django-easyfilters Documentation

Release 0.5

Luke Plant

June 20, 2016

Contents

django-easyfilters provides a UI for filtering a Django QuerySet by clicking on links. It is similar in some ways to `list_filter` and `date_hierarchy` in Django's admin, but for use outside the admin. Importantly, it also includes result counts for the choices, and it has a bigger emphasis on intelligent display and things 'just working'.

Contents:

CHAPTER 1

Installation

Install using pip or easy_install. Nothing further is required.

CHAPTER 2

Overview

Suppose your models.py looks something like this:

```
class Book(models.Model):
    name = models.CharField(max_length=100)
    binding = models.CharField(max_length=2, choices=BINDING_CHOICES)
    authors = models.ManyToManyField(Author)
    genre = models.ForeignKey(Genre)
    price = models.DecimalField(max_digits=6, decimal_places=2)
    date_published = models.DateField()
```

(with BINDING_CHOICES, Author and Genre omitted for brevity).

You might want to present a list of Book objects, allowing the user to filter on the various fields. Your views.py would be something like this:

```
from django.shortcuts import render

from myapp.models import Book

def booklist(request):
    books = Book.objects.all()
    return render(request, "booklist.html", {'books': books})
```

and the template is like this:

```
{% for book in books %}
   {# etc #}
{% endfor %}
```

To add the filters, in views.py add a FilterSet subclass and change the view code as follow:

```
from django.shortcuts import render
from django_easyfilters import FilterSet

from myapp.models import Book

class BookFilterSet(FilterSet):
    fields = [
        'binding',
        'authors',
        'genre',
        'price',
        ]
```

```
def booklist(request):
    books = Book.objects.all()
    booksfilter = BookFilterSet(books, request.GET)
    return render(request, "booklist.html", {'books': booksfilter.qs,
                                             'booksfilter': booksfilter})
```

Notice that the `books` item put in the context has been replaced by `bookfilter.qs`, so that the QuerySet passed to the template has filtering applied to it, as defined by BookFilterSet and the information from the query string (request.GET).

The `booksfilter` item has been added to the context in order for the filters to be displayed on the template.

Then, in the template, just add `{{ booksfilter }}` to the template. You can also use pagination e.g. using django-pagination:

```
{% autopaginate books 20 %}

<h2>Filters:</h2>
{{ booksfilter }}

{% paginate %}

<h2>Books found</h2>
{% for book in books %}
   {# etc #}
{% endfor %}
```

The `FilterSet` also provides a 'title' attribute that can be used to provide a simple summary of what has been selected so far. It is made up of a comma separated list of chosen fields. For example, if the user has selected genre 'Classics' and binding 'Hardback' in the example above, you would get the following:

```
>>> books = Book.objects.all()
>>> booksfilter = BookFilterSet(books, request.GET)
>>> booksfilter.title
u"Hardback, Classics"
```

The fields used for the `title` attribute, and the order they are used, can be customised by adding a `title_fields` attribute to your `FilterSet`:

```
class BookFilterSet(FilterSet):
    fields = [
        'binding',
        'authors',
        'genre',
        'price',
        ]

    title_fields = ['genre', 'binding']
```

Customisation of the filters can be done in various ways - see the FilterSet documentation for how to do this, and the Filters documentation for options that can be specified.

CHAPTER 3

FilterSet

class `django_easyfilters.filterset.`**`FilterSet`**

This is meant to be used by subclassing. The only required attribute is `fields`, which must be a list of fields to produce filters for. For example, given the following model definition:

```
class Book(models.Model):
    name = models.CharField(max_length=100)
    authors = models.ManyToManyField(Author)
    genre = models.ForeignKey(Genre)
    date_published = models.DateField()
```

...you could create a BookFilterSet like this:

```
class BookFilterSet(FilterSet):
    fields = [
        'genre',
        'authors',
        'date_published',
    ]
```

Each item in the fields attribute can also be a two-tuple containing first the field name and second a dictionary of options to be passed to the filters as keyword arguments, or a three-tuple containing the field name, a dictionary of options, and a Filter class. In this way you can override default options and the default filter type used e.g.:

```
from django_easyfilters.filters import ValuesFilter

class BookFilterSet(FilterSet):
    fields = [
        ('genre', dict(order_by_count=True)),
        ('date_published', {}, ValuesFilter),
    ]
```

This also allows *custom Filter classes* to be used.

To use the BookFilterSet, please see the overview instructions. The public API of `FilterSet` for use consists of:

`__init__`(*queryset*, *params*)

queryset must be a QuerySet, which can already be filtered.

params must be a QueryDict, normally request.GET.

`qs`

This attribute contains the input QuerySet filtered according to the data in `params`.

title
This attribute contains a title summarising the filters that have been selected.

In addition, there are methods/attributes that can be overridden to customise the FilterSet:

get_template (*field_name*)
This method is called for each field in the filterset, with the field name being passed in.

It is expected to return a Django Template instance. This template will then be rendered with the following Context data:

- `filterlabel` - the label for the filter (derived from verbose_name of the field)
- `choices` - a list of *choices* for the filter. Each one has the following attributes:
  - `link_type`: either `remove`, `add` or `display`, depending on the type of the choice.
  - `label`: the text to be displayed for this choice.
  - `url` for those that are `remove` or `add`, a URL for selecting that filter.
  - `count`: for those that are `add` links, the number of items in the QuerySet that match that choice.

template
A string containing a Django template, used to render all the filters. It is used by the default `get_template` method, see above.

title_fields
By default, the fields used to create the `title` attribute are all fields specified in the `fields` attribute, in that order. Specify `title_fields` to override this.

CHAPTER 4

Filters

When you specify the `fields` attribute on a `FilterSet` subclass, various different `Filter` classes will be chosen depending on the type of field. They are listed below, with the keyword argument options that they take.

class `django_easyfilters.filters.`**`Filter`**
This is the base class for all filters, and provides some options:

- `query_param`:

  The parameter in the query string that will be used for this field. This can be useful for shortening the query strings that are generated.

- `order_by_count`:

  Default: False

  If `True`, this will cause the choices to be sorted so that the choices with the largest 'count' appear first.

class `django_easyfilters.filters.`**`ForeignKeyFilter`**
This is used for ForeignKey fields

class `django_easyfilters.filters.`**`ManyToManyFilter`**
This is used for ManyToMany fields

class `django_easyfilters.filters.`**`ChoicesFilter`**
This is used for fields that have 'choices' defined (normally passed in to the field constructor). The choices presented will be in the order specified in 'choices'.

class `django_easyfilters.filters.`**`DateTimeFilter`**
This is the most complex of the filters, as it allows drill-down from year to month to day. It takes the following options:

- `max_links`

  Default: 12

  The maximum number of links to display. If the number of choices at any level does not fit into this value, ranges will be used to shrink the number of choices.

- `max_depth`

  Default: None

  If `'year'` or `'month'` is specified, the drill-down will be limited to that level.

class `django_easyfilters.filters.`**`NumericRangeFilter`**
This filter produces ranges of values for a numeric field. It is the default filter for decimal fields, but can also be used with integer fields. It attempts to make the ranges 'look nice' using rounded numbers in an automatic way. It uses 'drill-down' like DateTimeFilter.

It takes the following options:

- `max_links`

  Default: 5

  The maximum number of links to display. If there are fewer distinct values than this in the data, single values will be shown, and ranges otherwise.

- `ranges`

  Default: None

  If this is specified, it will override the (initial) automatic range. The value should be a list of ranges, where each item in the list is either:

  – a two-tuple containing the beginning and end range values

  – a three-tuple containing the beginning and end range values and a custom label.

- `drilldown`

  Default: True

  If `False`, only one level of choices will be displayed.

The 'end points' of ranges are handled in the following way: the lower bound is exclusive, and the upper bound is inclusive, apart from for the first range, where both are inclusive. This is designed for a fairly intuitive behaviour.

class `django_easyfilters.filters.`**`ValuesFilter`**
This is the fallback that is used when nothing else matches.

4.1 Custom Filter classes

As described in the `FilterSet` documentation, you can provide your own Filter class for a field. If you do so, it is expected to have the following API:

- `__init__(field, model, params, **kwargs)`

  Constructor. `field` is the string identifying the field, `model` is the model class, `params` is a QueryDict (i.e. normally request.GET). `kwargs` contains any custom options specified for the filter.

- `apply_filter(qs)`

  This method takes the QuerySet `qs` and returns a QuerySet that has filters applied to it, where the filter parameters are defined in the `params` that were passed to the constructor. The method must be able to extract the relevant parameter, if it exists, and filter the QuerySet accordingly.

- `get_choices(qs)`

  This method is passed a fully filtered QuerySet, and must return a list of choices to present to the user. The choices should be instances of `django_easyfilters.filters.FilterChoice`, which has the attributes:

  – label: User presentable text string for the choice

  – link_type: choice of FILTER_ADD, FILTER_REMOVE, FILTER_DISPLAY

  – count: the number of items for this choice (only for FILTER_ADD)

  – params: parameters used to create a link for this option, as a QueryDict

If you want to use a provided Filter and subclass from it, at the moment only the following additional methods are considered public:

- `render_choice_object(choice)`

  This method is responsible for generating the label for a choice (whether it is an 'add' or 'remove' choice). It is passed a choice object that is derived either from the query string (for 'remove' choices) or from the database (for 'add' choices).

  Different subclasses of Filter pass different types of object in. Currently the following can be relied on:

  - *ForeignKeyFilter* and *ManyToManyFilter* pass in the related database model instances as 'choice'.
  - *ValuesFilter* and *ChoicesFilter* pass in the underlying raw database value as 'choice'.

All other methods of Filter and subclasses are considered private implementation details and may change without warning.

CHAPTER 5

Development

5.1 Tests

To run the test suite, do:

```
./manage.py test django_easyfilters
```

This requires that the directory containing the `django_easyfilters` directory is on your Python path (virtualenv recommended), and Django is installed.

Alternatively, to run it on all supported platforms, install tox and do:

```
tox
```

This will create all the necessary virtualenvs for you, and is the preferred way of working, but will take longer initially. Once you have run it once, you can activate a specific virtualenv by doing, for example:

```
. .tox/py33-django15/bin/activate
```

5.2 Editing test fixtures

To edit the test fixtures, you can edit the fixtures in django_easyfilters/tests/fixtures/, or you can do it via an admin interface:

First create an empty db:

```
rm tests.db
./manage.py syncdb
```

Then load with current test fixture:

```
./manage.py loaddata django_easyfilters_tests
```

Then edit in admin at http://localhost:8000/admin/

```
./manage.py runserver
```

Or from a Python shell.

Then dump data:

```
./manage.py dumpdata tests --format=json --indent=2 > django_easyfilters/tests/fixtures/django_easyfi
```

5.3 Demo

Once the test fixtures have been loaded into the DB, and the devserver is running, as above, you can view a test page at http://localhost:8000/books/

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Symbols

C

D

F

G

M

N

Q

T

V