django-rest-framework-tricks Documentation

Release 0.2.11

Artur Barseghyan <artur.barseghyan@gmail.com>

Jan 22, 2020

Contents

Collection of various tricks for Django REST framework.

CHAPTER 1

Prerequisites

- Django 1.8, 1.9, 1.10, 1.11, 2.0, 2.1, 2.2 and 3.0.
- Python 2.7, 3.5, 3.6, 3.7, 3.8

CHAPTER 2

Dependencies

- djangorestframework: Written with 3.6.3, tested with >=3.5.0,<=3.11.0. May work on earlier versions, although not guaranteed.

CHAPTER 3

Installation

(1) Install latest stable version from PyPI:

```
pip install django-rest-framework-tricks
```

or latest stable version from GitHub:

```
pip install https://github.com/barseghyanartur/django-rest-framework-tricks/
↪archive/stable.tar.gz
```

or latest stable version from BitBucket:

```
pip install https://bitbucket.org/barseghyanartur/django-rest-framework-tricks/
↪get/stable.tar.gz
```

(2) Add `rest_framework` and `rest_framework_tricks` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # ...
    # REST framework
    'rest_framework',

    # REST framework tricks (this package)
    'rest_framework_tricks',

    # ...
)
```

CHAPTER 4

Documentation

Documentation is available on Read the Docs.

CHAPTER 5

Main features and highlights

- *Nested serializers*: Nested serializers for non-relational fields.
- *Ordering filter*: Developer friendly names for ordering options (for instance, for related field names).

CHAPTER 6

Usage examples

6.1 Nested serializers

Nested serializers for non-relational fields.

Our imaginary `Book` model consists of the following (non-relational) Django model fields:

- `title`: `CharField`
- `description`: `TextField`
- `summary`: `TextField`
- `publication_date`: `DateTimeField`
- `state`: `CharField` (with choices)
- `isbn`: `CharField`
- `price`: `DecimalField`
- `pages`: `IntegerField`
- `stock_count`: `IntegerField`

In our REST API, we want to split the Book serializer into parts using nested serializers to have the following structure:

```
{
    "id": "",
    "title": "",
    "description": "",
    "summary": "",
    "publishing_information": {
        "publication_date": "",
        "isbn": "",
        "pages": ""
    },
    "stock_information": {
```

(continues on next page)

(continued from previous page)

```
        "stock_count": "",
        "price": "",
        "state": ""
    }
}
```

6.1.1 Sample model

The only variation from standard implementation here is that we declare two `NestedProxyField` fields on the `Book` model level for to be used in `BookSerializer` serializer.

Note, that the change does not cause model change (no migrations or whatsoever).

6.1.1.1 Required imports

```
from django.db import models

from rest_framework_tricks.models.fields import NestedProxyField
```

6.1.1.2 Model definition

```
BOOK_PUBLISHING_STATUS_PUBLISHED = 'published'
BOOK_PUBLISHING_STATUS_NOT_PUBLISHED = 'not_published'
BOOK_PUBLISHING_STATUS_IN_PROGRESS = 'in_progress'
BOOK_PUBLISHING_STATUS_CHOICES = (
    (BOOK_PUBLISHING_STATUS_PUBLISHED, "Published"),
    (BOOK_PUBLISHING_STATUS_NOT_PUBLISHED, "Not published"),
    (BOOK_PUBLISHING_STATUS_IN_PROGRESS, "In progress"),
)
BOOK_PUBLISHING_STATUS_DEFAULT = BOOK_PUBLISHING_STATUS_PUBLISHED


class Book(models.Model):
    """Book."""

    title = models.CharField(max_length=100)
    description = models.TextField(null=True, blank=True)
    summary = models.TextField(null=True, blank=True)
    publication_date = models.DateField()
    state = models.CharField(max_length=100,
                             choices=BOOK_PUBLISHING_STATUS_CHOICES,
                             default=BOOK_PUBLISHING_STATUS_DEFAULT)
    isbn = models.CharField(max_length=100, unique=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    pages = models.PositiveIntegerField(default=200)
    stock_count = models.PositiveIntegerField(default=30)

    # List the fields for `PublishingInformationSerializer` nested
    # serializer. This does not cause a model change.
    publishing_information = NestedProxyField(
        'publication_date',
        'isbn',
```

(continues on next page)

(continued from previous page)

```
        'pages',
    )

    # List the fields for `StockInformationSerializer` nested serializer.
    # This does not cause a model change.
    stock_information = NestedProxyField(
        'stock_count',
        'price',
        'state',
    )

    class Meta(object):
        """Meta options."""

        ordering = ["isbn"]

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

6.1.2 Sample serializers

At first, we add `nested_proxy_field` property to the `Meta` class definitions of `PublishingInformationSerializer` and `StockInformationSerializer` nested serializers.

Then we define our (main) `BookSerializer` class, which is going to be used as a `serializer_class` of the `BookViewSet`. We inherit the `BookSerializer` from `rest_framework_tricks.serializers.HyperlinkedModelSerializer` instead of the one of the Django REST framework. There's also a `rest_framework_tricks.serializers.ModelSerializer` available.

6.1.2.1 Required imports

```
from rest_framework import serializers
from rest_framework_tricks.serializers import (
    HyperlinkedModelSerializer,
)

from .models import Book
```

6.1.2.2 Defining the serializers

Note: If you get validation errors about null-values, add `allow_null=True` next to the `required=False` for serializer field definitions.

Nested serializer

```
class PublishingInformationSerializer(serializers.ModelSerializer):
    """Publishing information serializer."""

    publication_date = serializers.DateField(required=False)
    isbn = serializers.CharField(required=False)
```

(continues on next page)

(continued from previous page)

```
    pages = serializers.IntegerField(required=False)

    class Meta(object):
        """Meta options."""

        model = Book
        fields = (
            'publication_date',
            'isbn',
            'pages',
        )
        # Note, that this should be set to True to identify that
        # this serializer is going to be used as `NestedProxyField`.
        nested_proxy_field = True
```

Nested serializer

```
class StockInformationSerializer(serializers.ModelSerializer):
    """Stock information serializer."""

    class Meta(object):
        """Meta options."""

        model = Book
        fields = (
            'stock_count',
            'price',
            'state',
        )
        # Note, that this should be set to True to identify that
        # this serializer is going to be used as `NestedProxyField`.
        nested_proxy_field = True
```

Main serializer to be used in the ViewSet

```
# Note, that we are importing the ``HyperlinkedModelSerializer`` from
# the `rest_framework_tricks.serializers`. Names of the serializers
# should match the names of model properties set with ``NestedProxyField``
# fields.
class BookSerializer(HyperlinkedModelSerializer):
    """Book serializer."""

    publishing_information = PublishingInformationSerializer(required=False)
    stock_information = StockInformationSerializer(required=False)

    class Meta(object):
        """Meta options."""

        model = Book
        fields = (
            'url',
            'id',
            'title',
            'description',
            'summary',
            'publishing_information',
            'stock_information',
```

(continues on next page)

(continued from previous page)

```
        )
```

6.1.3 Sample ViewSet

Absolutely no variations from standard implementation here.

6.1.3.1 Required imports

```
from rest_framework.viewsets import ModelViewSet
from rest_framework.permissions import AllowAny

from .models import Book
from .serializers import BookSerializer
```

6.1.3.2 ViewSet definition

```
class BookViewSet(ModelViewSet):
    """Book ViewSet."""

    queryset = Book.objects.all()
    serializer_class = BookSerializer
    permission_classes = [AllowAny]
```

6.1.3.3 Sample OPTIONS call

```
OPTIONS /books/api/books/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```

```
{
    "name": "Book List",
    "description": "Book ViewSet.",
    "renders": [
        "application/json",
        "text/html"
    ],
    "parses": [
        "application/json",
        "application/x-www-form-urlencoded",
        "multipart/form-data"
    ],
    "actions": {
        "POST": {
            "id": {
                "type": "integer",
                "required": false,
                "read_only": true,
```

(continues on next page)

(continued from previous page)


```
            "label": "ID"
        },
        "title": {
            "type": "string",
            "required": true,
            "read_only": false,
            "label": "Title",
            "max_length": 100
        },
        "description": {
            "type": "string",
            "required": false,
            "read_only": false,
            "label": "Description"
        },
        "summary": {
            "type": "string",
            "required": false,
            "read_only": false,
            "label": "Summary"
        },
        "publishing_information": {
            "type": "nested object",
            "required": false,
            "read_only": false,
            "label": "Publishing information",
            "children": {
                "publication_date": {
                    "type": "date",
                    "required": false,
                    "read_only": false,
                    "label": "Publication date"
                },
                "isbn": {
                    "type": "string",
                    "required": false,
                    "read_only": false,
                    "label": "Isbn"
                },
                "pages": {
                    "type": "integer",
                    "required": false,
                    "read_only": false,
                    "label": "Pages"
                }
            }
        },
        "stock_information": {
            "type": "nested object",
            "required": false,
            "read_only": false,
            "label": "Stock information",
            "children": {
                "stock_count": {
                    "type": "integer",
                    "required": false,
                    "read_only": false,
```


(continues on next page)

(continued from previous page)


```
                        "label": "Stock count"
                    },
                    "price": {
                        "type": "decimal",
                        "required": true,
                        "read_only": false,
                        "label": "Price"
                    },
                    "state": {
                        "type": "choice",
                        "required": false,
                        "read_only": false,
                        "label": "State",
                        "choices": [
                            {
                                "value": "published",
                                "display_name": "Published"
                            },
                            {
                                "value": "not_published",
                                "display_name": "Not published"
                            },
                            {
                                "value": "in_progress",
                                "display_name": "In progress"
                            }
                        ]
                    }
                }
            }
        }
    }
}
```


6.1.4 Unlimited nesting depth

Unlimited nesting depth is supported.

Our imaginary `Author` model could consist of the following (non-relational) Django model fields:

- `salutation`: `CharField`
- `name`: `CharField`
- `email`: `EmailField`
- `birth_date`: `DateField`
- `biography`: `TextField`
- `phone_number`: `CharField`
- `website`: `URLField`
- `company`: `CharField`
- `company_phone_number`: `CharField`
- `company_email`: `EmailField`

- `company_website`: `URLField`

In our REST API, we could split the Author serializer into parts using nested serializers to have the following structure:

```
{
    "id": "",
    "salutation": "",
    "name": "",
    "birth_date": "",
    "biography": "",
    "contact_information": {
        "personal_contact_information": {
            "email": "",
            "phone_number": "",
            "website": ""
        },
        "business_contact_information": {
            "company": "",
            "company_email": "",
            "company_phone_number": "",
            "company_website": ""
        }
    }
}
```

Our model would have to be defined as follows (see `Advanced usage examples` for complete model definition):

```
class Author(models.Model):
    """Author."""

    # ...

    # List the fields for `PersonalContactInformationSerializer` nested
    # serializer. This does not cause a model change.
    personal_contact_information = NestedProxyField(
        'email',
        'phone_number',
        'website',
    )

    # List the fields for `BusinessContactInformationSerializer` nested
    # serializer. This does not cause a model change.
    business_contact_information = NestedProxyField(
        'company',
        'company_email',
        'company_phone_number',
        'company_website',
    )

    # List the fields for `ContactInformationSerializer` nested
    # serializer. This does not cause a model change.
    contact_information = NestedProxyField(
        'personal_contact_information',
        'business_contact_information',
    )

    # ...
```

See the Advanced usage examples for complete example.

6.2 Ordering filter

Developer friendly names for ordering options (for instance, for related field names) for making better APIs.

6.2.1 Sample model

Absolutely no variations from standard implementation here.

6.2.1.1 Required imports

```
from django.db import models
```

6.2.1.2 Model definition

```
class Profile(models.Model):
    """Profile."""

    user = models.ForeignKey('auth.User')
    biography = models.TextField()
    hobbies = models.TextField()
```

6.2.2 Sample serializer

Absolutely no variations from standard implementation here.

6.2.2.1 Required imports

```
from rest_framework import serializers

from .models import Profile
```

6.2.2.2 Defining the serializers

```
class ProfileSerializer(serializers.ModelSerializer):
    """Profile serializer."""

    username = serializers.CharField(source='user.username', read_only=True)
    full_name = serializers.SerializerMethodField()
    email = serializers.CharField(source='user.email', read_only=True)

    class Meta(object):

    model = Profile
    fields = (
        'id',
        'username',
        'full_name',
```

(continues on next page)

(continued from previous page)

```
        'email',
        'biography',
        'hobbies',
    )

    def get_full_name(self, obj):
        return obj.user.get_full_name()
```

6.2.3 Sample ViewSet

The only variation from standard implementation here is that we use `rest_frameworks_tricks.filters.OrderingFilter` instead of `rest_framework.filters.OrderingFilter`.

6.2.3.1 Required imports

```
from rest_framework.viewsets import ModelViewSet
from rest_framework.permissions import AllowAny
from rest_framework_tricks.filters import OrderingFilter

from .models import Profile
from .serializers import ProfileSerializer
```

6.2.3.2 ViewSet definition

```
class ProfileViewSet(ModelViewSet):
    """Profile ViewSet."""

    queryset = Profile.objects.all()
    serializer_class = ProfileSerializer
    permission_classes = [AllowAny]
    filter_backends = (OrderingFilter,)
    ordering_fields = {
        'id': 'id',
        'username': 'user__username',
        'email': 'user__email',
        'full_name': ['user__first_name', 'user__last_name']
    }
    ordering = ('id',)
```

6.2.3.3 Sample GET calls

Note, that our ordering options are now equal to the field names in the serializer (JSON response). API becomes easier to use/understand that way.

```
GET /api/profile/?ordering=email
GET /api/profile/?ordering=-username
GET /api/profile/?ordering=full_name
GET /api/profile/?ordering=-full_name
```

CHAPTER 7

Demo

7.1 Run demo locally

In order to be able to quickly evaluate the `django-rest-framework-tricks`, a demo app (with a quick installer) has been created (works on Ubuntu/Debian, may work on other Linux systems as well, although not guaranteed). Follow the instructions below to have the demo running within a minute.

Grab and run the latest `rest_framework_tricks_demo_installer.sh` demo installer:

```
wget -O - https://raw.github.com/barseghyanartur/django-rest-framework-tricks/master/
↪examples/rest_framework_tricks_demo_installer.sh | bash
```

Open your browser and test the app.

```
http://127.0.0.1:8001/books/api/
```

CHAPTER 8

Testing

Project is covered with tests.

To test with all supported Python/Django versions type:

```
tox
```

To test against specific environment, type:

```
tox -e py38-django30
```

To test just your working environment type:

```
./runtests.py
```

To run a single test in your working environment type:

```
./runtests.py src/rest_framework_tricks/tests/test_nested_proxy_field.py
```

Or:

```
./manage.py test rest_framework_tricks.tests.test_nested_proxy_field
```

It's assumed that you have all the requirements installed. If not, first install the test requirements:

```
pip install -r examples/requirements/test.txt
```

CHAPTER 9

Writing documentation

Keep the following hierarchy.

```
=====
title
=====

header
======

sub-header
----------

sub-sub-header
~~~~~~~~~~~~~~

sub-sub-sub-header
^^^^^^^^^^^^^^^^^^

sub-sub-sub-sub-header
++++++++++++++++++++++

sub-sub-sub-sub-sub-header
**************************
```

CHAPTER 10

License

GPL-2.0-only OR LGPL-2.1-or-later

CHAPTER 11

Support

For any issues contact me at the e-mail given in the *Author* section.

CHAPTER 12

Author

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER 13

Project documentation

Contents:

Table of Contents

13.1 Advanced usage examples

Contents:

Table of Contents

13.1.1 Nested serializers

13.1.1.1 Unlimited nesting depth

Our imaginary `Author` model consist of the following (non-relational) Django model fields:

- `salutation`: `CharField`
- `name`: `CharField`
- `email`: `EmailField`
- `birth_date`: `DateField`
- `biography`: `TextField`
- `phone_number`: `CharField`
- `website`: `URLField`
- `company`: `CharField`
- `company_phone_number`: `CharField`
- `company_email`: `EmailField`
- `company_website`: `URLField`

In our REST API, we split the Author serializer into parts using nested serializers to have the following structure:

```
{
    "id": "",
    "salutation": "",
    "name": "",
    "birth_date": "",
    "biography": "",
    "contact_information": {
        "personal_contact_information": {
            "email": "",
```

(continues on next page)

(continued from previous page)

```
            "phone_number": "",
            "website": ""
        },
        "business_contact_information": {
            "company": "",
            "company_email": "",
            "company_phone_number": "",
            "company_website": ""
        }
    }
}
```

13.1.1.1.1 Sample models

The only variation from standard implementation here is that we declare two `NestedProxyField` fields on the `Author` model level for to be used in `AuthorSerializer` serializer.

Note, that the change does not cause model change (no migrations or whatsoever).

13.1.1.1.1.1 Required imports

```
from django.db import models

from rest_framework_tricks.models.fields import NestedProxyField
```

13.1.1.1.1.2 Model definition

```
class Author(models.Model):
    """Author."""

    salutation = models.CharField(max_length=10)
    name = models.CharField(max_length=200)
    email = models.EmailField()
    birth_date = models.DateField(null=True, blank=True)
    biography = models.TextField(null=True, blank=True)
    phone_number = models.CharField(max_length=200, null=True, blank=True)
    website = models.URLField(null=True, blank=True)
    company = models.CharField(max_length=200, null=True, blank=True)
    company_phone_number = models.CharField(max_length=200,
                                            null=True,
                                            blank=True)
    company_email = models.EmailField(null=True, blank=True)
    company_website = models.URLField(null=True, blank=True)

    # List the fields for `PersonalContactInformationSerializer` nested
    # serializer. This does not cause a model change.
    personal_contact_information = NestedProxyField(
        'email',
        'phone_number',
        'website',
    )
```

(continues on next page)

(continued from previous page)

```
    # List the fields for `BusinessContactInformationSerializer` nested
    # serializer. This does not cause a model change.
    business_contact_information = NestedProxyField(
        'company',
        'company_email',
        'company_phone_number',
        'company_website',
    )

    # List the fields for `ContactInformationSerializer` nested
    # serializer. This does not cause a model change.
    contact_information = NestedProxyField(
        'personal_contact_information',
        'business_contact_information',
    )

    class Meta(object):
        """Meta options."""

        ordering = ["id"]

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

Alternatively, you could rewrite the `contact_information` definition as follows (although at the moment it's not the recommended approach):

```
# ...
# List the fields for `ContactInformationSerializer` nested
# serializer. This does not cause a model change.
contact_information = NestedProxyField(
    {
        'personal_contact_information': (
            'email',
            'phone_number',
            'website',
        )
    },
    {
        'business_contact_information': (
            'company',
            'company_email',
            'company_phone_number',
            'company_website',
        )
    },
)
# ...
```

13.1.1.1.2 Sample serializers

At first, we add `nested_proxy_field` property to the `Meta` class definitions of `PersonalContactInformationSerializer`, `BusinessContactInformationSerializer` and `ContactInformationSerializer` nested serializers.

Then we define our (main) `AuthorSerializer` class, which is going to be used a `serializer_class` of the `AuthorViewSet`. We inherit the `AuthorSerializer` from `rest_framework_tricks.serializers.HyperlinkedModelSerializer` instead of the one of the Django REST framework. There's also a `rest_framework_tricks.serializers.ModelSerializer` available.

13.1.1.1.2.1 Required imports

```
from rest_framework import serializers
from rest_framework_tricks.serializers import (
    HyperlinkedModelSerializer,
    ModelSerializer,
)
```

13.1.1.1.2.2 Serializer definition

Note: If you get validation errors about null-values, add `allow_null=True` next to the `required=False` for serializer field definitions.

Nested serializer for 'ContactInformationSerializer' nested serializer

```
class PersonalContactInformationSerializer(serializers.ModelSerializer):
    """Personal contact information serializer."""

    class Meta(object):
        """Meta options."""

        model = Author
        fields = (
            'email',
            'phone_number',
            'website',
        )
        nested_proxy_field = True
```

Nested serializer for 'ContactInformationSerializer' nested serializer

```
class BusinessContactInformationSerializer(serializers.ModelSerializer):
    """Business contact information serializer."""

    class Meta(object):
        """Meta options."""

        model = Author
        fields = (
            'company',
            'company_email',
            'company_phone_number',
            'company_website',
        )
        nested_proxy_field = True
```

Nested serializer for 'AuthorSerializer' (main) serializer

```
class ContactInformationSerializer(serializers.ModelSerializer):
    """Contact information serializer."""

    personal_contact_information = PersonalContactInformationSerializer(
        required=False
    )
    business_contact_information = BusinessContactInformationSerializer(
        required=False
    )

    class Meta(object):
        """Meta options."""

        model = Author
        fields = (
            'personal_contact_information',
            'business_contact_information',
        )
        nested_proxy_field = True
```

Main serializer to be used in the ViewSet

```
class AuthorSerializer(ModelSerializer):
    """Author serializer."""

    contact_information = ContactInformationSerializer(required=False)

    class Meta(object):
        """Meta options."""

        model = Author
        fields = (
            'id',
            'salutation',
            'name',
            'birth_date',
            'biography',
            'contact_information',
        )
```

13.1.1.1.2.3 If you can't make use of *rest_framework_tricks* serializers

If somehow you can't make use of the `rest_framework_tricks.serializers.ModelSerializer` or `rest_framework_tricks.serializers.HyperlinkedModelSerializer` serializers, there are handy functions to help you to make your serializer to work with `NestedProxyField`.

See the following example:

13.1.1.1.2.4 Required imports

```
from rest_framework import serializers
from rest_framework_tricks.serializers.nested_proxy import (
    extract_nested_serializers,
```

(continues on next page)

(continued from previous page)

```
    set_instance_values,
)
```

13.1.1.1.2.5 Serializer definition

```
class BookSerializer(serializers.ModelSerializer):
    """BookSerializer."""

    # ...

    def create(self, validated_data):
        """Create.

        :param validated_data:
        :return:
        """
        # Collect information on nested serializers
        __nested_serializers, __nested_serializers_data = \
            extract_nested_serializers(
                self,
                validated_data,
            )

        # Create instance, but don't save it yet
        instance = self.Meta.model(**validated_data)

        # Assign fields to the `instance` one by one
        set_instance_values(
            __nested_serializers,
            __nested_serializers_data,
            instance
        )

        # Save the instance and return
        instance.save()
        return instance

    def update(self, instance, validated_data):
        """Update.

        :param instance:
        :param validated_data:
        :return:
        """
        # Collect information on nested serializers
        __nested_serializers, __nested_serializers_data = \
            extract_nested_serializers(
                self,
                validated_data,
            )

        # Update the instance
        instance = super(ModelSerializer, self).update(
            instance,
```

(continues on next page)

(continued from previous page)

```
            validated_data
        )

        # Assign fields to the `instance` one by one
        set_instance_values(
            __nested_serializers,
            __nested_serializers_data,
            instance
        )

        # Save the instance and return
        instance.save()
        return instance
```

13.1.1.1.3 Sample ViewSet

Absolutely no variations from standard implementation here.

13.1.1.1.3.1 Required imports

```
from rest_framework.viewsets import ModelViewSet
from rest_framework.permissions import AllowAny

from .models import Author
from .serializers import AuthorSerializer
```

13.1.1.1.3.2 ViewSet definition

```
class AuthorViewSet(ModelViewSet):
    """Author ViewSet."""

    queryset = Author.objects.all()
    serializer_class = AuthorSerializer
    permission_classes = [AllowAny]
```

13.1.1.1.4 Sample URLs/router definition

Absolutely no variations from standard implementation here.

13.1.1.1.4.1 Required imports

```
from django.conf.urls import url, include

from rest_framework_extensions.routers import ExtendedDefaultRouter

from .viewsets import AuthorViewSet
```

13.1.1.1.4.2 ViewSet definition

```
router = ExtendedDefaultRouter()
authors = router.register(r'authors',
                          AuthorViewSet,
                          base_name='author')

urlpatterns = [
    url(r'^api/', include(router.urls)),
]
```

13.1.1.1.5 Sample OPTIONS call

```
OPTIONS /books/api/authors/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```

```
{
    "name": "Author List",
    "description": "Author ViewSet.",
    "renders": [
        "application/json",
        "text/html"
    ],
    "parses": [
        "application/json",
        "application/x-www-form-urlencoded",
        "multipart/form-data"
    ],
    "actions": {
        "POST": {
            "id": {
                "type": "integer",
                "required": false,
                "read_only": true,
                "label": "ID"
            },
            "salutation": {
                "type": "string",
                "required": true,
                "read_only": false,
                "label": "Salutation",
                "max_length": 10
            },
            "name": {
                "type": "string",
                "required": true,
                "read_only": false,
                "label": "Name",
                "max_length": 200
            },
            "birth_date": {
                "type": "date",
```

(continues on next page)

(continued from previous page)

```
            "required": false,
            "read_only": false,
            "label": "Birth date"
        },
        "biography": {
            "type": "string",
            "required": false,
            "read_only": false,
            "label": "Biography"
        },
        "contact_information": {
            "type": "nested object",
            "required": false,
            "read_only": false,
            "label": "Contact information",
            "children": {
                "personal_contact_information": {
                    "type": "nested object",
                    "required": false,
                    "read_only": false,
                    "label": "Personal contact information",
                    "children": {
                        "email": {
                            "type": "email",
                            "required": true,
                            "read_only": false,
                            "label": "Email",
                            "max_length": 254
                        },
                        "phone_number": {
                            "type": "string",
                            "required": false,
                            "read_only": false,
                            "label": "Phone number",
                            "max_length": 200
                        },
                        "website": {
                            "type": "url",
                            "required": false,
                            "read_only": false,
                            "label": "Website",
                            "max_length": 200
                        }
                    }
                },
                "business_contact_information": {
                    "type": "nested object",
                    "required": false,
                    "read_only": false,
                    "label": "Business contact information",
                    "children": {
                        "company": {
                            "type": "string",
                            "required": false,
                            "read_only": false,
                            "label": "Company",
                            "max_length": 200
```

(continues on next page)

(continued from previous page)


```
                            },
                            "company_email": {
                                "type": "email",
                                "required": false,
                                "read_only": false,
                                "label": "Company email",
                                "max_length": 254
                            },
                            "company_phone_number": {
                                "type": "string",
                                "required": false,
                                "read_only": false,
                                "label": "Company phone number",
                                "max_length": 200
                            },
                            "company_website": {
                                "type": "url",
                                "required": false,
                                "read_only": false,
                                "label": "Company website",
                                "max_length": 200
                            }
                        }
                    }
                }
            }
        }
    }
}
```


13.1.1.1.6 Sample POST call

```
POST /books/api/authors/
HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```


```
{
    "salutation": "At eve",
    "name": "Shana Rodriquez",
    "birth_date": "2016-04-05",
    "biography": "Commodi facere voluptate ipsum veniam maxime obcaecati",
    "contact_information": {
        "personal_contact_information": {
            "email": "somasesu@yahoo.com",
            "phone_number": "+386-36-3715907",
            "website": "http://www.xazyvufugasi.biz"
        },
        "business_contact_information": {
            "company": "Hopkins and Mccoy Co",
            "company_email": "vevuciqa@yahoo.com",
            "company_phone_number": "+386-35-5689443",
            "company_website": "http://www.xifyhefiqom.com.au"
```


(continues on next page)

(continued from previous page)

```
            }
        }
}
```

13.2 Release history and notes

Sequence based identifiers are used for versioning (schema follows below):

```
major.minor[.revision]
```

- It's always safe to upgrade within the same minor version (for example, from 0.3 to 0.3.4).
- Minor version changes might be backwards incompatible. Read the release notes carefully before upgrading (for example, when upgrading from 0.3.4 to 0.4).
- All backwards incompatible changes are mentioned in this document.

13.2.1 0.2.11

2019-12-27

- Tested against Django 3.0.
- Tested against Python 3.8.
- Tested against Django REST Framework 3.11.

13.2.2 0.2.10

2019-04-12

- Tested against Django 2.1 and Django 2.2.
- Tested against Python 3.7.
- Dropping support for Python 3.4.
- Upgrade test suite.
- Temporary remove PyPy from tox (because of failing tests).

13.2.3 0.2.9

2018-02-03

- Make it possible to order by two (or more fields) at once, using the `OrderingFilter`.

13.2.4 0.2.8

2018-01-31

- Fixes in docs.

13.2.5 0.2.7

2018-01-28

- Fixes in docs.

13.2.6 0.2.6

2018-01-28

- Added `OrderingFilter`, which makes it possible to specify mapping (ordering option -> ORM field) for making more developer friendly ordering options in the API. An example of such could be a `Profile` model with `ForeignKey` relation to `User` model. In case if we want to order by `email` field in the `ProfileViewSet`, instead of ordering on `user__email` we could order just on `email`.

13.2.7 0.2.5

2017-12-30

- Update example project (and the tests that are dependant on the example project) to work with Django 2.0.

13.2.8 0.2.4

2017-07-14

- Fix issue #1 with non-required nested serializer fields.

13.2.9 0.2.3

2017-07-13

- More tests.
- Made tests DRY.

13.2.10 0.2.2

2017-07-04

- Documentation improvements.
- Tested against various Django REST framework versions (>=3.5.0,<=3.6.3).

13.2.11 0.2.1

2017-07-04

- Minor fixes.
- Documentation improvements.

13.2.12 0.2

2017-07-02

- Handle unlimited nesting depth for nested serializers of non-relational fields.
- Documentation improvements.

13.2.13 0.1.8

2017-07-01

- Initial beta release.

13.3 Package

Contents:

Table of Contents

13.3.1 rest_framework_tricks package

13.3.1.1 Subpackages

13.3.1.1.1 rest_framework_tricks.filters package

####### 13.3.1.1.1.1 Submodules

13.3.1.1.1.2 rest_framework_tricks.filters.ordering module

Ordering filter.

class rest_framework_tricks.filters.ordering.**OrderingFilter**

Bases: rest_framework.filters.OrderingFilter

Ordering filter improved.

Example:

```
>>> from rest_framework_tricks.filters import OrderingFilter
>>>
>>> class BooksViewSet(mixins.RetrieveModelMixin,
>>>                    mixins.ListModelMixin,
>>>                    viewsets.GenericViewSet):
>>>
>>>     serializer_class = BookSerializer
>>>     filter_backends = (
>>>         OrderingFilter,
>>>     )
```

(continues on next page)

(continued from previous page)

```
>>>     ordering_fields = {
>>>         'email': 'user__email',
>>>         'full_name': 'user__first_name',
>>>         'last_login': 'user__last_login',
>>>         'is_active': 'user__is_active',
>>>     }
```

Then it can be used in a view set as follows:

GET /books/api/proxy-books/?ordering=email

get_ordering(*request*, *queryset*, *view*)
Get ordering.

Important: list returned in this method is used directly in the filter_queryset method like:

```
>>> queryset.order_by(*ordering)
```

Ordering is set by a comma delimited ?ordering=... query parameter.

The *ordering* query parameter can be overridden by setting the *ordering_param* value on the OrderingFilter or by specifying an *ORDERING_PARAM* value in the API settings.

get_valid_fields(*queryset*, *view*, *context={}*)
Done.

Parameters

- **queryset** –
- **view** –
- **context** –

Returns

13.3.1.1.1.3 Module contents

Filters.

class rest_framework_tricks.filters.**OrderingFilter**
Bases: rest_framework.filters.OrderingFilter

Ordering filter improved.

Example:

```
>>> from rest_framework_tricks.filters import OrderingFilter
>>>
>>> class BooksViewSet(mixins.RetrieveModelMixin,
>>>                    mixins.ListModelMixin,
>>>                    viewsets.GenericViewSet):
>>>
>>>     serializer_class = BookSerializer
>>>     filter_backends = (
>>>         OrderingFilter,
>>>     )
>>>     ordering_fields = {
>>>         'email': 'user__email',
```

(continues on next page)

(continued from previous page)

```
>>>         'full_name': 'user__first_name',
>>>         'last_login': 'user__last_login',
>>>         'is_active': 'user__is_active',
>>>     }
```

Then it can be used in a view set as follows:

GET /books/api/proxy-books/?ordering=email

get_ordering(*request*, *queryset*, *view*)
Get ordering.

Important: list returned in this method is used directly in the filter_queryset method like:

```
>>> queryset.order_by(*ordering)
```

Ordering is set by a comma delimited ?ordering=... query parameter.

The *ordering* query parameter can be overridden by setting the *ordering_param* value on the OrderingFilter or by specifying an *ORDERING_PARAM* value in the API settings.

get_valid_fields(*queryset*, *view*, *context={}*)
Done.

Parameters

- **queryset** –
- **view** –
- **context** –

Returns

13.3.1.1.2 rest_framework_tricks.models package

13.3.1.1.2.1 Subpackages

13.3.1.1.2.2 rest_framework_tricks.models.fields package

13.3.1.1.2.3 Submodules

13.3.1.1.2.4 rest_framework_tricks.models.fields.nested_proxy module

Nested proxy field.

rest_framework_tricks.models.fields.nested_proxy.**NestedProxyField**(**fields*, ***options*)

NestedProxyField field.

Example:

```
>>> from django.db import models
>>> from rest_framework_tricks.models.fields import NestedProxyField
>>> from .constants import BOOK_STATUS_CHOICES, BOOK_STATUS_DEFAULT
>>>
>>>
```

(continues on next page)

(continued from previous page)

```
>>> class Book(models.Model):
>>>
>>>     title = models.CharField(max_length=100)
>>>     description = models.TextField(null=True, blank=True)
>>>     summary = models.TextField(null=True, blank=True)
>>>     publication_date = models.DateField()
>>>     state = models.CharField(max_length=100,
>>>                              choices=BOOK_STATUS_CHOICES,
>>>                              default=BOOK_STATUS_DEFAULT)
>>>     isbn = models.CharField(max_length=100, unique=True)
>>>     price = models.DecimalField(max_digits=10, decimal_places=2)
>>>     pages = models.PositiveIntegerField(default=200)
>>>     stock_count = models.PositiveIntegerField(default=30)
>>>
>>>     # This does not cause a model change
>>>     publishing_information = NestedProxyField(
>>>         'publication_date',
>>>         'isbn',
>>>         'pages',
>>>     )
>>>
>>>     # This does not cause a model change
>>>     stock_information = NestedProxyField(
>>>         'stock_count',
>>>         'price',
>>>         'state',
>>>     )
>>>
>>>     class Meta(object):
>>>
>>>         ordering = ["isbn"]
>>>
>>>     def __str__(self):
>>>         return self.title
```

Nesting depth is unlimited, so the following would be possible as well.

Example:

```
>>> class Author(models.Model):
>>>
>>>     salutation = models.CharField(max_length=10)
>>>     name = models.CharField(max_length=200)
>>>     email = models.EmailField()
>>>     birth_date = models.DateField(null=True, blank=True)
>>>     biography = models.TextField(null=True, blank=True)
>>>     phone_number = models.CharField(max_length=200,
>>>                                     null=True,
>>>                                     blank=True)
>>>     website = models.URLField(null=True, blank=True)
>>>     company = models.CharField(max_length=200,
>>>                                null=True,
>>>                                blank=True)
>>>     company_phone_number = models.CharField(max_length=200,
>>>                                             null=True,
>>>                                             blank=True)
>>>     company_email = models.EmailField(null=True, blank=True)
```

(continues on next page)

(continued from previous page)

```
>>>     company_website = models.URLField(null=True, blank=True)
>>>
>>>     # This does not cause a model change
>>>     personal_contact_information = NestedProxyField(
>>>         'email',
>>>         'phone_number',
>>>         'website',
>>>     )
>>>
>>>     # This does not cause a model change
>>>     business_contact_information = NestedProxyField(
>>>         'company',
>>>         'company_email',
>>>         'company_phone_number',
>>>         'company_website',
>>>     )
>>>
>>>     # This does not cause a model change
>>>     contact_information = NestedProxyField(
>>>         'personal_contact_information',
>>>         'business_contact_information',
>>>     )
```

You could even do this (although the way it's written above is at the moment the preferred/recommended way of dealing with unlimited nesting depth.

```
>>>     # This does not cause a model change
>>>     contact_information = NestedProxyField(
>>>         {
>>>             'personal_contact_information': (
>>>                 'email',
>>>                 'phone_number',
>>>                 'website',
>>>             )
>>>         },
>>>         {
>>>             'business_contact_information': (
>>>                 'company',
>>>                 'company_email',
>>>                 'company_phone_number',
>>>                 'company_website',
>>>             )
>>>         },
>>>     )
```

13.3.1.1.2.5 Module contents

Fields.

rest_framework_tricks.models.fields.**NestedProxyField**(**fields*, ***options*)

NestedProxyField field.

Example:

```
>>> from django.db import models
>>> from rest_framework_tricks.models.fields import NestedProxyField
>>> from .constants import BOOK_STATUS_CHOICES, BOOK_STATUS_DEFAULT
>>>
>>>
>>> class Book(models.Model):
>>>
>>>     title = models.CharField(max_length=100)
>>>     description = models.TextField(null=True, blank=True)
>>>     summary = models.TextField(null=True, blank=True)
>>>     publication_date = models.DateField()
>>>     state = models.CharField(max_length=100,
>>>                              choices=BOOK_STATUS_CHOICES,
>>>                              default=BOOK_STATUS_DEFAULT)
>>>     isbn = models.CharField(max_length=100, unique=True)
>>>     price = models.DecimalField(max_digits=10, decimal_places=2)
>>>     pages = models.PositiveIntegerField(default=200)
>>>     stock_count = models.PositiveIntegerField(default=30)
>>>
>>>     # This does not cause a model change
>>>     publishing_information = NestedProxyField(
>>>         'publication_date',
>>>         'isbn',
>>>         'pages',
>>>     )
>>>
>>>     # This does not cause a model change
>>>     stock_information = NestedProxyField(
>>>         'stock_count',
>>>         'price',
>>>         'state',
>>>     )
>>>
>>>     class Meta(object):
>>>
>>>         ordering = ["isbn"]
>>>
>>>     def __str__(self):
>>>         return self.title
```

Nesting depth is unlimited, so the following would be possible as well.

Example:

```
>>> class Author(models.Model):
>>>
>>>     salutation = models.CharField(max_length=10)
>>>     name = models.CharField(max_length=200)
>>>     email = models.EmailField()
>>>     birth_date = models.DateField(null=True, blank=True)
>>>     biography = models.TextField(null=True, blank=True)
>>>     phone_number = models.CharField(max_length=200,
>>>                                     null=True,
>>>                                     blank=True)
>>>     website = models.URLField(null=True, blank=True)
>>>     company = models.CharField(max_length=200,
>>>                                null=True,
>>>                                blank=True)
```

(continues on next page)

(continued from previous page)

```
>>>     company_phone_number = models.CharField(max_length=200,
>>>                                             null=True,
>>>                                             blank=True)
>>>     company_email = models.EmailField(null=True, blank=True)
>>>     company_website = models.URLField(null=True, blank=True)
>>>
>>>     # This does not cause a model change
>>>     personal_contact_information = NestedProxyField(
>>>         'email',
>>>         'phone_number',
>>>         'website',
>>>     )
>>>
>>>     # This does not cause a model change
>>>     business_contact_information = NestedProxyField(
>>>         'company',
>>>         'company_email',
>>>         'company_phone_number',
>>>         'company_website',
>>>     )
>>>
>>>     # This does not cause a model change
>>>     contact_information = NestedProxyField(
>>>         'personal_contact_information',
>>>         'business_contact_information',
>>>     )
```

You could even do this (although the way it's written above is at the moment the preferred/recommended way of dealing with unlimited nesting depth.

```
>>>     # This does not cause a model change
>>>     contact_information = NestedProxyField(
>>>         {
>>>             'personal_contact_information': (
>>>                 'email',
>>>                 'phone_number',
>>>                 'website',
>>>             )
>>>         },
>>>         {
>>>             'business_contact_information': (
>>>                 'company',
>>>                 'company_email',
>>>                 'company_phone_number',
>>>                 'company_website',
>>>             )
>>>         },
>>>     )
```

13.3.1.1.2.6 Module contents

13.3.1.1.3 rest_framework_tricks.serializers package

13.3.1.1.3.1 Submodules

13.3.1.1.3.2 rest_framework_tricks.serializers.nested_proxy module

Serializers.

The following code is used in the usage examples of the `ModelSerializer` and `HyperlinkedModelSerializer` classes.

```
>>> from rest_framework import serializers
>>>
>>> from .models import Book
>>>
>>>
>>> class PublishingInformationSerializer(serializers.ModelSerializer):
>>>
>>>     publication_date = serializers.DateField(required=False)
>>>     isbn = serializers.CharField(required=False)
>>>     pages = serializers.IntegerField(required=False)
>>>
>>>     class Meta(object):
>>>
>>>         model = Book
>>>         fields = (
>>>             'publication_date',
>>>             'isbn',
>>>             'pages',
>>>         )
>>>         nested_proxy_field = True
>>>
>>>
>>> class StockInformationSerializer(serializers.ModelSerializer):
>>>
>>>     class Meta(object):
>>>
>>>         model = Book
>>>         fields = (
>>>             'stock_count',
>>>             'price',
>>>             'state',
>>>         )
>>>         nested_proxy_field = True
```

rest_framework_tricks.serializers.nested_proxy.**extract_nested_serializers**(*serializer*, *validated_data*, *nested_serializers=None*, *nested_serializers_data=*

Extract nested serializers.

Parameters

- **serializer** (*rest_framework.serializers.Serializer*) – Serializer instance.
- **validated_data** (*dict*) – Validated data.
- **nested_serializers** (*dict*) –
- **nested_serializers_data** –

Returns

Return type tuple

class rest_framework_tricks.serializers.nested_proxy.**HyperlinkedModelSerializer**(*instance=None*, *data=<class 'rest_framework*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

HyperlinkedModelSerializer for models with NestedProxyField fields.

Example:

```
>>> from rest_framework_tricks.serializers import (
>>>     HyperlinkedModelSerializer,
>>> )
>>>
>>>
>>> class BookSerializer(HyperlinkedModelSerializer):
>>>
>>>     publishing_information = PublishingInformationSerializer(
>>>         required=False
>>>     )
>>>     stock_information = StockInformationSerializer(required=False)
>>>
>>>     class Meta(object):
>>>
>>>         model = Book
>>>         fields = (
>>>             'url',
>>>             'id',
>>>             'title',
>>>             'description',
>>>             'summary',
>>>             'publishing_information',
>>>             'stock_information',
>>>         )
```

create(*validated_data*)

Create.

Parameters **validated_data** –

Returns

update(*instance*, *validated_data*)

Update.

Parameters

- **instance** –
- **validated_data** –

Returns

rest_framework_tricks.serializers.nested_proxy.**is_nested_proxy_field**(*field*)

Check if field is nested proxy field.

Parameters **field** –

Returns True or False

Return type bool

class rest_framework_tricks.serializers.nested_proxy.**ModelSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

ModelSerializer for models with NestedProxyField fields.

Example:

```
>>> from rest_framework_tricks.serializers import ModelSerializer
>>>
>>>
>>> class BookSerializer(ModelSerializer):
>>>
>>>     publishing_information = PublishingInformationSerializer(
>>>         required=False
>>>     )
>>>     stock_information = StockInformationSerializer(required=False)
>>>
>>>     class Meta(object):
>>>
>>>         model = Book
>>>         fields = (
>>>             'url',
>>>             'id',
>>>             'title',
>>>             'description',
>>>             'summary',
>>>             'publishing_information',
>>>             'stock_information',
>>>         )
```

create(*validated_data*)

Create.

Parameters **validated_data** –

Returns

update(*instance*, *validated_data*)

Update.

Parameters

- **instance** –
- **validated_data** –

Returns

class rest_framework_tricks.serializers.nested_proxy.**NestedProxyFieldIdentifier**

Bases: object

NestedProxyField identifier.

rest_framework_tricks.serializers.nested_proxy.**set_instance_values**(*nested_serializers*, *nested_serializers_data*, *instance*)

Set values on instance.

Does not perform any save actions.

Parameters

- **nested_serializers** – Nested serializers.
- **nested_serializers_data** – Nested serializers data.
- **instance** – Instance (not yet saved)

Returns Same instance with values set.

Return type

13.3.1.1.3.3 Module contents

Serializers.

rest_framework_tricks.serializers.**extract_nested_serializers**(*serializer*, *validated_data*, *nested_serializers=None*, *nested_serializers_data=None*)

Extract nested serializers.

Parameters

- **serializer** (*rest_framework.serializers.Serializer*) – Serializer instance.
- **validated_data** (*dict*) – Validated data.
- **nested_serializers** (*dict*) –
- **nested_serializers_data** –

Returns

Return type tuple

class rest_framework_tricks.serializers.**HyperlinkedModelSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.HyperlinkedModelSerializer

HyperlinkedModelSerializer for models with NestedProxyField fields.

Example:

```
>>> from rest_framework_tricks.serializers import (
>>>     HyperlinkedModelSerializer,
>>> )
>>>
>>>
>>> class BookSerializer(HyperlinkedModelSerializer):
>>>
```

(continues on next page)

(continued from previous page)

```
>>>     publishing_information = PublishingInformationSerializer(
>>>         required=False
>>>     )
>>>     stock_information = StockInformationSerializer(required=False)
>>>
>>>     class Meta(object):
>>>
>>>         model = Book
>>>         fields = (
>>>             'url',
>>>             'id',
>>>             'title',
>>>             'description',
>>>             'summary',
>>>             'publishing_information',
>>>             'stock_information',
>>>         )
```

create(*validated_data*)

Create.

Parameters **validated_data** –

Returns

update(*instance*, *validated_data*)

Update.

Parameters

- **instance** –
- **validated_data** –

Returns

rest_framework_tricks.serializers.**is_nested_proxy_field**(*field*)

Check if field is nested proxy field.

Parameters **field** –

Returns True or False

Return type bool

class rest_framework_tricks.serializers.**ModelSerializer**(*instance=None*, *data=<class 'rest_framework.fields.empty'>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

ModelSerializer for models with NestedProxyField fields.

Example:

```
>>> from rest_framework_tricks.serializers import ModelSerializer
>>>
>>>
>>> class BookSerializer(ModelSerializer):
>>>
>>>     publishing_information = PublishingInformationSerializer(
```

(continues on next page)

(continued from previous page)

```
>>>         required=False
>>>     )
>>>     stock_information = StockInformationSerializer(required=False)
>>>
>>>     class Meta(object):
>>>
>>>         model = Book
>>>         fields = (
>>>             'url',
>>>             'id',
>>>             'title',
>>>             'description',
>>>             'summary',
>>>             'publishing_information',
>>>             'stock_information',
>>>         )
```

create(*validated_data*)
Create.

Parameters **validated_data** –

Returns

update(*instance*, *validated_data*)
Update.

Parameters

- **instance** –
- **validated_data** –

Returns

class rest_framework_tricks.serializers.**NestedProxyFieldIdentifier**
Bases: object

NestedProxyField identifier.

rest_framework_tricks.serializers.**set_instance_values**(*nested_serializers*, *nested_serializers_data*, *instance*)
Set values on instance.

Does not perform any save actions.

Parameters

- **nested_serializers** – Nested serializers.
- **nested_serializers_data** – Nested serializers data.
- **instance** – Instance (not yet saved)

Returns Same instance with values set.

Return type

13.3.1.1.4 rest_framework_tricks.tests package

13.3.1.1.4.1 Submodules

13.3.1.1.4.2 rest_framework_tricks.tests.base module

Base tests.

class rest_framework_tricks.tests.base.**BaseRestFrameworkTestCase**(*methodName='runTest'*)
Bases: rest_framework.test.APITestCase

Base REST framework test case.

authenticate()
Helper for logging the user in.

Returns

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUpTestData()
Set up class.

class rest_framework_tricks.tests.base.**BaseTestCase**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Base test case.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUpTestData()
Set up class.

13.3.1.1.4.3 rest_framework_tricks.tests.test_nested_proxy_field module

Test NestedProxyField.

class rest_framework_tricks.tests.test_nested_proxy_field.**TestNestedProxyFieldCreateAction**
Bases: rest_framework_tricks.tests.test_nested_proxy_field.TestNestedProxyFieldActionBase

Test NestedProxyField - create action.

get_client_action()
Get client action.

Returns Client action.

Return type callable

get_status_code()
Get status code.

Returns Status code expected as result of the action.

Return type str

classmethod setUpClass()
Set up.

test_another_nested_proxy_field_model_serializer_depth()
Test NestedProxyField and ModelSerializer with more depth.

test_another_nested_proxy_field_model_serializer_more_depth()
Test NestedProxyField and ModelSerializer with more depth.

test_nested_proxy_field_hyperlinked_model_serializer()
Test NestedProxyField and HyperlinkedModelSerializer.

test_nested_proxy_field_model_serializer()
Test NestedProxyField and ModelSerializer.

test_nested_proxy_field_model_serializer_depth()
Test NestedProxyField and ModelSerializer with more depth.

test_nested_proxy_field_model_serializer_depth_missing_fields()
Test NestedProxyField and ModelSerializer with more depth.

Several non-required fields are missing.

test_nested_proxy_field_model_serializer_depth_more_missing_fields()
Test NestedProxyField and ModelSerializer with more depth.

All of the non-required fields are missing.

test_nested_proxy_field_model_serializer_missing_all_nested_fields()
Test NestedProxyField and ModelSerializer.

test_nested_proxy_field_model_serializer_missing_fields()
Test NestedProxyField and ModelSerializer with missing fields.

class rest_framework_tricks.tests.test_nested_proxy_field.**TestNestedProxyFieldUpdateAction**
Bases: rest_framework_tricks.tests.test_nested_proxy_field.TestNestedProxyFieldActionBase

Test NestedProxyField - update action.

get_client_action()
Get client action.

Returns Client action.

Return type callable

get_status_code()
Get status code.

Returns Status code expected as result of the action.

Return type str

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUpClass()
Set up.

test_another_nested_proxy_field_model_serializer_depth()
Test NestedProxyField and ModelSerializer with more depth.

test_another_nested_proxy_field_model_serializer_more_depth()
Test NestedProxyField and ModelSerializer with more depth.

test_nested_proxy_field_hyperlinked_model_serializer()
Test NestedProxyField and HyperlinkedModelSerializer.

test_nested_proxy_field_model_serializer()
Test NestedProxyField and ModelSerializer.

test_nested_proxy_field_model_serializer_depth()
Test NestedProxyField and ModelSerializer with more depth.

test_nested_proxy_field_model_serializer_depth_missing_fields()
Test NestedProxyField and ModelSerializer with more depth.

Several non-required fields are missing.

test_nested_proxy_field_model_serializer_depth_more_missing_fields()
Test NestedProxyField and ModelSerializer with more depth.

All of the non-required fields are missing.

test_nested_proxy_field_model_serializer_missing_all_nested_fields()
Test NestedProxyField and ModelSerializer.

test_nested_proxy_field_model_serializer_missing_fields()
Test NestedProxyField and ModelSerializer with missing fields.

13.3.1.1.4.4 rest_framework_tricks.tests.test_ordering_filter module

Test OrderingFilter.

class rest_framework_tricks.tests.test_ordering_filter.**TestOrderingFilter**(*methodName='runTest'*)
Bases: *rest_framework_tricks.tests.base.BaseRestFrameworkTestCase*

Test OrderingFilter.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUpClass()
Set up.

test_ordering()
Test ordering (ascending).

Returns

test_ordering_descending()
Test ordering (descending).

Returns

test_ordering_list()
Test ordering list (ascending).

Returns

test_ordering_list_descending()
Test ordering list (descending).

Returns

test_standard_no_ordering()
Test standard no ordering.

Returns

test_standard_ordering()
Test standard ordering (ascending).

Returns

test_standard_ordering_descending()
Test standard ordering (descending).

Returns

13.3.1.1.4.5 rest_framework_tricks.tests.test_utils module

Test utils.

class rest_framework_tricks.tests.test_utils.**TestUtils**(*methodName='runTest'*)
Bases: *rest_framework_tricks.tests.base.BaseTestCase*

Test utils.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

test_dict_proxy()
Test DictProxy.

13.3.1.1.4.6 Module contents

Tests.

13.3.1.2 Submodules

13.3.1.3 rest_framework_tricks.apps module

Apps.

class rest_framework_tricks.apps.**Config**(*app_name*, *app_module*)
Bases: django.apps.config.AppConfig

Config.

label = 'rest_framework_tricks'

name = 'rest_framework_tricks'

13.3.1.4 rest_framework_tricks.utils module

Utils.

class rest_framework_tricks.utils.**DictProxy**(*mapping*)
Bases: object

Dictionary proxy.

Example:

```
>>> from rest_framework_tricks.utils import DictProxy
>>>
>>>
>>> __dict = {
>>>     'name': self.faker.name(),
>>>     'date': self.faker.date(),
>>> }
```

(continues on next page)

(continued from previous page)

```
>>>
>>> __dict_proxy = DictProxy(__dict)
```

13.3.1.5 Module contents

Collection of various tricks for Django REST framework.

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

N

O

P

R

S

T

U