django-mongoengine-filter Documentation

Release 0.1

Artur Barseghyan <artur.barseghyan@gmail.com>

Mar 22, 2020

Contents

`django-mongoengine-filter` is a reusable Django application for allowing users to filter mongoengine querysets dynamically. It's very similar to popular `django-filter` library and is designed to be used as a drop-in replacement (as much as it's possible) strictly tied to `MongoEngine`.

Full documentation on Read the docs.

CHAPTER 1

Requirements

- Python 2.7, 3.5, 3.6, 3.7, 3.8
- Django 1.11, 2.0, 2.1, 2.2, 3.0

CHAPTER 2

Installation

Install using pip:

```
pip install django-mongoengine-filter
```

Or latest development version:

```
pip install https://github.com/barseghyanartur/django-mongoengine-filter/archive/
↪master.zip
```

CHAPTER 3

Usage

Sample document

```
from mongoengine import fields, document
from .constants import PROFILE_TYPES, PROFILE_TYPE_FREE, GENDERS, GENDER_MALE

class Person(document.Document):

    name = fields.StringField(
        required=True,
        max_length=255,
        default="Robot",
        verbose_name="Name"
    )
    age = fields.IntField(required=True, verbose_name="Age")
    num_fingers = fields.IntField(
        required=False,
        verbose_name="Number of fingers"
    )
    profile_type = fields.StringField(
        required=False,
        blank=False,
        null=False,
        choices=PROFILE_TYPES,
        default=PROFILE_TYPE_FREE,
    )
    gender = fields.StringField(
        required=False,
        blank=False,
        null=False,
        choices=GENDERS,
        default=GENDER_MALE
    )

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

Sample filter

```
import django_mongoengine_filter

class PersonFilter(django_mongoengine_filter.FilterSet):

    profile_type = django_mongoengine_filter.StringFilter()
    ten_fingers = django_mongoengine_filter.MethodFilter(
        action="ten_fingers_filter"
    )

    class Meta:
        model = Person
        fields = ["profile_type", "ten_fingers"]

    def ten_fingers_filter(self, queryset, name, value):
        if value == 'yes':
            return queryset.filter(num_fingers=10)
        return queryset
```

Sample view

With function-based views:

```
def person_list(request):
    filter = PersonFilter(request.GET, queryset=Person.objects)
    return render(request, "dfm_app/person_list.html", {"object_list": filter.qs})
```

Or class-based views:

```
from django_mongoengine_filter.views import FilterView

class PersonListView(FilterView):

    filterset_class = PersonFilter
    template_name = "dfm_app/person_list.html"
```

Sample template

```
<ul>
{% for obj in object_list %}
    <li>{{ obj.name }} - {{ obj.age }}</li>
{% endfor %}
</ul>
```

Sample requests

- GET /persons/
- GET /persons/?profile_type=free&gender=male
- GET /persons/?profile_type=free&gender=female
- GET /persons/?profile_type=member&gender=female
- GET /persons/?ten_fingers=yes

CHAPTER 4

Development

4.1 Testing

To run tests in your working environment type:

```
./runtests.py
```

To test with all supported Python versions type:

```
tox
```

4.2 Running MongoDB

The easiest way is to run it via Docker:

```
docker pull mongo:latest
docker run -p 27017:27017 mongo:latest
```

4.3 Writing documentation

Keep the following hierarchy.

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

header
======

```

(continues on next page)

(continued from previous page)

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

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

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

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

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

CHAPTER 5

License

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

CHAPTER 6

Support

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

CHAPTER 7

Author

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER 8

Documentation

Contents:

8.1 Filter Reference

This is a reference document with a list of the filters and their arguments.

8.1.1 Filters

8.1.1.1 `CharFilter`

This filter does simple character matches, used with `CharField` and `TextField` by default.

8.1.1.2 `BooleanFilter`

This filter matches a boolean, either `True` or `False`, used with `BooleanField` and `NullBooleanField` by default.

8.1.1.3 `ChoiceFilter`

This filter matches an item of any type by choices, used with any field that has `choices`.

8.1.1.4 `MultipleChoiceFilter`

The same as `ChoiceFilter` except the user can select multiple items and it selects the OR of all the choices.

8.1.1.5 DateFilter

Matches on a date. Used with `DateField` by default.

8.1.1.6 DateTimeFilter

Matches on a date and time. Used with `DateTimeField` by default.

8.1.1.7 TimeFilter

Matches on a time. Used with `TimeField` by default.

8.1.1.8 ModelChoiceFilter

Similar to a `ChoiceFilter` except it works with related models, used for `ForeignKey` by default.

8.1.1.9 ModelMultipleChoiceFilter

Similar to a `MultipleChoiceFilter` except it works with related models, used for `ManyToManyField` by default.

8.1.1.10 NumberFilter

Filters based on a numerical value, used with `IntegerField`, `FloatField`, and `DecimalField` by default.

8.1.1.11 RangeFilter

Filters where a value is between two numerical values.

8.1.1.12 DateRangeFilter

Filter similar to the admin changelist date one, it has a number of common selections for working with date fields.

8.1.1.13 AllValuesFilter

This is a `ChoiceFilter` whose choices are the current values in the database. So if in the DB for the given field you have values of 5, 7, and 9 each of those is present as an option. This is similar to the default behavior of the admin.

8.1.2 Core Arguments

8.1.2.1 name

The name of the field this filter is supposed to filter on, if this is not provided it automatically becomes the filter's name on the `FilterSet`.

8.1.2.2 label

The label as it will apear in the HTML, analogous to a form field's label argument.

8.1.2.3 widget

The django.form Widget class which will represent the `Filter`. In addition to the widgets that are included with Django that you can use there are additional ones that django-filter provides which may be useful:

- `django_filters.widgets.LinkWidget` – this displays the options in a mannner similar to the way the Django Admin does, as a series of links. The link for the selected option will have `class="selected"`.

8.1.2.4 action

An optional callable that tells the filter how to handle the queryset. It recieves a `QuerySet` and the value to filter on and should return a `Queryset` that is filtered appropriately.

8.1.2.5 lookup_type

The type of lookup that should be performed using the Django ORM. All the normal options are allowed, and should be provided as a string. You can also provide either `None` or a `list` or a `tuple`. If `None` is provided, then the user can select the lookup type from all the ones available in the Django ORM. If a `list` or `tuple` is provided, then the user can select from those options.

8.1.2.6 distinct

A boolean value that specifies whether the Filter will use distinct on the queryset. This option can be used to eliminate duplicate results when using filters that span related models. Defaults to `False`.

8.1.2.7 exclude

A boolean value that specifies whether the Filter should use `filter` or `exclude` on the queryset. Defaults to `False`.

8.1.2.8 **kwargs

Any extra keyword arguments will be provided to the accompanying form Field. This can be used to provide arguments like `choices` or `queryset`.

8.2 Widget Reference

This is a reference document with a list of the provided widgets and their arguments.

8.2.1 `LinkWidget`

This widget renders each option as a link, instead of an actual <input>. It has one method that you can overide for additional customizability. `option_string()` should return a string with 3 Python keyword argument placeholders:

1. `attrs`: This is a string with all the attributes that will be on the final `<a>` tag.
2. `query_string`: This is the query string for use in the `href` option on the `<a>` elemeent.
3. `label`: This is the text to be displayed to the user.

8.3 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.

8.3.1 0.3.5

2020-03-23

- Tested against Python 3.8.
- Tested against Django 3.0.

8.3.2 0.3.4

2019-04-04

- Using lazy queries where possible.

8.3.3 0.3.3

2019-04-02

- Tested against Django 2.2.

8.3.4 0.3.2

2019-04-01

- Fixes in class-based views.
- Addition to docs.

8.3.5 0.3.1

2019-03-26

- More tests.
- Addition to docs.

8.3.6 0.3

2019-03-25

Got status beta

Note: Namespace changed from *django_filters_mongoengine* to *django_mongoengine_filter*. Modify your imports accordingly.

- Clean up.
- Added docs, manifest, tox.

8.3.7 0.2

2019-03-25

- Working method filters.

8.3.8 0.1

2019-03-25

- Initial alpha release.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search