django-easyfilters Documentation

Release 0.1

Luke Plant

June 12, 2013

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

ONE

INSTALLATION

Install using pip or easy_install. Nothing further is required.

CHAPTER

TWO

OVERVIEW

Suppose your model.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. Assuming your have a views.py is something like this:

```
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, you add the following:

```
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Notice that the `books` item put in the context has been replaced by `bookfilter.qs`, so that the QuerySet passed to the template has filtering applied to it, as defined by BookFilterSet and the information from the query string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, in order for the filters to be displayed on the template.

Then, in the template, just add `{{ booksfilter }}` to the template. books. 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 %}
```

Customisation of the filters can be done using a tuple containing (`field_name`, dict of options), instead of just `field_name`:

```
class BookFilterSet(FilterSet):
    model = Book
    fields = [
        'binding',
        ('genre', dict(order_by_count=True))
    ]
```

See *the Filters documentation* for options that can be specified. See *the FilterSet documentation* for ways to customize the rendering of the filters.

CHAPTER

THREE

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',
    ]
```

The items in the fields attribute can also be two-tuples containing first the field name and second a dictionary of options to be passed to the *filters* as keyword arguments.

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`.

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

`get_template`(*field_name*)

This method can be overriden to render the filterset. It 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.

CHAPTER

FOUR

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.

At the moment, all other methods of Filter and subclasses are considered private implementation details, until all the Filters are implemented and the API firms up.

class `django_easyfilters.filters.`**`Filter`**
This is the base class for all filters, and has 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. 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.`**`ValuesFilter`**
This is the fallback that is used when nothing else matches.

CHAPTER

FIVE

DEVELOPMENT

Python 2.6 is required for running the test suites and demo app.

First, ensure the directory containing the `django_easyfilters` directory is on your Python path (virtualenv recommended). Django is a required dependency.

5.1 Tests

To run the test suite, do:

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

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

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*