>[\d]+)/$
```

In order to get this functionality, use the DRF Toolbox router, which is provided in `drf_toolbox.routers.Router`. Registration of a nested view is done using a / separator, and must come only after the parent view has been registered:

```python
from drf_toolbox import routers
from myapp import views

router = routers.Router()
router.register(r'parent', views.ParentViewSet)
router.register(r'parent/child', views.ChildViewSet)
```

Note: Additionally, the DRF Toolbox router adds one other piece of functionality, the ability to specify viewsets as strings, rather than references. The following code, therefore, is indentical:

```python
from drf_toolbox import routers

router = routers.Router()
router.register(r'parent', 'myapp.views.ParentViewSet')
router.register(r'parent/child', 'myapp.views.ChildViewSet')
```

This ability does remove the ability to use strings in URI fragments the normal way. If you need this, use the stock DRF router for those views.

5.1.4 API Endpoint Fields

The DRF Toolbox includes `APIEndpointField` and `APIEndpointsField` classes for displaying API end-points. They will use the full URL, with the domain, if available, for the purpose of linking.

On the primary model being shown, the DRF Toolbox serializer will use the `APIEndpointsField`, which will display a dictionary of API endpoints, including any related endpoints registered as children to the router, or to the viewset with decorators like `@link` or `@action`.

On related models being shown in the same view, the `APIEndpointField` is used instead, which shows only the API endpoint URL for that object.

An `APIEndpointsField` is automatically available to serializers that serialize models which define a `get_absolute_url` method.

Note: Because of the additional complexity introduced by nesting viewsets, defining a `get_absolute_url` method is required; Django REST Framework's former intuition of URLs is no longer enabled.

5.2 Fields from django-pgfields

DRF Toolbox provides out of the box support for fields from django-pgfields if and only if django-pgfields is installed. No setup is required to get this functionality; it is auto-enabled if django-pgfields is present, and dormant if django-pgfields is absent.

Note: While the other three fields come with standard form elements, `CompositeField` classes do not have any straightforward form rendering.

As a result, when a `CompositeField` is used, the form data parser is *removed* from the viewset unless it was manually specified.

5.3 JSON Renderer

DRF Toolbox ships with a custom JSON renderer, available within the `drf_toolbox.renderers.json` module.

Currently, the only thing this offers is serialization of `date` or `datetime` objects to UNIX timestamps.

To enable this, use these classes instead of the stock Django REST Framework versions in your `DEFAULT_RENDERER_CLASSES` setting.