Django Reorder Documentation

Release 0.2.1

Baptiste Mispelon

September 13, 2016

Contents

Contents:

CHAPTER 1

Django Reorder

A project that helps sorting querysets in a specific order

1.1 Documentation

The full documentation is at https://django-reorder.readthedocs.org.

1.2 Quickstart

Install Django Reorder:

```
pip install django-reorder
```

Then use it in a project:

```
from django_reorder.reorder import reorder

Tshirt.objects.order_by(reorder(size=['S', 'M', 'L']))
```

Some more detailed examples can be foind on the Usage page.

1.3 Features

- Can be used in `order_by()` and in `annotate()` calls.
- Works across relationships.
- Lets you control the sorting order of `NULL` values explicitly (otherwise it can vary across databases).

1.4 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install -r requirements_test.txt
(myenv) $ python runtests.py
```

1.5 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ pip install django-reorder
```

CHAPTER 3

Usage

To use Django Reorder in a project:

```
from django_reorder.reorder import reorder
```

Consider the following model:

```
SIZES = [
    ('XXS', 'XX-Small'),
    ('XS', 'X-Small'),
    ('S', 'Small'),
    ('M', 'Medium'),
    ('L', 'Large'),
    ('XL', 'X-Large'),
    ('XXL', 'XX-Large'),
]


class Tshirt(models.Model):
    name = models.CharField(max_length=100)
    size = models.CharField(max_length=5, choices=SIZES)

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

If you try to do something like `Tshirt.objects.order_by('size')`, then you get alphabetical sorting on the size (L, M, S, XL, XS, XXL, XXS) which is probably not what you want.

With django-reorder, you can do:

```
Tshirt.objects.order_by(reorder(size=['XXS', 'XS', 'S', 'M', 'L', 'XL', 'XXL']))
```

And your tshirts will be sorted by their size.

3.1 Reference

`django_reorder.reorder.`**`reorder`**(*<fieldname>=<new order>*, *_default=AFTER*, *_reverse=False*)

This function generates a query expression (suitable for passing directly in `order_by` or `annotate`) that will sort the queryset by the given `<fieldname>` so that rows appear in the order given by `<new_order>`.

It takes two optional parameters (their names are prefixed by an underscore to prevent clashing with potential field names):

- `_default` controls whether values that don't appear in `<new order>` are sorted before or after (the default) those that do. Use the `BEFORE` and `AFTER` constants that can be imported from the module.
- `_reverse` lets you reverse the sorting.

`django_reorder.reorder.`**`null_first`** (*fieldname*)

This function is a small wrapper around `reorder()` and lets you control the sorting order of `NULL` values for a given field (otherwise your database will decide whether to put `NULL` values at the beginning or at the end, and different databases do it differently).

It takes a single required argument: the name of the field (as a string) whose `NULL` values should be sorted first.

class Tshirt(models.Model): ... purchased_on = models.DateTimeField(blank=True, null=True)

This will yield all Tshirt objects and those without a purchase date # will be listed first: Tshirt.objects.order_by(null_first('purchased_on'))

You can also combine null_first() with other sorting fields. # For example this will sort Tshirt by their purchase date but it will list # T-shirts without a purchase date first: Tshirt.objects.order_by(null_first('purchased_on'), 'purchased_on')

`django_reorder.reorder.`**`null_last`** (*fieldname*)

Works exactly like `null_first`, except that `NULL` values are sorted at the end.

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/bmispelon/django-reorder/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bmispelon/django-reorder/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django-reorder.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 django-reorder
$ cd django-reorder/
$ python setup.py develop
```

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:

```
$ flake8 django_reorder tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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.

4.3 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 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/bmispelon/django-reorder/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_django_reorder
```

CHAPTER 5

Credits

5.1 Development Lead

- Baptiste Mispelon <bmispelon@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.2.1 (2016-09-13)

- Forgot to `pull` before I ‘‘push‘‘ed...

6.2 0.2.0 (2016-09-13)

- Added `null_first()` and `null_last()` shortcuts.

6.3 0.1.0 (2016-07-26)

- First release on PyPI.

d

D

N

R