
django-elasticsearch-dsl-drf Documentation

Release 0.3.12

Artur Barseghyan <artur.barseghyan@gmail.com>

Sep 21, 2017

Contents

1	Prerequisites	3
2	Dependencies	5
3	Documentation	7
4	Main features and highlights	9
5	Installation	11
6	Quick start	13
7	Testing	15
8	Writing documentation	17
9	License	19
10	Support	21
11	Author	23
12	Project documentation	25
12.1	Quick start	25
12.1.1	Installation	27
12.1.2	Example app	28
12.1.2.1	Sample models	28
12.1.2.1.1	Required imports	28
12.1.2.1.2	Book statuses	28
12.1.2.1.3	Publisher model	29
12.1.2.1.4	Author model	29
12.1.2.1.5	Tag model	30
12.1.2.1.6	Book model	30
12.1.2.2	Admin classes	31
12.1.2.3	Create database tables	32
12.1.2.4	Fill in some data	32
12.1.2.5	Sample document	32
12.1.2.5.1	Required imports	33

12.1.2.5.2	Index definition	33
12.1.2.5.2.1	Settings	33
12.1.2.5.2.2	Document index	33
12.1.2.5.3	Custom analyzers	34
12.1.2.5.4	Document definition	34
12.1.2.6	Syncing Django's database with Elasticsearch indexes	35
12.1.2.6.1	Full database sync	35
12.1.2.6.2	Sample partial sync (using custom signals)	35
12.1.2.6.2.1	Required imports	36
12.1.2.6.2.2	Update book index on related model change	36
12.1.2.6.2.3	Update book index on related model removal	36
12.1.2.7	Sample serializer	37
12.1.2.7.1	Required imports	37
12.1.2.7.2	Serializer definition	37
12.1.2.8	ViewSet definition	39
12.1.2.8.1	Required imports	39
12.1.2.8.2	ViewSet definition	40
12.1.2.9	URLs	42
12.1.2.9.1	Required imports	42
12.1.2.9.2	Router definition	42
12.1.2.9.3	URL patterns	42
12.1.2.10	Check what you've done so far	42
12.1.2.10.1	URLs	42
12.1.2.10.2	Test in browser	43
12.1.3	Development and debugging	43
12.1.3.1	Profiling tools	43
12.1.3.1.1	Installation	43
12.1.3.1.2	Configuration	43
12.1.3.2	Debugging	44
12.2	Filter usage examples	44
12.2.1	Search	45
12.2.1.1	Search in all fields	45
12.2.1.2	Search a single term on specific field	45
12.2.1.3	Search for multiple terms	45
12.2.1.4	Search for multiple terms in specific fields	45
12.2.2	Filtering	46
12.2.2.1	Supported lookups	46
12.2.2.1.1	Native	46
12.2.2.1.1.1	term	46
12.2.2.1.1.2	terms	46
12.2.2.1.1.3	range	46
12.2.2.1.1.4	exists	46
12.2.2.1.1.5	prefix	46
12.2.2.1.1.6	wildcard	47
12.2.2.1.1.7	ids	47
12.2.2.1.2	Functional	47
12.2.2.1.2.1	contains	47
12.2.2.1.2.2	in	47
12.2.2.1.2.3	gt	47
12.2.2.1.2.4	gte	47
12.2.2.1.2.5	lt	47
12.2.2.1.2.6	lte	48
12.2.2.1.2.7	startswith	48
12.2.2.1.2.8	endswith	48

	12.2.2.1.2.9	isnull	48
	12.2.2.1.2.10	exclude	48
	12.2.3	Usage examples	48
12.3		Basic usage examples	48
	12.3.1	Example app	49
	12.3.1.1	Sample models	49
	12.3.1.2	Sample document	49
	12.3.1.3	Sample serializer	50
	12.3.1.4	Sample view	51
	12.3.1.5	Usage example	52
	12.3.1.5.1	Sample queries	52
	12.3.1.5.1.1	Search	52
	12.3.1.5.1.2	Filtering	53
	12.3.1.5.1.3	Ordering	53
12.4		Advanced usage examples	54
	12.4.1	Example app	55
	12.4.1.1	Sample models	55
	12.4.1.2	Sample document	57
	12.4.1.2.1	Index definition	57
	12.4.1.2.1.1	Settings	57
	12.4.1.2.1.2	Document index	58
	12.4.1.3	Sample serializer	59
	12.4.1.4	Sample view	60
	12.4.1.5	Usage example	62
	12.4.1.5.1	Sample queries	62
	12.4.1.5.1.1	Search	62
	12.4.1.5.1.2	Filtering	63
	12.4.1.5.1.3	Ordering	64
	12.4.1.6	Faceted search	64
	12.4.1.7	Suggestions	65
	12.4.1.7.1	Document definition	66
	12.4.1.7.2	ViewSet definition	67
	12.4.1.7.3	Sample requests/responses	68
	12.4.1.8	Pagination	70
	12.4.1.8.1	Page number pagination	70
	12.4.1.8.2	Limit/offset pagination	70
12.5		Various handy helpers	71
	12.5.1	More like this	71
12.6		Release history and notes	71
	12.6.1	0.3.12	72
	12.6.2	0.3.11	72
	12.6.3	0.3.10	72
	12.6.4	0.3.9	72
	12.6.5	0.3.8	72
	12.6.6	0.3.7	72
	12.6.7	0.3.6	72
	12.6.8	0.3.5	73
	12.6.9	0.3.4	73
	12.6.10	0.3.3	73
	12.6.11	0.3.2	73
	12.6.12	0.3.1	73
	12.6.13	0.3	73
	12.6.14	0.2.6	73
	12.6.15	0.2.5	73

12.6.16	0.2.4	74
12.6.17	0.2.3	74
12.6.18	0.2.2	74
12.6.19	0.2.1	74
12.6.20	0.2	74
12.6.21	0.1.8	74
12.6.22	0.1.7	74
12.6.23	0.1.6	75
12.6.24	0.1.5	75
12.6.25	0.1.4	75
12.6.26	0.1.3	75
12.6.27	0.1.2	75
12.6.28	0.1.1	75
12.6.29	0.1	76
12.7	django_elasticsearch_dsl_drf package	76
12.7.1	Subpackages	76
12.7.1.1	django_elasticsearch_dsl_drf.filter_backends package	76
12.7.1.1.1	Subpackages	76
12.7.1.1.1.1	django_elasticsearch_dsl_drf.filter_backends.filtering package	76
12.7.1.1.1.2	Submodules	76
12.7.1.1.1.3	django_elasticsearch_dsl_drf.filter_backends.filtering.common module	76
12.7.1.1.1.4	django_elasticsearch_dsl_drf.filter_backends.filtering.ids module	81
12.7.1.1.1.5	Module contents	83
12.7.1.1.2	Submodules	83
12.7.1.1.3	django_elasticsearch_dsl_drf.filter_backends.faceted_search module	83
12.7.1.1.4	django_elasticsearch_dsl_drf.filter_backends.mixins module	85
12.7.1.1.5	django_elasticsearch_dsl_drf.filter_backends.ordering module	86
12.7.1.1.6	django_elasticsearch_dsl_drf.filter_backends.search module	87
12.7.1.1.7	django_elasticsearch_dsl_drf.filter_backends.suggester module	88
12.7.1.1.8	Module contents	92
12.7.1.2	django_elasticsearch_dsl_drf.tests package	92
12.7.1.2.1	Submodules	92
12.7.1.2.2	django_elasticsearch_dsl_drf.tests.base module	92
12.7.1.2.3	django_elasticsearch_dsl_drf.tests.test_faceted_search module	93
12.7.1.2.4	django_elasticsearch_dsl_drf.tests.test_filtering module	93
12.7.1.2.5	django_elasticsearch_dsl_drf.tests.test_helpers module	96
12.7.1.2.6	django_elasticsearch_dsl_drf.tests.test_ordering module	96
12.7.1.2.7	django_elasticsearch_dsl_drf.tests.test_pagination module	97
12.7.1.2.8	django_elasticsearch_dsl_drf.tests.test_search module	97
12.7.1.2.9	django_elasticsearch_dsl_drf.tests.test_suggesters module	97
12.7.1.2.10	django_elasticsearch_dsl_drf.tests.test_views module	98
12.7.1.2.11	Module contents	98
12.7.2	Submodules	98
12.7.3	django_elasticsearch_dsl_drf.apps module	98
12.7.4	django_elasticsearch_dsl_drf.compat module	98
12.7.5	django_elasticsearch_dsl_drf.constants module	99
12.7.6	django_elasticsearch_dsl_drf.helpers module	99
12.7.7	django_elasticsearch_dsl_drf.pagination module	100
12.7.8	django_elasticsearch_dsl_drf.serializers module	101
12.7.9	django_elasticsearch_dsl_drf.utils module	103
12.7.10	django_elasticsearch_dsl_drf.views module	103
12.7.11	django_elasticsearch_dsl_drf.viewsets module	103
12.7.12	Module contents	104

13 Indices and tables	105
Python Module Index	107

Integrate Elasticsearch DSL with Django REST framework in the shortest way possible, with least efforts possible.

Package provides views, serializers, filter backends, pagination and other handy add-ons.

You are expected to use `django-elasticsearch-dsl` for defining your Elasticsearch documents.

CHAPTER 1

Prerequisites

- Django 1.8, 1.9, 1.10 and 1.11.
- Python 2.7, 3.4, 3.5, 3.6
- Elasticsearch 2.x, 5.x

CHAPTER 2

Dependencies

- `django-elasticsearch-dsl`
- `djangoRESTframework`

CHAPTER 3

Documentation

Documentation is available on [Read the Docs](#).

Main features and highlights

- *Dynamic serializer for Documents.*
- *Search filter backend.*
- *Ordering filter backend.*
- *Filtering filter backend* (big variety of native- and functional- query lookups, such as `gt`, `gte`, `lt`, `lte`, `endswith`, `contains`, `wildcard`, `exists`, `exclude`, `isnull`, `range`, `in`, `geo_distance`, `term` and `terms` is implemented).
- *Faceted search filter backend.*
- *Suggester filter backend.*
- *Pagination (Page number and limit/offset pagination).*
- *Ids filter backend.*

Installation

1. Install latest stable version from PyPI:

```
pip install django-elasticsearch-dsl-drf
```

or latest stable version from GitHub:

```
pip install https://github.com/barseghyanartur/django-elasticsearch-dsl-drf/  
↪archive/stable.tar.gz
```

or latest stable version from BitBucket:

```
pip install https://bitbucket.org/barseghyanartur/django-elasticsearch-dsl-drf/  
↪get/stable.tar.gz
```

2. Add `rest_framework`, `django_elasticsearch_dsl` and `django_elasticsearch_dsl_drf` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (  
    # ...  
    # REST framework  
    'rest_framework',  
  
    # Django Elasticsearch integration  
    'django_elasticsearch_dsl',  
  
    # Django REST framework Elasticsearch integration (this package)  
    'django_elasticsearch_dsl_drf',  
    # ...  
)
```


CHAPTER 6

Quick start

Perhaps the easiest way to get acquainted with `django-elasticsearch-dsl-drf` is to read the [*quick start tutorial*](#).

See it as a guide of diving into integration of Elasticsearch with Django with very low knowledge entry level.

CHAPTER 7

Testing

Project is covered with tests.

To test with all supported Python/Django versions type:

```
tox
```

To test against specific environment, type:

```
tox -e py36-django110
```

To test just your working environment type:

```
./runtests.py
```

To run a single test in your working environment type:

```
./runtests.py src/django_elasticsearch_dsl_drf/tests/test_filtering.py
```

Or:

```
./manage.py test django_elasticsearch_dsl_drf.tests.test_ordering
```

It's assumed that you have all the requirements installed. If not, first install the test requirements:

```
pip install -r examples/requirements/test.txt
```

Writing documentation

Keep the following hierarchy.

```
=====  
title  
=====  
  
header  
=====  
  
sub-header  
-----  
  
sub-sub-header  
~~~~~  
  
sub-sub-sub-header  
^^^^^  
  
sub-sub-sub-sub-header  
+++++  
  
sub-sub-sub-sub-sub-header  
*****
```


CHAPTER 9

License

GPL 2.0/LGPL 2.1

CHAPTER 10

Support

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

CHAPTER 11

Author

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER 12

Project documentation

Contents:

Table of Contents

- *django-elasticsearch-dsl-drf*
 - *Prerequisites*
 - *Dependencies*
 - *Documentation*
 - *Main features and highlights*
 - *Installation*
 - *Quick start*
 - *Testing*
 - *Writing documentation*
 - *License*
 - *Support*
 - *Author*
 - *Project documentation*
 - *Indices and tables*

Quick start

The best way to get acquainted with `django-elasticsearch-dsl-drf`.

See it as a guide of diving into integration of Elasticsearch with Django with very low knowledge entry level.

Contents:

Table of Contents

- *Quick start*
 - *Installation*
 - *Example app*
 - * *Sample models*
 - *Required imports*
 - *Book statuses*
 - *Publisher model*
 - *Author model*
 - *Tag model*
 - *Book model*
 - * *Admin classes*
 - * *Create database tables*
 - * *Fill in some data*
 - * *Sample document*
 - *Required imports*
 - *Index definition*
 - *Settings*
 - *Document index*
 - *Custom analyzers*
 - *Document definition*
 - * *Syncing Django's database with Elasticsearch indexes*
 - *Full database sync*
 - *Sample partial sync (using custom signals)*
 - *Required imports*
 - *Update book index on related model change*
 - *Update book index on related model removal*
 - * *Sample serializer*
 - *Required imports*
 - *Serializer definition*
 - * *ViewSet definition*
 - *Required imports*
 - *ViewSet definition*

- * *URLs*
 - *Required imports*
 - *Router definition*
 - *URL patterns*
- * *Check what you've done so far*
 - *URLs*
 - *Test in browser*
- *Development and debugging*
 - * *Profiling tools*
 - *Installation*
 - *Configuration*
 - * *Debugging*

Installation

1. Install latest stable version from PyPI:

```
pip install django-elasticsearch-dsl-drf
```

2. Add `rest_framework`, `django_elasticsearch_dsl` and `django_elasticsearch_dsl_drf` to `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    # ...
    # REST framework
    'rest_framework',

    # Django Elasticsearch integration
    'django_elasticsearch_dsl',

    # Django REST framework Elasticsearch integration (this package)
    'django_elasticsearch_dsl_drf',
    # ...
)
```

3. Basic Django REST framework and `django-elasticsearch-dsl` configuration:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ),
    'DEFAULT_PAGINATION_CLASS':
        'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 100,
    'ORDERING_PARAM': 'ordering',
}

# Elasticsearch configuration
```

```
ELASTICSEARCH_DSL = {
    'default': {
        'hosts': 'localhost:9200'
    },
}
```

Example app

To get started, let's imagine we have a simple book register with a couple of models.

- *Publisher model*: The book publisher model. Each book might have only one publisher (ForeignKey relation).
- *Author model*: The book author model. Each book might have unlimited number of authors (ManyToMany relation).
- *Tag model*: The tag model. Each book might have unlimited number of tags (ManyToMany relation).
- *Book model*: The book model.

To keep things separate, our Django models will reside in the `books` app. Elasticsearch documents and Django REST framework views will be defined in a `search_indexes` app. Both of the apps should be added to the `INSTALLED_APPS`.

```
INSTALLED_APPS = (
    # ...
    'books', # Books application
    'search_indexes', # Elasticsearch integration with the Django
                    # REST framework
    # ...
)
```

Sample models

Content of the `books/models.py` file. Additionally, see the code comments.

Required imports

Imports required for model definition.

books/models.py

```
import json

from django.conf import settings
from django.db import models
from django.utils.translation import ugettext, ugettext_lazy as _

from six import python_2_unicode_compatible
```

Book statuses

books/models.py

```
# States indicate the publishing status of the book. Publishing might
# be in-progress, not yet published, published, rejected, etc.
BOOK_PUBLISHING_STATUS_PUBLISHED = 'published'
BOOK_PUBLISHING_STATUS_NOT_PUBLISHED = 'not_published'
BOOK_PUBLISHING_STATUS_IN_PROGRESS = 'in_progress'
BOOK_PUBLISHING_STATUS_CANCELLED = 'cancelled'
BOOK_PUBLISHING_STATUS_REJECTED = 'rejected'
BOOK_PUBLISHING_STATUS_CHOICES = (
    (BOOK_PUBLISHING_STATUS_PUBLISHED, "Published"),
    (BOOK_PUBLISHING_STATUS_NOT_PUBLISHED, "Not published"),
    (BOOK_PUBLISHING_STATUS_IN_PROGRESS, "In progress"),
    (BOOK_PUBLISHING_STATUS_CANCELLED, "Cancelled"),
    (BOOK_PUBLISHING_STATUS_REJECTED, "Rejected"),
)
BOOK_PUBLISHING_STATUS_DEFAULT = BOOK_PUBLISHING_STATUS_PUBLISHED
```

Publisher model

books/models.py

```
@python_2_unicode_compatible
class Publisher(models.Model):
    """Publisher."""

    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()

    class Meta(object):
        """Meta options."""

        ordering = ["id"]

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

Author model

books/models.py

```
@python_2_unicode_compatible
class Author(models.Model):
    """Author."""

    salutation = models.CharField(max_length=10)
    name = models.CharField(max_length=200)
    email = models.EmailField()
    headshot = models.ImageField(upload_to='authors', null=True, blank=True)

    class Meta(object):
        """Meta options."""
```

```

        ordering = ["id"]

    def __str__(self):
        return self.name

```

Tag model

books/models.py

```

class Tag(models.Model):
    """Simple tag model."""

    title = models.CharField(max_length=255, unique=True)

    class Meta(object):
        """Meta options."""

        verbose_name = _("Tag")
        verbose_name_plural = _("Tags")

    def __str__(self):
        return self.title

```

Book model

books/models.py

```

@python_2_unicode_compatible
class Book(models.Model):
    """Book."""

    title = models.CharField(max_length=100)
    description = models.TextField(null=True, blank=True)
    summary = models.TextField(null=True, blank=True)
    authors = models.ManyToManyField('books.Author', related_name='books')
    publisher = models.ForeignKey(Publisher, related_name='books')
    publication_date = models.DateField()
    state = models.CharField(max_length=100,
                             choices=BOOK_PUBLISHING_STATUS_CHOICES,
                             default=BOOK_PUBLISHING_STATUS_DEFAULT)
    isbn = models.CharField(max_length=100, unique=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    pages = models.PositiveIntegerField(default=200)
    stock_count = models.PositiveIntegerField(default=30)
    tags = models.ManyToManyField('books.Tag',
                                  related_name='books',
                                  blank=True)

    class Meta(object):
        """Meta options."""

        ordering = ["isbn"]

```

```

def __str__(self):
    return self.title

# The only publisher information we're going to need in our document
# is the publisher name. Since publisher isn't a required field,
# we define a property on a model level to avoid indexing errors on
# non-existing relation.
@property
def publisher_indexing(self):
    """Publisher for indexing.

    Used in Elasticsearch indexing.
    """
    if self.publisher is not None:
        return self.publisher.name

# As of tags, again, we only need a flat list of tag names, on which
# we can filter. Therefore, we define a property on a model level,
# which will return a JSON dumped list of tags relevant to the
# current book model object.
@property
def tags_indexing(self):
    """Tags for indexing.

    Used in Elasticsearch indexing.
    """
    return json.dumps([tag.title for tag in self.tags.all()])

```

Admin classes

This is just trivial. A couple of correspondent admin classes in order to be able to fill some data.

books/admin.py

```

from django.contrib import admin

from .models import *

@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    """Book admin."""

    list_display = ('title', 'isbn', 'price', 'publication_date')
    search_fields = ('title',)
    filter_horizontal = ('authors', 'tags',)

@admin.register(Author)
class AuthorAdmin(admin.ModelAdmin):
    """Author admin."""

    list_display = ('name', 'email',)
    search_fields = ('name',)

@admin.register(Publisher)

```

```
class PublisherAdmin(admin.ModelAdmin):
    """Publisher admin."""

    list_display = ('name',)
    search_fields = ('name',)

@admin.register(Tag)
class TagAdmin(admin.ModelAdmin):
    """Tag admin."""

    list_display = ('title',)
    search_fields = ('title',)
```

Create database tables

For now, just run the migrations to create the database tables.

```
./manage.py makemigrations books
./manage.py migrate books
```

Fill in some data

If you have followed the instructions, you should now be able to log into the Django admin and create a dozen of Book/Author/Publisher/Tag records in admin.

```
http://localhost:8000/admin/books/publisher/
http://localhost:8000/admin/books/author/
http://localhost:8000/admin/books/tag/
http://localhost:8000/admin/books/book/
```

Once you've done that, proceed to the next step.

Sample document

In Elasticsearch, a document is a basic unit of information that can be indexed. For example, you can have a document for a single customer, another document for a single product, and yet another for a single order. This document is expressed in JSON (JavaScript Object Notation) which is an ubiquitous internet data interchange format.

Within an index/type, you can store as many documents as you want. Note that although a document physically resides in an index, a document actually must be indexed/assigned to a type inside an index.

Simply said, you could see an Elasticsearch index as a database and a document as a database table (which makes a Document definition in Elasticsearch DSL similar to a Django Model definition).

Often, complex SQL model structures are flattened in Elasticsearch indexes/documents. Nested relations are denormalized.

In our example, all 4 models (Author, Publisher, Tag, Book) would be flattened into a single `BookDocument`, which would hold all the required information.

Content of the `search_indexes/documents/book.py` file. Additionally, see the code comments.

Required imports

search_indexes/documents/book.py

```

from django.conf import settings
from django_elasticsearch_dsl import DocType, Index, fields
from elasticsearch_dsl import analyzer

from books.models import Book

```

Index definition

To separate dev/test/staging/production indexes, the following approach is recommended.

Settings

settings/base.py

```

# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.book': 'book',
    'search_indexes.documents.publisher': 'publisher',
}

```

settings/testing.py

```

# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.book': 'test_book',
    'search_indexes.documents.publisher': 'test_publisher',
}

```

settings/production.py

```

# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.book': 'prod_book',
    'search_indexes.documents.publisher': 'prod_publisher',
}

```

Document index

search_indexes/documents/books.py

```

# Name of the Elasticsearch index
INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

# See Elasticsearch Indices API reference for available settings
INDEX.settings(
    number_of_shards=1,
    number_of_replicas=1
)

```

Custom analyzers

```
html_strip = analyzer(  
    'html_strip',  
    tokenizer="standard",  
    filter=["standard", "lowercase", "stop", "snowball"],  
    char_filter=["html_strip"]  
)
```

Document definition

search_indexes/documents/book.py

```
@INDEX.doc_type  
class BookDocument(DocType):  
    """Book Elasticsearch document."""  
  
    id = fields.IntegerField(attr='id')  
  
    title = fields.StringField(  
        analyzer=html_strip,  
        fields={  
            'raw': fields.StringField(analyzer='keyword'),  
        }  
    )  
  
    description = fields.StringField(  
        analyzer=html_strip,  
        fields={  
            'raw': fields.StringField(analyzer='keyword'),  
        }  
    )  
  
    summary = fields.StringField(  
        analyzer=html_strip,  
        fields={  
            'raw': fields.StringField(analyzer='keyword'),  
        }  
    )  
  
    publisher = fields.StringField(  
        attr='publisher_indexing',  
        analyzer=html_strip,  
        fields={  
            'raw': fields.StringField(analyzer='keyword'),  
        }  
    )  
  
    publication_date = fields.DateField()  
  
    state = fields.StringField(  
        analyzer=html_strip,  
        fields={  
            'raw': fields.StringField(analyzer='keyword'),  
        }  
    )
```

```

isbn = fields.StringField(
    analyzer=html_strip,
    fields={
        'raw': fields.StringField(analyzer='keyword'),
    }
)

price = fields.FloatField()

pages = fields.IntegerField()

stock_count = fields.IntegerField()

tags = fields.StringField(
    attr='tags_indexing',
    analyzer=html_strip,
    fields={
        'raw': fields.StringField(analyzer='keyword', multi=True),
    },
    multi=True
)

class Meta(object):
    """Meta options."""

    model = Book  # The model associate with this DocType

```

Syncing Django's database with Elasticsearch indexes

So far, we have a couple of Django models and a single (decentralized) Elasticsearch index/document (Book).

Full database sync

The excellent `django-elasticsearch-dsl` library makes a good job of keeping the Book index fresh. It makes use of signals, so whenever the Book model is changed, the correspondent BookDocument indexes would be updated.

To simply run the full sync between Django's database and Elasticsearch, do as follows:

1. Create Elasticsearch indexes:

```
./manage.py search_index --create -f
```

2. Sync the data:

```
./manage.py search_index --populate -f
```

However, in case if a Tag, Publisher or Author models change, the Book index would not be automatically updated.

Sample partial sync (using custom signals)

In order to keep indexes fresh, you will have to write a couple of simple lines of code (using Django's signals). Whenever a change is made to any of the Tag, Publisher or Author models, we're going to update the correspondent BookDocument index.

Required imports

search_indexes/signals.py

```
from django.db.models.signals import post_save, post_delete
from django.dispatch import receiver

from django_elasticsearch_dsl.registries import registry
```

Update book index on related model change

search_indexes/signals.py

```
@receiver(post_save)
def update_document(sender, **kwargs):
    """Update document on added/changed records.

    Update Book document index if related `books.Publisher` (`publisher`),
    `books.Author` (`authors`), `books.Tag` (`tags`) fields have been updated
    in the database.
    """
    app_label = sender._meta.app_label
    model_name = sender._meta.model_name
    instance = kwargs['instance']

    if app_label == 'book':
        # If it is `books.Publisher` that is being updated.
        if model_name == 'publisher':
            instances = instance.books.all()
            for _instance in instances:
                registry.update(_instance)

        # If it is `books.Author` that is being updated.
        if model_name == 'author':
            instances = instance.books.all()
            for _instance in instances:
                registry.update(_instance)

        # If it is `books.Tag` that is being updated.
        if model_name == 'tag':
            instances = instance.books.all()
            for _instance in instances:
                registry.update(_instance)
```

Update book index on related model removal

search_indexes/signals.py

```
@receiver(post_delete)
def delete_document(sender, **kwargs):
    """Update document on deleted records.

    Updates Book document from index if related `books.Publisher`
    (`publisher`), `books.Author` (`authors`), `books.Tag` (`tags`) fields
```

```

have been removed from database.
"""

app_label = sender._meta.app_label
model_name = sender._meta.model_name
instance = kwargs['instance']

if app_label == 'books':
    # If it is `books.Publisher` that is being updated.
    if model_name == 'publisher':
        instances = instance.books.all()
        for _instance in instances:
            registry.update(_instance)
            # registry.delete(_instance, raise_on_error=False)

    # If it is `books.Author` that is being updated.
    if model_name == 'author':
        instances = instance.books.all()
        for _instance in instances:
            registry.update(_instance)
            # registry.delete(_instance, raise_on_error=False)

    # If it is `books.Tag` that is being updated.
    if model_name == 'tag':
        instances = instance.books.all()
        for _instance in instances:
            registry.update(_instance)
            # registry.delete(_instance, raise_on_error=False)

```

Sample serializer

At this step we're going to define a serializer to be used in the Django REST framework ViewSet.

Content of the `search_indexes/serializers.py` file. Additionally, see the code comments.

Required imports

search_indexes/serializers.py

```

import json

from rest_framework import serializers
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from .documents import BookDocument

```

Serializer definition

Simplest way to create a serializer, is to just specify which fields are needed to be serialized and leave it further to the dynamic serializer.

search_indexes/serializers.py

```
class BookDocumentSerializer(DocumentSerializer):
    """Serializer for the Book document."""

    tags = serializers.SerializerMethodField()

    class Meta(object):
        """Meta options."""

        # Specify the correspondent document class
        document = BookDocument

        # List the serializer fields. Note, that the order of the fields
        # is preserved in the ViewSet.
        fields = (
            'id',
            'title',
            'description',
            'summary',
            'publisher',
            'publication_date',
            'state',
            'isbn',
            'price',
            'pages',
            'stock_count',
            'tags',
        )

    def get_tags(self, obj):
        """Get tags."""
        return json.loads(obj.tags)
```

However, if dynamic serializer doesn't work for your or you want to customize too many things, you are free to use standard Serializer class of the Django REST framework.

search_indexes/serializers.py

```
class BookDocumentSerializer(serializers.Serializer):
    """Serializer for the Book document."""

    id = serializers.IntegerField(read_only=True)

    title = serializers.CharField(read_only=True)
    description = serializers.CharField(read_only=True)
    summary = serializers.CharField(read_only=True)

    publisher = serializers.CharField(read_only=True)
    publication_date = serializers.DateField(read_only=True)
    state = serializers.CharField(read_only=True)
    isbn = serializers.CharField(read_only=True)
    price = serializers.FloatField(read_only=True)
    pages = serializers.IntegerField(read_only=True)
    stock_count = serializers.IntegerField(read_only=True)
    tags = serializers.SerializerMethodField()

    class Meta(object):
        """Meta options."""
```

```

# List the serializer fields. Note, that the order of the fields
# is preserved in the ViewSet.
fields = (
    'id',
    'title',
    'description',
    'summary',
    'publisher',
    'publication_date',
    'state',
    'isbn',
    'price',
    'pages',
    'stock_count',
    'tags',
)

def get_tags(self, obj):
    """Get tags."""
    return json.loads(obj.tags)

```

ViewSet definition

At this step, we're going to define Django REST framework ViewSets.

Content of the `search_indexes/viewsets.py` file. Additionally, see the code comments.

Required imports

search_indexes/viewsets.py

```

from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_TERMS,
    LOOKUP_FILTER_RANGE,
    LOOKUP_FILTER_PREFIX,
    LOOKUP_FILTER_WILDCARD,
    LOOKUP_QUERY_IN,
    LOOKUP_QUERY_GT,
    LOOKUP_QUERY_GTE,
    LOOKUP_QUERY_LT,
    LOOKUP_QUERY_LTE,
    LOOKUP_QUERY_EXCLUDE,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    IdsFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet

from .documents import BookDocument, PublisherDocument
from .serializers import BookDocumentSerializer

```

ViewSet definition

search_indexes/viewsets.py

```
class BookDocumentView(BaseDocumentViewSet):
    """The BookDocument view."""

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        IdsFilterBackend,
        OrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'title',
        'description',
        'summary',
    )
    # Define filter fields
    filter_fields = {
        'id': {
            'field': 'id',
            # Note, that we limit the lookups of id field in this example,
            # to `range`, `in`, `gt`, `gte`, `lt` and `lte` filters.
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_GT,
                LOOKUP_QUERY_GTE,
                LOOKUP_QUERY_LT,
                LOOKUP_QUERY_LTE,
            ],
        },
        'title': 'title.raw',
        'publisher': 'publisher.raw',
        'publication_date': 'publication_date',
        'state': 'state.raw',
        'isbn': 'isbn.raw',
        'price': {
            'field': 'price.raw',
            # Note, that we limit the lookups of `price` field in this
            # example, to `range`, `gt`, `gte`, `lt` and `lte` filters.
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_GT,
                LOOKUP_QUERY_GTE,
                LOOKUP_QUERY_LT,
                LOOKUP_QUERY_LTE,
            ],
        },
        'pages': {
            'field': 'pages',
            # Note, that we limit the lookups of `pages` field in this
            # example, to `range`, `gt`, `gte`, `lt` and `lte` filters.
        },
    }
```



```

        'lookups': [
            LOOKUP_FILTER_RANGE,
            LOOKUP_QUERY_GT,
            LOOKUP_QUERY_GTE,
            LOOKUP_QUERY_LT,
            LOOKUP_QUERY_LTE,
        ],
    },
    'stock_count': {
        'field': 'stock_count',
        # Note, that we limit the lookups of `stock_count` field in
        # this example, to `range`, `gt`, `gte`, `lt` and `lte`
        # filters.
        'lookups': [
            LOOKUP_FILTER_RANGE,
            LOOKUP_QUERY_GT,
            LOOKUP_QUERY_GTE,
            LOOKUP_QUERY_LT,
            LOOKUP_QUERY_LTE,
        ],
    },
    'tags': {
        'field': 'tags',
        # Note, that we limit the lookups of `tags` field in
        # this example, to `terms`, `prefix`, `wildcard`, `in` and
        # `exclude` filters.
        'lookups': [
            LOOKUP_FILTER_TERMS,
            LOOKUP_FILTER_PREFIX,
            LOOKUP_FILTER_WILDCARD,
            LOOKUP_QUERY_IN,
            LOOKUP_QUERY_EXCLUDE,
        ],
    },
    'tags.raw': {
        'field': 'tags.raw',
        # Note, that we limit the lookups of `tags.raw` field in
        # this example, to `terms`, `prefix`, `wildcard`, `in` and
        # `exclude` filters.
        'lookups': [
            LOOKUP_FILTER_TERMS,
            LOOKUP_FILTER_PREFIX,
            LOOKUP_FILTER_WILDCARD,
            LOOKUP_QUERY_IN,
            LOOKUP_QUERY_EXCLUDE,
        ],
    },
}

# Define ordering fields
ordering_fields = {
    'id': 'id',
    'title': 'title.raw',
    'price': 'price.raw',
    'state': 'state.raw',
    'publication_date': 'publication_date',
}

# Specify default ordering
ordering = ('id', 'title', 'price',)

```

URLs

At this step, we're going to define url patterns.

Content of the `search_indexes/urls.py` file. Additionally, see the code comments.

Required imports

search_indexes/urls.py

```
from django.conf.urls import url, include
from rest_framework_extensions.routers import ExtendedDefaultRouter
from .views import BookDocumentView
```

Router definition

search_indexes/urls.py

```
router = ExtendedDefaultRouter()
books = router.register(r'books',
                       BookDocumentView,
                       base_name='bookdocument')
```

URL patterns

search_indexes/urls.py

```
urlpatterns = [
    url(r'^$', include(router.urls)),
]
```

Check what you've done so far

At this point, you are one step away from a working example of integrating Elasticsearch DSL with Django.

URLs

If you didn't add the urls of the `search_indexes` example application to your project's global url patterns, make sure to do it now.

```
from django.conf.urls import include, url
from search_indexes import urls as search_index_urls

urlpatterns = [
    # ...
    # Search URLs
    url(r'^search/', include(search_index_urls)),
    # ...
]
```

Test in browser

Open the following URL in your browser.

```
http://localhost:8000/search/books/
```

Perform a number of lookups:

```
http://localhost:8001/search/books/?ids=54|55|56
http://localhost:8001/search/books/?summary__contains=photography
http://localhost:8001/search/books/?tags__contains=ython
http://localhost:8001/search/books/?state=published
http://localhost:8001/search/books/?pages__gt=10&pages__lt=30
```

Development and debugging

Profiling tools

Looking for profiling tools for Elasticsearch?

Try [django-elasticsearch-debug-toolbar](#) package. It's implemented as a panel for the well known [Django Debug Toolbar](#) and gives you full insights on what's happening on the side of Elasticsearch.

Installation

```
pip install django-debug-toolbar
pip install django-elasticsearch-debug-toolbar
```

Configuration

Change your development settings in the following way:

settings/dev.py

```
MIDDLEWARE_CLASSES += (
    'debug_toolbar.middleware.DebugToolbarMiddleware',
    'debug_toolbar_force.middleware.ForceDebugToolbarMiddleware',
)

INSTALLED_APPS += (
    'debug_toolbar',
    'elastic_panel',
)

DEBUG_TOOLBAR_CONFIG = {
    'INTERCEPT_REDIRECTS': False,
}

DEBUG_TOOLBAR_PANELS = (
    # Defaults
    'debug_toolbar.panels.versions.VersionsPanel',
    'debug_toolbar.panels.timer.TimerPanel',
    'debug_toolbar.panels.settings.SettingsPanel',
)
```

```
'debug_toolbar.panels.headers.HeadersPanel',
'debug_toolbar.panels.request.RequestPanel',
'debug_toolbar.panels.sql.SQLPanel',
'debug_toolbar.panels.staticfiles.StaticFilesPanel',
'debug_toolbar.panels.templates.TemplatesPanel',
'debug_toolbar.panels.cache.CachePanel',
'debug_toolbar.panels.signals.SignalsPanel',
'debug_toolbar.panels.logging.LoggingPanel',
'debug_toolbar.panels.redirects.RedirectsPanel',
# Additional
'elastic_panel.panel.ElasticDebugPanel',
)
```

Debugging

Although (the unbeatable) Kibana is strongly recommended for data analyses, there are other good tools worth mentioning. One of them is elasticsearch-head [Elasticsearch 2.x](#) plugin or a correspondent [Chrome extension](#) of the same plugin. You may find it very useful for quick data preview or testing Elasticsearch queries.

Filter usage examples

Example usage of filtering backends.

Contents:

Table of Contents

- *Filter usage examples*
 - *Search*
 - * *Search in all fields*
 - * *Search a single term on specific field*
 - * *Search for multiple terms*
 - * *Search for multiple terms in specific fields*
 - *Filtering*
 - * *Supported lookups*
 - *Native*
 - *term*
 - *terms*
 - *range*
 - *exists*
 - *prefix*
 - *wildcard*
 - *ids*

- *Functional*
- *contains*
- *in*
- *gt*
- *gte*
- *lt*
- *lte*
- *startswith*
- *endswith*
- *isnull*
- *exclude*
- *Usage examples*

Search

Query param name reserved for search is `search`. Make sure your models and documents do not have it as a field or attribute.

Multiple search terms are joined with OR.

Let's assume we have a number of Book items with fields `title`, `description` and `summary`.

Search in all fields

Search in all fields (`name`, `address`, `city`, `state_province` and `country`) for word “reilly”.

```
http://127.0.0.1:8080/search/publisher/?search=reilly
```

Search a single term on specific field

In order to search in specific field (`name`) for term “reilly”, add the field name separated with `|` to the search term.

```
http://127.0.0.1:8080/search/publisher/?search=name|reilly
```

Search for multiple terms

In order to search for multiple terms “reilly”, “bloomsbury” add multiple `search` query params.

```
http://127.0.0.1:8080/search/publisher/?search=reilly&search=bloomsbury
```

Search for multiple terms in specific fields

In order to search for multiple terms “reilly”, “bloomsbury” in specific fields add multiple `search` query params and field names separated with `|` to each of the search terms.

```
http://127.0.0.1:8080/search/publisher/?search=name|reilly&search=city|london
```

Filtering

Supported lookups

Native

The following native (to Elasticsearch) filters/lookups are implemented:

- *term*
- *terms*
- *range*
- *exists*
- *prefix*
- *wildcard*
- *regexp*
- *fuzzy*
- *type*
- *ids*

term

Find documents which contain the exact term specified in the field specified.

```
http://127.0.0.1:8080/search/books/?tags__term=education&tags__term=economy
```

terms

Find documents which contain any of the exact terms specified in the field specified.

range

Find documents where the field specified contains values (dates, numbers, or strings) in the range specified.

exists

Find documents where the field specified contains any non-null value.

prefix

Find documents where the field specified contains terms which begin with the exact prefix specified.

wildcard

Find documents where the field specified contains terms which match the pattern specified, where the pattern supports single character wildcards (?) and multi-character wildcards (*)

ids

Find documents with the specified type and IDs.

Functional

The following functional (non-native to Elasticsearch, but common in Django) filters/lookups are implemented:

- *contains*
- *in*
- *gt*
- *gte*
- *lt*
- *lte*
- *startswith*
- *endswith*
- *isnull*
- *exclude*

contains

Case-insensitive containment test.

in

In a given list.

gt

Greater than.

gte

Greater than or equal to.

lt

Less than.

lte

Less than or equal to.

startswith

Case-sensitive starts-with.

endswith

Case-sensitive ends-with.

isnull

Takes either True or False.

exclude

Returns a new query set of containing objects that do not match the given lookup parameters.

Usage examples

See the [example project](#) for sample models/views/serializers.

- [models](#)
- [documents](#)
- [serializers](#)
- [views](#)

Additionally, see:

- [Basic usage examples](#)
- [Advanced usage examples](#)
- [Misc usage examples](#)

Basic usage examples

Basic Django REST framework integration example

See the [example project](#) for sample models/views/serializers.

- [models](#)
- [documents](#)
- [serializers](#)
- [views](#)

Contents:

Table of Contents

- *Basic usage examples*
 - *Example app*
 - * *Sample models*
 - * *Sample document*
 - * *Sample serializer*
 - * *Sample view*
 - * *Usage example*
 - *Sample queries*
 - *Search*
 - *Filtering*
 - *Ordering*

Example app

Sample models

books/models.py:

```
class Publisher(models.Model):
    """Publisher."""

    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()

    class Meta(object):
        """Meta options."""

        ordering = ["id"]

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

Sample document

search_indexes/documents/publisher.py:

```
from django_elasticsearch_dsl import DocType, Index, fields
from elasticsearch_dsl import analyzer
```

```

from books.models import Publisher

# Name of the Elasticsearch index
PUBLISHER_INDEX = Index('publisher')
# See Elasticsearch Indices API reference for available settings
PUBLISHER_INDEX.settings(
    number_of_shards=1,
    number_of_replicas=1
)

@PUBLISHER_INDEX.doc_type
class PublisherDocument(DocType):
    """Publisher Elasticsearch document."""

    id = fields.IntegerField(attr='id')

    name = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
    info = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
    address = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
    city = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
    state_province = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
    country = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
    website = fields.StringField()

    class Meta(object):
        """Meta options."""

        model = Publisher # The model associate with this DocType

```

Sample serializer

search_indexes/serializers.py:

```

import json

from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

class PublisherDocumentSerializer(DocumentSerializer):
    """Serializer for Publisher document."""

    class Meta(object):
        """Meta options."""

        # Note, that since we're using a dynamic serializer,
        # we only have to declare fields that we want to be shown. If
        # somehow, dynamic serializer doesn't work for you, either extend
        # or declare your serializer explicitly.
        fields = (
            'id',
            'name',
            'info',
            'address',
            'city',
            'state_province',
            'country',
            'website',
        )

```

Sample view

search_indexes/views.py:

```

from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet

# Example app models
from search_indexes.documents.publisher import PublisherDocument
from search_indexes.serializers import PublisherDocumentSerializer

class PublisherDocumentView(BaseDocumentViewSet):
    """The PublisherDocument view."""

    document = PublisherDocument
    serializer_class = PublisherDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'name',
        'info',
        'address',
        'city',
    )

```

```

        'state_province',
        'country',
    )
    # Define filtering fields
    filter_fields = {
        'id': None,
        'name': 'name.raw',
        'city': 'city.raw',
        'state_province': 'state_province.raw',
        'country': 'country.raw',
    }
    # Define ordering fields
    ordering_fields = {
        'id': None,
        'name': None,
        'city': None,
        'country': None,
    }
    # Specify default ordering
    ordering = ('id', 'name',)

```

Usage example

Considering samples above, you should be able to perform the search, sorting and filtering actions described below.

Sample queries

Search

Query param name reserved for search is `search`. Make sure your models and documents do not have it as a field or attribute.

Multiple search terms are joined with OR.

Let's assume we have a number of `Book` items with fields `title`, `description` and `summary`.

Search in all fields

Search in all fields (`name`, `address`, `city`, `state_province` and `country`) for word “reilly”.

```
http://127.0.0.1:8080/search/publisher/?search=reilly
```

Search a single term on specific field

In order to search in specific field (`name`) for term “reilly”, add the field name separated with `|` to the search term.

```
http://127.0.0.1:8080/search/publisher/?search=name|reilly
```

Search for multiple terms

In order to search for multiple terms “reilly”, “bloomsbury” add multiple search query params.

```
http://127.0.0.1:8080/search/publisher/?search=reilly&search=bloomsbury
```

Search for multiple terms in specific fields

In order to search for multiple terms “reilly”, “bloomsbury” in specific fields add multiple search query params and field names separated with | to each of the search terms.

```
http://127.0.0.1:8080/search/publisher/?search=name|reilly&search=city|london
```

Filtering

Let’s assume we have a number of Publisher documents with in cities (Yerevan, Groningen, Amsterdam, London).

Multiple filter terms are joined with AND.

Filter documents by single field

Filter documents by field (city) “yerevan”.

```
http://127.0.0.1:8080/search/publisher/?city=yerevan
```

Filter documents by multiple fields

Filter documents by city “Yerevan” and “Groningen”.

```
http://127.0.0.1:8080/search/publisher/?city__in=yerevan|groningen
```

Filter document by a single field

Filter documents by (field country) “Armenia”.

```
http://127.0.0.1:8080/search/publisher/?country=armenia
```

Filter documents by multiple fields

Filter documents by multiple fields (field city) “Yerevan” and “Amsterdam” with use of functional in query filter.

```
http://127.0.0.1:8080/search/publisher/?city__in=yerevan|amsterdam
```

You can achieve the same effect by specifying multiple filters (city) “Yerevan” and “Amsterdam”. Note, that in this case multiple filter terms are joined with OR.

```
http://127.0.0.1:8080/search/publisher/?city=yerevan&city=amsterdam
```

If you want the same as above, but joined with AND, add __term to each lookup.

```
http://127.0.0.1:8080/search/publisher/?city__term=education&city__term=economy
```

Filter documents by a word part of a single field

Filter documents by a part word part in single field (city) “ondon”.

```
http://127.0.0.1:8080/search/publisher/?city__wildcard=*ondon
```

Ordering

The – prefix means ordering should be descending.

Order documents by field (ascending)

Filter documents by field city (ascending).

```
http://127.0.0.1:8080/search/publisher/?search=country|armenia&ordering=city
```

Order documents by field (descending)

Filter documents by field `country` (descending).

```
http://127.0.0.1:8080/search/publisher/?ordering=-country
```

Order documents by multiple fields

If you want to order by multiple fields, use multiple ordering query params. In the example below, documents would be ordered first by field `country` (descending), then by field `city` (ascending).

```
http://127.0.0.1:8080/search/publisher/?ordering=-country&ordering=city
```

Advanced usage examples

Advanced Django REST framework integration examples.

See the [example project](#) for sample models/views/serializers.

- [models](#)
- [documents](#)
- [serializers](#)
- [views](#)

Contents:

Table of Contents

- *Advanced usage examples*
 - *Example app*
 - * *Sample models*
 - * *Sample document*
 - *Index definition*
 - *Settings*
 - *Document index*
 - * *Sample serializer*
 - * *Sample view*
 - * *Usage example*
 - *Sample queries*
 - *Search*
 - *Filtering*
 - *Ordering*
 - * *Faceted search*

- * *Suggestions*
 - *Document definition*
 - *ViewSet definition*
 - *Sample requests/responses*
- * *Pagination*
 - *Page number pagination*
 - *Limit/offset pagination*

Example app

Sample models

books/models.py

```
import json

from django.conf import settings
from django.db import models
from django.utils.translation import ugettext, ugettext_lazy as _

from six import python_2_unicode_compatible

BOOK_PUBLISHING_STATUS_PUBLISHED = 'published'
BOOK_PUBLISHING_STATUS_NOT_PUBLISHED = 'not_published'
BOOK_PUBLISHING_STATUS_IN_PROGRESS = 'in_progress'
BOOK_PUBLISHING_STATUS_CANCELLED = 'cancelled'
BOOK_PUBLISHING_STATUS_REJECTED = 'rejected'
BOOK_PUBLISHING_STATUS_CHOICES = (
    (BOOK_PUBLISHING_STATUS_PUBLISHED, "Published"),
    (BOOK_PUBLISHING_STATUS_NOT_PUBLISHED, "Not published"),
    (BOOK_PUBLISHING_STATUS_IN_PROGRESS, "In progress"),
    (BOOK_PUBLISHING_STATUS_CANCELLED, "Cancelled"),
    (BOOK_PUBLISHING_STATUS_REJECTED, "Rejected"),
)
BOOK_PUBLISHING_STATUS_DEFAULT = BOOK_PUBLISHING_STATUS_PUBLISHED

@python_2_unicode_compatible
class Publisher(models.Model):
    """Publisher."""

    name = models.CharField(max_length=30)
    info = models.TextField(null=True, blank=True)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()

    class Meta(object):
        """Meta options."""
```

```

        ordering = ["id"]

    def __str__(self):
        return self.name

@python_2_unicode_compatible
class Author(models.Model):
    """Author."""

    salutation = models.CharField(max_length=10)
    name = models.CharField(max_length=200)
    email = models.EmailField()
    headshot = models.ImageField(upload_to='authors', null=True, blank=True)

    class Meta(object):
        """Meta options."""

        ordering = ["id"]

    def __str__(self):
        return self.name

class Tag(models.Model):
    """Simple tag model."""

    title = models.CharField(max_length=255, unique=True)

    class Meta(object):
        """Meta options."""

        verbose_name = _("Tag")
        verbose_name_plural = _("Tags")

    def __str__(self):
        return self.title

@python_2_unicode_compatible
class Book(models.Model):
    """Book."""

    title = models.CharField(max_length=100)
    description = models.TextField(null=True, blank=True)
    summary = models.TextField(null=True, blank=True)
    authors = models.ManyToManyField('books.Author', related_name='books')
    publisher = models.ForeignKey(Publisher, related_name='books')
    publication_date = models.DateField()
    state = models.CharField(max_length=100,
                             choices=BOOK_PUBLISHING_STATUS_CHOICES,
                             default=BOOK_PUBLISHING_STATUS_DEFAULT)
    isbn = models.CharField(max_length=100, unique=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    pages = models.PositiveIntegerField(default=200)
    stock_count = models.PositiveIntegerField(default=30)
    tags = models.ManyToManyField('books.Tag',
                                  related_name='books',

```



```

blank=True)

class Meta(object):
    """Meta options."""

    ordering = ["isbn"]

    def __str__(self):
        return self.title

    @property
    def publisher_indexing(self):
        """Publisher for indexing.

        Used in Elasticsearch indexing.
        """
        if self.publisher is not None:
            return self.publisher.name

    @property
    def tags_indexing(self):
        """Tags for indexing.

        Used in Elasticsearch indexing.
        """
        return json.dumps([tag.title for tag in self.tags.all()])

```

Sample document

Index definition

To separate dev/test/staging/production indexes, the following approach is recommended.

Settings

settings/base.py

```

# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.book': 'book',
    'search_indexes.documents.publisher': 'publisher',
}

```

settings/testing.py

```

# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.book': 'test_book',
    'search_indexes.documents.publisher': 'test_publisher',
}

```

settings/production.py

```
# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.book': 'prod_book',
    'search_indexes.documents.publisher': 'prod_publisher',
}
```

Document index

search_indexes/documents/book.py

```
from django.conf import settings
from django_elasticsearch_dsl import DocType, Index, fields
from elasticsearch_dsl import analyzer

from books.models import Book

# Name of the Elasticsearch index
INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

# See Elasticsearch Indices API reference for available settings
INDEX.settings(
    number_of_shards=1,
    number_of_replicas=1
)

html_strip = analyzer(
    'html_strip',
    tokenizer="standard",
    filter=["standard", "lowercase", "stop", "snowball"],
    char_filter=["html_strip"]
)

@INDEX.doc_type
class BookDocument(DocType):
    """Book Elasticsearch document."""

    id = fields.IntegerField(attr='id')

    title = fields.StringField(
        analyzer=html_strip,
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )

    description = fields.StringField(
        analyzer=html_strip,
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )

    summary = fields.StringField(
        analyzer=html_strip,
        fields={
```

```

        'raw': fields.StringField(analyzer='keyword'),
    }
)

publisher = fields.StringField(
    attr='publisher_indexing',
    analyzer=html_strip,
    fields={
        'raw': fields.StringField(analyzer='keyword'),
    }
)

publication_date = fields.DateField()

state = fields.StringField(
    analyzer=html_strip,
    fields={
        'raw': fields.StringField(analyzer='keyword'),
    }
)

isbn = fields.StringField(
    analyzer=html_strip,
    fields={
        'raw': fields.StringField(analyzer='keyword'),
    }
)

price = fields.FloatField()

pages = fields.IntegerField()

stock_count = fields.IntegerField()

tags = fields.StringField(
    attr='tags_indexing',
    analyzer=html_strip,
    fields={
        'raw': fields.StringField(analyzer='keyword', multi=True),
    },
    multi=True
)

class Meta(object):
    """Meta options."""

    model = Book  # The model associate with this DocType

```

Sample serializer

search_indexes/serializers.py

```

import json

from rest_framework import serializers

class TagSerializer(serializers.Serializer):

```

```

"""Helper serializer for the Tag field of the Book document."""

title = serializers.CharField()

class Meta(object):
    """Meta options."""

    fields = ('title',)
    read_only_fields = ('title',)

class BookDocumentSerializer(serializers.Serializer):
    """Serializer for the Book document."""

    id = serializers.SerializerMethodField()

    title = serializers.CharField(read_only=True)
    description = serializers.CharField(read_only=True)
    summary = serializers.CharField(read_only=True)

    publisher = serializers.CharField(read_only=True)
    publication_date = serializers.DateField(read_only=True)
    state = serializers.CharField(read_only=True)
    isbn = serializers.CharField(read_only=True)
    price = serializers.FloatField(read_only=True)
    pages = serializers.IntegerField(read_only=True)
    stock_count = serializers.IntegerField(read_only=True)
    tags = serializers.SerializerMethodField()

class Meta(object):
    """Meta options."""

    fields = (
        'id',
        'title',
        'description',
        'summary',
        'publisher',
        'publication_date',
        'state',
        'isbn',
        'price',
        'pages',
        'stock_count',
        'tags',
    )
    read_only_fields = fields

    def get_tags(self, obj):
        """Get tags."""
        return json.loads(obj.tags)

```

Sample view

search_indexes/viewsets.py

```

from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_TERMS,
    LOOKUP_FILTER_RANGE,
    LOOKUP_FILTER_PREFIX,
    LOOKUP_FILTER_WILDCARD,
    LOOKUP_QUERY_IN,
    LOOKUP_QUERY_EXCLUDE,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet

# Example app models
from search_indexes.documents.book import BookDocument
from search_indexes.serializers import BookDocumentSerializer

class BookDocumentView(BaseDocumentViewSet):
    """The BookDocument view."""

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'title',
        'description',
        'summary',
    )
    # Define filtering fields
    filter_fields = {
        'id': {
            'field': '_id',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_IN,
            ],
        },
        'publisher': 'publisher.raw',
        'publication_date': 'publication_date',
        'isbn': 'isbn.raw',
        'tags': {
            'field': 'tags',
            'lookups': [
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
        },
    },

```

```
    },
    'tags.raw': {
        'field': 'tags.raw',
        'lookups': [
            LOOKUP_FILTER_TERMS,
            LOOKUP_FILTER_PREFIX,
            LOOKUP_FILTER_WILDCARD,
            LOOKUP_QUERY_IN,
            LOOKUP_QUERY_EXCLUDE,
        ],
    },
}

# Define ordering fields
ordering_fields = {
    'id': 'id',
    'title': 'title.raw',
    'price': 'price.raw',
    'state': 'state.raw',
    'publication_date': 'publication_date',
}

# Specify default ordering
ordering = ('id', 'title',)
```

Usage example

Considering samples above, you should be able to perform the search, sorting and filtering actions described below.

Sample queries

Search

Query param name reserved for search is `search`. Make sure your models and documents do not have it as a field or attribute.

Multiple search terms are joined with OR.

Let's assume we have a number of `Book` items with fields `title`, `description` and `summary`.

Search in all fields

Search in all fields (`title`, `description` and `summary`) for word “education”.

```
http://127.0.0.1:8080/search/books/?search=education
```

Search a single term on specific field

In order to search in specific field (`title`) for term “education”, add the field name separated with `|` to the search term.

```
http://127.0.0.1:8080/search/books/?search=title|education
```

Search for multiple terms

In order to search for multiple terms “education”, “technology” add multiple `search` query params.

```
http://127.0.0.1:8080/search/books/?search=education&search=technology
```

Search for multiple terms on specific fields

In order to search for multiple terms “education”, “technology” in specific fields add multiple `search` query params and field names separated with `|` to each of the search terms.

```
http://127.0.0.1:8080/search/books/?search=title|education&search=summary|technology
```

Filtering

Let’s assume we have a number of Book documents with the tags (education, politics, economy, biology, climate, environment, internet, technology).

Multiple filter terms are joined with AND.

Filter documents by field

Filter documents by field (`state`) “published”.

```
http://127.0.0.1:8080/search/books/?state=published
```

Filter documents by multiple fields

Filter documents by field (`states`) “published” and “in_progress”.

```
http://127.0.0.1:8080/search/books/?state__in=published|in_progress
```

Filter document by a single field

Filter documents by (field `tag`) “education”.

```
http://127.0.0.1:8080/search/books/?tag=education
```

Filter documents by multiple fields

Filter documents by multiple fields (field `tags`) “education” and “economy” with use of functional `in` query filter.

```
http://127.0.0.1:8080/search/books/?tags__in=education|economy
```

You can achieve the same effect by specifying multiple fields (`tags`) “education” and “economy”. Note, that in this case multiple filter terms are joined with OR.

```
http://127.0.0.1:8080/search/books/?tags=education&tags=economy
```

If you want the same as above, but joined with AND, add `__term` to each lookup.

```
http://127.0.0.1:8080/search/books/?tags__term=education&tags__term=economy
```

Filter documents by a word part of a single field

Filter documents by a part word part in single field (`tags`). Word part should match both “technology” and “biology”.

```
http://127.0.0.1:8080/search/books/?tags__wildcard=*logy
```

Geo-distance filtering

Assuming that our Address model has a location, which is `GeoPointField`, filter documents by radius of 2km from the given location.

```
http://localhost:8000/search/address/?location__geo_distance=2km|43.53455243|-12.
↪2344243
```

Ordering

The `-` prefix means ordering should be descending.

Order documents by field (ascending)

Order documents by field `price` (ascending).

```
http://127.0.0.1:8080/search/books/?search=title|lorem&ordering=price
```

Order documents by field (descending)

Order documents by field `price` (descending).

```
http://127.0.0.1:8080/search/books/?search=title|lorem&ordering=-price
```

Order documents by multiple fields

If you want to order by multiple fields, use multiple ordering query params. In the example below, documents would be ordered first by field `publication_date` (descending), then by field `price` (ascending).

```
http://127.0.0.1:8080/search/books/?search=title|lorem&ordering=-publication_date&
↪ordering=price
```

Faceted search

In order to add faceted search support, we would have to extend our view set in the following way:

search_indexes/viewsets.py

```
# ...

from django_elasticsearch_dsl_drf.filter_backends import (
    # ...
    FacetedSearchFilterBackend,
)

# ...

from elasticsearch_dsl import (
    DateHistogramFacet,
    RangeFacet,
    TermsFacet,
)

# ...

class BookDocumentView(BaseDocumentViewSet):
    """The BookDocument view."""

    # ...

    filter_backends = [
```



```

# ...
FacetedSearchFilterBackend,
]

# ...

faceted_search_fields = {
    'state': 'state.raw', # By default, TermsFacet is used
    'publisher': {
        'field': 'publisher.raw',
        'facet': TermsFacet, # But we can define it explicitly
        'enabled': True,
    },
    'publication_date': {
        'field': 'publication_date',
        'facet': DateHistogramFacet,
        'options': {
            'interval': 'year',
        }
    },
    'pages_count': {
        'field': 'pages',
        'facet': RangeFacet,
        'options': {
            'ranges': [
                ("<10", (None, 10)),
                ("11-20", (11, 20)),
                ("20-50", (20, 50)),
                (">50", (50, None)),
            ]
        }
    },
}

# ...

```

Note, that none of the facets is enabled by default, unless you explicitly specify it to be enabled. That means, that you will have to add a query string `facet={facet_field_name}` for each of the facets you want to see in results.

In the example below, we show results with faceted `state` and `pages_count` facets.

```
http://127.0.0.1:8000/search/books/?facet=state&facet=pages_count
```

Suggestions

The suggest feature suggests similar looking terms based on a provided text by using a suggester.

Note: The `SuggesterFilterBackend` filter backend can be used in the `suggest` custom view action/route only. Usages outside of the `suggest` action/route are restricted.

There are three options available here: `term`, `phrase` and `completion`.

Note: Suggestion functionality is exclusive. Once you have queried the `SuggesterFilterBackend`, the latter will transform your current search query into suggestion search query (which is very different). Therefore, always add

it as the very last filter backend.

Document definition

To make use of suggestions, you should properly indexed your documents using `fields.CompletionField`.
search_indexes/documents/publisher.py

```
from django.conf import settings

from django_elasticsearch_dsl import DocType, Index, fields

from books.models import Publisher

# Name of the Elasticsearch index
INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

# See Elasticsearch Indices API reference for available settings
INDEX.settings(
    number_of_shards=1,
    number_of_replicas=1
)

@INDEX.doc_type
class PublisherDocument(DocType):
    """Publisher Elasticsearch document."""

    id = fields.IntegerField(attr='id')

    name = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
            'suggest': fields.CompletionField(),
        }
    )

    info = fields.StringField()

    address = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword')
        }
    )

    city = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
            'suggest': fields.CompletionField(),
        }
    )

    state_province = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
            'suggest': fields.CompletionField(),
        }
    )
```

```

    )

    country = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
            'suggest': fields.CompletionField(),
        }
    )

    website = fields.StringField()

    class Meta(object):
        """Meta options."""

        model = Publisher # The model associate with this DocType

```

After that the `name.suggest`, `city.suggest`, `state_province.suggest` and `country.suggest` fields would be available for suggestions feature.

ViewSet definition

In order to add suggestions support, we would have to extend our view set in the following way:

search_indexes/viewsets.py

```

# ...

from django_elasticsearch_dsl_drf.constants import (
    SUGGESTER_TERM,
    SUGGESTER_PHRASE,
    SUGGESTER_COMPLETION,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    # ...
    SuggesterFilterBackend,
)

# ...

class PublisherDocumentViewSet(BaseDocumentViewSet):
    """The PublisherDocument view."""

    document = PublisherDocument

    # ...

    filter_backends = [
        # ...
        SuggesterFilterBackend,
    ]

    # ...

    # Suggester fields
    suggester_fields = {
        'name_suggest': {
            'field': 'name.suggest',

```

```
        'suggesters': [
            SUGGESTER_TERM,
            SUGGESTER_PHRASE,
            SUGGESTER_COMPLETION,
        ],
    },
    'city_suggest': {
        'field': 'city.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'state_province_suggest': {
        'field': 'state_province.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'country_suggest': {
        'field': 'country.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
}
```

In the example below, we show suggestion results (auto-completion) for `country` field.

Sample requests/responses

Once you have extended your view set with `SuggesterFilterBackend` functionality, you can make use of the `suggest` custom action of your view set.

Request

```
GET http://127.0.0.1:8000/search/publishers/suggest/?country_suggest__completion=Ar
```

Response

```
{
  "_shards": {
    "failed": 0,
    "successful": 1,
    "total": 1
  },
  "country_suggest__completion": [
    {
      "options": [
        {
          "score": 1.0,
          "text": "Armenia"
        },
        {
          "score": 1.0,
          "text": "Argentina"
        }
      ]
    }
  ]
}
```

```

        "offset": 0,
        "length": 2,
        "text": "Ar"
    }
]
}

```

You can also have multiple suggesters per request.

Request

```
GET http://127.0.0.1:8000/search/publishers/suggest/?name_suggest__completion=B&
↪country_suggest__completion=Ar
```

Response

```

{
  "_shards": {
    "successful": 1,
    "total": 1,
    "failed": 0
  },
  "country_suggest__completion": [
    {
      "text": "Ar",
      "options": [
        {
          "score": 1.0,
          "text": "Armenia"
        },
        {
          "score": 1.0,
          "text": "Argentina"
        }
      ],
      "offset": 0,
      "length": 2
    }
  ],
  "name_suggest__completion": [
    {
      "text": "B",
      "options": [
        {
          "score": 1.0,
          "text": "Book Works"
        },
        {
          "score": 1.0,
          "text": "Brumleve LLC"
        },
        {
          "score": 1.0,
          "text": "Booktrope"
        },
        {
          "score": 1.0,
          "text": "Borman, Post and Wendt"
        }
      ]
    }
  ]
}

```

```

        },
        {
            "score": 1.0,
            "text": "Book League of America"
        }
    ],
    "offset": 0,
    "length": 1
}
]
}
```

Pagination

Page number pagination

By default, the `PageNumberPagination` class is used on all view sets which inherit from `BaseDocumentViewSet`.

Example:

```

http://127.0.0.1:8000/search/books/?page=4
http://127.0.0.1:8000/search/books/?page=4&page_size=100
```

Limit/offset pagination

In order to use a different `pagination_class`, for instance the `LimitOffsetPagination`, specify it explicitly in the view.

search_indexes/viewsets.py

```

# ...

from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination

# ...

class BookDocumentView(BaseDocumentViewSet):
    """The BookDocument view."""

    # ...

    pagination_class = LimitOffsetPagination

    # ...
```

Example:

```

http://127.0.0.1:8000/search/books/?limit=100
http://127.0.0.1:8000/search/books/?offset=400&limit=100
```

Various handy helpers

Contents:

Table of Contents

- *Various handy helpers*
 - *More like this*

More like this

To get more-like-this results on a random registered model, do as follows:

```
from django_elasticsearch_dsl_drf.helpers import more_like_this
from books.models import Book
book = Book.objects.first()
similar_books = more_like_this(
    book,
    ['title', 'description', 'summary']
)
```

Customize results as follows:

```
from django_elasticsearch_dsl_drf.helpers import more_like_this
from elasticsearch_dsl.query import Q
from books.models import Book
book = Book.objects.first()
query = Q('bool', must_not=Q('term', **{'state.raw': 'cancelled'}))
similar_books = more_like_this(
    book,
    query=query,
    fields=['title', 'description', 'summary'],
    min_term_freq=2,
    min_doc_freq=1,
)
```

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.

0.3.12

2017-09-21

- Added `geo_distance` filter. Note, that although functionally the filter would not change its' behaviour, it is likely to be moved to a separate backend (`geo_spatial`). For now use as is.
- Minor fixes.

0.3.11

2017-09-21

- Added `query` argument to `more_like_this` helper.

0.3.10

2017-09-20

- Minor fixes.
- Simplified Elasticsearch version check.

0.3.9

2017-09-12

- Python 2.x compatibility fix.

0.3.8

2017-09-12

- Fixes tests on some environments.

0.3.7

2017-09-07

- Docs fixes.

0.3.6

2017-09-07

- Fixed suggestions test for Elasticsearch 5.x.
- Added `compat` module for painless testing of Elastic 2.x to Elastic 5.x transition.

0.3.5

2017-08-24

- Minor fixes in the ordering backend.
- Improved tests and coverage.

0.3.4

2017-08-23

- Minor fixes in the ordering backend.

0.3.3

2017-07-13

- Minor fixes and improvements.

0.3.2

2017-07-12

- Minor fixes and improvements.

0.3.1

2017-07-12

- Minor Python2 fixes.
- Minor documentation fixes.

0.3

2017-07-11

- Add suggestions support (*term*, *phrase* and *completion*).

0.2.6

2017-07-11

- Minor fixes.
- Fixes in documentation.

0.2.5

2017-07-11

- Fixes in documentation.

0.2.4

2017-07-11

- Fixes in documentation.

0.2.3

2017-07-11

- Fixes in documentation.

0.2.2

2017-07-11

- Fixes in documentation.

0.2.1

2017-07-11

- Fixes in documentation.

0.2

2017-07-11

- Initial faceted search support.
- Pagination support.

0.1.8

2017-06-26

- Python2 fixes.
- Documentation and example project improvements.

0.1.7

2017-06-25

- Dynamic serializer for Documents.
- Major improvements in documentation.

0.1.6

2017-06-23

- Implemented `gt``, ``gte`, `lt` and `lte` functional query lookups.
- Implemented `ids` native filter lookup.

0.1.5

2017-06-22

- Implemented `endswith` and `contains` functional filters.
- Added tests for `wildcard`, `exists`, `exclude` and `isnull` filters. Improved `range` filter tests.
- Improve `more_like_this` helper test.
- Improve ordering tests.
- Two additional arguments added to the `more_like_this` helper: `min_doc_freq` and `max_doc_freq`.
- Minor documentation improvements.

0.1.4

2017-06-22

- Added tests for `in`, `term` and `terms` filters.
- Minor documentation fixes.

0.1.3

2017-06-21

- Added tests for `more_like_this` helper, `range` and `prefix` filters.
- Minor documentation improvements.

0.1.2

2017-06-20

- Minor fixes in tests.

0.1.1

2017-06-20

- Fixes in `more_like_this` helper.
- Tiny documentation improvements.

0.1

2017-06-19

- Initial beta release.

django_elasticsearch_dsl_drf package

Subpackages

django_elasticsearch_dsl_drf.filter_backends package

Subpackages

django_elasticsearch_dsl_drf.filter_backends.filtering package

Submodules

django_elasticsearch_dsl_drf.filter_backends.filtering.common module

Common filtering backend.

```
class django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend
    Bases: rest_framework.filters.BaseFilterBackend, django_elasticsearch_dsl_drf.
filter_backends.mixins.FilterBackendMixin
```

Filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [FilteringFilterBackend,]
>>>     filter_fields = {
>>>         'title': 'title.raw',
```

```

>>>         'state': {
>>>             'field': 'state.raw',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_PREFIX,
>>>                 LOOKUP_FILTER_WILDCARD,
>>>                 LOOKUP_QUERY_EXCLUDE,
>>>                 LOOKUP_QUERY_ISNULL,
>>>             ],
>>>         }
>>> }

```

classmethod `apply_filter_prefix` (*queryset, options, value*)

Apply *prefix* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod `apply_filter_range` (*queryset, options, value*)

Apply *range* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod `apply_filter_term` (*queryset, options, value*)

Apply *term* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod `apply_filter_terms` (*queryset, options, value*)

Apply *terms* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*mixed: either str or iterable (list, tuple)*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod `apply_query_contains` (*queryset, options, value*)

Apply *contains* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod `apply_query_endswith` (*queryset, options, value*)

Apply *endswith* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod `apply_query_exclude` (*queryset, options, value*)

Apply *exclude* functional query.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod `apply_query_exists` (*queryset, options, value*)

Apply *exists* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod `apply_query_geo_distance` (*queryset, options, value*)

Apply *wildcard* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.

- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod **apply_query_gt** (*queryset, options, value*)
Apply *gt* functional query.

Parameters

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod **apply_query_gte** (*queryset, options, value*)
Apply *gte* functional query.

Parameters

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod **apply_query_in** (*queryset, options, value*)
Apply *in* functional query.

Parameters

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod **apply_query_isnull** (*queryset, options, value*)
Apply *isnull* functional query.

Parameters

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod **apply_query_lt** (*queryset, options, value*)
Apply *lt* functional query.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod **apply_query_lte** (*queryset, options, value*)

Apply *lte* functional query.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod **apply_query_wildcard** (*queryset, options, value*)

Apply *wildcard* filter.

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

filter_queryset (*request, queryset, view*)

Filter the queryset.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Updated queryset.

Return type *elasticsearch_dsl.search.Search*

get_filter_query_params (*request, view*)

Get query params to be filtered on.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Request query params to filter on.

Return type dict

classmethod `get_geo_distance_params` (*value*, *field*)

Get params for *geo_distance* query

Parameters *value* (*str*) –

Returns Params to be used in *geo_distance* query.

Return type dict

classmethod `get_gte_lte_params` (*value*, *lookup*)

Get params for *gte*, *gt*, *lte* and *lt* query.

Parameters

- *value* (*str*) –
- *lookup* (*str*) –

Returns Params to be used in *range* query.

Return type dict

classmethod `get_range_params` (*value*)

Get params for *range* query.

Parameters *value* –

Type str

Returns Params to be used in *range* query.

Return type dict

classmethod `prepare_filter_fields` (*view*)

Prepare filter fields.

Parameters *view* (*rest_framework.viewsets.ReadOnlyModelViewSet*) –

Returns Filtering options.

Return type dict

django_elasticsearch_dsl_drf.filter_backends.filtering.ids module

Ids filtering backend.

Filters documents that only have the provided ids. Note, this query uses the *_uid* field.

Elastic query:

```
{
  "query": {
    "ids": { "type": "book_document", "values": ["68", "64", "58"]
    }
  }
}
```

REST framework request equivalent:

- <http://localhost:8000/api/articles/?ids=68|64|58>

- <http://localhost:8000/api/articles/?ids=68&ids=64&ids=58>

Official Elastic docs:

- <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-ids-query.html>

class `django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend`
 Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Ids filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     IdsFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [IdsFilterBackend]
```

filter_queryset (*request, queryset, view*)

Filter the queryset.

Parameters

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **queryset** (`elasticsearch_dsl.search.Search`) – Base queryset.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

Returns Updated queryset.

Return type `elasticsearch_dsl.search.Search`

get_ids_query_params (*request*)

Get search query params.

Parameters **request** (`rest_framework.request.Request`) – Django REST framework request.

Returns List of search query params.

Return type list

get_ids_values (*request, view*)

Get ids values for query.

Parameters

- **request** (`rest_framework.request.Request`) – Django REST framework request.

- **queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Updated queryset.

Return type *elasticsearch_dsl.search.Search*

ids_query_param = 'ids'

Module contents

Term level filtering backends.

Submodules

django_elasticsearch_dsl_drf.filter_backends.faceted_search module

Faceted search backend.

class *django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend*
Bases: *rest_framework.filters.BaseFilterBackend*

Faceted search backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FacetedSearchFilterBackend
>>> )
>>> from elasticsearch_dsl import TermsFacet, DateHistogramFacet
>>> from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [FacetedSearchFilterBackend,]
>>>     faceted_search_fields = {
>>>         'title': 'title.raw', # Uses `TermsFacet` by default
>>>         'state': {
>>>             'field': 'state.raw',
>>>             'facet': TermsFacet,
>>>         },
>>>         'publisher': {
>>>             'field': 'publisher.raw',
>>>             'facet': TermsFacet,
>>>             'enabled': False,
>>>         },
>>>         'date_published': {
>>>             'field': 'date_published.raw',
>>>             'facet': DateHistogramFacet,
```

```
>>>         'options': {
>>>             'interval': 'month',
>>>         },
>>>         'enabled': True,
>>>     },
>>>
>>> }
```

Facets make queries to be more heavy. That's why by default all facets are disabled and enabled only explicitly either in the filter options (*enabled* set to *True*) or via query params *?facet=state&facet=date_published*.

aggregate (*request, queryset, view*)

Parameters

- **request** –
- **queryset** –
- **view** –

Returns

construct_facets (*request, view*)

Construct facets.

Turns the following structure:

```
>>> {
>>>     'publisher': {
>>>         'field': 'publisher.raw',
>>>         'facet': TermsFacet,
>>>         'enabled': False,
>>>     }
>>>     'date_published': {
>>>         'field': 'date_published',
>>>         'facet': DateHistogramFacet,
>>>         'options': {
>>>             'interval': 'month',
>>>         },
>>>         'enabled': True,
>>>     },
>>> }
```

Into the following structure:

```
>>> {
>>>     'publisher': TermsFacet(field='publisher.raw'),
>>>     'publishing_frequency': DateHistogramFacet(
>>>         field='date_published.raw',
>>>         interval='month'
>>>     ),
>>> }
```

faceted_search_param = 'facet'

filter_queryset (*request, queryset, view*)

Filter the queryset.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Updated queryset.

Return type *elasticsearch_dsl.search.Search*

get_faceted_search_query_params (*request*)

Get faceted search query params.

Parameters **request** (*rest_framework.request.Request*) – Django REST framework request.

Returns List of search query params.

Return type list

classmethod **prepare_faceted_search_fields** (*view*)

Prepare faceted search fields.

Prepares the following structure:

```
>>> {
>>>     'publisher': {
>>>         'field': 'publisher.raw',
>>>         'facet': TermsFacet,
>>>         'enabled': False,
>>>     }
>>>     'date_published': {
>>>         'field': 'date_published.raw',
>>>         'facet': DateHistogramFacet,
>>>         'options': {
>>>             'interval': 'month',
>>>         },
>>>         'enabled': True,
>>>     },
>>> }
```

Parameters **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) –

Returns Faceted search fields options.

Return type dict

django_elasticsearch_dsl_drf.filter_backends.mixins module

Mixins.

class *django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin*

Bases: object

Filter backend mixin.

classmethod **split_lookup_filter** (*value, maxsplit=-1*)

Split lookup filter.

Parameters

- **value** (*str*) – Value to split.
- **maxsplit** (*int*) – The *maxsplit* option of *string.split*.

Returns Lookup filter split into a list.

Return type list

classmethod **split_lookup_value** (*value*, *maxsplit=-1*)
Split lookup value.

Parameters

- **value** (*str*) – Value to split.
- **maxsplit** (*int*) – The *maxsplit* option of *string.split*.

Returns Lookup value split into a list.

Return type list

django_elasticsearch_dsl_drf.filter_backends.ordering module

Ordering backend.

class `django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend`
Bases: `rest_framework.filters.BaseFilterBackend`

Ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     OrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [OrderingFilterBackend,]
>>>     ordering_fields = {
>>>         'id': 'id',
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'state': {
>>>             'field': 'state.raw',
>>>         }
>>>     }
```

filter_queryset (*request*, *queryset*, *view*)
Filter the queryset.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Updated queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod **get_default_ordering_params** (*view*)

Get the default ordering params for the view.

Parameters **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Ordering params to be used for ordering.

Return type list

get_ordering_query_params (*request, view*)

Get ordering query params.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Ordering params to be used for ordering.

Return type list

ordering_param = 'ordering'

django_elasticsearch_dsl_drf.filter_backends.search module

Search backend.

class *django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend*
 Bases: *rest_framework.filters.BaseFilterBackend*, *django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin*

Search filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     SearchFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
```

```
>>> filter_backends = [SearchFilterBackend,]
>>> search_fields = (
>>>     'title',
>>>     'content',
>>> )
```

construct_search (*request, view*)

Construct search.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Updated queryset.

Return type *elasticsearch_dsl.search.Search*

filter_queryset (*request, queryset, view*)

Filter the queryset.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Updated queryset.

Return type *elasticsearch_dsl.search.Search*

get_search_query_params (*request*)

Get search query params.

Parameters **request** (*rest_framework.request.Request*) – Django REST framework request.

Returns List of search query params.

Return type list

search_param = 'search'

django_elasticsearch_dsl_drf.filter_backends.suggester module

Suggesters backend.

It's assumed, that fields you're planning to query suggestions for have been properly indexed using fields. CompletionField.

Example:

```
>>> from django_elasticsearch_dsl import DocType, Index, fields
>>>
>>> from books.models import Publisher
>>>
```



```

>>> # Name of the Elasticsearch index
>>> PUBLISHER_INDEX = Index(PUBLISHER_INDEX_NAME)
>>> # See Elasticsearch Indices API reference for available settings
>>> PUBLISHER_INDEX.settings(
>>>     number_of_shards=1,
>>>     number_of_replicas=1
>>> )
>>>
>>> @PUBLISHER_INDEX.doc_type
>>> class PublisherDocument(DocType):
>>>     "Publisher Elasticsearch document."
>>>
>>>     id = fields.IntegerField(attr='id')
>>>
>>>     name = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     info = fields.StringField()
>>>
>>>     address = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword')
>>>         }
>>>     )
>>>
>>>     city = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     state_province = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     country = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     website = fields.StringField()
>>>
>>>     class Meta(object):
>>>         "Meta options."
>>>
>>>     model = Publisher # The model associate with this DocType

```

```
class django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend
```

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Suggester filter backend for Elasticsearch.

Suggestion functionality is exclusive. Once you have queried the `SuggesterFilterBackend`, the latter will transform your current search query into suggestion search query (which is very different). Therefore, always add it as the very last filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     SUGGESTER_TERM,
>>>     SUGGESTER_PHRASE,
>>>     SUGGESTER_COMPLETION,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     SuggesterFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet
>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(BaseDocumentViewSet):
>>>
>>>     document = PublisherDocument
>>>     serializer_class = PublisherDocumentSerializer
>>>     filter_backends = [
>>>         # ...
>>>         SuggesterFilterBackend,
>>>     ]
>>>     # Suggester fields
>>>     suggester_fields = {
>>>         'name_suggest': {
>>>             'field': 'name.suggest',
>>>             'suggesters': [
>>>                 SUGGESTER_TERM,
>>>                 SUGGESTER_PHRASE,
>>>                 SUGGESTER_COMPLETION,
>>>             ],
>>>         },
>>>         'city_suggest': {
>>>             'field': 'city.suggest',
>>>             'suggesters': [
>>>                 SUGGESTER_COMPLETION,
>>>             ],
>>>         },
>>>         'state_province_suggest': {
>>>             'field': 'state_province.suggest',
>>>             'suggesters': [
>>>                 SUGGESTER_COMPLETION,
>>>             ],
>>>         },
>>>         'country_suggest': {
>>>             'field': 'country.suggest',
```

```
>>>         'suggesters': [
>>>             SUGGESTER_COMPLETION,
>>>         ],
>>>     },
>>> }
```

classmethod `apply_suggester_completion` (*suggester_name, queryset, options, value*)
 Apply *completion* suggester.

Parameters

- **suggester_name** (*str*) –
- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod `apply_suggester_phrase` (*suggester_name, queryset, options, value*)
 Apply *phrase* suggester.

Parameters

- **suggester_name** (*str*) –
- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

classmethod `apply_suggester_term` (*suggester_name, queryset, options, value*)
 Apply *term* suggester.

Parameters

- **suggester_name** (*str*) –
- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

`filter_queryset` (*request, queryset, view*)
 Filter the queryset.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Updated queryset.

Return type `elasticsearch_dsl.search.Search`

get_suggester_query_params (*request, view*)

Get query params to be for suggestions.

Parameters

- **request** (*rest_framework.request.Request*) – Django REST framework request.
- **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) – View.

Returns Request query params to filter on.

Return type `dict`

classmethod prepare_suggester_fields (*view*)

Prepare filter fields.

Parameters **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) –

Returns Filtering options.

Return type `dict`

Module contents

All filter backends.

django_elasticsearch_dsl_drf.tests package

Submodules

django_elasticsearch_dsl_drf.tests.base module

Base tests.

class `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase` (*methodName='runTest'*)

Bases: `django.test.testcases.TransactionTestCase`

Base REST framework test case.

authenticate ()

Helper for logging in Genre Coordinator user.

Returns

pytestmark = [`<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`, `<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`, ...]

classmethod setUpClass ()

Set up class.

class `django_elasticsearch_dsl_drf.tests.base.BaseTestCase` (*methodName='runTest'*)

Bases: `django.test.testcases.TransactionTestCase`

Base test case.

pytestmark = [`<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`, `<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`, ...]

```
classmethod setUpClass ()
    Set up class.
```

django_elasticsearch_dsl_drf.tests.test_faceted_search module

Test faceted search backend.

```
class django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch (methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test faceted search.

    pytestmark = [<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>, <MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>]

    classmethod setUp ()

    test_list_results_with_facets ()
        Test list results with facets.
```

django_elasticsearch_dsl_drf.tests.test_filtering module

Test filtering backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering (methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test filtering.

    pytestmark = [<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>, <MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>]

    classmethod setUpClass ()
        Set up.

    test_field_filter_contains ()
        Test filter contains.

        Example:

        http://localhost:8000/api/articles/?state__contains=lishe

    test_field_filter_endswith ()
        Test filter endswith.

        Example:

        http://localhost:8000/api/articles/?state__endswith=lished

    test_field_filter_exclude ()
        Test filter exclude.

        Example:

        http://localhost:8000/api/articles/?tags__exclude=children

    test_field_filter_exists_false ()
        Test filter exists.

        Example:

        http://localhost:8000/api/articles/?non_existent__exists=false
```

test_field_filter_exists_true()

Test filter exists true.

Example:

http://localhost:8000/api/articles/?tags__exists=true

test_field_filter_geo_distance()

Field filter term.

Example:

http://localhost:8000/api/books/?location__geo_distance=1km|48.8549|2.3000

test_field_filter_gt()

Field filter gt.

Example:

http://localhost:8000/api/users/?id__gt=10

Returns

test_field_filter_gt_with_boost()

Field filter gt with boost.

Example:

http://localhost:8000/api/users/?id__gt=10|2.0

Returns

test_field_filter_gte()

Field filter gte.

Example:

http://localhost:8000/api/users/?id__gte=10

Returns

test_field_filter_in()

Test filter in.

Example:

http://localhost:8000/api/articles/?id__in=1|2|3

test_field_filter_isnull_false()

Test filter isnull true.

Example:

http://localhost:8000/api/articles/?tags__isnull=false

test_field_filter_isnull_true()

Test filter isnull true.

Example:

http://localhost:8000/api/articles/?null_field__isnull=true

test_field_filter_lt()

Field filter lt.

Example:

http://localhost:8000/api/users/?id__lt=10

Returns

test_field_filter_lt_with_boost()

Field filter lt with boost.

Example:

http://localhost:8000/api/users/?id__lt=10|2.0

Returns

test_field_filter_lte()

Field filter lte.

Example:

http://localhost:8000/api/users/?id__lte=10

Returns

test_field_filter_prefix()

Test filter prefix.

Example:

http://localhost:8000/api/articles/?tags__prefix=bio

test_field_filter_range()

Field filter range.

Example:

http://localhost:8000/api/users/?age__range=16|67

test_field_filter_range_with_boost()

Field filter range.

Example:

http://localhost:8000/api/users/?age__range=16|67|2.0

test_field_filter_term()

Field filter term.

test_field_filter_term_explicit()

Field filter term.

test_field_filter_terms_list()

Test filter terms.

test_field_filter_terms_string()

Test filter terms.

Example:

http://localhost:8000/api/articles/?id__terms=1|2|3

`test_field_filter_wildcard()`

Test filter wildcard.

Example:

`http://localhost:8000/api/articles/?title__wildcard=*elusional*`

`test_ids_filter()`

Test ids filter.

Example:

`http://localhost:8000/api/articles/?ids=68|64|58` `http://localhost:8000/api/articles/?ids=68&ids=64&ids=58`

django_elasticsearch_dsl_drf.tests.test_helpers module

Test helpers.

class `django_elasticsearch_dsl_drf.tests.test_helpers.TestHelpers` (*methodName='runTest'*)

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseTestCase`

Test helpers.

pytestmark = [`<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`, `<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`], `<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`

classmethod `setUpClass()`

test_filter_by_field()

Filter by field.

django_elasticsearch_dsl_drf.tests.test_ordering module

Test ordering backend.

class `django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering` (*methodName='runTest'*)

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test ordering.

pytestmark = [`<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`, `<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`], `<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>`

classmethod `setUpClass()`

Set up class.

test_author_order_by_field_id_ascending()

Order by field *name* ascending.

test_author_order_by_field_id_descending()

Order by field *id* descending.

test_author_order_by_field_name_ascending()

Order by field *name* ascending.

test_author_order_by_field_name_descending()

Order by field *name* descending.

test_book_order_by_field_id_ascending()

Order by field *id* ascending.

test_book_order_by_field_id_descending()

Order by field *id* descending.


```
test_book_order_by_field_title_ascending()  
    Order by field title ascending.  
  
test_book_order_by_field_title_descending()  
    Order by field title descending.  
  
test_book_order_by_non_existent_field()  
    Order by non-existent field.
```

django_elasticsearch_dsl_drf.tests.test_pagination module

Test pagination.

```
class django_elasticsearch_dsl_drf.tests.test_pagination.TestPagination (methodName='runTest')  
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase  
  
    Test pagination.  
  
    pytestmark = [<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>, <MarkDecorator 'django_db' {'kwargs': {}, 'a'  
    classmethod setUpClass()  
        Set up class.  
  
    test_pagination()  
        Test pagination.
```

django_elasticsearch_dsl_drf.tests.test_search module

Test search backend.

```
class django_elasticsearch_dsl_drf.tests.test_search.TestSearch (methodName='runTest')  
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase  
  
    Test search.  
  
    pytestmark = [<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>, <MarkDecorator 'django_db' {'kwargs': {}, 'a'  
    classmethod setUp()  
  
    test_search_by_field()  
        Search by field.
```

django_elasticsearch_dsl_drf.tests.test_suggesters module

Test suggestions backend.

```
class django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters (methodName='runTest')  
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase  
  
    Test suggesters.  
  
    pytestmark = [<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>, <MarkDecorator 'django_db' {'kwargs': {}, 'a'  
    classmethod setUpClass()  
        Set up class.  
  
    test_suggesters()  
        Test suggesters.
```

```
test_suggesters_no_args_provided()
    Test suggesters with no args provided.
```

django_elasticsearch_dsl_drf.tests.test_views module

Test views.

```
class django_elasticsearch_dsl_drf.tests.test_views.TestViews (methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
```

Test views.

```
pytestmark = [<MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>, <MarkDecorator 'django_db' {'kwargs': {}, 'args': ()}>]
```

```
classmethod setUpClass()
    Set up class.
```

```
test_detail_view()
    Test detail view.
```

```
test_listing_view()
    Test listing view.
```

Module contents

Submodules

django_elasticsearch_dsl_drf.apps module

Apps.

```
class django_elasticsearch_dsl_drf.apps.Config (app_name, app_module)
    Bases: django.apps.config.AppConfig
```

Config.

```
label = 'django_elasticsearch_dsl_drf'
```

```
name = 'django_elasticsearch_dsl_drf'
```

django_elasticsearch_dsl_drf.compat module

Transitional compatibility module. Contains various field wrappers and helpers for painless (testing of) Elastic 2.x to Elastic 5.x transition. This module is not supposed to solve all transition issues for you. Better move to Elastic 5.x as soon as possible.

```
django_elasticsearch_dsl_drf.compat.get_elasticsearch_version (default=(2, 0, 0))
    Get Elasticsearch version.
```

Parameters `default (tuple)` – Default value. Mainly added for building the docs when Elasticsearch is not running.

Returns

Return type list

```
django_elasticsearch_dsl_drf.compat.KeywordField (**kwargs)
    Keyword field.
```

Parameters `kwargs` –

Returns

`django_elasticsearch_dsl_drf.compat.StringField(**kwargs)`
String field.

Parameters `kwargs` –

Returns

django_elasticsearch_dsl_drf.constants module

Constants module. Contains Elasticsearch constants, lookup constants, functional constants, suggesters, etc.

django_elasticsearch_dsl_drf.helpers module

Helpers.

`django_elasticsearch_dsl_drf.helpers.get_document_for_model(model)`
Get document for model given.

Parameters `model` (Subclass of `django.db.models.Model`.) – Model to get document index for.

Returns Document index for the given model.

Return type Subclass of `django_elasticsearch_dsl.DocType`.

`django_elasticsearch_dsl_drf.helpers.get_index_and_mapping_for_model(model)`
Get index and mapping for model.

Parameters `model` (Subclass of `django.db.models.Model`.) – Django model for which to get index and mapping for.

Returns Index and mapping values.

Return type tuple.

`django_elasticsearch_dsl_drf.helpers.more_like_this(obj, fields, max_query_terms=25, min_term_freq=2, min_doc_freq=5, max_doc_freq=0, query=None)`

More like this.

<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-mlt-query.html>

Parameters

- **obj** (Instance of `django.db.models.Model` (sub-classed) model.) – Django model instance for which similar objects shall be found.
- **fields** (`list`) – Fields to search in.
- **max_query_terms** (`int`) –
- **min_term_freq** (`int`) –
- **min_doc_freq** (`int`) –
- **max_doc_freq** (`int`) –
- **query** (`elasticsearch_dsl.query.Q`) – Q query

Returns List of objects.

Return type elasticsearch_dsl.search.Search

Example:

```
>>> from django_elasticsearch_dsl_drf.helpers import more_like_this
>>> from books.models import Book
>>> book = Book.objects.first()
>>> similar_books = more_like_this(
>>>     book,
>>>     ['title', 'description', 'summary']
>>> )
```

`django_elasticsearch_dsl_drf.helpers.sort_by_list` (*unsorted_dict*, *sorted_keys*)
Sort an OrderedDict by list of sorted keys.

Parameters

- **unsorted_dict** (*collections.OrderedDict*) – Source dictionary.
- **sorted_keys** (*list*) – Keys to sort on.

Returns Sorted dictionary.

Return type `collections.OrderedDict`

django_elasticsearch_dsl_drf.pagination module

Pagination.

class `django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination` (**args*,
***kwargs*)

Bases: `rest_framework.pagination.LimitOffsetPagination`

A limit/offset pagination.

Example:

<http://api.example.org/accounts/?limit=100> <http://api.example.org/accounts/?offset=400&limit=100>

get_facets (*facets=None*)

Get facets.

Parameters **facets** –

Returns

get_paginated_response (*data*)

Get paginated response.

Parameters **data** –

Returns

paginate_queryset (*queryset, request, view=None*)

class `django_elasticsearch_dsl_drf.pagination.Page` (*object_list*, *number*, *paginator*,
facets)

Bases: `django.core.paginator.Page`

Page for Elasticsearch.

```
class django_elasticsearch_dsl_drf.pagination.PageNumberPagination (*args,  
                                                                    **kwargs)
```

Bases: `rest_framework.pagination.PageNumberPagination`

Page number pagination.

A simple page number based style that supports page numbers as query parameters.

Example:

`http://api.example.org/accounts/?page=4` `http://api.example.org/accounts/?page=4&page_size=100`

django_paginator_class

alias of *Paginator*

get_facets (*page=None*)

Get facets.

Parameters *page* –

Returns

get_paginated_response (*data*)

Get paginated response.

Parameters *data* –

Returns

paginate_queryset (*queryset, request, view=None*)

Paginate a queryset.

Paginate a queryset if required, either returning a page object, or *None* if pagination is not configured for this view.

Parameters

- **queryset** –
- **request** –
- **view** –

Returns

```
class django_elasticsearch_dsl_drf.pagination.Paginator (object_list,          per_page,  
                                                         orphans=0,          al-  
                                                         low_empty_first_page=True)
```

Bases: `django.core.paginator.Paginator`

Paginator for Elasticsearch.

page (*number*)

Returns a Page object for the given 1-based page number.

Parameters *number* –

Returns

django_elasticsearch_dsl_drf.serializers module

Serializers.

```
class django_elasticsearch_dsl_drf.serializers.DocumentSerializer (instance=None,
                                                                    data=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    **kwargs)
```

Bases: `rest_framework.serializers.Serializer`

A dynamic DocumentSerializer class.

create (*validated_data*)
 Create.
 Do nothing.

Parameters *validated_data* –

Returns

get_fields ()
 Get the required fields for serializing the result.

update (*instance*, *validated_data*)
 Update.
 Do nothing.

Parameters

- **instance** –
- **validated_data** –

Returns

```
class django_elasticsearch_dsl_drf.serializers.DocumentSerializerMeta
    Bases: rest_framework.serializers.SerializerMetaclass

    Metaclass for the DocumentSerializer.

    Ensures that all declared subclasses implemented a Meta.
```

```
class django_elasticsearch_dsl_drf.serializers.Meta
    Bases: type

    Template for the DocumentSerializerMeta.Meta class.

    exclude = ()

    field_aliases = {}

    field_options = {}

    fields = ()

    ignore_fields = ()

    index_aliases = {}

    index_classes = ()

    search_fields = ()

    serializers = ()
```

django_elasticsearch_dsl_drf.utils module

Utils.

```
class django_elasticsearch_dsl_drf.utils.DictionaryProxy(mapping)
```

Bases: object

Dictionary proxy.

```
class django_elasticsearch_dsl_drf.utils.EmptySearch(**kwargs)
```

Bases: object

Empty Search.

django_elasticsearch_dsl_drf.views module

```
class django_elasticsearch_dsl_drf.views.BaseDocumentViewSet(*args, **kwargs)
```

Bases: rest_framework.viewsets.ReadOnlyModelViewSet

Base document ViewSet.

document = None

document_uid_field = 'id'

get_object()

Get object.

get_queryset()

Get queryset.

pagination_class

alias of PageNumberPagination

suggest(request)

Suggest functionality.

django_elasticsearch_dsl_drf.viewsets module

Base ViewSets.

```
class django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet(*args, **kwargs)
```

Bases: rest_framework.viewsets.ReadOnlyModelViewSet

Base document ViewSet.

document = None

document_uid_field = 'id'

get_object()

Get object.

get_queryset()

Get queryset.

pagination_class

alias of PageNumberPagination

suggest(request)

Suggest functionality.

Module contents

Integrate Elasticsearch DSL with Django REST framework.

CHAPTER 13

Indices and tables

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

d

`django_elasticsearch_dsl_drf`, 104

`django_elasticsearch_dsl_drf.apps`, 98

`django_elasticsearch_dsl_drf.compat`, 98

`django_elasticsearch_dsl_drf.constants`, 99

`django_elasticsearch_dsl_drf.filter_backends`, 92

`django_elasticsearch_dsl_drf.filter_backends.faceted_search`, 83

`django_elasticsearch_dsl_drf.filter_backends.filtering`, 83

`django_elasticsearch_dsl_drf.filter_backends.filtering.common`, 76

`django_elasticsearch_dsl_drf.filter_backends.filtering.ids`, 81

`django_elasticsearch_dsl_drf.filter_backends.mixins`, 85

`django_elasticsearch_dsl_drf.filter_backends.ordering`, 86

`django_elasticsearch_dsl_drf.filter_backends.search`, 87

`django_elasticsearch_dsl_drf.filter_backends.suggester`, 88

`django_elasticsearch_dsl_drf.helpers`, 99

`django_elasticsearch_dsl_drf.pagination`, 100

`django_elasticsearch_dsl_drf.serializers`, 101

`django_elasticsearch_dsl_drf.tests`, 98

`django_elasticsearch_dsl_drf.tests.base`, 92

`django_elasticsearch_dsl_drf.tests.test_faceted_search`, 93

`django_elasticsearch_dsl_drf.tests.test_filtering`, 93

`django_elasticsearch_dsl_drf.tests.test_helpers`, 96

`django_elasticsearch_dsl_drf.tests.test_ordering`, 96

`django_elasticsearch_dsl_drf.tests.test_pagination`, 97

`django_elasticsearch_dsl_drf.tests.test_search`, 97

`django_elasticsearch_dsl_drf.tests.test_suggesters`, 97

`django_elasticsearch_dsl_drf.tests.test_views`, 98

`django_elasticsearch_dsl_drf.utils`, 103

`django_elasticsearch_dsl_drf.views`, 103

`django_elasticsearch_dsl_drf.viewsets`, 103

A

- `aggregate()` (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend` class method), 84
- `apply_filter_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 77
- `apply_filter_range()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 77
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 77
- `apply_filter_terms()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 77
- `apply_query_contains()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 78
- `apply_query_endswith()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 78
- `apply_query_exclude()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 78
- `apply_query_exists()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 78
- `apply_query_geo_distance()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 78
- `apply_query_gt()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 79
- `apply_query_gte()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 79
- `apply_query_in()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 79
- `apply_query_isnull()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 79
- `apply_query_lt()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 79
- `apply_query_lte()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 80
- `apply_query_wildcard()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 80
- `apply_suggester_completion()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend` class method), 91
- `apply_suggester_phrase()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend` class method), 91
- `apply_suggester_term()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend` class method), 91
- `authenticate()` (`django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCases` class method), 92

B

- `BaseDocumentViewSet` (class in `django_elasticsearch_dsl_drf.views`), 103
- `BaseDocumentViewSet` (class in `django_elasticsearch_dsl_drf.views`), 103
- `BaseRestFrameworkTestCase` (class in `django_elasticsearch_dsl_drf.tests.base`), 92
- `BaseTestCase` (class in `django_elasticsearch_dsl_drf.tests.base`), 92

C

- `Config` (class in `django_elasticsearch_dsl_drf.apps`), 98
- `construct_facets()` (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend` class method), 84
- `construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend` class method), 88
- `create()` (`django_elasticsearch_dsl_drf.serializers.DocumentSerializer` class method), 102

D

- `DictionaryProxy` (class in `django_elasticsearch_dsl_drf.utils`), 103
- `django_elasticsearch_dsl_drf` (module), 104
- `django_elasticsearch_dsl_drf.apps` (module), 98
- `django_elasticsearch_dsl_drf.compat` (module), 98
- `django_elasticsearch_dsl_drf.constants` (module), 99
- `django_elasticsearch_dsl_drf.filter_backends` (module), 92
- `django_elasticsearch_dsl_drf.filter_backends.faceted_search` (module), 83

[django_elasticsearch_dsl_drf.filter_backends.filtering](#)
 (module), 83
[django_elasticsearch_dsl_drf.filter_backends.filtering.common](#)
 (module), 76
[django_elasticsearch_dsl_drf.filter_backends.filtering.ids](#)
 (module), 81
[django_elasticsearch_dsl_drf.filter_backends.mixins](#)
 (module), 85
[django_elasticsearch_dsl_drf.filter_backends.ordering](#)
 (module), 86
[django_elasticsearch_dsl_drf.filter_backends.search](#)
 (module), 87
[django_elasticsearch_dsl_drf.filter_backends.suggester](#)
 (module), 88
[django_elasticsearch_dsl_drf.helpers](#) (module), 99
[django_elasticsearch_dsl_drf.pagination](#) (module), 100
[django_elasticsearch_dsl_drf.serializers](#) (module), 101
[django_elasticsearch_dsl_drf.tests](#) (module), 98
[django_elasticsearch_dsl_drf.tests.base](#) (module), 92
[django_elasticsearch_dsl_drf.tests.test_faceted_search](#)
 (module), 93
[django_elasticsearch_dsl_drf.tests.test_filtering](#) (mod-
 ule), 93
[django_elasticsearch_dsl_drf.tests.test_helpers](#) (module),
 96
[django_elasticsearch_dsl_drf.tests.test_ordering](#) (mod-
 ule), 96
[django_elasticsearch_dsl_drf.tests.test_pagination](#) (mod-
 ule), 97
[django_elasticsearch_dsl_drf.tests.test_search](#) (module),
 97
[django_elasticsearch_dsl_drf.tests.test_suggesters](#) (mod-
 ule), 97
[django_elasticsearch_dsl_drf.tests.test_views](#) (module),
 98
[django_elasticsearch_dsl_drf.utils](#) (module), 103
[django_elasticsearch_dsl_drf.views](#) (module), 103
[django_elasticsearch_dsl_drf.viewsets](#) (module), 103
[django_paginator_class](#) ([django_elasticsearch_dsl_drf.pagination.PageNumberPagination](#)
 attribute), 101
[document](#) ([django_elasticsearch_dsl_drf.views.BaseDocumentViewSet](#)
 attribute), 103
[document](#) ([django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet](#)
 attribute), 103
[document_uid_field](#) ([django_elasticsearch_dsl_drf.views.BaseDocumentViewSet](#)
 attribute), 103
[document_uid_field](#) ([django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet](#)
 attribute), 103
[DocumentSerializer](#) (class in
[django_elasticsearch_dsl_drf.serializers](#)),
 101
[DocumentSerializerMeta](#) (class in
[django_elasticsearch_dsl_drf.serializers](#)),
 102

E

[EmptySearch](#) (class in
[django_elasticsearch_dsl_drf.utils](#)), 103
[exclude](#) ([django_elasticsearch_dsl_drf.serializers.Meta](#)
 attribute), 102

F

[faceted_search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.faceted_search](#)
 attribute), 84
[FacetedSearchFilterBackend](#) (class in
[django_elasticsearch_dsl_drf.filter_backends.faceted_search](#)),
 83
[field_aliases](#) ([django_elasticsearch_dsl_drf.serializers.Meta](#)
 attribute), 102
[field_options](#) ([django_elasticsearch_dsl_drf.serializers.Meta](#)
 attribute), 102
[fields](#) ([django_elasticsearch_dsl_drf.serializers.Meta](#) at-
 tribute), 102
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.faceted_search](#)
 method), 84
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.common](#)
 method), 80
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.ids](#)
 method), 82
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend](#)
 method), 86
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend](#)
 method), 88
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend](#)
 method), 91
[FilterBackendMixin](#) (class in
[django_elasticsearch_dsl_drf.filter_backends.mixins](#)),
 85
[FilteringFilterBackend](#) (class in
[django_elasticsearch_dsl_drf.filter_backends.filtering.common](#)),
 76

G

[get_default_ordering_params\(\)](#)
 ([django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend](#)
 class method), 87
[get_document_for_model\(\)](#) (in
[django_elasticsearch_dsl_drf.helpers](#)), 99
[get_elasticsearch_version\(\)](#) (in
[django_elasticsearch_dsl_drf.compat](#)), 98
[get_faceted_search_query_params\(\)](#)
 ([django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearch](#)
 method), 85
[get_facets\(\)](#) ([django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination](#)
 method), 100
[get_facets\(\)](#) ([django_elasticsearch_dsl_drf.pagination.PageNumberPagination](#)
 method), 101
[get_fields\(\)](#) ([django_elasticsearch_dsl_drf.serializers.DocumentSerializer](#)
 method), 102

get_filter_query_params()
 (django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend
 method), 80

get_geo_distance_params()
 (django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend
 class method), 81

get_gte_lte_params()
 (django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend
 class method), 81

get_ids_query_params()
 (django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend
 method), 82

get_ids_values()
 (django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend
 method), 82

get_index_and_mapping_for_model()
 (in module Meta (class in django_elasticsearch_dsl_drf.serializers),
 django_elasticsearch_dsl_drf.helpers), 99

get_object()
 (django_elasticsearch_dsl_drf.views.BaseDocumentViewSet
 method), 103

get_object()
 (django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet
 method), 103

get_ordering_query_params()
 (django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend
 method), 87

get_paginated_response()
 (django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination
 method), 100

get_paginated_response()
 (django_elasticsearch_dsl_drf.pagination.PageNumberPagination
 method), 101

get_queryset()
 (django_elasticsearch_dsl_drf.views.BaseDocumentViewSet
 method), 103

get_queryset()
 (django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet
 method), 103

get_range_params()
 (django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend
 class method), 81

get_search_query_params()
 (django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend
 method), 88

get_suggester_query_params()
 (django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend
 method), 92

ids_query_param
 (django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend
 attribute), 83

IdsFilterBackend
 (class in django_elasticsearch_dsl_drf.filter_backends.filtering.ids), 82

ignore_fields
 (django_elasticsearch_dsl_drf.serializers.Meta
 attribute), 102

index_aliases
 (django_elasticsearch_dsl_drf.serializers.Meta
 attribute), 102

index_classes
 (django_elasticsearch_dsl_drf.serializers.Meta
 attribute), 102

KeywordField
 (in module django_elasticsearch_dsl_drf.compat), 98

LimitOffsetPagination
 (class in django_elasticsearch_dsl_drf.pagination), 100

Meta
 (class in django_elasticsearch_dsl_drf.serializers), 102

more_like_this()
 (in module django_elasticsearch_dsl_drf.helpers), 99

name
 (django_elasticsearch_dsl_drf.apps.Config attribute), 98

OrderingFilterBackend
 (class in django_elasticsearch_dsl_drf.filter_backends.ordering), 86

Page
 (class in django_elasticsearch_dsl_drf.pagination), 100

PageNumberPagination
 (class in django_elasticsearch_dsl_drf.pagination), 100

paginate_queryset()
 (django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination
 method), 100

paginate_queryset()
 (django_elasticsearch_dsl_drf.pagination.PageNumberPagination
 method), 101

pagination_class
 (django_elasticsearch_dsl_drf.views.BaseDocumentViewSet
 attribute), 103

pagination_class
 (django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet
 attribute), 103

Paginator
 (class in django_elasticsearch_dsl_drf.pagination), 101

prepare_faceted_search_fields()
 (django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearch
 class method), 85

prepare_filter_fields()
 (django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend
 class method), 81

prepare_suggester_fields()
 (django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend
 class method), 92

[pytestmark \(django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase \(attribute\), 92](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.base.BaseTestCase \(attribute\), 92](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch \(attribute\), 93](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering \(attribute\), 93](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_helpers.TestHelpers \(attribute\), 96](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(attribute\), 96](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_pagination.TestPagination \(attribute\), 97](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_search.TestSearch \(attribute\), 97](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters \(attribute\), 97](#)
[pytestmark \(django_elasticsearch_dsl_drf.tests.test_views.TestViews \(attribute\), 98](#)

S

[search_fields \(django_elasticsearch_dsl_drf.serializers.Meta \(attribute\), 102](#)
[search_param \(django_elasticsearch_dsl_drf.filter_backends.SearchBackend \(attribute\), 88](#)
[SearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search \(87](#)
[serializers \(django_elasticsearch_dsl_drf.serializers.Meta \(attribute\), 102](#)
[setUp\(\) \(django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch \(class method\), 93](#)
[setUp\(\) \(django_elasticsearch_dsl_drf.tests.test_search.TestSearch \(class method\), 97](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase \(class method\), 92](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.base.BaseTestCase \(class method\), 92](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering \(class method\), 93](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.test_helpers.TestHelpers \(class method\), 96](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(class method\), 96](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.test_pagination.TestPagination \(class method\), 97](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters \(class method\), 97](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.test_views.TestViews \(class method\), 98](#)
[sort_by_list\(\) \(in module django_elasticsearch_dsl_drf.helpers\), 100](#)

[test_author_order_by_field_id_ascending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 96](#)
[test_author_order_by_field_id_descending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 96](#)
[test_author_order_by_field_name_ascending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 96](#)
[test_author_order_by_field_name_descending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 96](#)
[test_book_order_by_field_id_ascending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 96](#)
[test_book_order_by_field_id_descending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 96](#)
[test_book_order_by_field_title_ascending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 96](#)
[test_book_order_by_field_title_descending\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 97](#)
[test_book_order_by_non_existent_field\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering.TestOrdering \(method\), 97](#)
[test_detail_view\(\) \(django_elasticsearch_dsl_drf.tests.test_views.TestViews \(method\), 98](#)
[test_field_filter_contains\(\) \(django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering \(method\), 93](#)
[test_field_filter_endswith\(\) \(django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering \(method\), 93](#)
[test_field_filter_exclude\(\) \(django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering \(method\), 93](#)
[test_field_filter_exists_false\(\) \(django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering \(method\), 93](#)

(django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 96
 method), 93 test_list_results_with_facets() (django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch method), 93
 test_field_filter_exists_true() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 93
 method), 93 test_listing_view() (django_elasticsearch_dsl_drf.tests.test_views.TestViews method), 98
 test_field_filter_geo_distance() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 94
 method), 94 test_pagination() (django_elasticsearch_dsl_drf.tests.test_pagination.TestPagination method), 97
 test_field_filter_gt() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 94
 method), 97 test_search() (django_elasticsearch_dsl_drf.tests.test_search.TestSearch method), 97
 test_field_filter_gt_with_boost() test_suggesters() (django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters method), 97
 method), 94 test_suggesters_no_args_provided() (django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters method), 97
 test_field_filter_gte() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 94
 method), 97 test_faceted_search() (class in django_elasticsearch_dsl_drf.tests.test_faceted_search),
 test_field_filter_in() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 94
 method), 93
 test_field_filter_isnull_false() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 94
 method), 93
 test_field_filter_isnull_true() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 94
 method), 96
 test_field_filter_lt() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 94
 method), 96
 test_field_filter_lt_with_boost() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 method), 97
 test_field_filter_lte() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 TestSearch (class in django_elasticsearch_dsl_drf.tests.test_search),
 test_field_filter_prefix() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 TestSuggesters (class in django_elasticsearch_dsl_drf.tests.test_suggesters),
 test_field_filter_range() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 method), 97
 test_field_filter_range_with_boost() TestViews (class in django_elasticsearch_dsl_drf.tests.test_views),
 (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 test_field_filter_term() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 update() (django_elasticsearch_dsl_drf.serializers.DocumentSerializer method), 102
 test_field_filter_term_explicit() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 test_field_filter_terms_list() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 test_field_filter_terms_string() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 test_field_filter_wildcard() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering method), 95
 test_filter_by_field() (django_elasticsearch_dsl_drf.tests.test_helpers.TestHelpers method), 96
 test_ids_filter() (django_elasticsearch_dsl_drf.tests.test_filtering.TestFiltering