django-elasticsearch-dsl-drf Documentation

Release 0.5.1

Artur Barseghyan <artur.barseghyan@gmail.com>

Nov 02, 2017

Contents

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.

CHAPTER 4

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`, `term` and `terms` is implemented.
- *Geo-spatial filtering filter backend* (the following filters implemented: `geo_distance`, `geo_polygon` and `geo_bounding_box`).
- *Geo-spatial ordering filter backend* (the following filters implemented: `geo_distance`).
- *Faceted search filter backend*.
- *Suggester filter backend*.
- *Pagination (Page number and limit/offset pagination)*.
- *Ids filter backend*.

CHAPTER 5

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
```

CHAPTER 8

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

12.1 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

12.1.1 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'
    },
}
```

12.1.2 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
    # ...
)
```

12.1.2.1 Sample models

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

12.1.2.1.1 Required imports

Imports required for model definition.

books/models/book.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
```

12.1.2.1.2 Book statuses

books/models/book.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
```

12.1.2.1.3 Publisher model

books/models/book.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()
    latitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)
    longitude = models.DecimalField(null=True,
                                    blank=True,
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)

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

        ordering = ["id"]

    def __str__(self):
        return self.name

    @property
    def location_field_indexing(self):
        """Location for indexing.

        Used in Elasticsearch indexing/tests of `geo_distance` native filter.
        """
        return {
```

```
            'lat': self.latitude,
            'lon': self.longitude,
        }
```

12.1.2.1.4 Author model

books/models/author.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
```

12.1.2.1.5 Tag model

books/models/tag.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
```

12.1.2.1.6 Book model

books/models/book.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 properly 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 properly 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 [tag.title for tag in self.tags.all()]
```

12.1.2.2 Admin classes

This is just trivial. A couple of correspondent admin classes in order to ba 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',)
```

12.1.2.3 Create database tables

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

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

12.1.2.4 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.

12.1.2.5 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 flatterned in Elasticsearch indexes/documents. Nested relations are denormalized.

In our example, all 4 models (`Author`, `Publisher`, `Tag`, `Book`) would be flatterned 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.

12.1.2.5.1 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
```

12.1.2.5.2 Index definition

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

12.1.2.5.2.1 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',
}
```

12.1.2.5.2.2 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
)
```

12.1.2.5.3 Custom analyzers

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

12.1.2.5.4 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),
            'suggest': fields.CompletionField(multi=True),
        },
        multi=True
    )

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

        model = Book  # The model associate with this DocType
```

12.1.2.6 Syncing Django's database with Elasticsearch indexes

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

12.1.2.6.1 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.

12.1.2.6.2 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.

12.1.2.6.2.1 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
```

12.1.2.6.2.2 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)
```

12.1.2.6.2.3 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)
```

12.1.2.7 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.

12.1.2.7.1 Required imports

search_indexes/serializers/book.py

```
import json

from rest_framework import serializers
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from .documents import BookDocument
```

12.1.2.7.2 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/book.py

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

    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',
        )
```

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/book.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."""
        if obj.tags:
            return list(obj.tags)
        else:
            return []
```

12.1.2.8 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.

12.1.2.8.1 Required imports

search_indexes/viewsets/book.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,
    DefaultOrderingFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet

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

12.1.2.8.2 ViewSet definition

search_indexes/viewsets/book.py

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

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        IdsFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        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',)
```

12.1.2.9 URLs

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

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

12.1.2.9.1 Required imports

search_indexes/urls.py

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

from .views import BookDocumentView
```

12.1.2.9.2 Router definition

search_indexes/urls.py

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

12.1.2.9.3 URL patterns

search_indexes/urls.py

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

12.1.2.10 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.

12.1.2.10.1 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)),
    # ...
]
```

12.1.2.10.2 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
```

12.1.3 Development and debugging

12.1.3.1 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.

12.1.3.1.1 Installation

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

12.1.3.1.2 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',
)
```

12.1.3.2 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.

12.2 Filter usage examples

Example usage of filtering backends.

Contents:

Table of Contents

12.2.1 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`.

12.2.1.1 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
```

12.2.1.2 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
```

12.2.1.3 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
```

12.2.1.4 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
```

12.2.2 Filtering

12.2.2.1 Supported lookups

12.2.2.1.1 Native

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

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

12.2.2.1.1.1 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
```

12.2.2.1.1.2 terms

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

12.2.2.1.1.3 range

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

12.2.2.1.1.4 exists

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

12.2.2.1.1.5 prefix

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

12.2.2.1.1.6 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 (*)

12.2.2.1.1.7 ids

Find documents with the specified type and IDs.

12.2.2.1.2 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*

12.2.2.1.2.1 contains

Case-insensitive containment test.

12.2.2.1.2.2 in

In a given list.

12.2.2.1.2.3 gt

Greater than.

12.2.2.1.2.4 gte

Greater than or equal to.

12.2.2.1.2.5 lt

Less than.

12.2.2.1.2.6 lte

Less than or equal to.

12.2.2.1.2.7 startswith

Case-sensitive starts-with.

12.2.2.1.2.8 endswith

Case-sensitive ends-with.

12.2.2.1.2.9 isnull

Takes either True or False.

12.2.2.1.2.10 exclude

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

12.2.3 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

12.3 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

12.3.1 Example app

12.3.1.1 Sample models

books/models/publisher.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()
    latitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)
    longitude = models.DecimalField(null=True,
                                    blank=True,
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)

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

        ordering = ["id"]

    def __str__(self):
        return self.name

    @property
    def location_field_indexing(self):
        """Location for indexing.

        Used in Elasticsearch indexing/tests of `geo_distance` native filter.
        """
        return {
            'lat': self.latitude,
            'lon': self.longitude,
        }
```

12.3.1.2 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()

    # Location
    location = fields.GeoPointField(attr='location_field_indexing')

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

        model = Publisher  # The model associate with this DocType
```

12.3.1.3 Sample serializer

search_indexes/serializers/book.py

```
import json

from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

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

    location = serializers.SerializerMethodField()

    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',
        )

    def get_location(self, obj):
    """Represent location value."""
    try:
        return obj.location.to_dict()
    except:
        return {}
```

12.3.1.4 Sample view

search_indexes/views/publisher.py

```
from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_GEO_DISTANCE,
)
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_indxes.serializers import PublisherDocumentSerializer

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

    document = PublisherDocument
    serializer_class = PublisherDocumentSerializer
```

```
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        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',)
    # Define geo-spatial filtering fields
    geo_spatial_filter_fields = {
        'location': {
            'lookups': [
                LOOKUP_FILTER_GEO_DISTANCE,
            ],
        },
    }
```

12.3.1.5 Usage example

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

12.3.1.5.1 Sample queries

12.3.1.5.1.1 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
```

12.3.1.5.1.2 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
```

Geo-distance filtering

Filter documents by radius of 100000km from the given location.

```
http://127.0.0.1:8000/search/publishers/?location__geo_distance=100000km|12.04|-63.93
```

12.3.1.5.1.3 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
```

12.4 Advanced usage examples

Advanced Django REST framework integration examples.

See the example project for sample models/views/serializers.

- models
- documents
- serializers
- viewsets

Contents:

Table of Contents

- *Page number pagination*
- *Limit/offset pagination*
- *Customisations*

12.4.1 Example app

12.4.1.1 Sample models

books/models/publisher.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()
    latitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)
    longitude = models.DecimalField(null=True,
                                    blank=True,
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)
```

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

        ordering = ["id"]

    def __str__(self):
        return self.name

    @property
    def location_field_indexing(self):
        """Location for indexing.

        Used in Elasticsearch indexing/tests of `geo_distance` native filter.
        """
        return {
            'lat': self.latitude,
            'lon': self.longitude,
        }
```

books/models/author.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
```

books/models/tag.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
```

books/models/book.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

    @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 [tag.title for tag in self.tags.all()]
```

12.4.1.2 Sample document

12.4.1.2.1 Index definition

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

12.4.1.2.1.1 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',
}
```

12.4.1.2.1.2 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),
            'suggest': fields.CompletionField(multi=True),
        },
        multi=True
```

```
    )

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

        model = Book  # The model associate with this DocType
```

12.4.1.3 Sample serializer

search_indexes/serializers/tag.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',)
```

search_indexes/serializers/book.py

```
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."""
        if obj.tags:
            return list(obj.tags)
        else:
            return []
```

12.4.1.4 Sample view

search_indexes/viewsets/book.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,
    DefaultOrderingFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet

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


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

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        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',)
```

12.4.1.5 Usage example

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

12.4.1.5.1 Sample queries

12.4.1.5.1.1 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
```

12.4.1.5.1.2 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
```

12.4.1.5.1.3 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
```

12.4.1.6 Ids filter

Filters documents that only have the provided ids.

```
http://127.0.0.1:8000/api/articles/?ids=68|64|58
```

Or, alternatively:

```
http://127.0.0.1:8000/api/articles/?ids=68&ids=64&ids=58
```

12.4.1.7 Faceted search

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

search_indexes/viewsets/book.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
```

12.4.1.8 Geo-spatial features

For testing the boundaries the following online services might be helpful:

- geojson.io
- Bounding Box Tool

12.4.1.8.1 Filtering

Geo-distance filtering

Filter documents by radius of 100000km from the given location.

```
http://localhost:8000/search/publishers/?location__geo_distance=100000km|12.04|-63.93
```

Geo-polygon filtering

Filter documents that are located in the given polygon.

```
http://localhost:8000/search/publishers/?location__geo_polygon=40,-70|30,-80|20,-90
```

Geo-bounding-box filtering

Filter documents that are located in the given bounding box.

```
http://localhost:8000/search/publishers/?location__geo_bounding_box=44.87,40.07|43.87,
↪41.11
```

12.4.1.8.2 Ordering

Geo-distance ordering

```
http://localhost:8000/search/publishers/?ordering=location|48.85|2.30|km|plane
```

12.4.1.9 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 are `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.

12.4.1.9.1 Completion suggesters

12.4.1.9.1.1 Document definition

To make use of suggestions, you should properly index relevant fields of 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()

    # Location
    location = fields.GeoPointField(attr='location_field_indexing')

    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.

12.4.1.9.1.2 Serializer definition

This is how publisher serializer would look like.

search_indexes/serializers/publisher.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',
        )
```

12.4.1.9.1.3 ViewSet definition

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

search_indexes/viewsets/publisher.py

```
# ...

from django_elasticsearch_dsl_drf.constants import 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_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,
            ],
        },
    }

    # Geo-spatial filtering fields
    geo_spatial_filter_fields = {
        'location': {
            'lookups': [
                LOOKUP_FILTER_GEO_DISTANCE,
            ],
        },
    }
```

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

12.4.1.9.1.4 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
        }
    ]
}
```

12.4.1.9.1.5 Suggestions on Array/List field

Suggestions on Array/List fields (typical use case - tags, where Tag model would be a many-to-many relation to a Book model) work almost the same.

Before checking the *Sample requests/responses*, do have in mind the following:

- `Book` (see the *Sample models*)
- `BookSerializer` (see the *Sample serializer*)
- `BookDocumentView` (see the *Sample view*)

12.4.1.9.1.6 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/books/suggest/?tag_suggest__completion=bio
```

Response

```
{
    "_shards": {
        "failed": 0,
        "successful": 1,
        "total": 1
    },
    "country_suggest__completion": [
        {
            "options": [
                {
                    "score": 1.0,
                    "text": "Biography"
                },
                {
                    "score": 1.0,
                    "text": "Biology"
                }
            ],
            "offset": 0,
            "length": 2,
            "text": "bio"
        }
```

```
    ]
}
```

12.4.1.9.2 Term and Phrase suggestions

While for the `completion` suggesters to work the `CompletionField` shall be used, the `term` and `phrase` suggesters work on common text fields.

12.4.1.9.2.1 Document definition

search_indexes/documents/book.py

```
from django.conf import settings

from django_elasticsearch_dsl import DocType, Index, fields

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
)

@INDEX.doc_type
class BookDocument(DocType):
    """Book Elasticsearch document."""
    # ID
    id = fields.IntegerField(attr='id')

    title = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
            'suggest': fields.CompletionField(),
        }
    )

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

    summary = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField()
        }
    )
```

```
    # Publisher
    publisher = StringField(
        attr='publisher_indexing',
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
            'suggest': fields.CompletionField(),
        }
    )

    # Publication date
    publication_date = fields.DateField()

    # State
    state = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
        }
    )

    # ISBN
    isbn = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
        }
    )

    # Price
    price = fields.FloatField()

    # Pages
    pages = fields.IntegerField()

    # Stock count
    stock_count = fields.IntegerField()

    # Tags
    tags = StringField(
        attr='tags_indexing',
        analyzer=html_strip,
        fields={
            'raw': KeywordField(multi=True),
            'suggest': fields.CompletionField(multi=True),
        },
        multi=True
    )

    null_field = fields.StringField(attr='null_field_indexing')

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

        model = Book  # The model associate with this DocType
```

12.4.1.9.2.2 ViewSet definition

search_indexes/viewsets/book.py

```
from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_PREFIX,
    LOOKUP_FILTER_RANGE,
    LOOKUP_FILTER_TERMS,
    LOOKUP_FILTER_WILDCARD,
    LOOKUP_QUERY_EXCLUDE,
    LOOKUP_QUERY_GT,
    LOOKUP_QUERY_GTE,
    LOOKUP_QUERY_IN,
    LOOKUP_QUERY_IN,
    LOOKUP_QUERY_ISNULL,
    LOOKUP_QUERY_LT,
    LOOKUP_QUERY_LTE,
    SUGGESTER_PHRASE,
    SUGGESTER_TERM,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    # ...
    SuggesterFilterBackend,
)

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

    document = BookDocument
    # serializer_class = BookDocumentSerializer
    serializer_class = BookDocumentSimpleSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SearchFilterBackend,
        SuggesterFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'title',
        'description',
        'summary',
    )
    # Define filter fields
    filter_fields = {
        'id': {
            'field': 'id',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_GT,
                LOOKUP_QUERY_GTE,
                LOOKUP_QUERY_LT,
                LOOKUP_QUERY_LTE,
                LOOKUP_FILTER_TERMS,
            ],
```

```
        },
        'title': 'title.raw',
        'publisher': 'publisher.raw',
        'publication_date': 'publication_date',
        'state': 'state.raw',
        'isbn': 'isbn.raw',
        'price': {
            'field': 'price.raw',
            'lookups': [
                LOOKUP_FILTER_RANGE,
            ],
        },
        'pages': {
            'field': 'pages',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_GT,
                LOOKUP_QUERY_GTE,
                LOOKUP_QUERY_LT,
                LOOKUP_QUERY_LTE,
            ],
        },
        'stock_count': {
            # 'field': 'stock_count',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_GT,
                LOOKUP_QUERY_GTE,
                LOOKUP_QUERY_LT,
                LOOKUP_QUERY_LTE,
            ],
        },
        'tags': {
            'field': 'tags',
            'lookups': [
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
                LOOKUP_QUERY_ISNULL,
            ],
        },
        'tags.raw': {
            'field': 'tags.raw',
            'lookups': [
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
        },
        # This has been added to test `exists` filter.
        'non_existent_field': 'non_existent_field',
        # This has been added to test `isnull` filter.
        'null_field': 'null_field',
    }
```

```
    # 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',)

    # Suggester fields
    suggester_fields = {
        'title_suggest': 'title.suggest',
        'publisher_suggest': 'publisher.suggest',
        'tag_suggest': 'tags.suggest',
        'summary_suggest': 'summary',
    }
```

12.4.1.9.2.3 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.

Let's considering, that one of our books has the following text in the summary:

```
Twas brillig, and the slithy toves
Did gyre and gimble in the wabe.
All mimsy were the borogoves
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in his hand,
Long time the manxome foe he sought --
So rested he by the Tumtum tree,
And stood awhile in thought.
```

12.4.1.9.2.4 Term

Request

```
GET http://127.0.0.1:8000/search/books/suggest/?summary_suggest__term=tovse
```

Response

```
{
    "_shards": {
        "failed": 0,
        "total": 1,
        "successful": 1
    },
```

```
    "summary_suggest__term": [
        {
            "text": "tovs",
            "offset": 0,
            "options": [
                {
                    "text": "tove",
                    "score": 0.75,
                    "freq": 1
                },
                {
                    "text": "took",
                    "score": 0.5,
                    "freq": 1
                },
                {
                    "text": "twas",
                    "score": 0.5,
                    "freq": 1
                }
            ],
            "length": 5
        }
    ]
}
```


12.4.1.9.2.5 Phrase

Request

```
GET http://127.0.0.1:8000/search/books/suggest/?summary_suggest__phrase=slith%20tovs
```

Response


```
{
    "summary_suggest__phrase": [
        {
            "text": "slith tovs",
            "offset": 0,
            "options": [
                {
                    "text": "slithi tov",
                    "score": 0.00083028956
                }
            ],
            "length": 10
        }
    ],
    "_shards": {
        "failed": 0,
        "total": 1,
        "successful": 1
    }
}
```

12.4.1.10 Pagination

12.4.1.10.1 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
```

12.4.1.10.2 Limit/offset pagination

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

search_indexes/viewsets/book.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
```

12.4.1.10.3 Customisations

If you want to add additional data to the paginated response, for instance, the page size, subclass the correspondent pagination class and add your modifications in the `get_paginated_response_context` method as follows:

```
from django_elasticsearch_dsl_drf.pagination import PageNumberPagination


class CustomPageNumberPagination(PageNumberPagination):
    """Custom page number pagination."""

    def get_paginated_response_context(self, data):
        __data = super(
            CustomPageNumberPagination,
            self
        ).get_paginated_response_context(data)
```

```
        __data.append(
            ('current_page', int(self.request.query_params.get('page', 1)))
        )
        __data.append(
            ('page_size', self.get_page_size(self.request))
        )

        return sorted(__data)
```

Same applies to the customisations of the `LimitOffsetPagination`.

12.5 Nested fields usage examples

Advanced Django REST framework integration examples with object/nested fields.

See the example project for sample models/views/serializers.

- models
- documents
- serializers
- viewsets

Contents:

Table of Contents

12.5.1 Example app

12.5.1.1 Sample models

books/models/country.py

```
from django.db import models

from six import python_2_unicode_compatible

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

    name = models.CharField(max_length=255)
    info = models.TextField(null=True, blank=True)
    latitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)
    longitude = models.DecimalField(null=True,
                                    blank=True,
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)

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

        ordering = ["id"]

    def __str__(self):
        return self.name

    @property
    def location_field_indexing(self):
        """Location for indexing.

        Used in Elasticsearch indexing/tests of `geo_distance` native
        filter.
        """
        return {
            'lat': self.latitude,
            'lon': self.longitude,
        }
```

books/models/city.py

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

    name = models.CharField(max_length=255)
    info = models.TextField(null=True, blank=True)
    country = models.ForeignKey('books.Country')
    latitude = models.DecimalField(null=True,
                                   blank=True,
```

```
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)
    longitude = models.DecimalField(null=True,
                                     blank=True,
                                     decimal_places=15,
                                     max_digits=19,
                                     default=0)

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

        ordering = ["id"]

    def __str__(self):
        return self.name

    @property
    def location_field_indexing(self):
        """Location for indexing.

        Used in Elasticsearch indexing/tests of `geo_distance` native filter.
        """
        return {
            'lat': self.latitude,
            'lon': self.longitude,
        }
```

books/models/address.py

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

    street = models.CharField(max_length=255)
    house_number = models.CharField(max_length=60)
    appendix = models.CharField(max_length=30, null=True, blank=True)
    zip_code = models.CharField(max_length=60)
    city = models.ForeignKey('books.City')

    latitude = models.DecimalField(null=True,
                                    blank=True,
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)
    longitude = models.DecimalField(null=True,
                                     blank=True,
                                     decimal_places=15,
                                     max_digits=19,
                                     default=0)

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

        ordering = ["id"]

    def __str__(self):
        return "{} {} {} {}".format(
```

```
            self.street,
            self.house_number,
            self.appendix,
            self.zip_code
        )

    @property
    def location_field_indexing(self):
        """Location for indexing.

        Used in Elasticsearch indexing/tests of `geo_distance` native filter.
        """
        return {
            'lat': self.latitude,
            'lon': self.longitude,
        }
```

12.5.1.2 Sample document

12.5.1.2.1 Index definition

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

12.5.1.2.1.1 Settings

settings/base.py

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

settings/testing.py

```
# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.address': 'test_address',
}
```

settings/production.py

```
# Name of the Elasticsearch index
ELASTICSEARCH_INDEX_NAMES = {
    'search_indexes.documents.address': 'prod_address',
}
```

12.5.1.2.1.2 Document index

search_indexes/documents/address.py

```
from django.conf import settings
```

```
from django_elasticsearch_dsl import DocType, Index, fields
from django_elasticsearch_dsl_drf.compat import KeywordField, StringField

from books.models import Address

from .analyzers import html_strip


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 AddressDocument(DocType):
    """Address Elasticsearch document."""

    # In different parts of the code different fields are used. There are
    # a couple of use cases: (1) more-like-this functionality, where `title`,
    # `description` and `summary` fields are used, (2) search and filtering
    # functionality where all of the fields are used.

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

    # ***********************************************************************
    # ********************** Main data fields for search ********************
    # ***********************************************************************

    street = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
            'suggest': fields.CompletionField(),
        }
    )

    house_number = StringField(analyzer=html_strip)

    appendix = StringField(analyzer=html_strip)

    zip_code = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
            'suggest': fields.CompletionField(),
        }
    )

    # ***********************************************************************
    # ********** Additional fields for search and filtering **************
    # ***********************************************************************

    # City object
```

```
    city = fields.ObjectField(
        properties={
            'name': StringField(
                analyzer=html_strip,
                fields={
                    'raw': KeywordField(),
                    'suggest': fields.CompletionField(),
                }
            ),
            'info': StringField(analyzer=html_strip),
            'location': fields.GeoPointField(attr='location_field_indexing'),
            'country': fields.ObjectField(
                properties={
                    'name': StringField(
                        analyzer=html_strip,
                        fields={
                            'raw': KeywordField(),
                            'suggest': fields.CompletionField(),
                        }
                    ),
                    'info': StringField(analyzer=html_strip),
                    'location': fields.GeoPointField(
                        attr='location_field_indexing'
                    )
                }
            )
        }
    )

    location = fields.GeoPointField(attr='location_field_indexing')

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

        model = Address  # The model associate with this DocType
```

12.5.1.3 Sample serializer

search_indexes/serializers/address.py

```
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from ..documents import AddressDocument


class AddressDocumentSerializer(DocumentSerializer):
    """Serializer for address document."""

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

        document = AddressDocument
        fields = (
            'id',
            'name',
            'info',
            'city',
```

```
            'location',
        )
```

12.5.1.4 Sample view

search_indexes/viewsets/address.py

```
from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_GEO_DISTANCE,
    LOOKUP_FILTER_GEO_POLYGON,
    LOOKUP_FILTER_GEO_BOUNDING_BOX,
    # SUGGESTER_TERM,
    # SUGGESTER_PHRASE,
    SUGGESTER_COMPLETION,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    DefaultOrderingFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    SuggesterFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
)
from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination
from django_elasticsearch_dsl_drf.views import BaseDocumentViewSet

from ..documents import AddressDocument
from ..serializers import AddressDocumentSerializer


class AddressDocumentViewSet(BaseDocumentViewSet):
    """The AddressDocument view."""

    document = AddressDocument
    serializer_class = AddressDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        SearchFilterBackend,
        GeoSpatialFilteringFilterBackend,
        GeoSpatialOrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SuggesterFilterBackend,
    ]
    pagination_class = LimitOffsetPagination
    # Define search fields
    search_fields = (
        'street',
        'zip_code',
        'city.name',
        'city.country.name',
    )
    # Define filtering fields
    filter_fields = {
        'id': None,
```

```
        'city': 'city.name.raw',
        'country': 'city.country.name.raw',
    }
    # Define geo-spatial filtering fields
    geo_spatial_filter_fields = {
        'location': {
            'lookups': [
                LOOKUP_FILTER_GEO_BOUNDING_BOX,
                LOOKUP_FILTER_GEO_DISTANCE,
                LOOKUP_FILTER_GEO_POLYGON,

            ],
        },
    }
    # Define ordering fields
    ordering_fields = {
        'id': None,
        'street': None,
        'city': 'city.name.raw',
        'country': 'city.country.name.raw',
        'zip_code': None,
    }
    # Define ordering fields
    geo_spatial_ordering_fields = {
        'location': None,
    }
    # Specify default ordering
    ordering = (
        'id',
        'street.raw',
        'city.name.raw',
    )

    # Suggester fields
    suggester_fields = {
        'street_suggest': {
            'field': 'street.suggest',
            'suggesters': [
                SUGGESTER_COMPLETION,
            ],
        },
        'city_suggest': {
            'field': 'city.name.suggest',
            'suggesters': [
                SUGGESTER_COMPLETION,
            ],
        },
        'country_suggest': {
            'field': 'city.country.name.suggest',
            'suggesters': [
                SUGGESTER_COMPLETION,
            ],
        }
    }
```

12.5.1.5 Usage example

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

12.5.1.5.1 Sample queries

12.5.1.5.1.1 Search

Just a couple of examples, because searching in nested fields doesn't differ from searching in simple fields.

Search in all fields

Search in all fields (`street`, `zip_code` and `city`, `country`) for word "Picadilly".

```
http://127.0.0.1:8000/search/addresses/?search=Piccadilly
```

Search a single term on specific field

In order to search in specific field (`country`) for term "Armenia", add the field name separated with `|` to the search term.

```
http://127.0.0.1:8000/search/addresses/?search=city.country.name|Armenia
```

12.5.1.5.1.2 Filtering

Filter documents by field

Filter documents by field (`city`) "Dublin".

```
http://127.0.0.1:8000/search/addresses/?city=Dublin
```

Filter documents by multiple fields

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

```
http://127.0.0.1:8000/search/addresses/?city__in=Yerevan|Dublin
```

12.5.1.5.1.3 Ordering

The − prefix means ordering should be descending.

Order documents by field (descending)

Order documents by field `country` (ascending).

```
http://127.0.0.1:8000/search/addresses/?ordering=-country
```

12.5.1.6 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 are `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.

Suggest completion for field `country`.

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

Suggest completion for field `city`.

```
http://127.0.0.1:8000/search/addresses/suggest/?city_suggest__completion=Ye
```

12.6 Various handy helpers

Contents:

Table of Contents

12.6.1 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,
)
```

12.7 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.

0.5.1 — 2017-10-18

- Fixed serialization of complex nested structures (lists of nested objects).
- Documentation fixes.

12.7.1 0.5

2017-10-05

Note: This release contains changes that might be backwards incompatible for your project. If you have used dynamic document serializer `django_elasticsearch_dsl_drf.serializers.DocumentSerializer` with customisations (with use of `serializers.SerializerMethodField`, having the value parsed to JSON), just remove the custom parts.

- Support for `ObjectField`, `NestedField`, `GeoPointField`, `ListField`, `GeoShapeField` (and in general, nesting fields either as a dictionary or list should not be a problem at all).
- Dynamic serializer has been made less strict.
- Added `get_paginated_response_context` methods to both `PageNumberPagination` and `LimitOffsetPagination` pagination classes to simplify customisations.

12.7.2 0.4.4

2017-10-02

- Documentation improvements (Elasticsearch suggestions).
- More tests (term and phrase suggestions).
- Code style fixes.

12.7.3 0.4.3

2017-09-28

- Documentation fixes.
- Fixes in tests.
- Improved factories.

12.7.4 0.4.2

2017-09-28

- Added `geo_bounding_box` query support to the geo-spatial features.

12.7.5 0.4.1

2017-09-26

- Fixes in docs.

12.7.6 0.4

2017-09-26

Note: This release contains changes that might be backwards incompatible for your project. Make sure to add the `DefaultOrderingFilterBackend` everywhere you have used the `OrderingFilterBackend`, right after the latter.

- `GeoSpatialFilteringFilterBackend` filtering backend, supporting `geo_distance` and `geo_polygon` geo-spatial queries.
- `GeoSpatialOrderingFilterBackend` ordering backend, supporting ordering of results for `geo_distance` filter.
- `OrderingFilterBackend` no longer provides defaults when no ordering is given. In order to take care of the defaults include the `DefaultOrderingFilterBackend` in the list of `filter_backends` (after all other ordering backends).

12.7.7 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.

12.7.8 0.3.11

2017-09-21

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

12.7.9 0.3.10

2017-09-20

- Minor fixes.
- Simplified Elasticsearch version check.

12.7.10 0.3.9

2017-09-12

- Python 2.x compatibility fix.

12.7.11 0.3.8

2017-09-12

- Fixes tests on some environments.

12.7.12 0.3.7

2017-09-07

- Docs fixes.

12.7.13 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.

12.7.14 0.3.5

2017-08-24

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

12.7.15 0.3.4

2017-08-23

- Minor fixes in the ordering backend.

12.7.16 0.3.3

2017-07-13

- Minor fixes and improvements.

12.7.17 0.3.2

2017-07-12

- Minor fixes and improvements.

12.7.18 0.3.1

2017-07-12

- Minor Python2 fixes.
- Minor documentation fixes.

12.7.19 0.3

2017-07-11

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

12.7.20 0.2.6

2017-07-11

- Minor fixes.
- Fixes in documentation.

12.7.21 0.2.5

2017-07-11

- Fixes in documentation.

12.7.22 0.2.4

2017-07-11

- Fixes in documentation.

12.7.23 0.2.3

2017-07-11

- Fixes in documentation.

12.7.24 0.2.2

2017-07-11

- Fixes in documentation.

12.7.25 0.2.1

2017-07-11

- Fixes in documentation.

12.7.26 0.2

2017-07-11

- Initial faceted search support.
- Pagination support.

12.7.27 0.1.8

2017-06-26

- Python2 fixes.
- Documentation and example project improvements.

12.7.28 0.1.7

2017-06-25

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

12.7.29 0.1.6

2017-06-23

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

12.7.30 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.

12.7.31 0.1.4

2017-06-22

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

12.7.32 0.1.3

2017-06-21

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

12.7.33 0.1.2

2017-06-20

- Minor fixes in tests.

12.7.34 0.1.1

2017-06-20

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

12.7.35 0.1

2017-06-19

- Initial beta release.

12.8 django_elasticsearch_dsl_drf package

12.8.1 Subpackages

12.8.1.1 django_elasticsearch_dsl_drf.fields package

12.8.1.1.1 Submodules

12.8.1.1.2 django_elasticsearch_dsl_drf.fields.nested_fields module

Nested fields.

class django_elasticsearch_dsl_drf.fields.nested_fields.**GeoPointField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: *django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField*

Geo point field.

class django_elasticsearch_dsl_drf.fields.nested_fields.**GeoShapeField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: *django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField*

Geo shape field.

class django_elasticsearch_dsl_drf.fields.nested_fields.**NestedField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: *django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField*

Nested field.

class django_elasticsearch_dsl_drf.fields.nested_fields.**ObjectField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: rest_framework.fields.Field

Object field.

get_value(*dictionary*)
Get value.

to_internal_value(*data*)
To internal value.

to_representation(*value*)

To representation.

class django_elasticsearch_dsl_drf.fields.nested_fields.**ListField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: rest_framework.fields.Field

List field.

get_value(*dictionary*)
Get value.

to_internal_value(*data*)
To internal value.

to_representation(*value*)
To representation.

12.8.1.1.3 Module contents

Fields.

class django_elasticsearch_dsl_drf.fields.**BooleanField**(***kwargs*)
Bases: rest_framework.fields.BooleanField

Object field.

get_value(*dictionary*)
Get value.

to_representation(*value*)
To representation.

class django_elasticsearch_dsl_drf.fields.**CharField**(***kwargs*)
Bases: rest_framework.fields.CharField

Object field.

get_value(*dictionary*)
Get value.

to_representation(*value*)
To representation.

class django_elasticsearch_dsl_drf.fields.**DateField**(*format=<class 'rest_framework.fields.empty'>*, *input_formats=None*, **args*, ***kwargs*)

Bases: rest_framework.fields.DateField

Object field.

get_value(*dictionary*)
Get value.

to_representation(*value*)
To representation.

class django_elasticsearch_dsl_drf.fields.**FloatField**(***kwargs*)

Bases: rest_framework.fields.FloatField

Object field.

get_value(*dictionary*)
Get value.

to_representation(*value*)
To representation.

class django_elasticsearch_dsl_drf.fields.**GeoPointField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: *django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField*

Geo point field.

class django_elasticsearch_dsl_drf.fields.**GeoShapeField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: *django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField*

Geo shape field.

class django_elasticsearch_dsl_drf.fields.**IntegerField**(***kwargs*)

Bases: rest_framework.fields.IntegerField

Object field.

get_value(*dictionary*)
Get value.

to_representation(*value*)
To representation.

class django_elasticsearch_dsl_drf.fields.**IPAddressField**(*protocol='both'*, ***kwargs*)
Bases: rest_framework.fields.IPAddressField

Object field.

get_value(*dictionary*)
Get value.

to_representation(*value*)
To representation.

class django_elasticsearch_dsl_drf.fields.**ListField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)
Bases: rest_framework.fields.Field

List field.

get_value(*dictionary*)
Get value.

to_internal_value(*data*)
To internal value.

to_representation(*value*)
To representation.

class django_elasticsearch_dsl_drf.fields.**NestedField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)
Bases: *django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField*

Nested field.

class django_elasticsearch_dsl_drf.fields.**ObjectField**(*read_only=False*, *write_only=False*, *required=None*, *default=<class 'rest_framework.fields.empty'>*, *initial=<class 'rest_framework.fields.empty'>*, *source=None*, *label=None*, *help_text=None*, *style=None*, *error_messages=None*, *validators=None*, *allow_null=False*)

Bases: `rest_framework.fields.Field`

Object field.

get_value(*dictionary*)
Get value.

to_internal_value(*data*)
To internal value.

to_representation(*value*)
To representation.

12.8.1.2 django_elasticsearch_dsl_drf.filter_backends package

12.8.1.2.1 Subpackages

12.8.1.2.1.1 django_elasticsearch_dsl_drf.filter_backends.filtering package

12.8.1.2.1.2 Submodules

12.8.1.2.1.3 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_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_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

12.8.1.2.1.4 django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial module

Geo spatial filtering backend.

Elasticsearch supports two types of geo data:

- geo_point fields which support lat/lon pairs
- geo_shape fields, which support points, lines, circles, polygons, multi-polygons etc.

The queries in this group are:

- geo_shape query: Find document with geo-shapes which either intersect, are contained by, or do not intersect with the specified geo-shape.
- geo_bounding_box query: Finds documents with geo-points that fall into the specified rectangle.
- geo_distance query: Finds document with geo-points within the specified distance of a central point.
- geo_distance_range query: Like the geo_distance query, but the range starts at a specified distance from the central point. Note, that this one is deprecated and this isn't implemented.
- geo_polygon query: Find documents with geo-points within the specified polygon.

class django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.**GeoSpatialFilteringF**
Bases: rest_framework.filters.BaseFilterBackend, *django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin*

Geo-spatial filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_GEO_DISTANCE,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialFilteringFilterBackend
>>> )
>>> 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 = [GeoSpatialFilteringFilterBackend,]
>>>     geo_spatial_filter_fields = {
>>>         'loc': 'location',
>>>         'location': {
>>>             'field': 'location',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_GEO_DISTANCE,
>>>             ],
>>>         }
>>> }
```

classmethod apply_query_geo_bounding_box(*queryset*, *options*, *value*)
Apply *geo_bounding_box* 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_geo_distance(*queryset*, *options*, *value*)
Apply *geo_distance* 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_geo_polygon(*queryset*, *options*, *value*)
Apply *geo_polygon* 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

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_bounding_box_params (*value*, *field*)
Get params for *geo_bounding_box* query.

Example:

/api/articles/?location__geo_bounding_box=40.73,-74.1|40.01,-71.12

Example:

/api/articles/?location__geo_polygon=40.73,-74.1|40.01,-71.12
|_name:myname|validation_method:IGNORE_MALFORMED|type:indexed

Elasticsearch:

{

"query": {

"bool" [{]

"must" [{] "match_all" : {}

}, "filter" : {

"geo_bounding_box" [{]

"person.location" [{]

"top_left" [{] "lat" : 40.73, "lon" : -74.1

}, "bottom_right" : {

"lat" : 40.01, "lon" : -71.12

}

}

}

}

}

}

}

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_bounding_box* query.

Return type dict

classmethod get_geo_distance_params(*value*, *field*)

Get params for *geo_distance* query.

Example:
/api/articles/?location__geo_distance=2km|43.53|-12.23

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_distance* query.

Return type dict

classmethod get_geo_polygon_params(*value*, *field*)

Get params for *geo_polygon* query.

Example:
/api/articles/?location__geo_polygon=40,-70|30,-80|20,-90

Example:
/api/articles/?location__geo_polygon=40,-70|30,-80|20,-90
|_name:myname|validation_method:IGNORE_MALFORMED

Elasticsearch:

{

"query": {

"bool" [{]

"must" [{] "match_all" : {}

}, "filter" : {

"geo_polygon" [{]

"person.location" [{]

"points" [[] {"lat" : 40, "lon" : -70}, {"lat" : 30, "lon" : -80}, {"lat" : 20, "lon" : -90}

]

}

}

}

}

}

}

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_distance* query.
Return type dict

classmethod prepare_filter_fields(*view*)
Prepare filter fields.
Parameters **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) –
Returns Filtering options.
Return type dict

12.8.1.2.1.5 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'

12.8.1.2.1.6 Module contents

Term level filtering backends.

class django_elasticsearch_dsl_drf.filter_backends.filtering.**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_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_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

class django_elasticsearch_dsl_drf.filter_backends.filtering.**GeoSpatialFilteringFilterBackend**
Bases: rest_framework.filters.BaseFilterBackend, *django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin*

Geo-spatial filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_GEO_DISTANCE,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialFilteringFilterBackend
>>> )
>>> 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 = [GeoSpatialFilteringFilterBackend,]
>>>     geo_spatial_filter_fields = {
>>>         'loc': 'location',
>>>         'location': {
>>>             'field': 'location',
>>>             'lookups': [
```

```
>>>                     LOOKUP_FILTER_GEO_DISTANCE,
>>>                 ],
>>>             }
>>> }
```

classmethod apply_query_geo_bounding_box(*queryset*, *options*, *value*)
Apply *geo_bounding_box* 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_geo_distance(*queryset*, *options*, *value*)
Apply *geo_distance* 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_geo_polygon(*queryset*, *options*, *value*)
Apply *geo_polygon* 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

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_bounding_box_params(*value*, *field*)
Get params for *geo_bounding_box* query.

Example:
/api/articles/?location__geo_bounding_box=40.73,-74.1|40.01,-71.12
Example:
/api/articles/?location__geo_polygon=40.73,-74.1|40.01,-71.12
|_name:myname|validation_method:IGNORE_MALFORMED|type:indexed
Elasticsearch:
{
"query": {
"bool" [{]
"must" [{] "match_all" : {}
}, "filter" : {
"geo_bounding_box" [{]
"person.location" [{]
"top_left" [{] "lat" : 40.73, "lon" : -74.1
}, "bottom_right" : {
"lat" : 40.01, "lon" : -71.12
}
}
}
}
}
}

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_bounding_box* query.
Return type dict

classmethod get_geo_distance_params(*value*, *field*)
Get params for *geo_distance* query.

Example:
/api/articles/?location__geo_distance=2km|43.53|-12.23

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_distance* query.
Return type dict

classmethod get_geo_polygon_params(*value*, *field*)
Get params for *geo_polygon* query.

Example:
/api/articles/?location__geo_polygon=40,-70|30,-80|20,-90
Example:
/api/articles/?location__geo_polygon=40,-70|30,-80|20,-90
|_name:myname|validation_method:IGNORE_MALFORMED
Elasticsearch:
{
"query": {
"bool" [{]
"must" [{] "match_all" : {}
}, "filter" : {
"geo_polygon" [{]
"person.location" [{]
"points" [[] {"lat" : 40, "lon" : -70}, {"lat" : 30, "lon" : -80}, {"lat" : 20, "lon" : -90}
]
}
}
}
}
}

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_distance* query.
Return type dict

classmethod prepare_filter_fields(*view*)
Prepare filter fields.
Parameters **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) –
Returns Filtering options.
Return type dict

class django_elasticsearch_dsl_drf.filter_backends.filtering.**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'

12.8.1.2.1.7 django_elasticsearch_dsl_drf.filter_backends.ordering package

12.8.1.2.1.8 Submodules

12.8.1.2.1.9 django_elasticsearch_dsl_drf.filter_backends.ordering.common module

Ordering backend.

class `django_elasticsearch_dsl_drf.filter_backends.ordering.common.`**`DefaultOrderingFilterBacke`**
Bases: `rest_framework.filters.BaseFilterBackend`

Default ordering filter backend for Elasticsearch.

Make sure this is your last ordering backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     DefaultOrderingFilterBackend,
>>>     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 = [
>>>         DefaultOrderingFilterBackend,
>>>         OrderingFilterBackend,
>>>     ]
>>>     ordering_fields = {
>>>         'id': 'id',
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'state': {
>>>             'field': 'state.raw',
>>>         }
>>>     }
>>>     ordering = 'name'
```

`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'

class django_elasticsearch_dsl_drf.filter_backends.ordering.common.**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',
>>>         }
>>>     }
>>>     ordering = ('id', 'name',)
```

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_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'

12.8.1.2.1.10 django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial module

Geo-spatial ordering backend.

class django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.**GeoSpatialOrderingFil**
Bases: rest_framework.filters.BaseFilterBackend, *django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin*

Geo-spatial ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialOrderingFilterBackend
>>> )
>>> 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 = [GeoSpatialOrderingFilterBackend,]
>>>     geo_spatial_ordering_fields = {
>>>         'location': {
>>>             'field': 'location',
>>>         }
>>>     }
```

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_geo_distance_params(*value*, *field*)
Get params for *geo_distance* ordering.

Example:
/api/articles/?ordering=-location|45.3214|-34.3421|km|planes

Parameters
- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_distance* query.
Return type dict

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'

12.8.1.2.1.11 Module contents

Ordering backends.

class django_elasticsearch_dsl_drf.filter_backends.ordering.**DefaultOrderingFilterBackend**
Bases: rest_framework.filters.BaseFilterBackend

Default ordering filter backend for Elasticsearch.

Make sure this is your last ordering backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     DefaultOrderingFilterBackend,
>>>     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 = [
>>>         DefaultOrderingFilterBackend,
>>>         OrderingFilterBackend,
>>>     ]
>>>     ordering_fields = {
>>>         'id': 'id',
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'state': {
>>>             'field': 'state.raw',
>>>         }
>>>     }
>>>     ordering = 'name'
```

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'

class django_elasticsearch_dsl_drf.filter_backends.ordering.**GeoSpatialOrderingFilterBackend**
Bases: rest_framework.filters.BaseFilterBackend, *django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin*

Geo-spatial ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialOrderingFilterBackend
>>> )
>>> 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 = [GeoSpatialOrderingFilterBackend,]
>>>     geo_spatial_ordering_fields = {
>>>         'location': {
>>>             'field': 'location',
>>>         }
>>>     }
```

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_geo_distance_params(*value*, *field*)
Get params for *geo_distance* ordering.

Example:
/api/articles/?ordering=-location|45.3214|-34.3421|km|planes

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_distance* query.
Return type dict

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'

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',
>>>         }
>>>     }
>>>     ordering = ('id', 'name',)
```

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_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'

12.8.1.2.2 Submodules

12.8.1.2.3 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*)
Aggregate.

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

12.8.1.2.4 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_complex_value(*value*, *maxsplit=-1*)
Split lookup complex value.
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_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

12.8.1.2.5 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'

12.8.1.2.6 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

12.8.1.2.7 Module contents

All filter backends.

12.8.1.3 django_elasticsearch_dsl_drf.tests package

12.8.1.3.1 Submodules

12.8.1.3.2 django_elasticsearch_dsl_drf.tests.base module

12.8.1.3.3 django_elasticsearch_dsl_drf.tests.data_mixins module

Data mixins.

class django_elasticsearch_dsl_drf.tests.data_mixins.**AddressesMixin**
Bases: object

Addresses mixin.

classmethod created_addresses()
Create addresses.
Returns

class django_elasticsearch_dsl_drf.tests.data_mixins.**BooksMixin**
Bases: object

Books mixin.

classmethod create_books()
Create books.
Returns

12.8.1.3.4 django_elasticsearch_dsl_drf.tests.test_faceted_search module

Test faceted search backend.

class django_elasticsearch_dsl_drf.tests.test_faceted_search.**TestFacetedSearch**(*methodName='runT*
Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

Test faceted search.

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

classmethod setUp()

test_list_results_with_facets()
Test list results with facets.

12.8.1.3.5 django_elasticsearch_dsl_drf.tests.test_filtering_common module

Test filtering backend.

class django_elasticsearch_dsl_drf.tests.test_filtering_common.**TestFilteringCommon**(*methodName=*
Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase, *django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin*, *django_elasticsearch_dsl_drf.tests.data_mixins.BooksMixin*

Test filtering common.

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

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_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

test_nested_field_filter_term()
Nested field filter term.

test_various_complex_fields()
Test various complex fields.
Returns

12.8.1.3.6 django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial module

Test geo-spatial filtering backend.

class django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.**TestFilteringGeoSpatial**(*m*
Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

Test filtering geo-spatial.

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

classmethod setUpClass()
Set up.

test_field_filter_geo_bounding_box()
Test field filter geo-bounding-box.
Returns

test_field_filter_geo_bounding_box_fail_test()
Test field filter geo-bounding-box (fail test).
Returns

test_field_filter_geo_distance()
Field filter geo-distance.

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

test_field_filter_geo_polygon()
Test field filter geo-polygon.
Returns

test_field_filter_geo_polygon_fail_test()
Test field filter geo-polygon (fail test).
Returns

test_field_filter_geo_polygon_string_options()
Test field filter geo-polygon.
Returns

test_field_filter_geo_polygon_string_options_fail_test()
Test field filter geo-polygon (fail test).
Returns

12.8.1.3.7 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' {'args': (), 'kwargs': {}}>, <MarkDecorator 'django_db' {'args': (), 'kwa

classmethod setUpClass()

test_filter_by_field()
Filter by field.

12.8.1.3.8 django_elasticsearch_dsl_drf.tests.test_ordering_common module

Test ordering backend.

class django_elasticsearch_dsl_drf.tests.test_ordering_common.**TestOrdering**(*methodName='runTest'*)
Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

Test ordering.

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

classmethod setUpClass()
Set up class.

test_author_default_order_by()
Author order by default.

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_default_order_by()
Book order by default.

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.

12.8.1.3.9 django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial module

Test geo-spatial ordering filter backend.

class django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial.**TestOrderingGeoSpatial**(*met*
Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

Test ordering geo-spatial.

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

classmethod setUpClass()
Set up.

test_field_filter_geo_distance()
Field filter geo_distance.

Example:
http://localhost:8000 /api/publisher/?ordering=location|48.85|2.30|km|plane

12.8.1.3.10 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' {'args': (), 'kwargs': {}}>, <MarkDecorator 'django_db' {'args': (), 'kwa

classmethod setUpClass()
Set up class.

test_pagination()
Test pagination.

12.8.1.3.11 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' {'args': (), 'kwargs': {}}>, <MarkDecorator 'django_db' {'args': (), 'kwa

classmethod setUp()

test_search_by_field()
Search by field.

12.8.1.3.12 django_elasticsearch_dsl_drf.tests.test_suggesters module

12.8.1.3.13 django_elasticsearch_dsl_drf.tests.test_views module

12.8.1.3.14 Module contents

12.8.2 Submodules

12.8.3 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'

12.8.4 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

12.8.5 django_elasticsearch_dsl_drf.constants module

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

12.8.6 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

12.8.7 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

get_paginated_response_context(*data*)
Get paginated response data.
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

get_paginated_response_context(*data*)
Get paginated response data.

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*, *allow_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

12.8.8 django_elasticsearch_dsl_drf.serializers module

12.8.9 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.

12.8.10 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.

12.8.11 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.

12.8.12 Module contents

Integrate Elasticsearch DSL with Django REST framework.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

A

B

C

D

S

T