

---

# **django-mongoengine-filter Documentation**

*Release 0.4.2*

**Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>**

**May 23, 2023**



# CONTENTS

|          |                                     |           |
|----------|-------------------------------------|-----------|
| <b>1</b> | <b>Requirements</b>                 | <b>3</b>  |
| <b>2</b> | <b>Installation</b>                 | <b>5</b>  |
| <b>3</b> | <b>Usage</b>                        | <b>7</b>  |
| <b>4</b> | <b>Development</b>                  | <b>9</b>  |
| 4.1      | Testing . . . . .                   | 9         |
| 4.2      | Running MongoDB . . . . .           | 9         |
| 4.3      | Writing documentation . . . . .     | 9         |
| <b>5</b> | <b>License</b>                      | <b>11</b> |
| <b>6</b> | <b>Support</b>                      | <b>13</b> |
| <b>7</b> | <b>Author</b>                       | <b>15</b> |
| <b>8</b> | <b>Documentation</b>                | <b>17</b> |
| 8.1      | Filter Reference . . . . .          | 17        |
| 8.1.1    | Filters . . . . .                   | 17        |
| 8.1.1.1  | CharFilter . . . . .                | 17        |
| 8.1.1.2  | BooleanFilter . . . . .             | 17        |
| 8.1.1.3  | ChoiceFilter . . . . .              | 17        |
| 8.1.1.4  | MultipleChoiceFilter . . . . .      | 17        |
| 8.1.1.5  | DateFilter . . . . .                | 17        |
| 8.1.1.6  | DateTimeFilter . . . . .            | 18        |
| 8.1.1.7  | TimeFilter . . . . .                | 18        |
| 8.1.1.8  | ModelChoiceFilter . . . . .         | 18        |
| 8.1.1.9  | ModelMultipleChoiceFilter . . . . . | 18        |
| 8.1.1.10 | NumberFilter . . . . .              | 18        |
| 8.1.1.11 | RangeFilter . . . . .               | 18        |
| 8.1.1.12 | DateRangeFilter . . . . .           | 18        |
| 8.1.1.13 | AllValuesFilter . . . . .           | 18        |
| 8.1.2    | Core Arguments . . . . .            | 18        |
| 8.1.2.1  | name . . . . .                      | 18        |
| 8.1.2.2  | label . . . . .                     | 19        |
| 8.1.2.3  | widget . . . . .                    | 19        |
| 8.1.2.4  | action . . . . .                    | 19        |
| 8.1.2.5  | lookup_type . . . . .               | 19        |
| 8.1.2.6  | distinct . . . . .                  | 19        |
| 8.1.2.7  | exclude . . . . .                   | 19        |

|         |                                           |    |
|---------|-------------------------------------------|----|
| 8.1.2.8 | <b>**kwargs</b>                           | 19 |
| 8.2     | Widget Reference                          | 19 |
| 8.2.1   | LinkWidget                                | 20 |
| 8.3     | Using django-mongoengine-filter           | 20 |
| 8.3.1   | The model                                 | 20 |
| 8.3.2   | The filter                                | 20 |
| 8.3.3   | The view                                  | 22 |
| 8.3.4   | The URL conf                              | 22 |
| 8.3.5   | The template                              | 22 |
| 8.3.6   | Other Meta options                        | 22 |
| 8.3.6.1 | Ordering using <code>order_by</code>      | 22 |
| 8.3.6.2 | Custom Forms using <code>form</code>      | 23 |
| 8.3.7   | Non-Meta options                          | 23 |
| 8.3.7.1 | <code>strict</code>                       | 23 |
| 8.3.8   | Overriding <code>FilterSet</code> methods | 23 |
| 8.3.8.1 | <code>get_ordering_field()</code>         | 23 |
| 8.3.9   | Generic View                              | 24 |
| 8.4     | Recipes                                   | 24 |
| 8.5     | Release history and notes                 | 24 |
| 8.5.1   | 0.4.2                                     | 25 |
| 8.5.2   | 0.4.1                                     | 25 |
| 8.5.3   | 0.4.0                                     | 25 |
| 8.5.4   | 0.3.5                                     | 25 |
| 8.5.5   | 0.3.4                                     | 25 |
| 8.5.6   | 0.3.3                                     | 25 |
| 8.5.7   | 0.3.2                                     | 26 |
| 8.5.8   | 0.3.1                                     | 26 |
| 8.5.9   | 0.3                                       | 26 |
| 8.5.10  | 0.2                                       | 26 |
| 8.5.11  | 0.1                                       | 26 |

|          |                           |           |
|----------|---------------------------|-----------|
| <b>9</b> | <b>Indices and tables</b> | <b>27</b> |
|----------|---------------------------|-----------|

`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](#).



## REQUIREMENTS

- Python 3.7, 3.8, 3.9, 3.10 or 3.11.
- MongoDB 3.x, 4.x, 5.x.
- Django 2.2, 3.0, 3.1, 3.2, 4.0 or 4.1.



## 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
```



## 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):
```

(continues on next page)

(continued from previous page)

```
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

## DEVELOPMENT

### 4.1 Testing

To run tests in your working environment type:

```
pytest -vrx
```

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  
=====  
  
sub-header  
-----  
  
sub-sub-header  
~~~~~  
  
sub-sub-sub-header  
^^^^^^^^^^^^^^^^
```

(continues on next page)

(continued from previous page)

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

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

**LICENSE**

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



## SUPPORT

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

For overall issues, go to [GitHub](#).



---

CHAPTER  
SEVEN

---

AUTHOR

Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>



## 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 appear 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 manner 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 receives 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 override for additional customization. `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>` element.
3. `label`: This is the text to be displayed to the user.

## 8.3 Using django-mongoengine-filter

django-mongoengine-filter provides a simple way to filter down a queryset based on parameters a user provides. Say we have a `Product` model and we want to let our users filter which products they see on a list page.

### 8.3.1 The model

Let's start with our model:

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=255)
    price = models.DecimalField()
    description = models.TextField()
    release_date = models.DateField()
    manufacturer = models.ForeignKey(Manufacturer)
```

### 8.3.2 The filter

We have a number of fields and we want to let our users filter based on the price or the release\_date. We create a `FilterSet` for this:

```
import django_mongoengine_filter

class ProductFilter(django_mongoengine_filter.FilterSet):
    class Meta:
        model = Product
        fields = ['price', 'release_date']
```

As you can see this uses a very similar API to Django's `ModelForm`. Just like with a `ModelForm` we can also override filters, or add new ones using a declarative syntax:

```
import django_filters

class ProductFilter(django_mongoengine_filter.FilterSet):
    price = django_filters.NumberFilter(lookup_type='lt')
    class Meta:
        model = Product
        fields = ['price', 'release_date']
```

Filters take a `lookup_type` argument which specifies what lookup type to use with Django’s ORM. So here when a user entered a price it would show all Products with a price less than that.

**It’s quite common to forget to set lookup type for `CharField`’s/`TextField`’s and wonder why search for “foo” doesn’t return result for “foobar”. It’s because default lookup type is exact text, but you probably want `icontains` lookup field.**

Items in the `fields` sequence in the Meta class may include “relationship paths” using Django’s `__` syntax to filter on fields on a related model:

```
class ProductFilter(django_mongoengine_filter.FilterSet):
    class Meta:
        model = Product
        fields = ['manufacturer__country']
```

Filters also take any arbitrary keyword arguments which get passed onto the `django.forms.Field` initializer. These extra keyword arguments get stored in `Filter.extra`, so it’s possible to override the initializer of a `FilterSet` to add extra ones:

```
class ProductFilter(django_mongoengine_filter.FilterSet):
    class Meta:
        model = Product
        fields = ['manufacturer']

    def __init__(self, *args, **kwargs):
        super(ProductFilter, self).__init__(*args, **kwargs)
        self.filters['manufacturer'].extra.update(
            {'empty_label': 'All Manufacturers'})
```

Like `django.contrib.admin.ModelAdmin` does it is possible to override default filters for all the models fields of the same kind using `filter_overrides`:

```
class ProductFilter(django_mongoengine_filter.FilterSet):
    filter_overrides = {
        models.CharField: {
            'filter_class': django_filters.CharFilter,
            'extra': lambda f: {
                'lookup_type': 'icontains',
            }
        }
    }

    class Meta:
        model = Product
        fields = ['name']
```

### 8.3.3 The view

Now we need to write a view:

```
def product_list(request):
    f = ProductFilter(request.GET, queryset=Product.objects)
    return render_to_response('my_app/template.html', {'filter': f})
```

If a `queryset` argument isn't provided then all the items in the default manager of the model will be used.

### 8.3.4 The URL conf

We need a URL pattern to call the view:

```
re_path(r'^list$', views.product_list)
```

### 8.3.5 The template

And lastly we need a template:

```
{% extends "base.html" %}

{% block content %}
    <form action="" method="get">
        {{ filter.form.as_p }}
        <input type="submit" />
    </form>
    {% for obj in filter %}
        {{ obj.name }} - ${{ obj.price }}<br />
    {% endfor %}
{% endblock %}
```

And that's all there is to it! The `form` attribute contains a normal Django form, and when we iterate over the `FilterSet` we get the objects in the resulting `queryset`.

### 8.3.6 Other Meta options

#### 8.3.6.1 Ordering using `order_by`

You can allow the user to control ordering by providing the `order_by` argument in the Filter's Meta class. `order_by` can be either a list or tuple of field names, in which case those are the options, or it can be a bool which, if True, indicates that all fields that the user can filter on can also be sorted on. An example of ordering using a list:

```
import django_filters

class ProductFilter(django_filters.FilterSet):

    price = django_filters.NumberFilter(lookup_type='lt')

    class Meta:
        model = Product
```

(continues on next page)

(continued from previous page)

```
fields = ['price', 'release_date']
order_by = ['price']
```

If you want to control the display of items in `order_by`, you can set it to a list or tuple of 2-tuples in the format (field\_name, display\_name). This lets you override the displayed names for your ordering fields:

```
order_by = (
    ('name', 'Company Name'),
    ('average_rating', 'Stars'),
)
```

Note that the default query parameter name used for ordering is `o`. You can override this by setting an `order_by_field` attribute on the `FilterSet` class to the string value you would like to use.

### 8.3.6.2 Custom Forms using form

The inner `Meta` class also takes an optional `form` argument. This is a form class from which `FilterSet.form` will subclass. This works similar to the `form` option on a `ModelAdmin`.

## 8.3.7 Non-Meta options

Note that these options do not go in the `Meta` class, they are specified directly in your `FilterSet` class.

### 8.3.7.1 strict

The `strict` option controls whether results are returned when an invalid value is specified by the user for any filter field. By default, `strict` is set to `True` meaning that an empty queryset is returned if any field contains an invalid value. You can loosen this behavior by setting `strict` to `False` which will effectively ignore a filter field if its value is invalid.

## 8.3.8 Overriding FilterSet methods

### 8.3.8.1 get\_ordering\_field()

If you want to use a custom widget, or in any other way override the ordering field you can override the `get_ordering_field()` method on a `FilterSet`. This method just needs to return a Form Field.

Ordering on multiple fields, or other complex orderings can be achieved by overriding the `FilterSet.get_order_by()` method. This is passed the selected `order_by` value, and is expected to return an iterable of values to pass to `QuerySet.order_by`. For example, to sort a `User` table by last name, then first name:

```
class UserFilter(django_filters.FilterSet):
    class Meta:
        order_by = (
            ('username', 'Username'),
            ('last_name', 'Last Name')
        )

    def get_order_by(self, order_value):
```

(continues on next page)

(continued from previous page)

```
if order_value == 'last_name':
    return ['last_name', 'first_name']
return super(UserFilter, self).get_order_by(order_value)
```

### 8.3.9 Generic View

In addition to the above usage there is also a class-based generic view included in django-filter, which lives at `django_filters.views.FilterView`. You must provide either a `model` or `filterset_class` argument, similar to `ListView` in Django itself:

```
# urls.py
from django.urls import re_path
from django_filters.views import FilterView
from myapp.models import Product

urlpatterns = [
    re_path(r'^list/$', FilterView.as_view(model=Product)),
]
```

You must provide a template at `<app>/<model>_filter.html` which gets the context parameter `filter`. Additionally, the context will contain `object_list` which holds the filtered queryset.

A legacy functional generic view is still included in django-filter, although its use is deprecated. It can be found at `django_filters.views.object_filter`. You must provide the same arguments to it as the class based view:

```
# urls.py
from django.urls import re_path
from myapp.models import Product

urlpatterns = [
    re_path(r'^list/$', 'django_filters.views.object_filter', {'model': Product}),
]
```

The needed template and its context variables will also be the same as the class-based view above.

## 8.4 Recipes

## 8.5 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.

### 8.5.1 0.4.2

2023-05-24

- Fix *AttributeError: 'BaseFilterSet' object has no attribute 'is\_valid'* issue.

### 8.5.2 0.4.1

2023-02-23

- Fix issue with adding `help_text`.

### 8.5.3 0.4.0

2022-12-24

- Drop support for Python < 3.7.
- Drop support for Django < 2.2.
- Tested against Python 3.9, 3.10 and 3.11.
- Tested against Django 3.1, 3.2, 4.0 and 4.1.
- Apply `black`, `isort` and `ruff`.
- Fix GitHub CI.

### 8.5.4 0.3.5

2020-03-23

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

### 8.5.5 0.3.4

2019-04-04

- Using lazy queries where possible.

### 8.5.6 0.3.3

2019-04-02

- Tested against Django 2.2.

### 8.5.7 0.3.2

2019-04-01

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

### 8.5.8 0.3.1

2019-03-26

- More tests.
- Addition to docs.

### 8.5.9 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.

### 8.5.10 0.2

2019-03-25

- Working method filters.

### 8.5.11 0.1

2019-03-25

- Initial alpha release.

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`