Django Rest Framework Braces

Release 0.1

Miroslav Shubernetskiy

July 01, 2015

Contents

CHAPTER 1

Contents

1.1 Overview

Django-rest-braces (drf-braces) is all about adding useful utilities for working with DRF. This overview will go over some of the most useful ones. You can refer to the API docs or source code to see documentation about all of the utilities.

1.1.1 Forms

Many Django applications are built using standard Django practices following basic request-response data flow. Safe way of dealing with user-input in such applications is to use Django forms. That works really well until application needs to be extended to introduce services since many of the forms might need to be rewritten (as serializers when using DRF). That situation becomes much worse custom forms (not `ModelForm`) need to be migrated.

Same issue presents itself when Django application is initially started by using services but later needs to add basic UI using Django forms.

DRF-braces attempts to solve these challenges by providing converters to go from form to serializer and vise-versa - `FormSerializer` and `SerializerForm`.

FormSerializer

`FormSerializer` is a special serializer class which converts existing `Form` to `Serializer` while preserving form validation logic. It works very similar to `ModelForm` or `ModelSerializer`:

```
from django import forms
from drf_braces.serializers.form_serializer import FormSerializer

class MyForm(forms.Form):
    foo = forms.CharField(max_length=32)
    bar = forms.DateTimeField()

class MySerializer(FormSerializer):
    class Meta(object):
        form = MyForm
```

SerializerForm

`SerializerForm` is a special form class which converts existing `Serializer` to `Form` while preserving serializer validation logic. It works very similar to `ModelForm` or `ModelSerializer`:

```
from rest_framework import serializers
from drf_braces.forms.serializer_form import SerializerForm

class MySerializer(serializers.Serializer):
    foo = serializers.CharField(max_length=32)
    bar = serializers.DateTimeField()

class MyForm(SerializerForm):
    class Meta(object):
        serializer = MySerializer
```

Warning: Currently `SerializerForm` does not support nested serializers.

1.1.2 Serializers

Enforce Validation

DRF has a concept of partial serializers which then only validate data supplied in request payload. The problem is that if the data is sent, it must be valid and if a single field is invalid, the whole serializer validation fails and error is returned to the consumer. That however is not always desired if the application must accept the payload as is and ignore invalid data.

DRF-braces provides `enforce_validation_serializer` which returns a recursive serializer copy does just that. It only enforces validation on specified fields and if validation fails on non-specified fields, it ignores that data:

```
from rest_framework import serializers
from drf_braces.serializers import enforce_validation_serializer

class MySerializer(serializers.Serializer):
    must_validate_fields = ['foo']

    foo = serializers.CharField(max_length=32)
    bar = serializers.DateTimeField()

MyEnforceValidationSerializer = enforce_validation_serializer(MySerializer)
```

Note: Even though above `MySerializer` defines `must_validate_fields`, `MySerializer` still enforces validation on all fields. Only serializers returned by `enforce_validation_serializer` consider `must_validate_fields` in field validation.

1.1.3 Mixins

- `MultipleSerializersViewMixin`
- `StrippingJSONViewMixin`
- `MapDataViewMixin`

1.1.4 Parsers

- `SortedJSONParser`
- `StrippingJSONParser`

1.1.5 Fields

Some fields:

- `UnvalidatedField`
- `PositiveIntegerField`
- `NonValidatingChoiceField`

and mixins:

- `EmptyStringFieldMixin`
- `AllowBlankFieldMixin`
- `ValueAsTextFieldMixin`

1.2 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.2.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/dealertrack/django-rest-framework-braces/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Django Rest Framework Braces could always use more documentation, whether as part of the official Django Rest Framework Braces docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dealertrack/django-rest-framework-braces/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.2.2 Get Started!

Ready to contribute? Here's how to set up *django-rest-framework-braces* for local development.

1. Fork the *django-rest-framework-braces* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-rest-framework-braces.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv drf-braces
$ cd django-rest-framework-braces/
$ make install
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make lint
$ make test-all
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.2.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.

2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, and for PyPy. Check https://travis-ci.org/dealertrack/django-rest-framework-braces/pull_requests and make sure that the tests pass for all supported Python versions.

1.3 Credits

1.3.1 Development Lead

- Mike Waters - https://github.com/mikewaters
- Miroslav Shubernetskiy - https://github.com/miki725

1.3.2 Contributors

- Khaled Porlin - https://github.com/porlin72

1.4 History

1.4.1 0.1.0 (2015-06-15)

- First release on PyPI.

CHAPTER 2

Django Rest Framework Braces

Collection of utilities for working with DRF. Name inspired by django-braces.

- Free software: MIT license
- GitHub: https://github.com/dealertrack/django-rest-framework-braces
- Documentation: https://django-rest-framework-braces.readthedocs.org.

2.1 Installing

Easiest way to install `django-rest-framework-braces` is by using `pip`:

```
$ pip install django-rest-framework-braces
```

2.2 Usage

Once installed, you can use any of the supplied utilities by simply importing them. For example:

```
from drf_braces.mixins import MultipleSerializersViewMixin

class MyViewSet(MultipleSerializersViewMixin, GenericViewSet):
    def create(self, request):
        serializer = self.get_serializer(serializer_class=MySerializer)
        ...
```

For full list of available utilities, please refer to the documentation.

2.3 Testing

To run the tests you need to install testing requirements first:

```
$ make install
```

Then to run tests, you can use use Makefile command:

```
$ make test
```

2.3.1 Indices and tables

- genindex
- modindex
- search